An Efficient OLAP Query Processing Technique Using Measure Attribute Indexes*

T.S. Jung[1], M.S. Ahn[2], and W.S. Cho[2]

[1] Department of Information Industrial Engineering, Chungbuk National University, 361763 Choungju, Chungbuk, Korea
[2] Department of Management Information Systems, Chungbuk National University, 361763 Choungju, Chungbuk, Korea
{mispro, epita55, wscho}@chungbuk.ac.kr

Abstract. We propose an index structure, called *measure attribute (MA) index*, and a query processing technique to improve OLAP query performance. OLAP queries are extremely complicated due to representing the intricate business logic of the company on a huge quantity of data. This is why the efficient query evaluation becomes a critical issue in OLAP systems. Proposed query processing technique supports an efficient evaluation of the star joins and grouping operators known as the most frequently used but very expensive operators in OLAP queries. The MA index is a variation of the path index in object databases and supports index-only processing for the star joins and grouping operators. Index-only processing is a well known efficient technique in the query evaluation area. We implemented the MA index on top of an object-relational DBMS. Performance analysis shows that the MA index provides speedups of orders of magnitude for typical OLAP queries.

1 Introduction

Data warehouse is a collection of integrated, subject-oriented, nonvolatile, time-variant information for decision making in an organization[10,15]. The data warehouse is becoming an essential infrastructure in most organizations such as business companies, hospitals, and government organizations.

OLAP (On-Line Analytical Processing) is the process that end users directly analyze the data warehouse in an interactive mode for various decision making applications[3]. Conventional OLAP systems can be implemented either MOLAP (multidimensional OLAP) or ROLAP (relational OLAP)[3]. In this paper, we assume another type of OLAP implementation, called OOLAP (object-relational database OLAP). Recently, the necessity of OOLAP is increasing by the emergence of ORDBMSs as the next generation DBMSs[1,4,6,7,14].

* This research was supported by the Program for the Training of Graduate Students in Regional Innovation which was conducted by the Ministry of Commerce, Industry and Energy of the Korean Government.

X. Zhou et al. (Eds.): WISE 2004, LNCS 3306, pp. 218–228, 2004.

OLAP queries are extremely complicated due to representing the intricate business logic of the company. They also require long response time because of the complex queries on a huge quantity of data. This is why the efficient query evaluation is an important issue in OLAP systems.

In this paper, we propose an index structure, called *MA* (*measure attribute*) *index*, and a query processing technique to improve the performance of OLAP queries. The MA index is a variation of the path index in object databases and supports index-only processing for the star joins and grouping operators. Note that they are the most frequently used operators in OLAP queries[5]. An MA index directly associates the values of the measure attribute with OIDs (or RIDs) of the dimension class (or table) for index-only processing of the star joins and groupings. Index-only processing is a well-known query processing technique in the database performance area. We also propose algorithms for index-only processing of the star joins and groupings.

Proposed query processing technique with the MA indexes supports an efficient evaluation of the star join and grouping operators. We have implemented the MA index on top of an object-relational DBMS, and the query performance analysis shows speedups of orders of magnitude for typical OLAP queries. Although the idea has been implemented in the OOLAP environment, it can be applied to the ROLAP environments with no modification.

The paper is organized as follows. In Section 2, we discuss OOLAP systems and the existing index structures. In Section 3, we propose the MA index and a query processing technique utilizing the index. In Section 4, we describe mathematical analysis and performance evaluation results. In Section 5, we present conclusions and future work.

2 Related Work

We discuss about the star schema and OLAP queries in OOLAP environments. We then present existing indexes to speedup OLAP query evaluation.

2.1 Star Schema and OLAP Query

There are some differences between ROLAP and OOLAP environments. First, OOLAP uses Object DBMSs instead of Relational DBMSs. Second, relationships between primary key and foreign key in ROLAP are replaced by attribute-domain relationships in OOLAP. Each class in the OOLAP may have subclasses, and the domain of an attribute may be another class. Figure 1 shows a star schema in OOLAP. Various analyses can be done on Figure 1 by using OLAP queries. Query 1 is a typical OLAP query on the star schema.

In query Q1, star joins of Sales, Store, Time, and Product are represented by the path expressions in the object SQL. Note that the cost of the star joins is very expensive because the fact class Sales contains a huge number of objects in most cases.

