© 2015 IJSRSET | Volume 1 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Profile Based Concurrent Data Download – Cloud, Data Sharing & Load Balancing

M. Gayathri, K. Jayanth, S. Gubendhirapandian, P. Ashokkumar
Computer Science and Engineering, Dhanalakshmi College of Engineering, Chennai, Tamilnadu, India

ABSTRACT

In the EXISTING SYSTEM each processing node will partition a data set independently. There is no data sharing. In the PROPOSED SYSTEM, we are developing Two Techniques namely Data Download & Data Sharing. In Data Download Model, Priority based Retrieval is achieved based on the Query. The requested data is downloaded from different Servers as the Data are partitioned. In Data sharing, the data are divided into different chunks and stored as threads in the partition matrix. From the partition matrix the data will be retrieved for the read/write purpose without overlapping. The MODIFICATION part of the project is to implement Real Time Cloud (Drop Box) along with load balancing &Automatic and Continuous Data retrieval. Data sharing and download is achieved as said in proposed system, except data is encrypted using AES. Data is partitioned into different sub cloud servers (SCS). For Example 'DO' Splits the File into 8 Parts, and starts uploading to the Cloud. During uploading Process, a High Priority user sends File request to the Cloud then Cloud will Start Transmitting the Data even before entire content is stored in the cloud. We implement Load Monitoring system among SCS.

Keywords: Cloud Deployment Module, Priority based Profile Filtering Module, Data Encryption and Chunking Module, Drop box Setup Module, Concurrent Data Transfer Module, Load Balancing and Data Delivery Module.

I. INTRODUCTION

Parallel database systems [1], [2] 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, however, the underlying system architecture should be conducive. To that 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 enforced either at load-time, with data pre-processed 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 enclosing multiple processing cores in a single chip. However, they differ from shared-nothing machines as there is nothing in the execution model enforcing that all cores 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 concept 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 paradigm 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 pros and cons, some of which generically appear in any type of processing, whereas

others manifest specifically in 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 identifying 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 based on 1) a combination of data partitioning and task based processing for effective parallelization, 2) managing memory in a way that eliminates the need for per-thread memory pools while, at the same time, allows the system to scalability cater for the memory needs of the concurrently executing threads, and 3) data structures that are efficiently manipulated by multiple threads without any need for complex synchronization. Having such a design in place allows us to implement efficient multithreaded versions of the fundamental query processing algorithms, namely selections, projections, partitioning, sorting, join evaluation, and aggregation. For each algorithm we present multiple 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 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.

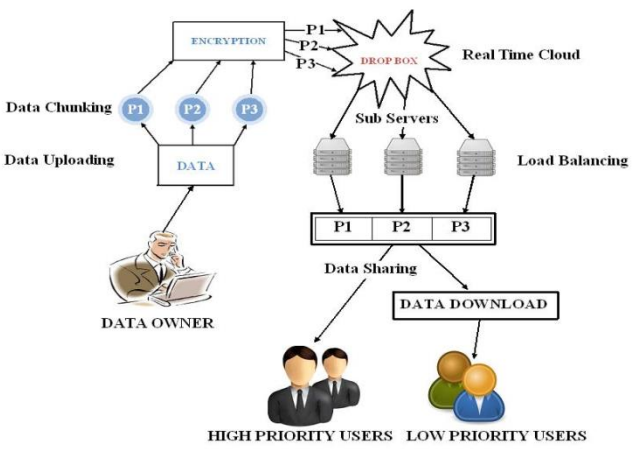


Figure 1: The System

II. METHODS AND MATERIAL

A. Cloud Deployment Module

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/ Data Owner information to authenticate when they want to login into their account. The User / Data Owner information will be stored in the Database of the Cloud Service Provider. Also the Cloud Server will redirect the User requested job to the any of the Queue to process the User requested Job. The Request of all the Users will process by the Virtual Machines in the Queue. To communicate with the Client and the with the other modules of the Cloud Network, the Cloud Server will establish connection between them. For this Purpose we are going to create a User Interface Frame. Also the Cloud Service Provider will send the User Job request to the Queues in Fist in First out (FIFO) manner.

B. Priority Based Profile Filtering Module

In this project we introduce a concept of priority based profile filtering that is we have two type profile in our project first one is paid user and the second one non paid user .so based on the profile the user can able to download and upload the data. By using this technique we can easily reduce the burden of the cloud server to provide the service.

Multicore systems and multithreaded processing are now the de facto standards of enterprise and personal computing. If used in an uninformed way, however, multithreaded processing might actually degrade performance. We present the facets of the memory access bottleneck as they manifest in multithreaded processing and show their impact on query evaluation.

