Parallel ROLAP Data Cubes performance Comparison and Analysis

Wei Shi, Qi Deng, Boyong Liang
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
swei4,qdeng, byliang@scs.carleton.ca

December 29, 2003

Abstract

The pre-computation of data cubes is critical to improving the response time of On-Line Analytical Processing (OLAP) systems and can be instrumental in accelerating data mining tasks in large data warehouses. In order to meet the need for improved performance created by growing data sizes, parallel solutions for generating the data cube are becoming increasingly important. This Project Report reports the test result of building and querying data cube between large parallel database: DB2 and Panda system. We have investigated two large parallel database: DB2 and Oracle 9i. Under the default setup of DB2 and Panda system, we tested the performance of building a cube in both Panda system and DB2 and also some data cube queries. We mainly studied the Partition difference between Panda system and DB2, then analyze the reason of the test result after the serious study of these two systems.

1 Introduction

In recent years, there has been tremendous growth in the data warehousing market. Despite the sophistication and maturity of conventional database technologies, the ever-increasing size of corporate databases, coupled with the emergence of the new global Internet "database", suggests that new computing models may soon be required to fully support many crucial data management tasks. In particular, the exploitation of parallel algorithms and architectures holds considerable promise, given their inherent capacity for both concurrent computation and data access.

We are interested in the design parallel data mining and OLAP algorithms and their implementation on coarse grained parallel multicomputers and PC-based clusters. Now, under the supervision of Professor Frank Dehne, our Data Cube group explores the use of parallel algorithms and data structures in the context of high performance On-line Analytical Processing (OLAP). OLAP is the foundation for a wide range of essential business applications, including sales and marketing analysis, planning, budgeting, performance measurement and data warehouse reporting. To support this functionality, OLAP relies heavily upon a data model known as the data cube. Conceptually, the data cube allows users to view organizational data from different perspectives and at a variety of summarization levels. In fact, the data cube model is central to our parallelization efforts[5].

Usually the size of the data cube is potentially very large. In order to meet the need for improved performance created by growing data sizes in OLAP applications, parallel solutions for generating the data cube have become increasingly important. The current parallel approaches can be grouped into two broad categories: (1) *work partitioning* [3, 4, 9, 10, 11] and (2) *data partitioning* [6, 7]. Work partitioning methods assign different view computations to different processors. Given a parallel computer with p processors, work partitioning schemes partition the set of views into p groups and assign the computation of the views in each group to a different processor. The main challenges for these methods are load balancing and scalability, which are addressed in different ways by the different techniques studied in [3, 4, 9, 11, 10]. One distinguishing feature of work partitioning methods is that all processors need simultaneous access to the entire raw data set. This access is usually provided through the use of a shared disk system (available e.g. for SunFire 6800 and IBM SP systems).

Data partitioning methods partition the raw data set into p subsets and store each subset locally on one processor. All views are computed on every processor but only with respect to the subset of data available at each processor. A subsequent "merge" procedure is required to agglomerate data across processors. The advantage of data partitioning methods is that they do not require all processors to have access to the entire raw data set. Each processor only requires a local copy of a portion of the raw data which can, e.g., be stored on its local disk. This makes such methods feasible for shared-nothing parallel machines like the popular, low cost, Beowulf style clusters consisting of standard PCs connected via a data switch and without any (expensive) shared disk array. The main problem with data partitioning methods is that the "merge", which has to be performed for every view of the data cube, has the potential to create massive data movements between processors with serious consequences for performance and scalability of the entire system.

In the previous work of our research group, Dr.Todd [5] built the full data cube and partial data cube in parallel, and tested the performance of five queries, namely: rollup; drawdown; slice; dice; pivot. We realized that it is crucial to investigate the existent parallel databases and invest a mount of time on testing the efficiency of building cube and execute five queries on the cube in parallel databases and in our our way so that we can analysis the advantages and disadvantages, then improve our work. We investigate Oracle9i RAC, DB2 these two large and well developed database. We are going to explain the difference of data partition which decides the efficiency of five queries, and the difference between building a cube using different algorithms both in parallel database but also Todd's project

2 Parallel Database

With the widespread adoption of the relational data model first appeared in 1983 in the marketplace, parallel database systems become more than a research curiosity. Relational queries are ideally suited to parallel execution; they consist of uniform operations applied to uniform streams of data. Each operator produces a new relation, so the operators can be composed into highly parallel dataflow graphs. By streaming the output of one operator into the input of another operator, the two operators can work in series giving pipelined parallelism. By partitioning the input data among multiple processors and memories, an operator can often be split into many independent operators each working on a part of the data. This partitioned data and execution gives partitioned parallelism.[8]

The parallelism in databases is inherited from their underlying data model. In particular, the relational data model (RDM) provides many opportunities for parallelization. For this reason, existing research projects, academic and industrial alike, on parallel databases are nearly exclusively centered on relational systems. In addition to the parallel potential of the relational data model, the worldwide utilization of relational database management systems has further justified the investment in parallel relational databases research. It is, therefore, the objective of this book to review the latest techniques in parallel relational databases.

