Online Management of Massive Data on Tertiary Storage: Dream or Reality?

BAOLIANG LIU[1]
JIANZHONG LI[1]
KIMUTAI KIMELI[2]

[1]Harbin Institute of Technology
School of Computer Science and Technology
90 Xidazhi Street, Harbin City, Heilongjiang Province, P.R.China
[2]Moi University, Eldoret, Kenya
[1](liubl,lijzh)@hit.edu.cn
[2]vkkimeli@hotmail.com

Abstract. Despite the decrease in disk price and the increase in disk storage capacity, the storage requirements of many organizations still can't be met economically using disk system alone. Tertiary storage offers a lower-cost alternative. Whether it is feasible to manage massive data on tertiary storage or not is still a question with no answer. In this paper, we take the join operation, which is one of most common and time consuming database operations, for example to demonstrate the feasibility of online management of massive data on tertiary storage. The experimental results show that it is more important to choose the right operation algorithm than to choose the storage devices in the analysis and mining of massive data. Tertiary storages can be used to manage massive data as effective as disks and they provide a new solution to the problem of online querying and accommodating massive data.

Keywords: Massive Data Management, Tertiary Storage, Join Operation, Online.

(Received August 9, 2005 / Accepted November 29, 2005)

1 Introduction

Enormous quantities of data have been accumulated by enterprises or scientific communities, the analysis and mining of which will greatly help the decision making and scientific discovery. Many enterprises keep a record of every customer transaction they have ever performed, typically in some DBMS (Database Management System). Yet, the total volume of data generated by such an enterprise is quickly exceeds the disks storage capacity (affordably). The data have to be migrated onto less-costly off-line tertiary storage, such as tape library. Tertiary storage has the characteristics of unlimited storage capacity, low cost and space saving. Unfortunately, DBMS can only operate on disk resident data. It can't manage tertiary resident data unless migrates the data back onto disk again.

Tertiary storage has long been chosen as primary storage devices to store observation data and deducing data in scientific area. These data is typically organized as files and they can't be managed by DBMS. Whereas many scientific applications, such as the global change studies that the Earth Observing System (EOS) [1] and Project Sequoia [2] will enable, require advanced data analysis capabilities, and would like to use a DBMS for relating and tracking the data. Unfortunately, aside from metadata management, today's DBMS has little to offer these applications. They can neither handle the huge volume of data required by the applications, nor can they access the media on which the data is stored.

Although tertiary storage has long been important to the commercial and scientific communities, but from DBMS prospective, tertiary storage, such as tapes and optical disks, are second class citizens comparing to magnetic disks and main memory. One reason is the low data transfer rate and the high positioning cost that prevent the online usage of these devices. The other reason is discrepancies between devices produced by different vendors. For example, tape libraries are sequen-

tial access devices with larger capacity whereas optical disk libraries are random access devices with smaller capacity.

Since 1990's researchers have put much effort in technologies to manage massive data on tertiary storage and many encouraging results have come up. These results can be divided into the following five areas: (1) Improving data transfer rate of tertiary storage. Not only physical data transfer rate are improved greatly, but also Striping [3, 4, 5, 6] technology was introduced to organize tertiary resident data. Striping technology distributes data onto many cartridges, such as tape and optical disk, to improve the data retrieval rate by parallel accessing these cartridges. (2) Reducing the overall positioning time of tertiary storage by scheduling the requests. Requests execution are reordered that requests for online cartridge are served first. Other technologies such as caching and data pre-fetching are also used. (3) Tertiary resident data organization methods. Specially designed data organization methods for scientific multi-array data [7, 8] and data distribution model [9] were presented. Data accessing efficiency can be further improved by changing the dimension order of multi-array based on dimension access frequency. (4) Database operations on tertiary resident data. Many tape join operations [10] and sort [11] operations for tertiary storage were presented. (5) Query and query optimization methods on tertiary resident data. Query processing technologies such as Pre-execution and Batching [12] were presented. Queries are first executed on disk surrogate information, and then the tertiary resident data access order which can help to schedule queries is known. Finally the query is executed on tertiary resident data in the batching phase. Re-ordering [13, 14] query optimization method had also been presented. Re-ordering technology first divides a query into many sub-queries. Each sub-query only access data on one cartridge. Reordering the execution of these sub-queries and serves those sub-queries that access online cartridge data first. It can improve the query processing efficiency by changing the order of cartridge swapping in and out.