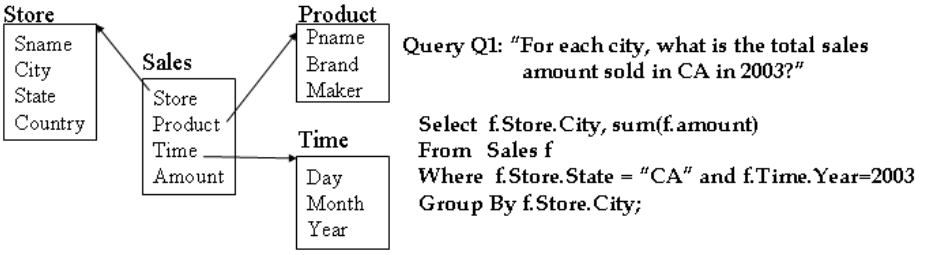


Fig. 1. A Star schema for sales information

2.2 Conventional Index Structures for OLAP Query Processing

For efficient OLAP query processing, various indexes such as *Bitmap* (BM for short) index and *Bitmap Join* (BJ for short) index have been proposed in OLAP systems[2,8,9,12,13,18,19]. Generally, BM index is superior to the B^+-Tree index in space and retrieval efficiency. BJ[12] index is an integration of the Join Index[17] and BM index for efficient star joins.

However, space overhead of the indexes BM index and BJ index becomes serious as the number of distinct values in the attribute increases[18]. *Encoded Bitmap* (EB for short) index is a solution for the space overhead[19]. In Figure 1, let assume 1,000,000 tuples in the fact table *Sales*, 12,000 tuples in the dimension table *Store*. The BJ index between *Sales* and *Store* requires 12,000 bit vectors, each of which has 1,000,000 bits (12Gbits). However, the EB index requires just $\lceil \log_2 12000 \rceil = 14$ bit vectors, each of which has 1,000,000 bits (14Mbits) [19].

Although these indexes provide a reasonable space overhead and reduced access time compared with conventional indexes, their query evaluation cost is still expensive especially when a large number of objects have to be accessed from the fact table. Note that the index provides not the objects themselves but the locations of the qualified objects. So the query processor should access the qualified objects from the fact table via the locations obtained from the indexes. Our experiments show that the latter activity (i.e., accessing qualified objects via the locations) requires a significant time overhead when a great number of objects should be accessed from the fact table. We confirm this situation in Sections 4.2 and 4.3.

3 The MA Index and Query Processing Technique

We present the MA index which intends to minimize the access cost even though the query processor accesses a great number of objects from the fact table. We then propose an efficient query processing algorithm that fully utilizes the MA indexes.

3.1 Index Structure

Storage structure for the MA index can be considered as a simple relational algebra operators. Let F and D_i, i=1, 2, 3, ..., n denote the fact table and dimension tables respectively. Figure 2(a) shows a star schema of F and D_i's. The MA index MA(F.d_i) on the path expression F.d_i can be constructed by the join of F and D_i followed by a projection as follows:

$$MA(F.d_i) = \prod_{Di.oid,\ MA} (D_i \text{ JOIN } F)$$

For the star schema in Figure 2(a), we create four MA indexes for the joins (D_i JOIN F), i=1, 2, 3, 4. Each index MA(F.d_i) is conceptually a binary relation whose attributes are *oid* of D_i and the *measure attribute (MA)* of F. Figure 2(b) shows the structure of MA(F.d_i), i=1, 2, 3, 4. In the implementation, all the MA indexes share the MA values stored in the file MAF rather than duplicating them in each index.