The topic of parallel databases is large and no single manuscript could be expected to cover this field in a comprehensive manner. In particular, this manuscript does not address parallel object-oriented database systems. However, it is noteworthy that several projects described in this manuscript make use of hybrid relational DBMSs. The emergence of hybrid relational DBMSs such as multimedia object/relational[2] database systems has been made necessary by new database applications as well as the need for commercial corporations to preserve their initial investment in the RDM. These hybrid systems require an extension to the underlying structure of the RDM to support unstructured data types such as text, audio, video, and object-oriented data. Towards this end, many commercial relational DBMSs are now offering support for large data types (also known as binary large objects, or BLOBs) that may require several gigabytes of storage space per row.

3 Panda System

PANDA system focus on designing efficient algorithms for the computation of the cube (full and partial). Panda works within the relational paradigm to produce a set S of 2d summarized views (where d is the number of dimensions). Its general approach is to partition the workload in advance, thereby allowing the construction of individual views to be fully localized. Initially, Panda computes an optimized task graph, called a schedule tree, that represents the cheapest way to compute each view in the data cube from a previously computed view (Figure 1 shows A four dimensional minimum cost PipeSort spanning tree). Panda employs a partitioning strategy based on a modified k-min-max partitioning algorithm that divides the schedule tree into p equally weighted sub-trees (where p is the number of processors). Panda system supports schedule tree construction with a rich cost model that accurately represents the relationship between the primary algorithmic components of the design. In addition, it also presents efficient algorithms and data structures for the physical disk-based materialization of the views contained in each sub-tree.[5]

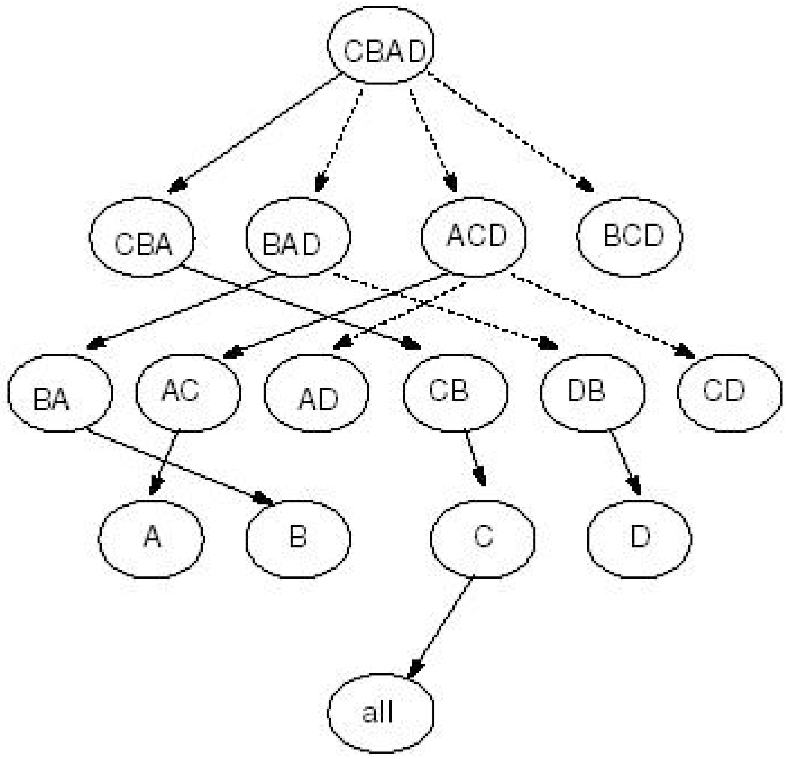

Figure 1: A four dimensional minimum cost PipeSort spanning tree

3.1 Data Organization

3.1.1 Data Generator

Parallel Data Cube Generator- pipesort.exe The data cube generation module is responsible for computing both full and partial data cubes. In other words, it can generate all 2d views in the d-dimensional space, or it can construct some user-defined subset[5].

PANDA provides Data Generator to generate the input sets that correspond to the data warehouse fact table. The module takes as input a binary input set (or fact table) that has been previously produced by the data generator. On a Linux cluster, then a copy of the input set must be placed on each node (i.e., on the local file system of each disk). Once the output is produced, it will be written back to the appropriate disk configuration. If this is a disk array then the view will simply be striped by the hardware to a single logical drive. If it is a cluster, then the views will be written to the local disks associated with each processor. The primary data cube generator will write a group of views to each disk. An extension to the system, designed specifically for cluster environments, stripes every individual view across the disk set[5].

