Disk-Tape Joins: Synchronizing Disk and Tape Access

Jussi Myllymaki
Miron Livny

Technical Report #1270

March 1995

Disk–Tape Joins: Synchronizing Disk and Tape Access *

Jussi Myllymaki Miron Livny

Computer Sciences Department
University of Wisconsin–Madison
{jussi,miron}@cs.wisc.edu

Abstract

Today large amounts of data are stored on tertiary storage media such as magnetic tapes and optical disks. DBMSs typically operate only on magnetic disks since they know how to maneuver disks and how to optimize accesses on them. Tertiary devices present a problem for DBMSs since these devices have dismountable media and have very different operational characteristics compared to magnetic disks. For instance, most tape drives offer very high capacity at low cost but are accessed sequentially, involve lengthy latencies, and deliver lower bandwidth. Typically, the scope of a DBMS's query optimizer does not include tertiary devices, and the DBMS might not even know how to control and operate upon tertiary-resident data. In a three-level hierarchy of storage devices (main memory, disk, tape), the typical solution is to elevate tape-resident data to disk devices, thus bringing such data into the DBMS' control, and then to perform the required operations on disk. This requires additional space on disk and may not give the lowest response time possible. With this challenge in mind, we studied the trade-offs between memory and disk requirements and the execution time of a join with the help of two well-known join methods. The conventional, disk-based Nested Block Join and Hybrid Hash Join were modified to operate directly on tapes. An experimental implementation of the modified algorithms gave us more insight into how the algorithms perform in practice. Our performance analysis shows that a DBMS desiring to operate on tertiary storage will benefit from special algorithms that operate directly on tape-resident data and take into account and exploit the mismatch in disk and tape characteristics.

Keywords: tertiary storage, join methods, concurrent I/O

1 Introduction

Over the last few years, a need to record and process vast quantities of data has emerged in business environments as well as in scientific settings. Since tertiary devices offer lower cost per megabyte and smaller footprint than magnetic disks, large volumes of data are routinely stored on optical disks or magnetic tapes rather than on secondary storage devices. At processing time, however, this practice may lead to a situation where different parts of the input data reside on storage devices with very different characteristics. In a database environment, a query may be asked to operate

*An abridged version of this paper appears in the proceedings of the ACM SIGMETRICS Conference, May 1995

on relations that are stored on devices with different loading times, access times, and/or transfer rates. A typical query optimizer of a database management system (DBMS) knows how to deal with only two types of storage devices—primary (i.e. main memory) and secondary (i.e. magnetic disks). Tertiary-resident data is commonly perceived as archived data [6] which means that the DBMS is not capable of maneuvering tertiary devices directly. Using services of the operating system, the DBMS first moves all tertiary-resident data to disk, and then optimizes and processes the query. The underlying assumption of these systems is that there is always enough free disk space to store the data and that the cost of the extra copying steps cannot be avoided.

The trade-offs between main memory space and processing time for relational operators that operate on disk-resident data have been extensively studied by the database community. However, when such an operator has to deal with data that resides on two different device types, it faces a new situation with different space-time trade-offs. To illustrate such a situation consider the following scenario: A company records every customer transaction in a tape-resident relation S. In a disk-resident relation R, it stores the names of a subset of its customer population. In order to analyze the activities of these selected customers, the DBMS of the company has to perform a join of the two relations. Since the DBMS cannot operate on tape-resident data directly, before the join can take place, space on disk has to be made available for staging the customer relation S and it has to be transferred from tape to disk. The disk requirement of such a join may be as much as twice the size of S[1] while the I/O cost of the join will be the sum of the cost of reading S from tape and the cost of joining S with R from disk. This scenario brings up natural questions: If the DBMS were capable of operating directly on tertiary data, could the disk space requirement be reduced? If these requirements are partially or completely eliminated, how does it affect processing time? How does the amount of main memory space allocated to the join operator affect the interplay between disk space and processing time?

In this paper we examine these questions with the help of two well-known *ad hoc* join methods: Nested Block Join [8] and Hybrid Hash Join [4,11]. Two join algorithms for tape-resident data that are based on these methods are presented and used to profile the interplay between the amount of disk and memory space allocated to a join operator and how long it takes to process the join. By means of a simple cost model, we show how a DBMS can exploit the special characteristics of a memory-disk-tape hierarchy when joining a tape-resident relation with a relation stored on disk. When compared to the two-stage approach employed by existing DBMSs, both algorithms deliver the same or even better processing time with a much smaller disk space requirement. The Hash Join based algorithm reduces the disk space requirement by hashing the tape-resident relation directly from tape to disk. The tape version of Nested Block Join requires very little disk space or none at all and reduces processing time by overlapping tape and disk transfers. If all disk I/Os

[1] In a Hash Join, disk space is required for the original copy of relation S plus a hashed version.

required to perform the join overlap with tape I/Os, the overall execution-time of the join is the time it takes to read the relation from tape.

For a DBMS to efficiently join a tape-resident relation with a disk-resident relation, all its components, most notably the query optimizer, the operators, and the job scheduler, must be aware of the characteristics of the different I/O devices. The task of a well-informed DBMS would be to accommodate and exploit these differences. For instance, some tape drives don't have enough buffer memory to mask tape stops and restarts[2]. Each stop/start can mean a one-second delay on a typical low-end drive. The DBMS should therefore try to keep the tape drive *streaming* in order to avoid these delays.

