Abstract - Constructing fact tables and performing OLAP operations are some of the most essential but expensive operations in data warehousing. OLAP operations require aggregation on many combinations of each dimension attribute. As the number of dimensions increase, it becomes very expensive to compute summary data cubes because the cost of required computations grows exponentially with the increase of number of dimensions. Query processing for these applications requires different views of data for analysis and effective decision-making. As data warehouses grow, parallel processing techniques can be applied to enable the use of larger data sets and reduce the time for analysis, thereby enabling evaluation of many more options for decision-making. Architecture for computing OLAP operations in parallel, using multiple processors is presented in this paper.

Keywords: Parallel Computing; Big Data; OLAP; Data Warehousing;

1 Introduction

The challenges of maintaining response times of data warehouses as they grow larger and more complex have given rise to new technology solutions like OLAP, and multi-dimensional databases. These solutions are attractive since they ensure the integration of data and give users a refined approach to decision-making. Online Analytical Processing (OLAP) is a database acceleration technique used for deductive analysis. The main objective of OLAP is to have constant-time or near constant-time answers for the many typical queries. For example, in a database containing salesmen’s performance data, one may want to compute, online, the amount of sales done in a given city for the last 10 days, including only salesmen who have 2 or more years of experience. Using a relational database containing sales information, such a computation may be expensive. Using OLAP, however, the computation can typically be done online. To achieve such acceleration one can create a cube of data, a map from all attribute values to a given measure. Using OLAP, the computation can typically be done online. To achieve such acceleration one can create a cube of data, a map from all attribute values to a given measure. The biggest challenge in OLAP now is the sheer volume of data which needs to be handled. Large organizations are usually awash with data, and OLAP techniques are being used to try to make sense of these vast data volumes. However, most OLAP systems impose limits on data volumes, and cannot scale to the level now demanded of them.

Creating a common fact table containing information from all the data sources and issuing queries against that large fact table requires a considerable amount of time. It takes a significant amount of time to perform any of the basic OLAP operations such as roll-up, drill down, pivot, slice and dice on a big fact table. Performing these operations in parallel by dividing the work among all the available processors (nodes) should reduce the overall response time. As shown in figure 1, data from the operational databases are fetched in parallel by multiple processors, and each processor creates a fact table locally. Once the creation of the fact table is completed, the user can start querying that fact table using the available OLAP tool. All the processors compute the results in parallel and then the manager combines results from all the processors and displays the result in the form of a grid or a chart.

1.1 Bottom-up Approach

The Initial approach to parallelism that we considered was based on the Bottom-Up Cubing approach. Bottom-up approaches reuse previously computed sort orders and generate more detailed group-by result sets from less detailed ones. Bottom-up data cube construction methods calculate the group-by result sets in an order that emphasizes the reuse of previously computed sorts. All OLAP operations (Roll Up, Drill Down, Pivot and Slice & Dice) can be done using the bottom-up approach. For example, to do a Roll Up operation on a particular level, the only procedure required by this method is the aggregation level. By passing an aggregation level, this method will distribute cube dimensions to all the computing processors [1][2]. The main logic behind this bottom up sorting is that if it is known that values of attribute A are already sorted, then there is no need for the complete resorting of AB. The total computation time of the bottom up method is dominated by the number of such single attribute
sorts. However, after running some experiments it proved that the communication cost of this approach was much higher than the new approach.

Figure 2 provides an illustration of the recursive subdivision of a four dimensional space. In this case, BUC (Bottom-Up Computation) will first partition (i.e., sort) the input set on A. It will then aggregate on <a1> before partitioning <a1> into its <a1b1> and <a1b2> components. This recursive partitioning will continue until the last dimension has been reached. Eventually, the backtracking will return the algorithm to the <a2> partition, at which point the whole process is repeated.

As noted above, the order in which the attributes are partitioned is also important. Specifically, BUC partitions the attributes in order of decreasing cardinality. In so doing, it minimizes the use of large sorts since the maximum cardinality dimension will immediately split the data into as many small partitions as possible. Experimental results for BUC demonstrate approximately a factor of two performance advantage with respect to alternative data cube algorithms in sparse spaces. However, the benefit of BUC tends to be limited to these full cube high dimension problems. When denser views - the kind typically found in the lower levels of the lattice - are required, the time to recursively partition input sets could dominate the run time since very little short-circuiting takes place. As a result, BUC is ill suited to problems in which large sparse views either not exist or are not required.