3.2 Query

3.2.1 Build Multiple Index - Rcube

Parallel RCUBE Index Generator - rcube.exe The RCUBE index generation module builds multi-dimensional packed r-tree indexes for the data cube constructed by the data cube generator. Specifically, it will build a separate index for each of the O(2d) aggregate views that have been previously distributed across the cluster[5].

With respect to the output, a pair of files will actually be generated for each data cube view. The first is a Hilbert-sorted data set. Simply put, this file is a reorganization of the aggregate view in Hilbert order. Moreover, the records are organized into block sized units. Records do not cross block boundaries and padding is used where necessary[5].

The second file is the index proper. It is a tree-based construct that provides access to the records contained in the blocks of the Hilbert sorted data set[5].

Unlike the original data cube, the index files are not localized on a given disk. Instead, each Hilbert/Index pair is striped across every disk. Each Hilbert/Index pair represents a partial index that will, in turn, be used by the query generator[5].

3.2.2 Parallel Query Engine

The query engine module accepts multi-dimensional user queries and resolves them against the indexed data cube produced by the data cube and index generators. Queries can either be point queries, range queries or, in special cases, hierarchical queries[5].

4 DB2 Parallel Computing

4.1 Architecture

p1 - **share nothing**
DB2 PE employs a Shared noting architecture in which the database system consists of a set of independent logical database nodes. Each logical node represents a collection of system resources including, processes, main memory, disk storage and communications, managed by an independent database manager. The logical nodes use message passing to exchange data with each other. Tables are partitioned across nodes using a hash partitioning strategy. The cost-based parallel query optimizer takes table partitioning information into account when generating parallel plans for execution by the runtime system. A DB2 PE system can be configured to contain one or more logical nodes per physical processor. For example, the system can be configured to implement one node per processor in a shared-nothing, MPP system or multiple nodes in a symmetric multiprocessor (SMP) system. This paper provides an overview of the storage model, query optimization, runtime system, utilities, and performance of DB2 Parallel Edition. See figure 2

p2 - **share nothing**
DB2 PE supports a partitioned storage model I which tables in a database are partitioned

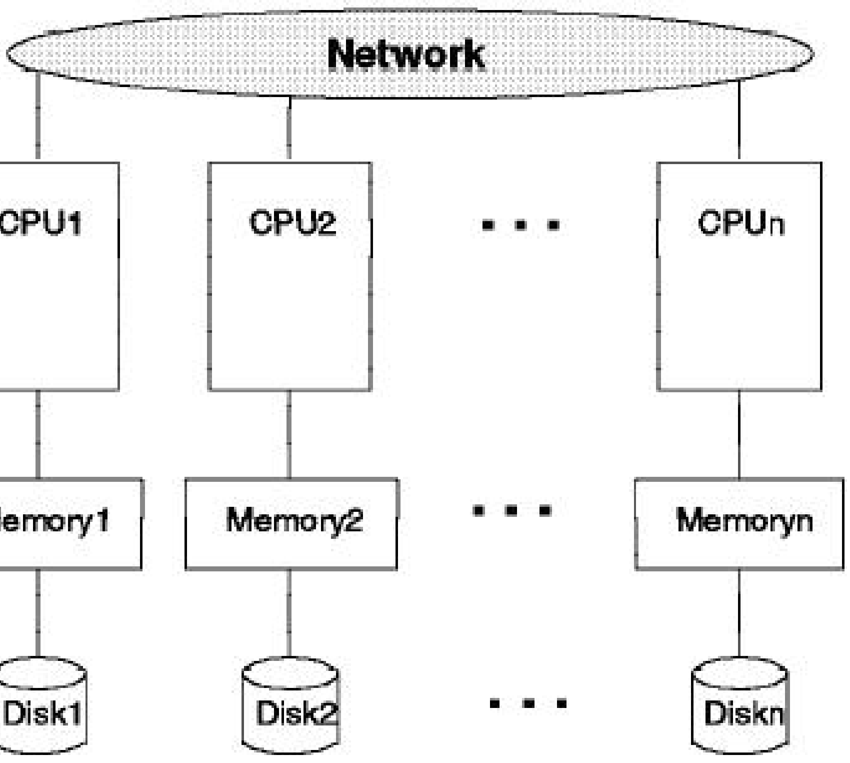

Figure 2: DB2 parallel computing architecture

across a set of database nodes. Indexes are partitioned along with their corresponding tales and stored locally at each node. The system supports hash partitioning and partial *declustering* a of database tables. Extensions to the CREATE TABLE statement allow users to specify a partitioning key and nodegroup associated with a table. The partitioning key is a set of columns in the table row is stored. Nodegroups are named subsets of database nodes. Each nodegroup is associated with a data structure called the partitioning map(PM) partitioning map (PM) which consists of 4096 entries each containing a node number. If x represents the partitioning key value of a row in a table, then node at which this row is stored is given by, PM (H) which consists of 4096 entries. If x represents the partitioning key value of a row in a table, then the node at which this row is stored is given by, PM[H(x)], where H(.) represents the system hash function and PM represents the partitioning map associated with the nodegroup in which the table is created. Data definition language (DDL) extensions are provided to allow users to create nodegroups in a data base.