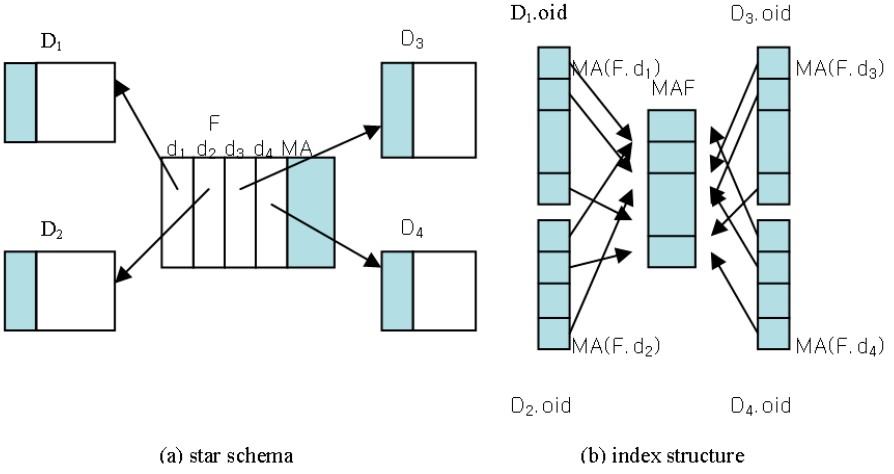


Fig. 2. MA Index Structure

(Example 1)
Figure 3 is showing the example of storage and index structures for the classes Sales, Store, and Time for the star schema in Figure 1. Figure 3 (b) shows the states of Sales, Store, and Time classes and Figure 3 (a) shows two measure index on the attributes Sales.Store and Sales.Time : MA(Sales.Store) and MA(Sales.Time). Note that objects in the fact class are not stored in the original form; only measure attribute(s) is physically stored in the file MAF(measure attribute file). And the relationships between fact and dimension classes are stored in the index MA. For example, MA(Sales.Store) includes two fields; one for the search key, and the other for the offsets of the values in MAF. The key of the index MA(Sales.Store) is the OID of Store, and the pointer field has the offsets of the values in the MAF file. For example, Amount values related to the Store OID 3312 are 5, 10, 45, … because their offset in MAF are 1,3,5,… respectively. In MA(Sales.Store), we can see the index entry [3312|1,3,5,…] denoting this information. MA(Sales.Time) can be constructed in the same way. We can access Amount values sold by a specific Store object from the

index MA(Sales.Store), and this leads a significant advantage in the processing of star joins and grouping operators.

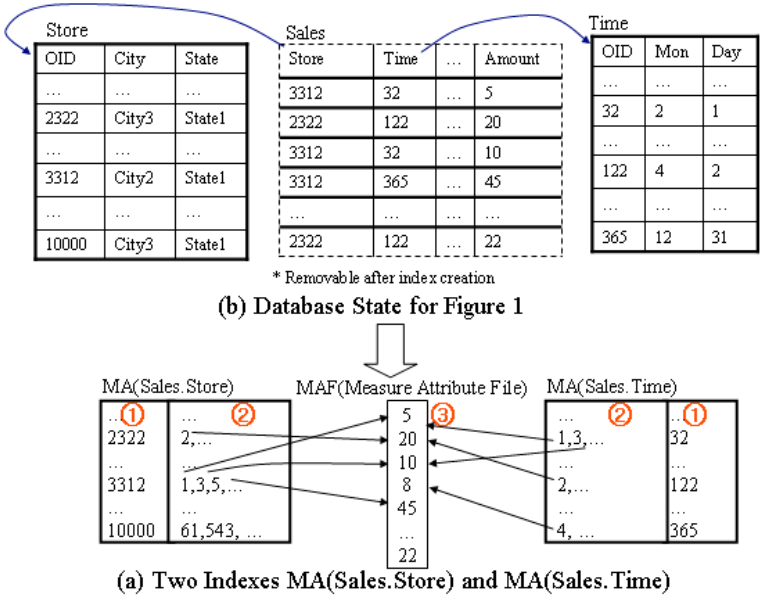


Fig. 3. Database State and Index Structure

Note that the user interface is Figure 3(b), not Figure 3(a). That is to say, users see the star schema in the form of Figure 1, and issue OLAP query as the form of Query Q1 in Section 2.1. The OLAP engine described in Section 3.2 decomposes the OLAP query for the star schema in Figure 3(b) into a number of small queries in accordance with the storage structure in Figure 3(a).

Although conventional EB and join indexes also support star joins of F and D_i, they still need significant access costs for the measure attributes. However, the MA indexes overcome this overhead by containing the measure values directly in the index. For example, we can aggregate measure attribute values (*5, 10, 45, …*) in MAF after joining store (*OID = 3312*) and sales objects by using the index *MA(Sales.Store)*. I.e., index-only processing for the joins and aggregations is possible.