We present a system design based on partition parallelism, memory pooling, and data structures conducive to multithreaded processing. Based on this design, we present alternative implementations of the most common query processing algorithms, which we experimentally evaluate using multiple scenarios and hardware platforms. Our results show that the design and algorithms are indeed scalable across platforms, but the choice of optimal algorithm largely depends on the problem parameters and underlying hardware.

C. Data Encryption And Chunking Module

In this module we are going describe about the chunking, that is chunking [8], [9] mean a large file is split into eight parts of files and they have been stored in different server. The use of splitting is to reduce the load of server and to provide quality of service to the cloud user while they were uploading or downloading files .And another thing we are implement security to the file while we split form main cloud, we are going to encrypted file AES algorithm to provide security to the cloud owner. Before you upload to cloud the data is encrypted and then it is chunked. Data that is passed through the de-duplication engine is chunked into smaller units and assigned identities using cryptographic hash functions. Thereafter, two chunks of data are compared to ascertain whether they have the same identity. If the answer is yes, a link to the data (and not the chunk), is included in the incremental backup. If the answer is no, it is accepted as an independent chunk of data and uploaded to the backup server. While some algorithms accept the premise and create algorithms for identifying similar chunks of data based on similar identities, other algorithms take into consideration the pigeonhole concept while designing the algorithm. The pigeonhole principle in mathematics and computer science states that if "n" items are put into "m" pigeonholes with n>m, then at least one pigeon hole must contain more than one item. Chunking for de- duplication can be frequency based or content based. Frequency based chunking identifies high frequencies of occurrences of data chunks. The algorithm uses this frequency information to enhance data duplication gain. Content based chunking is a stateless chunking algorithm which partitions a long stream of data into smaller units or chunks and removes duplicate ones. However, this type of algorithm is random and does not provide performance guarantee. Commercial implementations of de-duplication vary primarily in the chunking method and architecture that is used. Some online backup algorithms chunk data according to physical layer constraints (for instance a 4KB block size); others only use complete files (as in single instance storage algorithms) as data chunks. But, the most intelligent, though CPU intensive, chunking methodology is considered to be the sliding block methodology. In this methodology, a window is passed along the file stream to identify the natural internal file boundaries.

D. Advanced Encryption Standard Algorithm

For encryption, each round consists of the following four steps: 1) Substitute bytes, 2) Shift rows, 3) Mix columns, and 4) Add round key. The last step consists of XORing the output of the previous three steps with four words from the key schedule. For decryption, each round consists of the following four steps: 1) Inverse shift rows, 2) Inverse substitute bytes, 3) Add round key, and 4) Inverse mix columns. The third step consists of XORing the output of the previous two steps with four words from the key schedule. Note the differences between the order in which substitution and shifting operations are carried out in a decryption round vis-a-vis the order in which similar operations are carried out in an encryption round. The last round for encryption does not involve the "Mix columns "step. The last round for decryption does not involve the "Inverse mix columns" step.

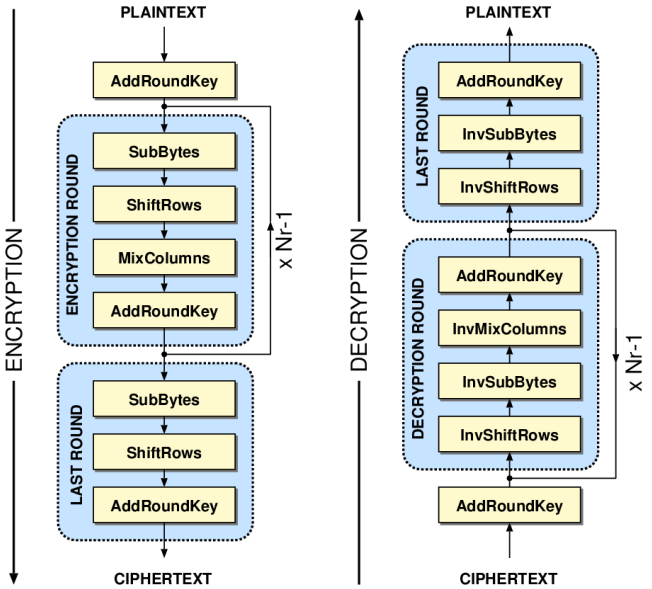


Figure 2: Encryption Standard Algorithm

E. Dropbox Setup Module

In this module we implement the real cloud as Drop box. Drop box is a file hosting service operated by Drop box software. Dropbox allows users to create a special folder

on each of their computers, which Dropbox then synchronizes so that it appears to be the same folder (with the same contents) regardless of which computer is used to view it. Files placed in this folder also are accessible through a website and mobile phone applications.