4.2 Partition mechanism

DB2 Parallel Edition supports the hash partitioning technique to assign each row of a table to the node to which the row is hashed. You need to define a partitioning key before applying the hashing algorithm. The hashing algorithm uses the partitioning key as an input to generate a partition number. The partition number then is used as an index into the partitioning map. The partitioning map contains the node number(s) of the nodegroup. There are three major steps to perform the data partitioning:

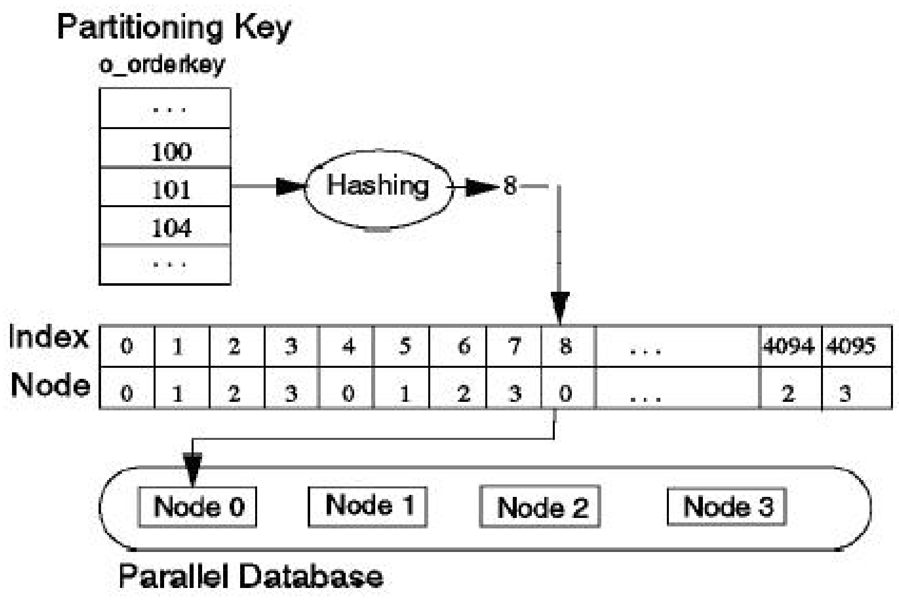

Figure 3: Parallel DB2 Partition diagram

1. Partitioning key selection: A partitioning key is a set of one or more columns of a given table. It is used by the hashing algorithm to determine on which node the row is placed.

2. Partitioning map creation: In DB2 Parallel Edition, a partitioning map is an array of 4096 node numbers. Each nodegroup has a partitioning map. The content of the partitioning map consists of the node numbers which are defined for that nodegroup. The hashing algorithm uses the partitioning key on input to generate a partition number. The partition number has a value between 0 and 4095. It is then used as an index into the partitioning map for the nodegroup. The node number in the partitioning map is used to indicate to which node the operation should be sent.
3. Partitioning rows: The steps for row partitioning consist of splitting and inserting the data across the target nodes.. DB2 Parallel Edition provides tools to partition the data and then load the data to the system via a fast-load utility or import. Data can also be inserted into the database via an application that uses buffered inserts. See figure 3

4.3 Table - Table Space - Partition Roup

Nodegroups and Data Partitioning
In DB2 Parallel Edition, data placement is one of the more challenging tasks. It determines the best placement strategy for all tables defined in a parallel database system. The rows of a table can be distributed across all the nodes (fully declustered), or a subset of the nodes (partially declustered). Tables can be assigned to a particular set of nodes. Two tables in a parallel database system can share exactly the same set of nodes (fully overlapped), at least one node (partially overlapped), or no common nodes (nonoverlapped). Parallel Edition supports the hash partitioning technique to assign a row of a table to a table partition.

