Functional Disk System as a High Performance Relational Storage

Masaru KITSUBREGAW, Miyuki NAKANO, Mikio TAKAGI
Institute of Industrial Science
University of Tokyo
22-1, Roppongi 7, Minato-ku, Tokyo
Japan

ABSTRACT

Functional Disk System with Relational database engine (FDS-R) is a relational storage system designed to accelerate relational algebraic operations. FDS-R employs a filtering and dynamic clustering mechanism as a special hardware function and provides an intelligent data management and an efficient data processing. The relation size which could be handled in the first prototype, however, was limited to the size of a staging buffer. Then we built up the second version of FDS-R, FDS-R II, which is designed to handle large relations efficiently. We have presented the processing strategy: the "Extended Task Cycle" for relational algebraic operations on FDS-R II, where the strategy is selected at run time from two algorithms (Nested Loop and Grace Hash Algorithms) by comparing the estimated I/O costs of them.

In this paper, we present the overview of Hardware configuration and Software system of FDS-R II. First we show the basic Task Cycle and give the performance evaluation results using the original Wisconsin Benchmark for small relations, where filtered relation can be staged on the staging buffer. FDS attains higher performance than the current database systems such as INGRES, Oracle and IDL. Next we explain the "Extended Task Cycle" which was introduce to handle the relations larger than the staging buffer size. With the expanded version of Wisconsin Benchmark, we measured the FDS-R II performance. FDS-R II attained a high performance for processing large relations as compared to other large database systems such as Gamma and Teradata. While FDS-R II uses just one disk and three MC68020, Teradata uses 40 disks and 20 AMP's and Gamma requires 8 disks and 17 VAX 11/750's.

Key Words
secondary storage, relational database, performance evaluation

1. Introduction

Recently the demands for large database processing have been increasing more and more. While the CPU performance has very much improved during the past decade, the same cannot be stated about the secondary storage, whose performance has been a very serious problem on large data processing. Aiming at the dissolution of this "I/O bottleneck" between CPU and the secondary storage, we have proposed the idea of "Functional Disk System (FDS)" in [1]. FDS is not a simple storage system but introduces some semantics of data processing in the storage system itself, incorporating a large staging buffer and a parallel processing mechanism. FDS can reduce the large overhead of the I/O drivers of conventional computer systems and provide a high-level data interface with the host.

In order to clarify the effectiveness of FDS, we have developed FDS-R II which we called "Functional Disk System with Relational database engine". FDS-R II employs a filtering and dynamic clustering mechanisms as a special hardware function to accelerate database operations, provides an intelligent data management and performs efficient data processing using multiple processors. The first version of the FDS-R prototype (FDS-R I) has been constructed and the preliminary performance evaluation of the system with the Wisconsin Benchmark has already been reported in [1]. It attained a very high performance in comparison with other database systems. In order to improve the performance, we have developed a revised system, FDS-R II. We have presented the processing algorithm employed in FDS-R II to handle large relations in [12]. On FDS-R II, we select the processing algorithm at run time from two algorithms, the Nested Loop and the Grace Hash Algorithms, by comparing their estimated I/O costs. The performance of basic relational operations was evaluated extensively by measuring the execution time. The I/O cost function used for algorithm selection was shown to be very effective.

In this paper, we describe the overview of Functional Disk System and report the basic query processing on FDS-R, which is called "Task Cycle". We evaluate the basic performance of FDS-R II for small relations with the Wisconsin Benchmark. FDS-R II attains a high performance in comparison with the other database systems such as INGRES, ORACLE, IDL and DIRECT. Then, we briefly introduce the "Extended Task Cycle" for large data processing. With the expanded version of Wisconsin Benchmark, we evaluate the performance of FDS-R II for large relations. FDS-R II...
attained a very high performance in comparison with Gamma and Teradata. While FDS-RII uses just one disk and 3 MC68020 CPU's, Teradata uses 40 disks and 20 AMP's and Gamma requires 8 disks and 17 VAX11/750's.

Section 2 briefly introduces the hardware and software configurations of FDS-RII. In section 3, we present the data flow of FDS-R and the query processing cycle when the size of the filtered relation is less than that of the staging buffer. In section 4 the basic performance of FDS-RII is measured using selection, projection, aggregation and join queries with Wisconsin Benchmark. Then we introduce the query processing method for large relations whose size far exceeds that of the staging buffer in section 5. The cost formula to choose the run time algorithm is also presented. In section 6, the performance evaluation of FDS-RII with the expanded version of Wisconsin Benchmark is presented. The last section concludes this paper and discusses future topics of FDS-R.