3.2 Query Processing Algorithm

Here, we describe a query processing algorithm for utilizing the MA indexes. We assume the star schema in Figure 2. For the following Query Q in Figure 4, the algorithm QueryDecompAndProc() shows the query transformation process.

```
[Query Q]
Select      d1.a1, d2.a2, …, dk.ak, Agg_Func(f.m1)
From        F
Where       σ(d1.a1), σ(d2.a2), ..., σ(dN.aN)
Group by    d1.a1, d2.a2, …, dk.ak

[QueryDecompAndProc( )]
Input:  OLAP Query Q
Output: Query Result
Step1: Convert Q into Qi, i=1,2,...,N where Oi is the query on the dimension Di
Step2: For each Qi, compute the offset list Li by using the index MI(F.di)
Step3: L = L1 ∩ L2 ∩ ... ∩ LN
Step4: For each group attribute dj.aj, generate the offset list GLj by using the
       index MI(F.dj)
Step5: GLj = L ∩ GLj
Step6: For each GLj , compute aggregation by accessing the MAF.
```

Fig. 4. Query Q and Query Processing Algorithm

(Example 2)
Figure 5 shows the query processing in detail. For the query Q1 in Figure 1, the OLAP system converts Q1 into Q1-1 and Q1-2 in order to utilize the measure indexes. Q1-1 and Q1-2 can be processed by using *MA(Sales.Store)* and *MA(Sales.Time)* respectively. *MA(Sales.Store)*, as shown in Figure 3 (a), provides the offsets which point to the amounts sold by the state "*CA*". Similarly, *MA(Sales.Time)* generates the offsets for the amounts sold in *2003*. Two sets of the offsets (described by L_1={1,3,5,7,9,21,33} and L_2={1,3,5,21,34,45}) are merged into the set L={1,3,5,21} to process the logical AND operator in the WHERE clause. Each offset in L denotes the location where the qualified measure value is stored in MAF. If there are grouping attributes, the list L is decomposed into the group lists GLs by using the MA indexes: GL_1 and GL_2. Finally, measure values corresponding to GL_i are accessed and aggregated into the find results.

4 Performance Analysis

In this section, we analyze the space and time complexity for the measure attribute indexes. We first describe mathematical analysis and then query response time for a large data warehouse. Symbols in Table 1 show the parameters used in the mathematical analysis.

4.1 Space Comparison

In Figure 2, we assume the following statistics for space analysis. Here the length of OID is 8 bytes.

