ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (157-163), Month: January - March 2015, Available at: www.researchpublish.com

Customized Stream Query Scheduling In Parallel Database System

[1]Mr. Niraj Kumar, [2]Mr. Mahadeo Prasad, [3]Mr. Rajesh Singh

[1,2,3] Assistant Professor, Department of Computer Science & Engineering,
RTC Institute of Technology, Anandi, Ormanjhi, Ranchi (Jharkhand), India

Abstract: **The growth of World Wide Web requires large database the organizations are using large volume of data. A number of commercial and research systems application required power and scalability of parallel query processing. Web-based applications, such as e-commerce sites, are faced with highly changeable workloads. The number of customers browsing and purchasing items varies through-out the day and business managers can further complicate the workload by requesting complex reports on sales data. This means the load on a database system can vary considerably with a sudden arrival of requests or a request involving a complex query. If there are too many requests operating in the DBMS concurrently, then resources are stressed and performance drops. To keep the DBMS's performance consistent across varying loads, a load control system can be used. We focus on scheduling of queries for parallel database systems by dividing the workload into batches. We propose scheduling algorithms which exploit the common operations within the queries in a batch. One scheduling algorithm cannot optimally meet an arbitrary set of Quality of Service (QoS) requirements. Therefore, to meet unique features of specific monitoring applications, an adaptive strategy selector guidable by QoS requirements was developed. The adaptive algorithm is general, being able to use any scheduling algorithm and to react to any combination of quality of service preferences.**

1. INTRODUCTION

Database Management Systems (DBMSs) are the primary tools used for storing and accessing data and they are the backbone of many applications. A single DBMS can receive many concurrent requests which it must handle. The type of requests a database receives, also called the workload, can vary. These two situations represent the two fundamental database workload types: online transaction processing (OLTP) and online analytical processing (OLAP)|also referred to as Business Intelligence (BI). An OLTP workload is characterized by many short transactions and numerous updates. For instance, an update of an inventory number is quick and needs to only touch a very small amount of data. OLAP workloads on the other hand, usually consist of longer, more resource intensive queries. They tend to require reading large amounts of data and more complex processing (sorting, finding the maximum, calculating totals). Certainly, these workloads are not always distinct and it is possible to have both OLTP and OLAP type requests acting on a single database. Since a database has limited physical resources such as CPU and memory, there is a limit to the number of requests it can process concurrently. Too many concurrent requests lead to resource contention, which can cause the performance of the database to drop drastically. The number of requests that a database is able to handle depends on a variety of factors such as the system hardware, the system configuration and the workload.

2. MOTIVATION

Controlling load on a DBMS is not an easy task since not all requests are equal in the amount of resources they require. Setting a static limit for the total number of requests that are allowed to execute may work well if requests are relatively equal in their resource requirements, but will lead to suboptimal performance if the requests are extremely varied, for instance, a mix of OLTP and OLAP queries. Beyond the amount of resource demand, queries can also differ in the type of resources they require. For instance, I/O intensive queries primarily read data, CPU intensive queries require a lot of

calculations, and memory intensive queries may store many partial results. If the mix of queries being executed is not balanced, then one resource may be overloaded while others are idle. A load control system should be able to effectively handle all of these factors and adapt to current conditions. DBMS has to be able to manage these complex queries at any time and be able to perform optimally no matter what type, or how many queries are presented.

3. RESEARCH STATEMENT

The objective of our research is to investigate the feasibility of a database load control system based on regulating individual resource consumption in a predictive manner. A significant difference between our work and previous proposals is that compile time optimization techniques tend to ignore issues of resource allocation. We focus on scheduling of queries for parallel database systems by dividing the workload into batches. In general, a customized system is a system that changes its behavior in response to a changing environment with the goal of improving performance [4].The adaptive scheduler selector will periodically evaluate the current scheduling algorithm's performance for the administration-specified QoS requirements and compare this with the other candidate algorithms' performance. This qualitative comparison is based upon assigning a fitness score [13] to each algorithm that captures how well it performed in several metrics, such as throughput, memory size, and output rate.

4. DATABASE LOAD CONTROL SYSTEM

Load control in a database system can be achieved through admission control and scheduling. Admission control limits the number of queries that can enter the system to avoid resource contention. Scheduling, with respect to load control, means selecting which queries to execute so that resource contention is minimal. Several load control approaches that focus on controlling resource contention are presented in the following subsections. Other approaches, such as the work by Niu et al. [14] and Brown et al. [15] focus on attaining service level objectives (SLOs) for different groups of queries by controlling access to physical resources.