The Bottom-Up approach requires reuse of previously computed cubes, which is expected to reduce the number of required computations [3]. However, a small experiment using the Bottom-Up approach showed that there is a lot of overhead in reusing a computed cube. Because each worker node starts with its own computed cube, sends that cube to the other workers, and other workers send computed cubes to that worker, the time required to send cubes was greater than the time required to compute cubes locally. Thus, this project doesn’t use a data-local approach where cubes are computed locally instead of sending them back and forth. Another problem of using the Bottom-Up approach is that it is recursive [3]. This recursive nature turned a small experiment into a complex job. The excessive use of memory made this approach unfeasible using desktop nodes. Balancing the load between workers is also a problem. In some queries, one worker could do all the computation while other worker nodes remain idle. Finally, the data size at worker machines increases tremendously when a single worker machine tries to compute the whole tree.

2  Parallel Data Loader and OLAP Operations

The appeal of parallel processing is especially strong for data warehouse environments, due to the inherent nature of these environments. OLAP systems emphasize interactive
processing of complex queries. Given this requirement, as well as the often extreme size of warehouses, methods are clearly needed for more rapid query execution. By partitioning data among a set of processors, OLAP queries can be executed concurrently, in parallel, potentially achieving linear speedup and thus significantly improving query response time [4][5][6]. In addition to these OLAP queries, another problem data warehouses face is the creation of fact tables. Using a large amount of data and large number of data tables increases the time required for creating the fact table. In order to solve this problem, new approaches for creating fact tables need to be explored. This chapter discusses a new approach that proved to address these efficiency issues.

2.1 Data Loader Design

The main function of the data loader portion of this research was to connect to remote databases and get the appropriate data to create a local fact table. The data loader is also independent of the platform in the sense that the remote database can be implemented on any RDBMS. Thus, one of the main requirements is to have Internet connectivity at the remote sites and to have a DBMS server running on each client’s machine. The following section discusses the available modes for storing data cubes.

Storage Modes: Once data is fetched from multiple sources, there must be a way to store it into a centralized location. There are three main choices for storing data. These are Multidimensional OLAP, Relational OLAP and Hybrid OLAP [3][7]. ROLAP was used to take advantage of combining transactional data and historical data stored in a data warehouse. Processing ROLAP can be very slow on single processor systems, but it can be improved greatly by utilizing parallel processors. The design of the data warehouse database schema should incorporate the principles of dimensional modeling so the dimension tables and fact tables represent the business data and the way clients will view and query the data. Most dimensional modeling results in a star or snowflake database schema. Such schemas facilitate cube design and reduce the number of multiple-table joins when querying the database to process dimensions and cubes [1]. This project uses a star schema as the database schema.

Creating the Fact Table: Initially all workers wait for a request from the manager. The manager broadcasts a message to all workers that the user wants to create the fact table. Each worker gets the address from where it has to fetch the data. Workers will connect to the remote databases and start retrieving data. Depending on the criteria selected by the user to create the fact table, the workers perform queries on their remote databases and fetch the needed rows. This approach assumes all the dimension tables needed to build the fact table are already stored locally in each worker machine, as illustrated in figure 4. This makes it unnecessary to transfer dimension tables from the manager machine to the worker machines. In summary, worker machines need to fetch the transaction and historical data from the remote databases. Using the dimension tables and the transaction table, worker machines create fact tables locally. Once the creation of the fact table for one remote database is complete, the worker checks whether there are any more addresses available for which data needs to be fetched. If there are not any addresses available, it sends a notification message to the manager indicating that it is done with fact table creation. If there is some address available, the worker fetches data from that address and appends rows to the already existing fact table.

The initial approach investigated was to send the fact table data from the workers to the manager. However, after conducting further research and collecting results from a small experiment it was clear that since the manager sends the OLAP queries back to the workers, it is better to keep data local to the worker nodes. The biggest advantage of using this approach is that a large space on the manager computer is not needed, because the fact table is divided into several sub fact tables on worker machines. The second advantage is that only minimal communication is needed between the manager and workers; the manager does not send data to workers but rather sends processing requests.