2. Hardware and Software Configurations of FDS-RII

2.1. Hardware Configuration of FDS-RII

Fig. 1 shows the hardware configuration of the revised version of FDS-R. FDS-RII consists of three parts: the processing part, the staging buffer and the disk part. The processing part consists of four processors, and the staging buffer is shared by these four processors, one of them being the master and the other three the slave processors. Each processor board is a commercial one which contains the MC68020 CPU(16MHz) and 1 MB of local memory. The staging buffer consists of 6MB dynamic memory. The processing part, the staging buffer and the disk part are connected by Versa Bus, the standard MOTOROLA bus. Also, in order to increase the I/O bandwidth, another special bus which we call "SO Bus" is added to the system. It connects the processor boards to the disk part and is used only to store the result relations into the disk.

The disk part is composed of 8 inch disks(NEC D2257) and the "Intelligent Disk Controller(IDC)". The IDC is the heart of this disk part and is responsible for the control of the disks and for the filtering and dynamic clustering of the data read from them.

2.2. Software Configuration

Fig. 2 shows the configuration of the FDS-R system software.

A ROM based MICROWARE OS-9 is used as the kernel on each processor. Since OS-9 is an operating system for a single processor, we developed the FDS-R communication system(FDS-R CS). FDS-R CS supports the message passing among processes on multiple processors.

The QUEL Subset System(QSS) is constructed on top of FDS-R CS. It consists of two processes: the QUEL subset process(Q-PS) on the master processor and the procedure handler process(PH-PS) on each of the four processors.

QUEL Subset Process runs on the master processor and is composed of the four elements shown in Fig. 3. Q-PS provides the user interface, parses the queries, interprets the query tree and controls both IDC and PH-PS on each processor via FDS-R CS. Q-PS generates a query packet corresponding to one relational algebraic operation after interpreting the query. A query packet consists of an IDC packet, a procedure packet and a system packet. The IDC packet contains informations about the source relations and the filtering and clustering of their tuples. The procedure packet contains informations about the relational operation to be executed. The system packet contains the information necessary to the multiple processors and the staging buffer management. According to a query packet, Q-PS drives the IDC, controls the synchronization of the multiple processors and manages the space of the staging buffer. Each of the PH-PS invokes a procedure packet sent by Q-PS in parallel.
3. Task Cycle

In FDS-R a query is decomposed into a combination of simple relational operations such as selection, projection, join, aggregation. The hash-based algorithm is applied for each of the simple relational algebraic operations. As described in [3], [4], [5], [6], [11] and [12], hash-based algorithms are efficient for large data processing. One of the properties of the hash-based algorithms is the load reduction of the operations, and the other is the natural opportunity of parallel processing provided by them.

In the following, we consider the execution of a simple relational operation and present the processing cycle for a query execution on FDS-R, in detail.

The execution of a relational operation consists of 2 phases: the Staging Phase and the Processing Phase.

1) Staging Phase

IDC reads relations from the disk and clusters them into the staging buffer. When reading a relation from the disk and transferring it to the staging buffer, IDC examines each tuple; according to some specified parameters, it filters some attribute fields and applies a hash function to a target attribute field, and then clusters the relation on the fly. At the same time, IDC also builds a hash table.

2) Processing Phase

After the Staging Phase, each processor gets a cluster from the hash table of IDC and transfers it into its own local memory. The clusters are processed in parallel and the result tuples are written back to the disks via the SO Bus.

Since one query is composed of several relational operations, the query execution on FDS-R implies that the data repeatedly streams out the disk, and then back to the disk. This one cycle of dataflow out disk to disk is the unit of a query execution on FDS-R and here we call it "Task Cycle". In FDS-R, we only considered the case in which the data could be processed in one Task Cycle, that is, the source data could fit into the staging buffer. The Task Cycle of FDS-R is shown in Fig.4 and explained in detail as follows.

4. Performance Evaluation of FDS-RII with the original Wisconsin Benchmark

In this section we show some results of the performance measurement of simple relational operations on FDS-R for small relations. FDS-RII is compared with current relational database systems with the original Wisconsin Benchmark[8].