4.1 Load Control Balancing:

The goal of a load control system is to keep a database system running efficiently, even under heavy and variable loads. This can be achieved through a variety of methods. We approach the problem of load control by considering the demand on individual resources. A DBMS has limited physical resources and excess demand on these resources can lead to poor performance. Therefore, resource demand should be regulated. We study the feasibility of this kind of load control approach by focusing on the sort heap as a resource. We have implemented a prototype load control system which schedules queries according to their sort heap requirements. Three different scheduling methods are proposed. Each of these scheduling methods acts as a gate-keeping mechanism, only executing those queries whose sort heap requirement fit into the currently available sort heap space. When more than the available amount of sort heap is demanded, sort heap contention arises. This means that the amount of sort heap space that some of the queries are allowed to use is less than the amount required by the query. This leads to slower execution time. Without enough sort heap memory, partial results of a sort or hash-join may have to be written to disk, which is a costly operation. Hence, the goal of our load control system is to limit the number of concurrently running queries. So that their combined sort heap requirement does not exceed sort heap space.

4.2 Basic Schedulers:

Three scheduling methods are proposed:

Blocking Queue Scheduler (BQS):

The Blocking Queue Scheduler's functionality consists solely of gate keeping. All the queries that enter the system are put in a queue in the order they arrived. The query at the front of the queue is only executed if there is enough sort heap space for it. It follows a first-in-first-out (FIFO) policy. If there is not enough space, the scheduler waits until enough space is available. The advantages of this scheduler are that it is very simple to implement, there is very little overhead, and the issue of starvation| when a query is never executed |is avoided. The main disadvantage is that it is not flexible in terms of being able to pick which query runs next. There may be a query in the queue for which there is enough sort heap space available, but it cannot be run until it is at the front of the queue.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (157-163), Month: January - March 2015, Available at: www.researchpublish.com

Smallest" Job First Scheduler (SJFS):

An alternative to the FIFO policy is a shortest-job-first policy. We modify this policy to a smallest-job-first policy. This means ordering the incoming queries by their sort heap requirements from smallest to largest and then performing gate keeping just like the Blocking Queue Scheduler. The advantage of this approach is that if a query fits into the currently available sort heap, it will be allowed to run. However, this type of scheduling induces more overhead than BQS since the waiting queries need to be sorted. Also, there is the risk of starvation since queries are re-ordered when new ones arrive.

First Fit Scheduler (FFS):

The First Fit Scheduler keeps a list of all the queries that have been submitted to the system in the order they were submitted. It traverses through this list until a query whose sort heap requirement is less than or equal to the currently available sort heap space is found. Once found, the query is executed and removed from the list. Then, the search for the next query to execute is repeated from the beginning of the list. A first-fit approach was chosen rather than a best-fit since the available sort heap space is constantly changing; by the time the best-fit query is found, it may no longer be the best fit. Therefore, the extra overhead involved in finding the best fit brings little benefit.

The advantage of this scheduler is that, like SJFS, if there is a query for which there is enough sort heap space, it will be executed. However, FFS is more likely to run a balanced mix of queries than SJFS, since queries of all sizes are considered for execution, not just the one requiring the least sort heap. Nevertheless, of all the proposed schedulers, FFS is the one that requires the largest overhead since the list of waiting queries is constantly traversed. This is not a problem as long as long as the list of waiting queries is small. However, under very heavy loads the waiting query list could get very long. In these cases, overhead could be reduced by only considering the first n queries in the list. This way, the overhead of searching for the next query to execute is constant, no matter how heavy the workload. FFS is also susceptible to starvation. In order to assess the effectiveness of each of the proposed scheduling methods, a prototype external load control system was implemented. An overview of this system is shown in Figure 1. However, this overhead of maintaining the query queue is minimal when compared to the overhead of retrieving the query plans.

5. BATCH SCHEDULING SYSTEM

We assume that the workload for the database system consists of batches of queries and that each query is composed of several operators. In addition, there is a partial order defined on the operators in a query. For example, the probe phase of a hash join operator cannot begin until the build phase has completed. There are several ways in which the operators from different queries can be combined into a single schedule. The aim of our scheduling algorithms is to find the global schedule for all queries that minimizes the total execution time for a batch of queries without violating the partial order constraints. All the algorithms operate on a query graph defined on the queries in a batch. The nodes of this graph are the relations accessed by at least one query.