$n(D_i) = 10{,}000,\ n(F) = 100{,}000 \sim 100{,}000{,}000$

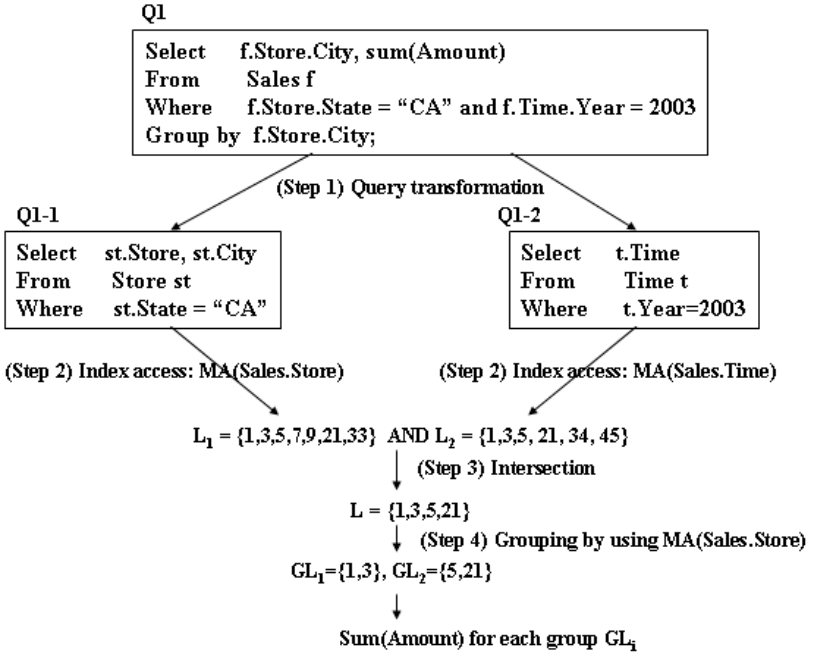


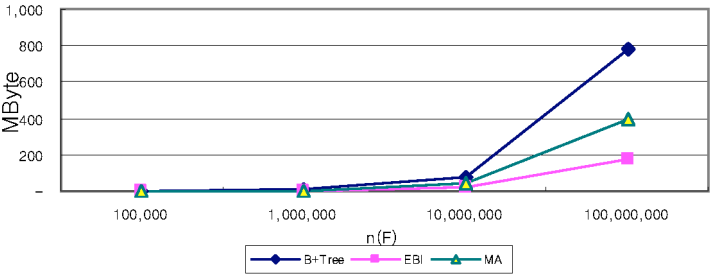
Fig. 5. Query Processing by Transformation

Table 1. The parameters used in the analysis.

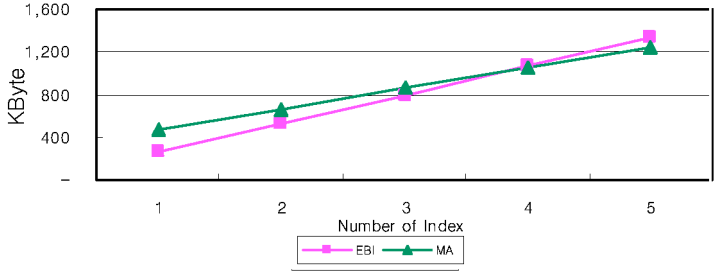
Symbols	Meaning
n(C)	Number of objects in the class C
p(C)	Number of blocks in the class C
s(pred)	Selectivity for the condition *pred*
s(C)	Selectivity of the class C
B(b, bf, k)	The expected number of block I/Os when *k* records are accessed in the address order from the file composed of *b* blocks with *bf* blocking factor [20]
c(I)	Index access cost for the index I
bf(C)	Blocking factor for the class C
G(I, attr)	Grouping cost using the index I

From these statistics, we compute the space requirements for B^+-Tree, Encoded Bit-map(EB) index, and Measure Attribute (MA) indexes. Figure 6(a) shows the comparison of the space requirements for one index. As *n(F)* increases, space requirements of B^+-Tree and the MA index increase rapidly compared with EB index. Figure 5(b) shows the space of each index structure as the number of indexes to be created increases. If we create more than 4 indexes, the space of the MA index is

smaller than that of the EB index due to the sharing of the measure attribute values in MAF as shown in Figures 2 and 3.



(a) Index Space for a single index



(b) Index Space as the number of indexes to be created increases

Fig. 6. Index Space Comparisons

4.2 Query Evaluation Cost

Here, we compare query evaluation costs for B^+-Tree, EB, and MA indexes. Various parameters shown in Table 1 are used in the mathematical cost modeling.

For query Q in Section 3.2, equations (1), (2), and (3) are the cost formulas when we use B+-Tree index I_1, EB index I_2, and MA index I_3, respectively. Aggregation cost is excluded because it has the same cost regardless of the index types. Here, we measure the query evaluation cost in the number of disk block IOs.

$$Cost1(Q) = p(D_i) + n(D_i)\times s(C)\times c(I_1) + b(\, p(F),\, bf(F),\, n(F)\times s(C)\,) + G(I_1, attr\text{-}list) \quad (1)$$

$$Cost2(Q) = p(D_i) + n(D_i)\times s(C)\times c(I_2) + b(\, p(F),\, bf(F),\, n(F)\times s(C)\,) + G(I_2, attr\text{-}list) \quad (2)$$

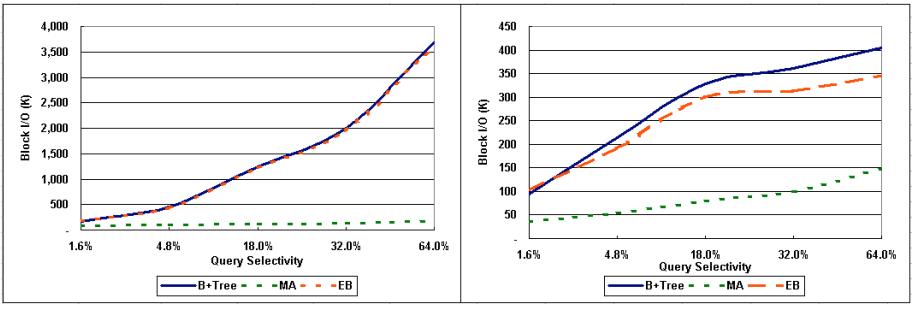
$$Cost3(Q) = p(D_i) + n(D_i)\times s(C)\times c(I_3) + b(\, p(MAF),\, bf(MAF),\, n(F)\times s(C)\,) + G(\, I_3, attr\text{-}list) \quad (3)$$