One disk drive, 6 MB of staging buffer memory and 3 processors are activated under the measurement. According to the Wisconsin Benchmark[8], we use 1,000, 10,000 tuples as benchmark relations and each tuple is 182 bytes long. The attribute a0, a1 and a2 are 2-byte integers. The attribute a0 is unique and randomly ordered integer between 0 and 9999. The attribute a2 is randomly ordered integer between 0 and 99.

We use the following four queries for the performance measurement.

Table 1 shows the result of selection query. In this query, the benchmark relation is 10,000 tuples and the selectivity is 10%. Therefore, we get the result relation whose size is 1,000 tuples. The result relation is written back to the disk. On FDS-R, the predicate e.a0 < 1000 is evaluated by IDC.

Table 2 shows the results of the projection query. This query is the duplicate elimination. In this query, the benchmark relation is 1,000 tuples and each tuple has an unique value so that the result relation is equal to the source relation. The result relation is written back into the disk.

Table 3 shows the results of the join query. In this query, both of the two relation have 10,000 tuples. The predicate e.a0 < 1000 is executed by IDC and the selected tuples are clustered with the value of attribute a0. Dynamically clustered tuples are processed in parallel by activated three processors. We have the result relation whose size is 1,000 tuples. The result relation is written back into the disk.
Table 4 shows the results of the aggregation query. In this query, the attribute a2 is randomly ordered 2 byte integer between 0 to 99 so that the source relation whose size is 10,000 tuples is partitioned into 100 clusters with the value of the attribute a2. On FDS_R, only the two attribute fields are extracted and staged on the staging buffer. At the same time, tuples are clustered with the value of the attribute a2 by IDC and each cluster is processed in parallel.

All results shows that FDS-R attains a high performance in comparison with the other database systems. Especially, the result of join query is very fast. This is because Functional Disk System can reduce the large overhead of the I/O driver and IDC can provide the efficient data management.

5. Query Execution for Large Relations on FDS-R

5.1. Extended Task Cycle[12]

For processing relations larger than the size of the staging buffer, it is important to divide the source relation into the size of a staging buffer and to manage the staging buffer for processing divided data. On FDS-R, we employ two algorithms for large data processing in order to get high performance. FDS-R selects the processing method from two algorithms: the Nested Loop and the Grace Hash Algorithm[3,4].

Table 1. The execution time of selection queries with the Wisconsin Benchmark

range of e is tenKtuple
retrieve into tmp (e.all) where e.a0 < 1000

<table>
<thead>
<tr>
<th>Selection Query(10% selectivity)</th>
<th>U-INGRES</th>
<th>64.4 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-INGRES</td>
<td>53.9 sec</td>
<td></td>
</tr>
<tr>
<td>ORACLE</td>
<td>230.6 sec</td>
<td></td>
</tr>
<tr>
<td>IDM noDAC</td>
<td>33.4 sec</td>
<td></td>
</tr>
<tr>
<td>IDM DAC</td>
<td>23.6 sec</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td>46.0 sec</td>
<td></td>
</tr>
<tr>
<td>FDS-R</td>
<td>3.4 sec</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. The execution time of projection query with the Wisconsin Benchmark

range of e is oneKtuple
retrieve unique into tmp (e.all)

<table>
<thead>
<tr>
<th>Projection Query(Duplicate Elimination)</th>
<th>U-INGRES</th>
<th>236.8 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-INGRES</td>
<td>122.0 sec</td>
<td></td>
</tr>
<tr>
<td>ORACLE</td>
<td>199.8 sec</td>
<td></td>
</tr>
<tr>
<td>IDM noDAC</td>
<td>122.2 sec</td>
<td></td>
</tr>
<tr>
<td>IDM DAC</td>
<td>68.1 sec</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td>58.0 sec</td>
<td></td>
</tr>
<tr>
<td>FDS-R</td>
<td>1.5 sec</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. The execution time of join query with the Wisconsin Benchmark

range of e is tenKtuple
range of f is tenKtuple
retrieve into tmp (e.all,f.all) where e.a1 = f.a1 and e.a1 < 1000