Despite the above results the study of database technologies on managing tertiary resident data is still in its beginning phase. One of key question is if tertiary storage can be used to online manage massive data, which is still no answer. In this paper, we take the join operation, which is one of the most common and time consuming database operations, for example to illustrate the feasibility of online management of tertiary resident data. Join operation is the base operator for data analysis and mining. We believe the study of join operation can sufficiently shows if online management of tertiary resident data could be fulfilled. Previous join operation methods are all migrated from disk join operation, which are not efficient enough. In this paper we presented a more efficient join method that is designed for tape library. During the join processing, join attributes are first separated and the join is performs on these join attributes, the results is join attributes index. Then non-join attributes are scanned using the join result index. The experimental results show that it is more important to choose the right operation algorithm than to choose the storage devices in analysis and mining of massive data. Tertiary storage can be used to manage massive data online as effective as disks. They provide a new solution to the problem of online querying and accommodating massive data.

The rest of paper is organized as follows: In section 2 we briefly describe the architecture of tertiary storage. In section 3 we present the Attribute Separating Join method. The experimental results in section 4 demonstrate the effectiveness of using tape library to manage the massive data. Section 5 concludes the paper.

2 Tertiary Storage System Model

The tertiary storage system is illustrated in figure 1. Tertiary storage system consists of main memory, disks and tertiary storage such as tape library. They all connect to system bus. The capacities of the devices increase from top to bottom in the storage level and the access speeds decrease in the same order. Conventionally the upper level device is used as buffer for the lower level device. The data exchange between disk and tape drives and the data exchange among different tape drives are through main memory.

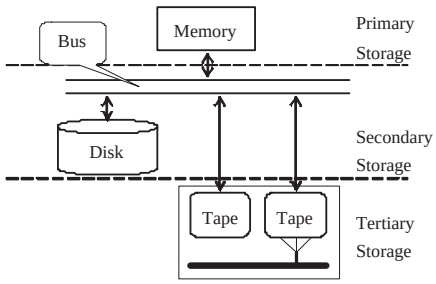

Figure 1: Tertiary storage architecture

Tape libraries consist of three parts: tape drives, mechanic arms and tape shelf. Tape library has many tape drives. There are many tapes on the shelf. The mechanic arm could be instructed to fetch a tape and load it into a tape drive, and it could also unload a tape from a drive and put it back onto some slot on the shelf. We

Table 1: Parameters of Tape Library

Parameter	Experimental Devices	High End Tertiary Devices
Tape Library Type	Exabyte 220	IBM LTO 3584
Tape Drive Type	Eliant 820	Ultrum 2
Tape Type	8mm	SDLT Tape
Tape Drive Number	2	192
Tape Capacity	10G	400GB
Tape Mount Time	71s	49
Positioning speed	5MB/s	unknown
Data transfer rate	3.5MB/s	70MB/s
Slot number	20	6881
Capacity	200GB	2.75PB

give some parameters of our experimental tape library in table 1 and we list the current high end tape library for comparison.

Table 2: Parameters Used in this Paper

Parameter	Description
$\|R\|$	Size of R (similar for S, M, D)
$\|\|R\|\|$	Number of tuples in R (similar for S)
J_R	Join attribute relation of R (similar for S)
A_R	Non join attribute relation of R (similar for S)
$c(R)$	Reducing ratio of R (similar for S)
JRI	Join result index
RC	Temporary semi join results
$\|D\|$	Disk cache size
$\|M\|$	Memory cache size
X_D	Data transfer rate of disk
X_T	Data transfer rate of tape

3 Attribute Separation Join

Attribute Separation Join (ASJ) can be divided into three phases, namely attribute separation phase, join phase and materialization phase. We will take relation $R(r_j, r_1, \cdots, r_n)$ and $S(s_j, s_1, \cdots, s_n)$ for example to illustrate the procedure of ASJ. First, relation R (S) are separated into join attribute relation J_R (J_S) and non-join attribute relation A_R (A_S) in the attribute separation phase. Second, join are performed on the two join attribute relations J_R and J_S, the result of which is join result index (JRI). In the materialization phase, non-join attribute relation A_R and A_S are scanned using JRI to get the final join results.