F. Concurrent Data Transfer Module

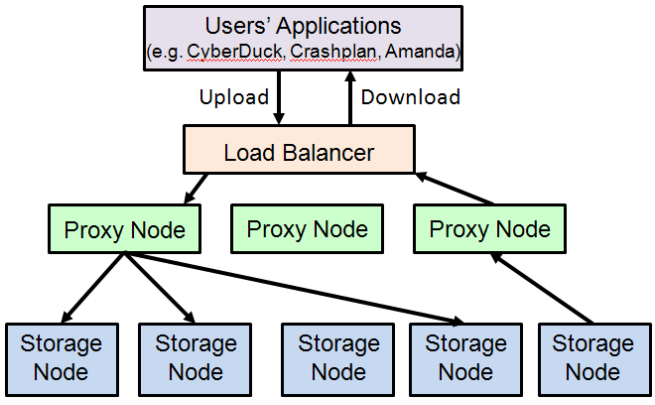
In this module we going to implement the Concurrent data transfer processing in a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent [5] means something that happens at the same time as something else. Tasks are broken down into subtasks that are then assigned to separate processors to perform simultaneously, instead of sequentially as they would have to be carried out by a single processor. Concurrent processing is sometimes said to be synonymous with parallel processing. Here paid user download and upload the files at the same time so that quality of service are achieved in this project by introducing the concurrent data transfer.

G. Secure Data Sharing In Cloud

Cloud systems can be used to enable data sharing capabilities and this can provide an abundant of benefits to the user. The benefits organisations can gain from data sharing are higher productivity. With multiple users from different organisations contributing to data in the Cloud, the time and cost will be much less compared to having to manually exchange data and hence creating a clutter of redundant and possibly out-of-date documents. Some of major requirements of secure data sharing in the Cloud are as follows. Firstly the data owner should be able to specify a group of users that are allowed to view his or her data. Any member within the group should be able to gain access to the data anytime, anywhere without the data owner's intervention. No-one, other than the data owner and the members of the group, should gain access to the data, including the Cloud Service Provider. The data owner should be able to add new users to the group. The data owner should also be able to revoke access rights against any member of the group over his or her shared data. No member of the group should be allowed to revoke rights or join new users to the group.

H. Load Balancing And Data Delivery Module

Load balancing [6] is the process of reassigning the total loads to the individual nodes of the collective system to make the best response time and also good utilization of the resources. Cloud computing is an internet computing in which the load balancing is the one of the challenging task. Various methods are to be used to make a better system by allocating the loads to the nodes in a balancing manner but due to network congestion, bandwidth usage etc., there were problems are occurred. These problems were solved by some of the existing techniques. A load balancing [10] algorithm which is dynamic in nature does not consider the previous state or behavior of the system, that is, it depends on the current behavior of the system. There were various goals that related to the load balancing such as to improve the performance substantially, to maintain the system stability. Once the User send the request to process the job, the Cloud Service Provider will pass the request to the any of the sub cloud server in the Cloud Server Provider. Each three data owner has one database based request they retrieve the data via cloud server. So that the User requested Job will be assigned to the available sub cloud server via cloud service provider which contains minimum load and the concerned sub cloud server will process the User requested Job. The load balancing strategy involves creating cloud partitions. A cloud partition is a sub area of public cloud. Here the divisions are based on the VDBSCAN algorithm. Once the public cloud is partitioned, then the load balancing starts when a job arrives at the system.



The main controller decides which cloud partition should receive the job. The load balancer assigned for each partition then decides how to assign the jobs to the nodes. When the load status of a cloud partition is idle or normal, this task can is accomplished locally. If the

cloud partition is overloaded, this job should be transferred to another partition.

III. RESULTS AND DISCUSSION

A. Feasibility Study

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

B. Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

C. Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

D. Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity.

IV. 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: 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. That the less uniform a distribution is, the more appropriate techniques like size-bound partitioning become; 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 count-partitioning will likely perform well. 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.

V. REFERENCES

[1] R. Acker et al., "Parallel Query Processing in Databases on Multicore Architectures," Proc. Eigth Int'1 Conf. Algorithms and Architecture Parallel Processing (ICA3PP), 2008.

[2] D.A. Alcantara at 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'1 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'1 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., "Load Balancing for Parallel Query Execution on NUMA Multiprocessors," Distributed and Parallel Databases, vol. 7, no. 1, pp. 99-121, 1999.

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

[8] K. Eshghi and H. Tang, "A framework for analyzing and improving content- based chunking algorithms", Hewlett-Packard Labs Technical Report TR. 30, (2005).

[9] L. Ramshaw and M. Marcus, "Text Chunking Using Transformation- Based Learning," In: D. Yarovsky and K. Church, Eds., Proceedings of the Third Workshop on Very Large Corpora, Association for Computational Linguistics, Somerset, 1995, pp. 82-94.

[10] A.K. Sidhu, S. Kinger, "Analysis of Load Balancing Techniques in Cloud Computing", International Journal of Computers & Technology,Volume 4 No. 2, ISSN 2277-3061, March-April, 2013.