<table>
<thead>
<tr>
<th>Join Query(10% Join selectivity)</th>
<th>U-INGRES</th>
<th>10.2 min</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-INGRES</td>
<td>1.8 min</td>
<td></td>
</tr>
<tr>
<td>ORACLE</td>
<td>300 min</td>
<td></td>
</tr>
<tr>
<td>IDM noDAC</td>
<td>300 min</td>
<td></td>
</tr>
<tr>
<td>IDM DAC</td>
<td>10.2 min</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td>6.5 sec</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. The execution time of aggregation query with the Wisconsin Benchmark

range of e is tenKtuple
retrieve (e.a2, sum=sum(e.a1 by e.a2))

<table>
<thead>
<tr>
<th>Aggregation Query(Partitions = 100)</th>
<th>U-INGRES</th>
<th>174.2 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-INGRES</td>
<td>484.8 sec</td>
<td></td>
</tr>
<tr>
<td>ORACLE</td>
<td>1487.5 sec</td>
<td></td>
</tr>
<tr>
<td>IDM noDAC</td>
<td>67.5 sec</td>
<td></td>
</tr>
<tr>
<td>IDM DAC</td>
<td>38.2 sec</td>
<td></td>
</tr>
<tr>
<td>DIRECT</td>
<td>229.5 sec</td>
<td></td>
</tr>
<tr>
<td>FDS-R</td>
<td>2.5 sec</td>
<td></td>
</tr>
</tbody>
</table>
(1) Nested Loop Algorithm

It may not be necessary to explain this algorithm. This is the most simple iterative method. Usually this method is considered to be inefficient, since the outer pages of the relation are read many times. Later it is shown, however, that it outperforms the next partitioning algorithm for relations under a certain size.

(2) Grace Hash Algorithm

In this method, the source data is partitioned into clusters which do not exceed the size of the staging buffer by applying an appropriate hash function, and the generated clusters are stored back into the disk. Then the clusters are read one by one from the disk. The tuples of the cluster on the staging buffer are processed as described above.

ExtendedTaskCycle(data,operation)

StagingPhase(data,operation)
Begin
select the processing method
from Nested Loop Algorithm and Grace Hash
Algorithm
if Grace Hash Algorithm is selected
Begin
  determine the number of I/O clusters
  make I/O clusters of source data
  for each overflowed I/O cluster
    StagingPhase(the overflowed I/O cluster,
operation)
  End
else Nested Loop Algorithm is selected
Begin
  determine the number of outerloops and inner-loops
End
End
ProcessingPhase(data,operation)
Begin
for each I/O cluster of outer relation
  for each I/O cluster of inner relation
    load the specified data into the staging buffer
    process data on the staging buffer according to
the operation
End

End

Fig.5 The Extended Task Cycle

FDS-RII supports large data processing by extending the Task Cycle. Here we introduce the extended Task Cycle where the processing algorithm is chosen at run time from two algorithms and the staging buffer is managed properly. Fig.5 shows this Extended Task Cycle. In the Staging Phase of the Extended Task Cycle, a run time method is selected from two algorithms with using the estimated I/O cost and the source relations are divided into small data partitions which are termed "I/O clusters". The Processing Phase is invoked as many times as the number of I/O clusters generated at the Staging Phase.

5.2. Selection Criteria for the Run Time Algorithm

On FDS-RII, the run time algorithm is determined with using the I/O cost formula, which are described in [12] in detail. In this paragraph, we show only conditions for choosing the run time routine from the two algorithms. The following parameters is used to express the selection criteria for the run time algorithm on FDS-RII.

\[ R \] : size of relation R in bytes
\[ M \] : size of staging buffer in bytes
\[ f_r, f_s \] : filtering factor of relation R,S

(1) Join Operation

The number \( n \) of outerloops for the smaller relation R is given by \( n = \left\lfloor \frac{f_r R}{M - \alpha} \right\rfloor \) (\( \alpha = 1 \) page). The condition under which the I/O cost of the Nested Loop Algorithm is better than that of the Grace Hash Algorithm is given as follows.

\[ n < 1 + 2f_r + 2f_s < 5 \quad \text{cond}(1) \]

The condition under which the I/O cost of the Nested Loop Algorithm with the filtered inner relation is better than that of the GRACE Hash Algorithm is given as follows.

\[ n < 1 + 4f_r, \quad \text{cond}(2) \]

The following condition determines whether the inner relation S is written back at the Nested Loop Algorithm.