ASJ algorithm is described as follows:
Algorithm: AttrSepJoin
Input: $R(r_j, r_1, \cdots, r_n)$, $S(s_j, s_1, \cdots, s_n)$
Output: Join Result
(1) J_R, A_R = JoinAttrSep(R);
(2) J_S, A_S = JoinAttrSep(S);
(3) JRI = JoinPhase(J_R, J_S);
(4) Materialization(JRI, A_R, A_S);

3.1 Attributes Separation Phase

Suppose $R(r_j, r_1, \cdots, r_n)$ and $S(s_j, s_1, \cdots, s_n)$ are two relations, $r_1, \cdots, r_n$ and $s_1, \cdots, s_n$ are non-join attributes of R and S respectively, r_j and s_j are the join attributes. In the attribute separation phase relation R is separated into join attribute relation $J\_R(r_{id}, r_j)$ and non-join attributes relation $A\_R(r_{id}, r_1, \cdots, r_n)$. r_{id} is auto generated tuple identifier. One tuple in relation R corresponds to one tuple in J_R and one tuple in A_R respectively. Relation S is processed the same with relation R.

Attribute separation algorithm is described below:
Algorithm: JoinAttrSep
Input: relation $T(t_j, t_1, \cdots, t_k)$
Output: relation $J\_T(t_{id}, t_j)$ and $A\_T(t_{id}, t_1, \cdots, t_k)$
(1) t_{id} = 0;
(2) **FOR** each tuple in T **DO**
(3) Do selection and projection;
(4) t_{id}++;
(5) Write tuple (t_{id}, t_j) into J_T;
(6) Write tuple $(t_{id}, t_1, \cdots, t_k)$ into A_T;
(7) **END FOR**

Tertiary storage systems have many drives and data can be accessed in parallel. The join attributes separation phase requires at least three drives, one for reading relation T and the other two for write join attribute relation J_T and non-join attribute relation A_T. Tertiary storage typically has many dives, and drive can be plugged into the system if needed, so ASJ algorithm has good adaptively.

3.2 Join Phase

There are two steps in the join phase: in step I relation J_R are divided into many smaller divisions and

each division can be loaded into memory. In order to reduce the positioning time of drive, each division must continuously stored on tapes, which requires scanning relation J_R at least times and in each scan at most divisions can be produced. In step II we read J_S cyclically into disk cache and at the same time divide J_S with the same hash function with J_R. When disk get full, we join the part of relation in disk with relation J_R. This process is ended until J_S is finished processing. The results of join phase is the tuple identifier pair (r_{id}, s_{id}). It can be used as index to scan the two non-join attribute relation, so we call it Join Result Index.

We describe the join algorithm below:

Algorithm: JoinPhase
Input: J_R and J_S
Output: JRI
(1) **FOR** i=1 **TO** DO
(2) Generate divisions of J_R;
(3) Write these divisions into tapes.
(4) **END FOR**
(5) **WHILE** J_S is not end **DO** (exeute (6), (7) (10) in parallel)
(6) Read J_S_{i+1} into disk and hash them with the same hash functions
(7) Join J_S_i (read in the previous cycle) with J_R and write join result into memory cache
(8) **IF** memory cache full **THEN**
(9) Sort JRI based on s_{id} and write into disk
(10) **END IF**
(11) **END WHILE**

3.3 Materialization Phase

In materialization phase the two non-join attribute relations A_R and A_S are merged with the help of JRI. The basic idea is to sort JRI based on s_{id} and then merge s_{id} with A_S which is naturally sorted on s_{id}. Then we substitute s_{id} of A_S with r_{id} of JRI. The merge result is temp relation $RC(r_id, s_1, \cdots, s_m)$. RC is then sorted on r_{id}. Finally we merge RC with A_R and get the final join results.