There exists an edge from node Ri to node Rj for every query Q that references relations i and j. A batch of queries defines a graph which is a collection of disjoint connected sub graphs. Each sub graph represents a subset of the queries in the batch which share some relation with other queries in the same sub graph. Consequently, sharing of operators is possible only within a connected sub graph. Figure 2 shows an example of a query graph for a batch of 7 queries divided into 3 sub graphs.

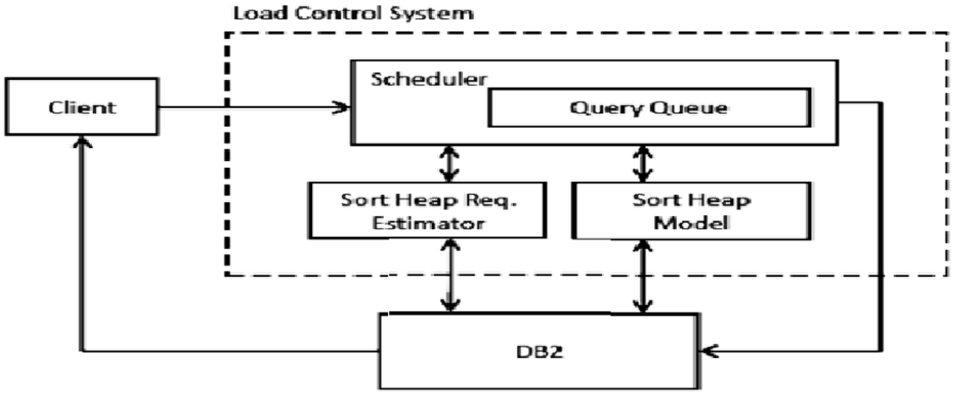

Figure 1: Architecture of the Load Control System

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (157-163), Month: January - March 2015, Available at: www.researchpublish.com

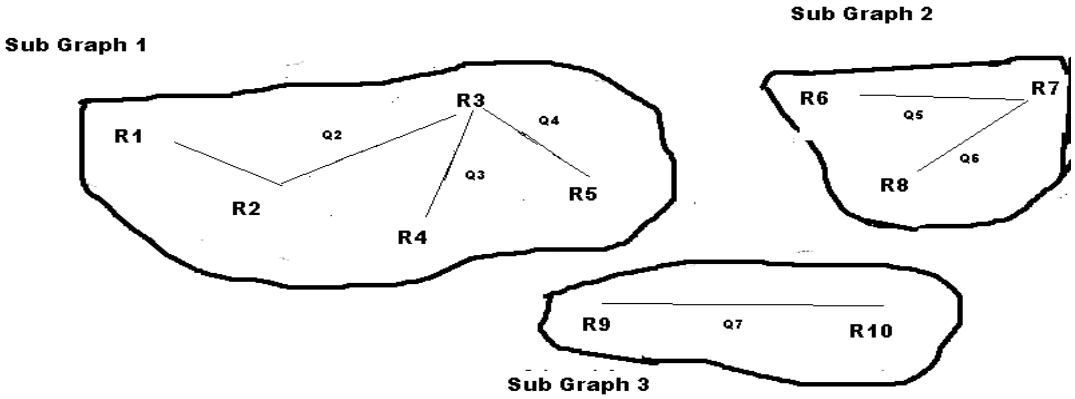

Figure 2: Simple query graph

5.1 Batch Computing:

Batch computing refers to a system of computing in which a user does not directly dispatch programs interactively for execution but rather delegates this responsibility to a batch scheduling system. The batch scheduler then in turn dispatches the programs for execution, monitors their status, and returns their output upon completion to

the user. Although batch schedulers are typically software programs themselves, and we will refer to them as such in this dissertation, this is not necessarily the case; in fact, in the early days of card reading computers, the "batch scheduler" was often a human operator responsible for feeding cards into a computer [8].

Due to this separation between the user and the programs, batch computing is neither generally well-suited nor designed for applications which require frequent interactions with the users. More appropriate are long running programs that do not require user input after initialization. Batch schedulers are appreciated by users because they assume the drudgery of program dispatching and monitoring and free the user for more creative endeavors. In addition to being useful for executing and monitoring long running programs, batch schedulers are useful when users have multiple programs to execute. Here again, batch schedulers can assume the drudgery and the time-consuming process of dispatching, monitor, and collecting the output of these multiple programs.