\[ f_r < 1 - \frac{1}{n} \quad (n \geq 2) \quad \text{cond}(3) \]

(2) Projection and Aggregation Operations

As well as the join operation, the number \( n \) of outer-loops is given by \( n = \left\lfloor \frac{f_r S}{M - \alpha} \right\rfloor \) (\( \alpha = 1 \) page). The condition under which the I/O cost of the Nested Loop Algorithm is better than that of the Grace Algorithm is given as follows.

\[ n < 1 + 4f_r, \quad \text{cond}(4) \]
6. Performance Evaluation of FDS-II with the expanded version of Wisconsin Benchmark

In this section we show some results of the performance measurement of the prototype machine of FDS-II for large relations. For the evaluation of results, we compare them with the result of the expanded version of Wisconsin Benchmark[7].

One disk drive, 6 MB of staging buffer memory and 3 processors are activated under the measurement. According to the Wisconsin Benchmark[7,8], we use 10,000, 100,000, 1,000,000 tuples as benchmark relations and each tuple is 208 bytes long. In this evaluation, we do not support an index structure in the database, that is, our results are measured by using no indexed relations only.

6.1. Selection Query

We measured the performance of the following selection query. The result relation is written back into the disk.

range of e is relation R
retrieve into tmp (e.all) where e.a1 ∈ X

Table 5. The execution time of selection queries with the expanded version of Wisconsin Benchmark

<table>
<thead>
<tr>
<th>Selection Query (1% selectivity)</th>
<th>Tuple</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>1.6 sec</td>
<td>13.8 sec</td>
<td>134 sec</td>
<td></td>
</tr>
<tr>
<td>Teradata</td>
<td>6.9 sec</td>
<td>28.2 sec</td>
<td>213 sec</td>
<td></td>
</tr>
<tr>
<td>FDS-II</td>
<td>2.9 sec</td>
<td>24.8 sec</td>
<td>244 sec</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Selection Query (10% selectivity)</th>
<th>Tuple</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>2.1 sec</td>
<td>17.4 sec</td>
<td>182 sec</td>
<td></td>
</tr>
<tr>
<td>Teradata</td>
<td>16.0 sec</td>
<td>111.0 sec</td>
<td>1107 sec</td>
<td></td>
</tr>
<tr>
<td>FDS-II</td>
<td>3.8 sec</td>
<td>34.9 sec</td>
<td>355 sec</td>
<td></td>
</tr>
</tbody>
</table>

The selection query is the basic query which characterizes the performance of the system. On FDS-II, the predicate e.a1 < X is executed by IDC on the fly. So we expect that the execution costs of a selection query will be almost equal to the I/O costs and that the execution time will increase as a linear function of the reading and writing data sizes. Our measurement used two sets of selection queries; first with 1% selectivity and then with 10% selectivity. Table 5 shows the results of FDS-II and the other systems, Gamma and Teradata whose performance is shown in [7]. The execution time of FDS-II is 2.5 times as large as that of Gamma. Although FDS-II activates just one disk, while Gamma and Teradata activate 8 disks and 20 disks respectively, FDS-II attains a high performance.

6.2. Join Query

We execute the following join query for the performance measurement. The result relation is written back into disk.

range of e is relation R
range of f is relation S
retrieve into tmp (e.all,f.all) where e.a1 = f.a1 and e.a1 < X

On FDS-II, the predicate e.a1 < X is executed by the filtering function of IDC and selected tuples are clustered by the value of attribute a1. Then the clustered data are staged on the staging buffer and are joined by processors. The relation R and S have the same number of tuples.

| Table 6. The execution time of join query with the expanded version of Wisconsin Benchmark |
|------------------------------------------|----------|----------|----------|
| Join Query (1% selectivity)              | Tuple | 10,000 | 100,000 | 1,000,000 |
| FDS-II                                   | 7.4 sec | 69.7 sec | 994 sec |
| Teradata                                 | 35.6 sec | 331.7 sec | 3535 sec |
| Gamma                                    | 5.1 sec | 36.3 sec | 703 sec |

Table 6 shows the measurement results of join query. From the results of measurement in Table 6, the execution time of join query increases as a linear function of the input and output data sizes as discussed in section 4. As shown in Table 6, the execution time of FDS-II is three times as large as that of Gamma. However, while the measurement environment of Gamma consists of 8 disk nodes (the processors with disks), 8 nondisk processors and 1 scheduler[7], that of FDS-II consists of only one disk and 3 CPU's. Since the cost of join query on FDS-II is almost the same as the I/O cost, we expect that the disk-reading cost will be easily decreased by using multiple disks.