We describe the materialization phase below:

Algorithm: Materialization
Input: JRI, A_R, A_S
Output: Final join results
(1) Multi-way sort JRI on s_{id};
(2) **WHILE** (not end JRI) **DO**
(3) **IF** disk cache is full **THEN**
(4) sort RC based on r_{id}
(5) Merge RC and A_R, then output the results.
(6) **END IF**
(7) Substitute r_{id} with s_{id} in A_S;
(8) Write merging result into memory cache;
(9) **IF** memory cache is full **THEN**
(10) Sort tuples in memory cache based on r_{id} and write into RC
(11) **END IF**
(12) **END WHILE**

4 Experimental Results

We implemented ASJ algorithm on a 550MHz Pentium system running Linux 7.2. The computer has 128MB of main memory, 20GB of disk space and an INITIO SCSI-2 bus, to which a Exabyte 220L tape library connected. The tape library has two Eliant 820 drives. The cartridge (tape) capacity is 10GB without compression. There are 20 tapes on the shelf. The parameters of Exabyte 220L are listed in table 2.

We first compared ASJ algorithm with other tape join algorithm. It is indicated in [11] that $CTT - GH$ is the candidates for massive data management in efficiency and scalability, so we only compare ASJ with CTT-GH. We briefly describe the working procedure of CTT-GH below:

(1) In Step I, relation R is scanned $\lceil |R|/|D| \rceil$ times. In each scan, a fraction of R partitions are generated, in full, on the disk. The number of partitions completed in each scan is $|D|/|M|$, the product of the total number of partitions $|R|/|M|$ and the fraction of R that fits on disk $|D|/|R|$. Once all scans have been performed, all partitions of R are stored contiguously on tapes. In the case of $|R| < |D|$, which reduced to CDT-GH, we only scan relation R once and hash it on the disk.

(2) Step II is iterated until S is exhausted. In each iteration i, $|S_i| = |D|$ ($|S_i| = |D| - |R|$ if $|R| < |D|$) blocks of S data are read from tape, hashed, and written into partitions which are on the disk. A join process then reads each partition of R into memory and joins it with the corresponding partition of S. Note that a hash process can simultaneously read more data from S and produce the hash partitions needed in the next iteration.

There are three factors that affect the performance of ASJ, namely dataset size, reducing ratio of each relation and the cardinality of JRI. Reducing ratio influences the reduction of redundant I/O and relation scan times. The cardinality of JRI affects the size of RC, and hence affects the cost of materialization phase. We compared ASJ with CTT-GH on these three factors.

To further demonstrate the effectiveness of managing massive data with tertiary storage, we implemented three disk join algorithms, namely nested-loop join, hash based join and sort merge join, and compared ASJ with these three algorithms. The results show that it is

more import to choose the right join algorithm than to choose the storage device in massive data management, which in turn indicates the effectiveness and feasibility of managing massive data with tertiary storage.

In all the experiments we set $|S| = 2|R|$, $\|S\| = 2\|R\|$ and reducing ratio $c(R) = c(S)$. In order to test the influence of the cardinality of JRI, we make the join attribute of R distinct in the data we generated, the join attribute of S references join attribute of R, but the join attributes of S is generated randomly. So the cardinality of JRI equals $\|S\|$ without other filters. In order to vary the cardinality of JRI, we use a random switch-variable to multiply the tuple just generated. When the switch-variable equals 1, we output the join results and when the switch-variable equals 0, we discard the join results. Thus we can vary the cardinality of JRI by changing the possibility that switch-variable equals 1. In massive data environment, the capacity of memory and disk cache is much smaller than the data amount. It requires high end devices to accommodate massive data. In this paper we take a method of reducing the size of memory, disk and dataset proportionally to compare the performance of ASJ and CTT-GH in the same environments. We set the size of main memory to 40MB and disk size to 650MB respectively in all the experiments.