In the formula, $p(D_i)$ is the access cost for the dimension table D_i to evaluate the local conditions. The next item $n(D_i)\times s(C)\times c(I_j)$ is the index access cost for qualified objects in D_i. The third item $b(\, p(F),\, bf(F),\, n(F)\times s(C)\,)$ represents the access cost for the fact table F. The last item $G(I_j,\, attr\text{-}list)$ denotes grouping cost by using the index I_j.

We assume that the query processor evaluates the conditions of each dimension table at first and obtains OIDs of the qualified objects in the dimension tables. In the

next, an index is used to find OIDs of the qualified objects in the fact table. In Equation (1) and (2), the query processor accesses the fact table by using the OIDs of the fact table returned from the index. However, in Equation (3), the values in the measure attribute can be obtained from MAF instead of the fact table.

In Equations (1), (2), and (3), the first and the second items have almost the same cost for all indexes; but, the third and the fourth items have a great difference as the number of qualified objects in the fact table increases. The parameter *b(p(MAF), bf(MAF), n(F)*s(C))* is much smaller than *b(p(F), bf(F), n(F)×s(C))* in Equations (1) and (2) since MAF is small compared with the fact table F. Furthermore, group by operator can be processed by the index-only fashion in the MA index. Therefore, $G(I_3, attr\text{-}list)$ in Equation (3) is much smaller than those of Equations (1) and (2).



(a) Selection Query (b) Selection and Grouping Query

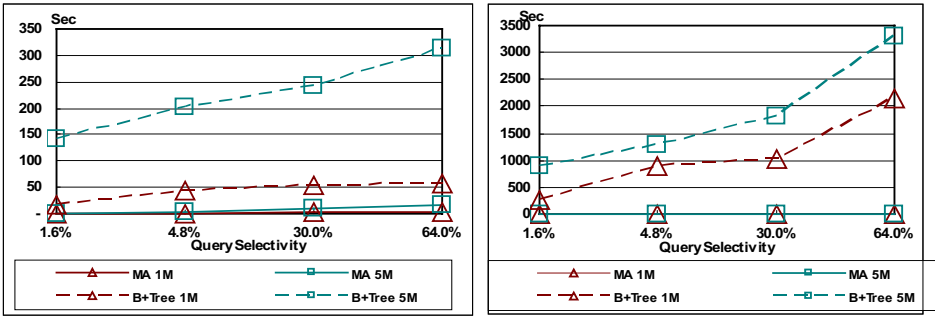
Fig. 7. Comparison of query evaluation costs for various indexes

Figure 7 shows the costs of Equations (1), (2), and (3) in a graphical form. Figure 7(a) compares the costs of queries where group by operators are not included; but Figure 7(b) shows the costs of queries having group by operators. In Figure 7(a), there is a little difference between B^+-Tree and EB indexes in the query evaluation cost. In Figure 7(b), the difference between B^+-Tree and EB is not significant due to the dominant grouping cost. This is because the conventional query processor does not utilize B^+-Tree or EB indexes in the evaluation of the grouping operators. Note that the MA index provides a superior performance regardless of the query selectivity. There are two reasons for this superiority; (1) the third item in Equation (3) is very small since it accesses the smaller MAF file instead of the huge fact table F. (2) the fourth item in Equation (3) is also very small due to the index-only processing of the grouping operators.