6.3. Projection Query

We execute the following projection query for the performance measurement. This query is the duplicate elimination. All tuples have an unique value, which means that the source relation is no duplicate data. The result relation is written back into disk.

range of e is relation R
retrieve unique into tmp (e.all)

On FDS-II, tuples are clustered by the value of attribute a1. Then the clustered data are staged on the staging buffer and are searched by processors in parallel.
Table 7 The execution time of projection queries with the expanded version of Wisconsin Benchmark

<table>
<thead>
<tr>
<th>Projection Query (Duplicate Elimination)</th>
<th>Tuples</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDS-R</td>
<td>sec</td>
<td>1.7</td>
<td>13.9</td>
<td>286</td>
</tr>
</tbody>
</table>

Table 7 shows the measurement results of projection query. From the results of measurement in Table 7, the execution time of 100,000 tuples is 20 times larger than that of 10,000 tuples. This is because Grace Hash Algorithm is used for 100,000 tuple source relation. In Grace Hash, the source relation is at first read and partitioned with hash function. Then each partition is written back to the disk. The partitions are read one by one to be processed by the processors. Finally the result tuples are written into the disk. Thus two disk reads and two disk writes are required. In this case, the result relation has the same number of tuples as the source relation. Therefore the I/O cost for the Grace Hash Algorithm is two times larger than that of one Task Cycle, which means the time for 100,000 tuple processing, 286 sec is almost 20 times as large as 10,000 tuple processing, 13.9 sec.

6.4. Aggregation Query

We execute the following aggregation query for the performance measurement.

range of e is relation R

retrieve (e.a2, sum := sum(e.a1 by e.a2))

On FDS-RII, tuples are clustered by the value of attribute a2. Then the clustered data are staged on the staging buffer and are processed by processors in parallel.

Table 8 The execution time of aggregation queries with the expanded version of Wisconsin Benchmark

<table>
<thead>
<tr>
<th>Aggregation Query (Partitions = 100)</th>
<th>Tuples</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDS-R</td>
<td>sec</td>
<td>2.9</td>
<td>24.9</td>
<td>307</td>
</tr>
</tbody>
</table>

Table 8 shows the measurement results of aggregation query. In case of 1,000,000 relation, the Grace Hash Algorithm is applied. From Table 8, we can find that the execution time increases as a linear function of the relation sizes contrary to the projection query. The execution time of 100,000 relation is almost ten times larger than that of 100,000 relation. On FDS-RII, only two aggregation fields are extracted from original tuples and the I/O clusters are very small. Therefore the reading cost of source relation dominate the total processing time, while the cost of making I/O clusters and reading these clusters are very small.

7. Conclusions

We present the performance of FDS-RII, which is designed to handle large relations efficiently. At first, we present the Task Cycle which is the unit of the query execution on FDS-R. We measure the basic performance of FDS-RII with simple relational operations. With the Wisconsin Benchmark, we evaluate the performance of FDS-RII and find that FDS-RII attains a high performance in comparison with the other database systems such as INGRES, ORACLE, IDM and DIRECT. We introduce the Extended Task Cycle to handle large relations on FDS-RII. On FDS-RII, the processing algorithm is selected at run time from two algorithms (Nested Loop Algorithm, Grace Hash Algorithm), by comparing their estimated I/O costs. With the expanded version of Wisconsin Benchmark, we measured the performance of FDS-RII for large relations. FDS-RII attained a high performance for very large relations as compared to other large database systems such as Gamma and Teradata. While FDS uses just one disk and three MC68020, Teradata uses 40 disk(20 AMP's) and Gamma requires 8 disks and 17 VAX750's.

In this paper, we measured the performance of FDS-RII by using the uniformly distributed data. We are planning to implement the Dynamic Hybrid Grace Hash Algorithm[9] and to measure the performance for large relations which are not uniformly distributed and to analyze results when the I/O cluster overflows.

Acknowledgement

This research is partially supported by Casio Foundation, Denki-Tsushin-Fukyu Foundation, Inamori Foundation and the Murata Science Foundation.

[References]


- 249 -