4.1 Performance Related to Dataset Size

In the first experiment, we compared the performance of ASJ and CTT-GH with varying dataset size. The dataset size varied from 3GB to 30GB (relation R size varied from 1GB to 10GB). We set $c(R) = c(S) = 10$ and $\|JRI\| = 0.1\|S\|$. When R size was smaller than 3G bytes, both J_R and J_S could fit in disk at the same time. Neither J_R nor J_S could fit in disk when R size was larger than 6GB. We listed the execution time of both algorithms in figure 2. Obviously the performance of ASJ is much better than that of CTT-GH even both J_R and J_S can't fit on disk. One reason is that there is no redundant I/O in ASJ. Relations are often tuple wise organized in external memories (disks, tertiary storage) and the data on these external memories are read in block. Not all the attribute of a relation are join attribute, but they are loaded into memory together with join attributes. They are redundant I/O of the join operation, which should be avoid. The other reason is that the relation scan times of partitioning relation R is greatly reduced. Tape libraries are sequential access devices, in order to avoid the high positioning cost when read the corresponding partitions of both relations, the data of each partition need to be stored continuously on tape. In order to generate these physically continuous partition, we need to scan the relation for many times. We can notice that ASJ algorithm scales well with the increase of dataset size.

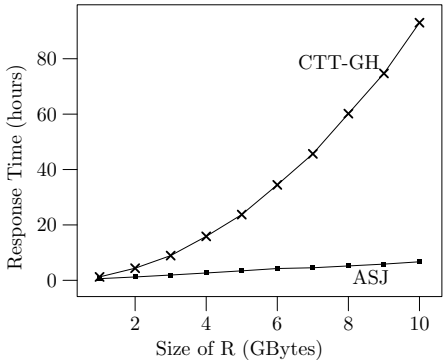

Figure 2: The influence of dataset size to performance of algorithms

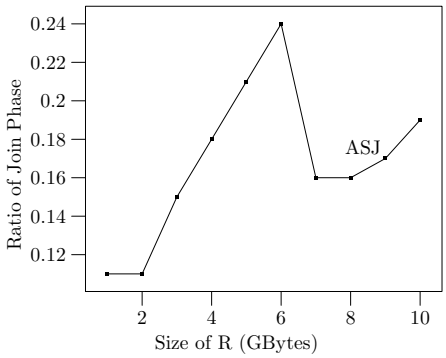

Figure 3: Ratio of join phase to the total execution time

From figure 3, we can see that the execution time of join phase only take a minor part of the total execution time, notice that there is no redundant I/O in the join phase. We can draw a conclusion from figure 2 and 3 that ASJ can effectively avoid redundant I/O problem and greatly reduce the relation scan times and hence improve the effectiveness and scalability of tape join operation.

4.2 Performance Related to Reducing Ratio

In the second experiment we use 30GB (R size is 10GB) dataset to test the influence of reducing ratio to join performance. We set $\|JRI\| = 0.1\|S\|$. The reducing ratio was varied from 10 to 100 by increasing the tuple size of R and S while decreasing the number of tuples accordingly in order to keep the size of the dataset. We fixed the size of the join attribute because generally the size of join attribute is not large. Since the CPU cost

was negligible compared to I/O cost, the reduction of response time couldn't be caused by the reduction of the number of tuples. The experimental result was listed in figure 4. As we had imagined, the total execution time of ASJ decreased as we increased the reducing ratio. The execution time of join phase with larger reducing ratio took less proportion in the total execution time than that with smaller one. There were more chances that the join attribute relations could fit on disk with larger reducing ratio. We can see clearly from figure 4 that when we increased the reducing ratio, the execution time would converge to a constant value which is the least I/O cost of join operation.

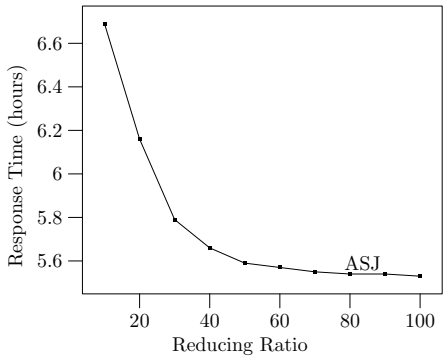

Figure 4: The influence of reducing ratio to the performance of ASJ

4.3 Performance Related to the Cardinality of JRI

In the third experiment we used 30G dataset and set $c(R) = c(S) = 10$. The experimental result was shown in figure 5. It is obvious that the cardinality of JRI had much influence to the response time. The larger the JRI cardinality, the larger the RC is. And there is less chance to put RC in disks, we have to multi-pass scan A_R when merging with A_R. In the experiments, the size of relation S is 20G and the size of JRI can't fit into disk even when $\|JRI\|$ equals $0.1\|S\|$, so A_R needs to be multi-pass scanned. But we can see that the performance of ASJ is still better than that of CTT-GH even we multi-pass scan A_R. We can see from the figure that the execution time of ASJ is only half of that of CTT-GH even when $\|JRI\|$ equals $\|S\|$.