4.3 Experimental Results

For performance comparison, we create a data warehouse for Figure 3 containing 1,000,000 objects and 5,000,000 objects in the fact table F(Sales). Two queries Q1 and Q2 are used : Q1(F JOIN D_1) and Q2(F JOIN D_1 JOIN D_2 with Grouping). B^+-Tree and MA indexes are compared. Remind that B^+-Tree and EB indexes have almost the same cost as shown in Figure 7, especially for the queries having grouping

operators. Figure 8 shows that the MA index provides a significant performance advantage compared with B^+-Tree. For small query selectivity, there is relatively little performance difference. However, as the query selectivity increases or if the query includes group by operators, the MA index has a significant performance advantage. This performance advantage comes from the index-only processing for the expensive star joins and grouping operators. Since most OLAP queries include grouping operators with high query selectivity compared with the OLTP queries[16], this performance advantage is especially valuable.



(a) Q1(F JOIN D_1 JOIN D_2) (b) Q2(F JOIN D_1 JOIN D_2 with Grouping)

Fig. 8. Response times of B^+-Tree and MA indexes for 1M and 5M

5 Conclusions

We proposed a new index structure, called *measure attribute(MA) index*, and a query processing technique to improve the performance of OLAP queries. OLAP queries are extremely complicated due to representing the intricate business logic of the company on a huge quantity of data. That is why the performance is an important issue in the OLAP queries.

The MA index supports an efficient evaluation of the star joins and groupings that are frequently used but most expensive operators in the OLAP queries. The MA index is a variation of the path index in object databases and it supports index-only processing of the star join and grouping operators. We have implemented the MA index on top of an object-relational DBMS. The performance analysis shows that the MA index provides speedups of orders of magnitude for typical OLAP queries.

References

1. Y. Zhao, et al., "Array-based evaluation of multi-dimensional queries in object-relational database system," In Proc. Int'l Conf. ICDE, 1998.
2. C. Y. Chan and Y. Ioannidis, "Bitmap index design and evaluation," In Proc. ACM SIGMOD Conference, pp. 355-366, 1998.

3. G. Colliat, "OLAP, relational and multidimensional database system," In Proc. ACM SIGMOD Record, 25(3), 1996.
4. B. D. Czejdo, et al., "Design of a data warehouse over object-oriented and dynamically evolving data sources," DEXA Workshop, pp. 128-132, 2001.
5. R. Elmasri and S. B. Navathe, Fundamentals of database Systems, Addison-wesley, 2000.
6. V. Gopalkrishnan, et al., "Star/snow-flake schema driven object-relational data warehouse - design and query processing strategies," DaWaK, pp. 11-22, 1999.
7. J. Gu, et al., "OLAP++: Powerful and easy-to-use federations of OLAP and object databases," In Proc. Int'l Conf. on VLDB, pp. 599-602, 2000.
8. T. Johnson, "Performance measurements of compressed bitmap indices," In Proc. Int'l Conf. on VLDB, pp. 278-289, 1999.
9. M, Jurgens and H. J. Lenz, "Tree based indexes vs. bitmap indexes : a performance study," In Proc, Int'l Workshop on DMDW, 1999.
10. L. Do, et al., "Issues in developing very large data warehouse," In Proc. Int'l. Conf. on VLDB, pp.633-640, 1998.
11. C. Mohan, et al., "Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques," In Proc. EDBT, pp. 29-43, 1990. 3
12. P. O'neil and G. Graefe, "Multi-table joins through bitmapped join indics," In Proc. ACM SIGMOD Record, 24(3), pp. 8-11, Sept, 1995.
13. P. O'neil and D. Quass, "Improved query performance with variant indexes," In Proc. ACM SIGMOD, pp. 38-49, 1997.
14. F. Ravat and O. Teste, "A temporal object-oriented data warehouse model," In Proc. DEXA Workshop, pp. 583-592, 2000.
15. Stanford Technology Group, "Designing the data warehouse on relational database," White Paper.
16. TPC-D, http://www.tpc.org, 2002
17. P. Valduriez, "Join indices," ACM Trans. on Database Systems, 12(2), pp.218-246, 1987.
18. K. L. Wu and P. S. Yu, Range-based bitmap indexing for high cardinality attributes with skew, Research Report, IBM Waston Research Center, May, 1996.
19. M. C. Wu and A. P. Buchmann, "Encoded bitmap indexing for data warehouses," In Proc. Int'l Conf. ICDE, 1998.
20. S. B. Yao., "Approximating block accesses in database organizations," Comm. of the ACM, 20(4), pp. 260-261, 1977.