Nodegroups

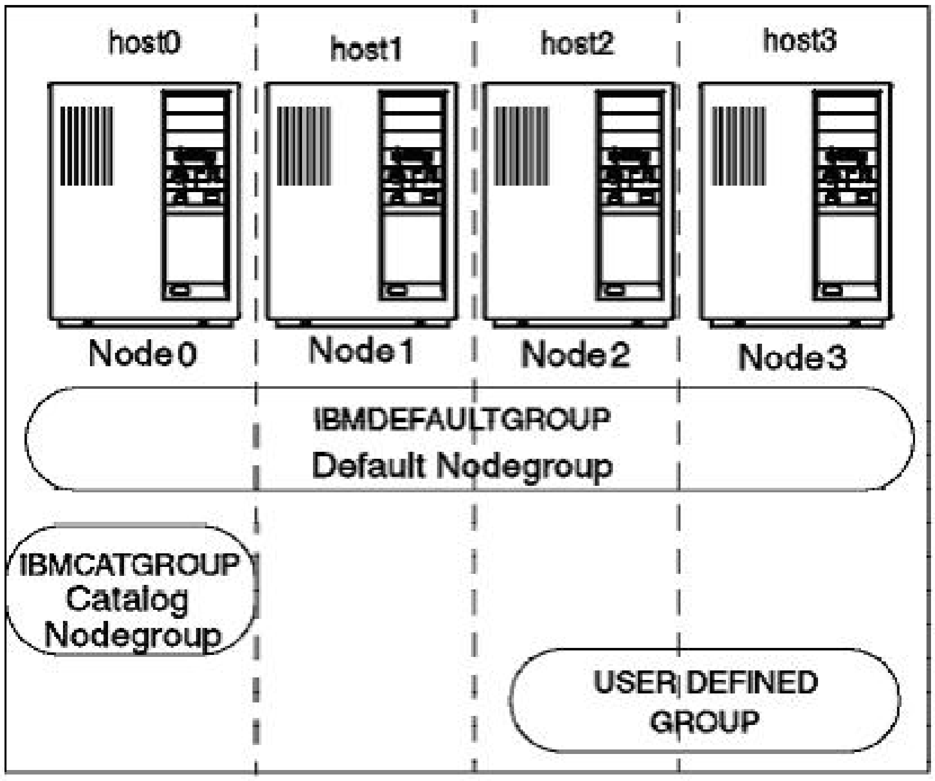

Figure 4: Nodegroup diagram

Nodegroups in DB2 Parallel Edition are used to support declustering (full or partial), overlapped assignment and hash partitioning of tables in the parallel database system. A nodegroup is a named subset of one or more of the nodes defined in the node configuration file, $HOME/sqllib/db2nodes.cfg, of DB2 Parallel Edition. It is used to manage the distribution of table partitions. In DB2 Parallel Edition, there are system-defined and user-defined nodegroups. Tables must be created within a nodegroup. Figure xx shows a DB2 Parallel Edition instance consisting of four physical machines. The default nodegroup, IBMDEFAULTGROUP, spans all of the nodes in the instance. The catalog nodegroup, IBMCATGROUP, is created only on the node where the create database command was issued. The catalog node is on host0 in this example. Also shown is a user-defined group that spans host2 and host3. See figure 4.

4.4 Query optimize and summary table MQT

We use CUBE grouping to generate full data cube in DB2. CUBE grouping is an extension to the GROUP BY clause that produces a result set that contains all the rows of a ROLLUP aggregation. A CUBE grouping can also be thought of as a series of grouping-sets. In the case of a CUBE, all permutations of the cubed grouping-expression-list are computed along with the grand total. Therefore, the n elements of a CUBE translate to 2**n (2 to the power n) grouping-sets[1]. The following is an example of create full data cube using CUBE grouping:

SELECT a, b, c, d, e, f, sum(fact) as amount,
COUNT(fact) AS COUNTAMOUNT FROM fact8
GROUP BY CUBE (a, b, c, d, e, f)
We use materialized query table (MQT) that stores the results of CUBE grouping to issue the cube query. DB2 optimizer will automatically choose MQT if a range query happens on the base table.
The following is an example of create full data cube as MQT.
CREATE TABLE fact8cube
AS (SELECT a, b, c, d, e, f, sum(fact) as amount, COUNT(fact) AS COUNT_AMOUNT
FROM fact8
GROUP BY CUBE (a, b, c, d, e, f))
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION
IN cubets8
The REFRESH TABLE statement refreshes the data in a MQT, in our case, builds the full data cube.

5 Experiment Evaluation

In this section, we present and compare experimental results for the parallel full cube generation and parallel cube query in Panda system and DB2 Parallel Edition V8.0 system. We will also discuss the experimental DB2 environment that might infect the system performance.
We use Panda's Data Generator to generate a group of data which contains difference number of records, dimensions and cardinalities. The data are used to build full data cube and query on 1 to 11 processors in Panda system. The same data are loaded into DB2 tables and are used to build full data cube and query on 1 to 11 processors in DB2 system as well. We use the default configuration on both systems and ignore the CPU workload, system buffer and others factors which might affect the performance.

5.1 Full data cube generation

We use Panda's data cube generator utility in Panda system (see section 3.1). In DB2 system, we use Materialized Query Tables (MQT) for cube groupBy (see section 4.3).