4.4 Comparison of ASJ with three disk join algorithms

We implemented three disk join algorithms, namely nested-loop join, hash based join and sort merge join, and we compared these three join algorithms with ASJ to sufficiently demonstrate the effectiveness of managing massive data using tertiary storage. In the three disk algorithms, we set disk cache large enough and we set disk cache is set to 1GB for ASJ. We varied the dataset size from 3GB to 30GB (R size is varied from 1GB to 10GB), we set $c(R) = c(S) = 10$ and $\|JRI\| = 0.1\|S\|$. The experimental result was depicted in figure 6. In all the four join algorithms, the performance of nested loop join is worst, the performance of hash based join and sort merge join is similar and hash based join is slightly better than sort merge join. When dataset size exceeds 18GB ($|R| > 6GB$), ASJ outperforms the other three join algorithms. In order to show the reason, we compared the relation scan times of all the algorithms and drew the results in figure 7. It is obvious that the relation scan times of nested loop join are far more than that of the other three algorithms. Hash based join and sort merge join both requires scanning relations for many times although their relation scan times are similar. Relation scan times of ASJ are the least. ASJ can greatly reduce the relation scan times through join attribute separation, and hence reduce the I/O cost. This experiment also indicates that in join operation of massive data, it is more important to choose the right algorithm than to choose the storage devices. Tape libraries have larger storage capacity and lower storage price. It is an effective way to solve the problem of massive data management by choosing tertiary storage with the help of right and efficient algorithms.

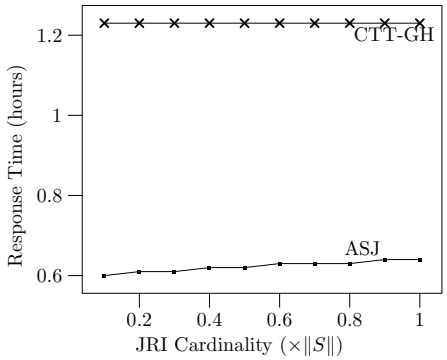

Figure 5: The influence of cadinality of JRI to the performance of ASJ

4.5 Performance Related to Tape Transfer Rate

The data transfer rates of various tape libraries produced by different vendors differ much. In all the experiments above we use devices with very low data transfer rate, which make the experimental results be propitious to disk join algorithms. In the simulation experiment below, we test the influence of date transfer rate to the