Finally, batch schedulers are extremely useful in distributed computational settings where they can dispatch multiple programs in parallel across multiple computational resources. In such a case, in addition to its other duties of dispatching, monitoring, and collecting output, the batch scheduler is also responsible for monitoring a collection of computational resources and implementing a scheme for matching computational resources with the programs that need them.

6. ADAPTIVE SCHEDULING

There are several scheduling algorithms for execution scheduling of query operators. The Execution Engine will ask a scheduler to choose the next operator to run and to determine its workload. After the operator is run, the controller may decide to choose another scheduling algorithm if it deems the current algorithm is not meeting the user's QoS requirements for execution behavior. There is a growing trend to provide parallel scientific computation services through the web interface, especially for computation- and data-intensive tasks such as scientific database queries, data mining, and visualization. Rather than having users download large volumes of shared data and run stand-alone applications, scientific web services allow them to perform common data processing/ analysis tasks through intuitive web interfaces. For example, an online bio-sequence search service can be viewed as the equivalent of web search engine in the bioinformatics world.

A scheduling algorithm is responsible for two tasks: choosing the operator to run next and assigning a workload to that operator. The next operator decision depends on the algorithm itself while the workload assignment is often fixed regardless of the scheduler. In Raindrop, the workload assignment is controlled by two administrator-defined parameters. The first parameter, RATIO, is the ratio of tuples that an operator should dequeues relative to the total number tuples available. Currently this ratio is fixed for each strategy, but future work could adapt this depending on statistics. The

second parameter, THRESHOLD, aids in calculating how much work to assign to an operator. It aims to reduce the chances that an operator is underutilized by setting a limit for when to use the RATIO and when to use the total number of tuples available. Illustrates the intuition by Pseudo code for determining operator workload.

N = the number of tuples in operator O's input queue A = N x RATIO

if A > THRESHOLD

Then O dequeues A tuples.

Else

Then O dequeues N tuples

We now describe several scheduling strategies employed by our adaptive scheduling framework, and explain their advantages and disadvantages.

6.1 Round Robin:

Round Robin (RR) is perhaps the most basic scheduling algorithm. It works by placing all run able operators in a circular queue and allocating a fixed time slice to each. Round Robin's best quality is the avoidance of starvation. An operator is guaranteed to be scheduled within a fixed period of time. In fact, as long as an operator always has worked to do,

no operator will be run more times than any other. However, Round Robin does not adapt at all to changing stream conditions. It also does not consider many possibly important factors, such as an operator's performance relative to other operators, size of the input queues, or the selectivity. Therefore, the intermediate queue sizes can grow rapidly be-

cause RR may spend its time running other operators that have less work to do or are less favorable for other reasons.

6.2 FIFO:

FIFO (first in first out) chooses a leaf operator to execute and attempts to push its tuples through the system as far as possible. FIFO typically yields a consistent throughput, because it tries to execute older tuples until completion before it considers newly arrived tuples. But it has the same drawbacks as Round Robin - no addictiveness and no consideration of operator properties.

6.3 Greedy:

Greedy scheduling assigns a priority to each operator and always tries to run the operator with the highest priority. the operator with the highest priority has no work to do (i.e. empty input queues), Greedy will choose the next highest priority. The priority, calculated dynamically, was originally shown in [3]. Greedy eliminates some of the drawbacks of Round Robin because it considers the cost of each operator before choosing which operator to run. However, it is prone to starvation. If the high priority operator, O, is proceeded by lower priority operators,

6.4 Most Tuples in Queue:

The Most Tuples in Queue (MTIQ) scheduler is a greedy algorithm that assigns a priority to each operator equivalent to the number of the tuples in its input queues. MTIQ is a simplified batch scheduler similar to [5]. Batch schedulers work under the assumption that the average tuple processing cost can be reduced if an operator works on more tuples at a time. Operators typically have a start-up cost associated with their execution and the batch scheduler can amortize this cost over a larger group of tuples. Round Robin and FIFO do not have this property and thus those algorithms tend to under-utilize operators. Second, MTIQ tends to have a burst output pattern. Typically it takes a relatively long period of time for enough tuples to make it through the system such that the root operator has more work to do than the operators below it. However, when the root operator runs, it then will output a large block of tuples. Some tuples will experience little delay while others will be enquired for long periods of time, but on average, the mean delay will not be much worse than the other algorithms. The most obvious advantage is that MTIQ works well at minimizing memory consumption. By running the operator with the most tuples enqueued, the algorithm will have a better chance than the previous algorithms at ensuring that no queue will grow unbounded. If the data arrives faster than MTIQ can process it, then that queue will grow infinite in size.