We noticed that in Figure 5.2, the run time of building data cube on 8 nodes in DB2 is extremely fast. It occurred when rebuild the cube several times, the buffer system inside the DBMS might accelerate the performance.

5.2 Parallel Cube Query

We use Panda's parallel indexes generator and query engine utilities for data cube query in Panda system (see section 3.2, 3.3). We use MQT optimization mechanism for cube query in DB2
Figure 5is the run time of four parallel query run time of 1M data on Panda and DB2

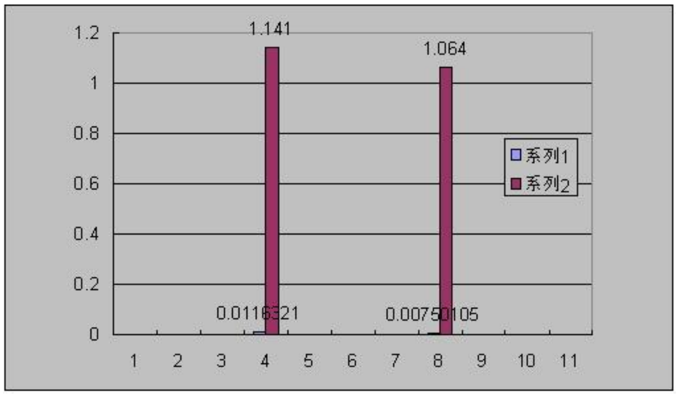

Figure 5: Parallel query run time of 1M data on Panda and DB2 system (file query36.xsl)

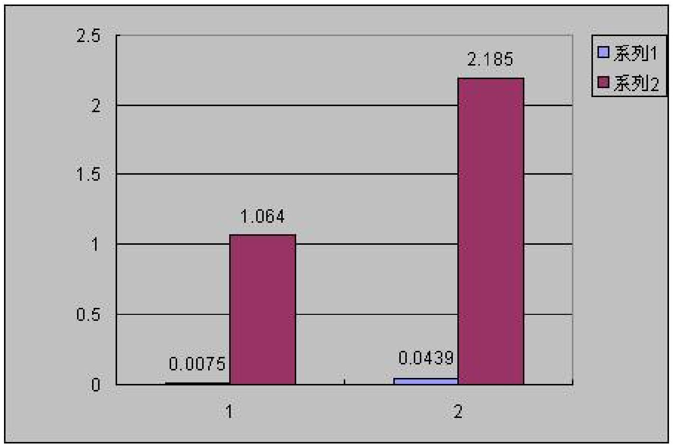

Figure 6: The run time of four parallel query on 8 processor for 1M and 2M data on Panda and DB2 system(file query63.xsl)

system. Figure6 is the run time of four parallel query on 8 processor for 1M and 2M data on Panda and DB2 system.

6 Analysis and conclusion

NOTE: Because there are only two groups of experimental result, the following analyze and conclusion are based on the current result and to be proved and confirmed by more experimental.

6.1 Parallel data cube generation

Figure 7 and 8 show that the run time of Panda's parallel full data cube generation is much faster than DB2 system. The result also shows Panda system can balance the workload very well, if we consider the average run-time of generation, Panda's algorithm has very stable speedup in most cases. DB2 system also has good speedup, but the result of partial data