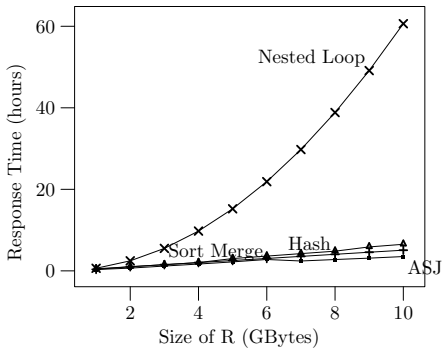

Figure 6: Performance comparison of ASJ and three disk join algorithms

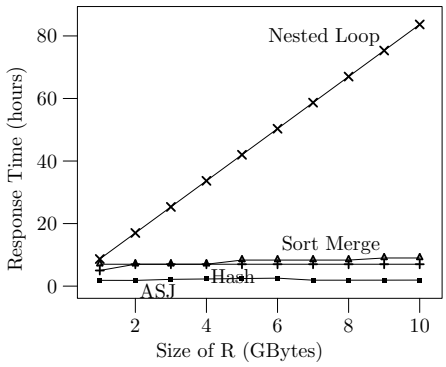

Figure 7: The comparison between dataset size and dataset scan times

performance of ASJ. The dataset size is 30GB ($|R| = 10GB$), $c(R) = c(S) = 10$, $\|JRI\| = 0.1\|S\|$. We use disk to simulate tape library and we change simulated tape data transfer rate by inserting redundant data. The experimental results are depicted in figure 8. We can see from the figure that the data transfer rate of tape drive have much influence to the performance of ASJ. When tape data transfer rate is one tenth of that of disk, the performance of ASJ is worse than that of disk algorithms. As we increase the tape transfer rate to three tenth of that of disk, the performance of ASJ exceeds that of disk algorithms. The data transfer rate of current high end tape library is about 70MB/s while the disk data transfer rate of personal computer is about 10MB/s. The performance of ASJ would be much better than that of disk algorithms if ASJ is running on high end tertiary storage, which further demonstrates the effectiveness and feasibility of using tertiary storage to manage massive data.

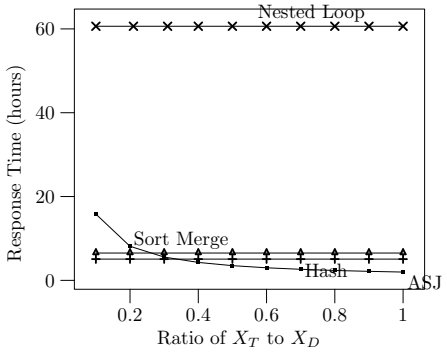

Figure 8: The influence of tape data transfer rate to the performance of ASJ

5 Conclusions

The question of whether tertiary storage can be used to online manage massive data is still no answer. In this paper, we take join operation, which is the most common and time consuming database operation, for example to demonstrate the feasibility of online management of tertiary resident data. Join is the base operator for data analysis and mining. We believe the study of join operation can sufficiently shows if online management of tertiary resident data could be fulfilled. The experimental results show that in analysis and mining of massive data, it is more important to choose the right operation algorithm than to choose the storage devices. Tertiary storage can be used to manage massive data online as effective as disks, which provide a new way to

solve the key problem of online querying and accommodating massive data.

References

[1] Frew J, Dozier J. Data Management for Earth System Science. ACM SIGMOD Record, 26(1), p. 27-31, 1997.

[2] Stonebraker M, Frew J, Gardels K, Meredith J. The Sequoia 2000 Storage Benchmark. Proccedings of ACM SIGMOD, p. 2-11, 1993.

[3] Gibson A. Redundant Disk Arrays: Reliable,Paraleel Secondary Storage. Ph.d thesis, U.C. Berkeley, 1991.

[4] Drapeau, Katz H. Striped Tape Arrays. Proceedings of 12th IEEE Mass Storage systems Symposium, p.257-265, 1993.

[5] Drapeau L, Katz R. Striping in Large Tape Libraries. Proceedings of Supercomputing, pp 378-387, 1993.

[6] Golubchik L, Muntz R. Analysis of striping techniques in robotic storage libraries. Proceedings of the 14th IEEE Symposium on Mass Storage Systems, p.225-38, 1995.

[7] Chen L, Drach R, Keating M, Louise S, Rotem D, Shoshani A. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. Information Systems, v.20, p.155-83, 1995.

[8] Sarawagi S. Stonebraker M: Efficient Organization of Large Multidimensional Arrays. Proceedings of International Conference on Data Engineering, pp 328-336, 1994.

[9] Christodoulakis S, Triantafillou P, Zioga F. Principles of optimally placing data in tertiary storage libraries. Proceedings of International Conference on Very Large Data Bases, p. 236-245, 1997.

[10] Myllymaki J, Livny M. Disk-Tape Joins: Synchronizing Disk and Tape Access. Proceedings of SIGMETRICS, p.279-290, 1995.

[11] Myllymaki J, Livny M. Relational Joins for Data on Tertiary Storage. Proceedings of International Conference on Data Engineering, p.159-168, 1997.

[12] Yu J, DeWitt D. Query Pre-Execution and Batching in Paradise: A Two-Pronged Approach to the Efficient Processing of Queries on Tape-Resident Raster Images. SSDBM, p.64-78, 1997.

[13] Sarawagi S, Stonebraker M. Reordering Query Execution in Tertiary Memory Databases. Proceedings of 22 VLDB, p.156-167, 1996.

[14] Sarawagi S. Execution Reordering for Tertiary Memory Access. Data Engineering Bulletin, v.20, p.46-54, 1997.