2.2 OLAP Design

Figure 5 shows that the manager sends OLAP queries to all the worker machines. Each worker then computes the results locally and sends them back to the manager. Finally, the manager combines the results collected from the different workers and presents them to the user in the form of a table or a chart. Since each worker has part of the fact table and the dimension tables locally, communication cost between manager and worker is minimized. In addition, there is no need for inter-worker communication where workers need to communicate only with the manager. Each worker has the same copy of dimension tables, which are typically not very big. It is also possible that the manager makes sure each worker has the latest copy of dimension tables. Thus, whenever the manager sends an OLAP query to the workers, they do not need to fetch the data from the manager, but only compute parts of the fact table they store locally and send the results back to the manager. Since the number of workers available divides the fact table and each worker has to manage only its part, queries are processed faster when compared with a single node.

![Figure 5: Parallel OLAP Architecture](image)
2.3 Data Loader Implementation

User can select the criteria for building the fact table. He can select the dimension tables and the fields from the remote table he wants to select as measures for that fact table. Once user has selected dimension tables, primary keys for those tables are automatically imported in the fact table. These primary keys become foreign keys in the fact table. User can also select measure fields and the type of operation he wants to perform on them. Once the creation of the fact table is completed, workers start populating the fact table with values. All the information regarding the remote tables, such as connection string and remote table name are stored in the remote URL table. Once the manager sends a message to the workers to start working on the creation of the fact table, each worker fetches one connection string from this remote URL table. Once a worker is done with creating fact table data for that connection string, it asks the manager whether there is any more data to be fetched. For example, if there are four remote databases and if there are two worker machines, the first worker will fetch data from the first remote database and the second worker will fetch data from the second database. The worker that finishes earliest will start fetching data from the third database. Once a worker is done with the creation of the fact table, it sends a message to the manager that it is done. Once the manager receives acknowledgement from all the workers it notifies the user that the creation of the fact table is completed.

2.4 OLAP Implementation

Initially users have to select the attributes from the dimension tables to perform queries, as shown in figure 6. Users are also allowed to select the measure on which they want to perform the query, so the user can control the way the query is performed.

After selecting attributes, user can select the value of those attributes; for example, if he selects an attribute named “Year” from the Period table, he can select which year he wants to look into, say 2003, for example, as illustrated in figure 7. According to this all OLAP operations become generic, i.e. there is no need for customized implementation of the different OLAP operations. Once the user has selected all the fields he wants to see, the manager will send all requests to all available worker nodes. Each worker gets this information and starts working on computing the results. Each worker node uses its dimension tables and the locally stored part of the fact table to compute the results. Once it computes the results of all the queries, it sends them back to the manager. The manager waits for a response from all workers. Once all workers send their results to the manager, it combines the results and presents them in the form of a table or a chart as seen in figure 7.

Users also have the option to select the measure on which to perform the analysis. In addition, users can select to view the results in the form of bar or pie charts (Figure 8), as well as the default tabular format. These charts are implemented using .Net Charting graphics library.

3 Conclusion

The communication overhead of the Bottom Up approach increases as the size of data and the number of processors increase to the point that makes it not efficient for real applications. The Bottom Up approach reuses the cube computed by one worker node to compute other cubes by other worker nodes, which involves a large number of big communication messages between worker nodes. This paper presents a more efficient alternative that processes sub-fact tables on worker nodes locally eliminating the inter-node communication overhead. Though the new approach is more
efficient and scalable than its predecessors, it does not address important issues such as fault tolerance. If one of the worker nodes goes down, the partition of the data set allocated to those nodes (sub-fact table) will be lost. Since no other workers have access to, or retain a copy, the absence of that data might lead to an incomplete fact table. One solution to this problem is to maintain a record of the work being done by all worker nodes and the status of each node (redundancy). One viable option to implement this solution is to designate a node primarily for managing node failures (failure manager). This node can reassign work from a failed node to other active worker nodes. Extensive experimentation is needed to assess the efficiency of this solution and its impact on the regular computation.

4 References