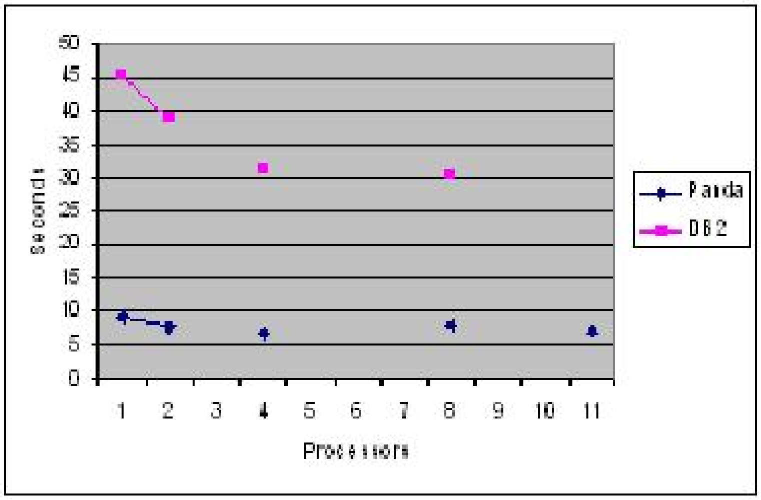

Figure 7: Test Result 1

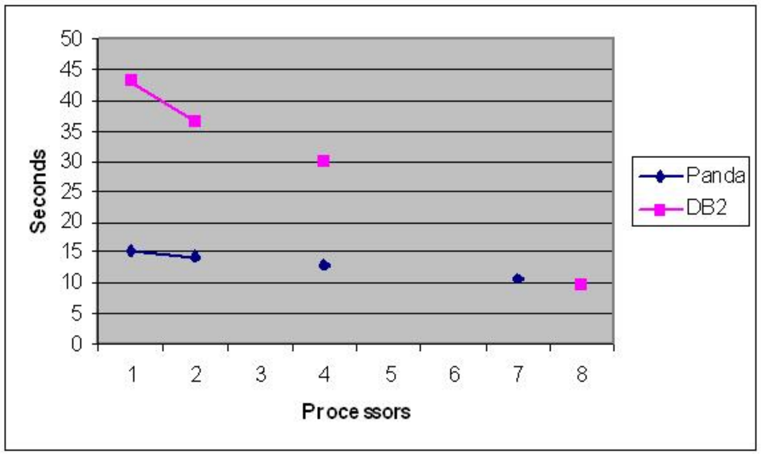

Figure 8: Test Result 2

cube generation (groupBy MQT result in experimental report) shows in many case, the run time might increase depends on data volume and partition number, say 11 partitions

6.2 Parallel data cube query

Figure 5 and 6 show that the run time of Panda's parallel data cube query is extremely faster than DB2

Of course, we realized that Panda has to take long to build multiple dimension indexes before it can handle queries. Consider we are working on the data warehouse, taking time to build efficient index is worth to prompt the performance of the coming huge number of queries. We also realized we do not take the benefit of DB2 index mechanism to achieve the highest performance of DB2 query.

6.3 The efficient of data cube organization

The experimental result shows Panda's data cube and indexes partition is very efficient for cube query We realized that, unlike Panda system, DB2 PE, as a commercial DBMS, is designed to handle large number of concurrent transactions processing. Its high availability and reliability features, such as Logging system, Lock mechanism, will counteract the system to achieve highest performance.

7 Future Work

Due to the restriction of current experimental DB2 environment, our next work is to review the comparisons of parallel data cube generation and query on more processors and larger data volume. Analyze the performance for various kind of data cube, including the different dimension numbers and cardinalities To reach the high performance of OLAP operation in DB2 system, we will also keep on researching DB2 data organization and partition mechanism. DB2 OLAP facilities, such as Cube View, OLAP Server, are also the interesting relative topics we will work on.

8 Appendix

8.1 Experiment Report

Target
Analyze the data partition mechanism in RDBMS
Implementing the data cube functionalities in parallel DB2 systems
Running the Panda system and DB2 parallel data cube building and query to compare and analyze the performance of these to system
Steps a) Data generate and transform Data generator -fHex Transform to text file -fTxt
b) Panda Run the Panda cube building (pipesort.exe) and cube query(query.exe) based the data file -fHex See previous files
c) DB2 Run the DB2 parallel cube building (refresh MQT) and cube query(summary table) based the data file -fHex
i. DB2 environment setup
Define database - testcube
Adjust transaction logs
Define the partition group 'datacubegroup x', where x means the number of nodes (1, 2, 4, 8,13) Define the tabespaces on deferent nodes as following: 'cubets x', where x means the number of nodes(1,2, 4, 8,13), in the corresponding 'datacubegroup x'
already defined, just use them in the coming steps
ii. Create base tables
Create the base table in the corresponding tablesspace
Naming convention: factXdYp X: number of dimensions of data OR the sequence number of testing Y: number of partition On the tablespace - "cubetsY"
Statement: db2 "create table fact6d8p (a int not null, b int not null, c int not null, d int not null, e int not null, f int not null, fact int) IN cubets8 index in cubets8"

iii. Load data to base table
copy data file (in text format) into current node, say c1-2 scp input.dat db2inst1@c1-2:/home/db2inst1/data/cubeXX.load.data
load data

iv. Define cube and build
Define the cube(MQT) on the same tablespace as its base table FactXdYpCube On the same tablespace- "cubetsY" for the base table "factXdYp"
Steps
a) Define cubes: db2batch -d testcube -f build.sql -r build.result
b) Build (refresh) db2batch -d testcube -f rAll -r rAll.result

c) Check test result in rAll.result

v. Cube query
Query
"db2batch -d testcube -f mqt_ex.sql -r mqt.result"
Explain
db2expln -d testcube -f mqt_ex.sql -o temp
Result mqt.result

2. result see the result file, collect data into the table and analyze.