Algorithm	Advantages	Disadvantages
Round Robin	Guarantee that every operator is scheduled	Over time poor output rate
FIFO	Schedules operator with the same frequency Output tuples sooner and at a constant rate	Queue size grow quickly Output rate is low Does not utilize operator as fully as gready
MTTQ	Queue sizes are smaller Higher output rate Fully utilizes operators	Brusly output pattern Tuples takes long time in the system

7. CONCLUSIONS

As DBMS workloads are becoming more complex, effective load management systems are needed. Resource-aware load management systems are one way to handle the varying resource requirements of queries. The objective of this thesis is to investigate the feasibility of a database load control system based on regulating resource consumption in a predictive manner.

This Paper addressed the issues relating to creating an adaptive execution strategy for the execution of a continuous query over streaming data. The proposed adaptive strategy chooses the next scheduling algorithm to utilize among several candidate algorithms based on their performance thus far relative to the user's quality of service requirements. We also showed that the user's service preferences do in fact have an effect on the behavior of the adaptive algorithm. In our study, the adaptive algorithm that was optimized for a given metric outperformed the other adaptive algorithm that was optimized for an other metric. This is an important conclusion because it shows that the adaptive algorithm behaves intelligently and does not win simply because it combines the other algorithms. Given the presence of a single algorithm that optimally met the requirement, the adaptive strategy chose that algorithm more than the other. When the adaptive algorithm periodically switched to one of the other candidates for exploratory purposes, the additive's overall performance decreased. Thus, the adaptive was never able to outperform that single strategy.

8. FUTURE WORK

There are many future topics to investigate based on the preliminary results produced By this Paper. The first direction involves augmenting the experimental study with additional data distributions and more complex query plans. Another direction involves tweaking the various experiment parameters. Further testing to find the optimal values for the weight to give to old values for weighted average, workload ratio, and frequency of updating statistics should result in improved performance. The adaptive strategy can be further tweaked by altering the data decay and algorithm switch parameters or by running multiple operators at the same time. Another direction involves investigating incorporating alternate adaptive techniques such as those used in [5][9]. Combining these techniques with the adaptive scheduling strategy yields an interesting research question - could we find a formula to weigh the benefits of one technique over the other and always choose the adaptive technique that will meet the user's quality of service best.

REFERENCES

[1] I. T. Archive. http://www.acm.org/sigcomm/ita/, 2003.

[2] R. Avnur and J. M. Hellerstein. Eddies: continuously adaptive query processing. In SIGMOD Conference 2000, pages 261-272, 2000.

[3] B. Babcock, S. Babu, M. Datar, and R. Motwan. Chain: Operator scheduling for memory minimization in data stream systems. In Proc. of SIGMOD 2003, pages 253-264, 2003.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002), pages 1-16, 2002.

[5] D. Carney, U. Cetintemel, M. Cherniack, C.Convey, S. Lee, G. Seidman, M. Stone-Braker, N. Tatbul, and S. Zdonik Monitoring streams: A new class of data management applications.In Proceedings of the 28th International Conference on Very Large DataBases (VLDB'02), 2002.

[6] J.Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In SIGMOD, pages 379-390, 2000.

[7] L.Golab and M. T. Ozsu. Processing sliding window multi-joins in continuous queries over data streams. In VLDB, pages 500-511, September 2003.

[8] J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive Query Processing: Technology in Evolution. IEEE Data Engineering Bulletin, 23(2), June 2000.

[9] Z. G. Ives, D. Florescu, M. Friedman, A. Levy, and D. S. Weld. An adaptive query execution system for data integration. In Proceedings of SIGMOD, pages 299-310, 1999.

[10] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In L. M. Haas and A. Tiwary, editors, SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, pages 106-117. ACM Press, 1998.

[11] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In ICDE, 2002.

[12] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams.In ACM SIGMOD Conference 2002, 2002.

[13] M. Mitchell. An Introduction to Genetic Algorithms. MIT Press, 1999.

[14] T. Mitchell. Machine Learning. McGraw Hill, 1997.

[15] J. F. Naughton, D. J. DeWitt, D. Maier, et al. The Niagara internet query system. EEE Data Engineering Bulletin, 24(2):27-33, 2001.

[16] R. Motwani, J. Widom and A. Arasu et al. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In Proceedings of CIDR, pages 245-256, 2003.

[17] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. In I Proceedings of USENIX, 8, pages 13-24, 1998.