8.1.1 Data combination

1. Data size
1 2 3
500k 1M 2M

2. Number of dimensions
1 2 3
6 8 10

3.Shew
1 2 3
0 1 2

4. Cadinalities
1 2 3
high mix low

high: 256,256,256,256,...
mix: 256,128,64,32,16,8,6,6,...
low: 16,16,16,16,16,16,16,16,...
5. Processors: 2,4,6,8,13

Add a check mark at at the end of the data you've tested,
1) 500k, 6, 0, high x

2) 500k, 6, 0, mix
3) 500k, 6, 0, low
4) 500k, 6, 1, high
5) 500k, 6, 1, mix
6) 500k, 6, 1, low
7) 500k, 6, 2,high
8) 500k, 6, 2,mix
9) 500k, 6, 2,low

10) 500k, 8, 0, high
11) 500k, 8, 0, mix
12) 500k, 8, 0, low
13) 500k, 8, 1, high
14) 500k, 8, 1, mix
15) 500k, 8, 1, low
16) 500k, 8, 2,high
17) 500k, 8, 2,mix
18) 500k, 8, 2,low

19) 500k, 10, 0, high
20) 500k, 10, 0, mix
21) 500k, 10, 0, low
22) 500k, 10, 1, high
23) 500k, 10, 1, mix
24) 500k, 10, 1, low
25) 500k, 10, 2,high
26) 500k, 10, 2,mix
27) 500k, 10, 2,low

28) 1M, 6, 0, high
29) 1M, 6, 0, mix
30) 1M, 6, 0, low
31) 1M, 6, 1, high
32) 1M, 6, 1, mix
33) 1M, 6, 1, low
34) 1M, 6, 2,high
35) 1M, 6, 2,mix
36) 1M, 6, 2,low

37) 1M, 8, 0, high
38) 1M, 8, 0, mix
39) 1M, 8, 0, low
40) 1M, 8, 1, high
41) 1M, 8, 1, mix
42) 1M, 8, 1, low
43) 1M, 8, 2,high
44) 1M, 8, 2,mix
45) 1M, 8, 2,low

46) 1M, 10, 0, high
47) 1M, 10, 0, mix
48) 1M, 10, 0, low
49) 1M, 10, 1, high
50) 1M, 10, 1, mix
51) 1M, 10, 1, low
52) 1M, 10, 2, high
53) 1M, 10, 2, mix
54) 1M, 10, 2, low

55) 2M, 6, 0, high
56) 2M, 6, 0, mix
57) 2M, 6, 0, low
58) 2M, 6, 1, high
59) 2M, 6, 1, mix
60) 2M, 6, 1, low
61) 2M, 6, 2,high
62) 2M, 6, 2,mix
63) 2M, 6, 2,low

64) 2M, 8, 0, high
65) 2M, 8, 0, mix
66) 2M, 8, 0, low
67) 2M, 8, 1, high
68) 2M, 8, 1, mix
69) 2M, 8, 1, low
70) 2M, 8, 2,high
71) 2M, 8, 2,mix
72) 2M, 8, 2,low

73) 2M, 10, 0, high
74) 2M, 10, 0, mix
75) 2M, 10, 0, low
76) 2M, 10, 1, high
77) 2M, 10, 1, mix
78) 2M, 10, 1, low
79) 2M, 10, 2,high
80) 2M, 10, 2,mix
81) 2M, 10, 2,low

References

[1] Ibm db2 universal database sql reference version 8.

[2] F. Carino and W. Sterling. Parallel strategies and new concepts for a petabyte multimedia database computer,. *Parallel Database Techniques, M. Abdelguerfi and K.F. Wong, eds., chapter 7, IEEE CS Press, Los Alamitos, CA*, pages pp. 139–164, 1998.

[3] F. Dehne, T. Eavis, S. Hambrusch, and A. Rau-Chaplin. Parallelizing the data cube. *Distributed and Parallel Databases*, 11(2):181–201, 2002.

[4] F. Dehne, T. Eavis, and Andrew Rau-Chaplin. A cluster architecture for parallel data warehousing. In *Proc IEEE International Conference on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, 2001.

[5] T. Eavis. Parallel relational olap. June, 2003.

[6] S. Goil and A. Choudhary. High performance OLAP and data mining on parallel computers. *Journal of Data Mining and Knowledge Discovery*, 1(4):391–417, 1997.

[7] S. Goil and A. Choudhary. A parallel scalable infrastructure for OLAP and data mining. In *Proc. International Data Engineering and Applications Symposium (IDEAS'99)*, Montreal, 1999.

[8] David J. DeWitt J. Gray. Parallel database systems: The future of high performance database processing,. January 1992.

[9] H. Lu, X. Huang, and Z. Li. Computing data cubes using massively parallel processors. In *Proc. 7th Parallel Computing Workshop (PCW'97)*, Canberra, Australia, 1997.

[10] Seigo Muto and Masaru Kitsuregawa. A dynamic load balancing strategy for parallel datacube computation. In *ACM Second International Workshop on Data Warehousing and OLAP (DOLAP 1999)*, pages 67–72, 1999.

[11] R.T. Ng, A. Wagner, and Y. Yin. Iceberg-cube computation with pc clusters. In *ACM SIGMOD Conference on Management of Data*, pages 25–36, 2001.