The work reported here is part of an ongoing effort to address the challenge of expanding the scope of DBMSs to include tertiary storage devices. Therefore, the main objective of our cost analysis is to identify the major factors that affect the performance of joins of tertiary-resident data. In particular, we are interested in exploring how to minimize the disk space requirements of such joins and to profile the interplay between the space requirements of the join and its execution time. Our purpose is to develop an understanding of the issues involved rather than to identify the fastest possible way of performing a join of tertiary-resident data under a given set of conditions. We hope that the results of this paper demonstrate the potential of making a more informed choice of the method and resource allocation for this type of joins.

This paper is organized as follows. Related work is discussed in Section 2. In Section 3 we discuss the role of magnetic tape as a tertiary storage medium and compare it with other storage technologies. The system model used in this paper is described in Section 4. Section 5 describes the modified algorithms in detail and an analysis of the algorithms is presented in Section 6. Further discussion appears in Sections 7 and 8. An experimental implementation is described in Section 9. Section 10 describes future work and Section 11 concludes this paper.

2 Related Work

Prompted by a SIGMOD Database Challenges paper which highlighted the importance of DBMS access to and control of tertiary storage [6], we set out to study the factors determining the performance of tertiary join operations. In particular, we look at magnetic tape devices which seem particularly poorly suited for joins, and are indeed commonly viewed as "second class citizens" in DBMSs, i.e. suitable for offline or nearline processing only.

Joining two relations is one of the most common operations in a relational DBMS and one of the most costly if done naively. Since the seminal paper on computing joins of relations by Blasgen and Eswaran [1], the database research community has shown great interest in optimizing joins.

[2]On the other hand, more expensive tape drives with very short stop/start latencies, such as an IBM 3490, can make drive motions completely transparent to the application.

An important subclass of joins are *ad hoc* joins which do not rely on the existence of pre-computed access structures such as indices. Joins can be further classified as equi-joins (exact match of join attributes requested) or θ-joins (arbitrary matching criteria). Much effort has been devoted to the fine-tuning of the various suggested *ad hoc* equi-join algorithms.

While studies on disk-based joins are numerous, to our knowledge there are few studies on joins that consider magnetic tape as a storage medium or that develop a system model for analyzing tape joins. Most studies on disk-based joins employ a system model comprising main memory and disks where disks are represented by a transfer-only cost model [1,2,4,8,11]. In such a cost model the number of pages transferred is the cost metric while the latency penalty of small requests (seek and platter rotation delay) is disregarded. In other studies [7] the cost metric has been the number of multi-page I/O requests, regardless of request size. A detailed cost model which combines both cost metrics has been considered in [5].

A tape join study done as part of the Sequioa 2000 project [12] assumes a configuration of two tapes and one tape drive, and hence focuses on media switch delays [10]. The study employs a transfer-only cost model for tape drives, counting the number of chunks, or extents, transferred. CPU and disk I/O costs are ignored in the cost model.

Memory size, relative to the size of relations, is typically a key factor in selecting a join method. For disk-based equi-joins, Hybrid Hash Join is commonly viewed as the method of choice when only a small amount of main memory is available, compared to the smaller relation. In [5] it is suggested that when a moderate fraction of the smaller relation can fit in memory, Nested Block Join with optimal buffer allocation provides better performance. Given that such conclusions can be made for disk joins, in this paper we explore which factors a query optimizer needs to consider when picking a method for disk-tape joins.

3 Magnetic Tape as a Tertiary Storage Medium

The performance and cost effectiveness of magnetic tape storage depend largely on the technology used (Table 1). Low-end tape drives, such as the 8 millimeter and 4 millimeter technologies, typically store several gigabytes of data on a single cassette in uncompressed format but transfer less than 1 MB/s uncompressed. Mid-range tape drives, such as Digital Linear Tape (DLT) drives, offer native capacities of up to 20 gigabytes and transfer rates of up to 1.5 MB/s. In the high-end range, more expensive tape systems such as the IBM 3490E and Metrum drives offer a high data transfer rate (faster than many disk systems) but at a much higher cost. In library configurations the cost of automation (robotic picker mechanism) can add up to a significant portion of the total cost.

For comparison, Table 1 also shows the characteristics of a typical magnetic disk and a rewritable, magneto-optic (MO) optical drive. In terms of their performance, MO optical drives are similar to

Table 1: Comparison of Magnetic Tape Technologies

tape technology	native capacity	transfer rate	drive cost	drive cost per MB/s	media cost
8 mm	5 GB	0.5 MB/s	$2,000	$4,000 / MB/s	$.002 / MB
4 mm (DDS 2)	4 GB	0.4 MB/s	$1,000	$2,500 / MB/s	$.005 / MB
DLT	20 GB	1.5 MB/s	$5,000	$3,333 / MB/s	$.006 / MB
1/2" 3490E	0.8 GB	3 MB/s	$20,000	$6,700 / MB/s	$.03 / MB
1/2" Metrum	14.5 GB	2 MB/s	$40,000	$20,000 / MB/s	$.001 / MB
Ampex DST 600	25 GB	15 MB/s	$150,000	$10,000 / MB/s	
1" Metrum	12.5 GB	22 MB/s	$160,000	$7,273 / MB/s	
magnetic disk	2 GB	4 MB/s	$1,000	$250 / MB/s	$0.50 / MB
MO optical disk	1.3 GB	1.6 MB/s	$2,100	$1,312 / MB/s	$0.10 / MB

magnetic disks and appear to bridge the gap between secondary and tertiary storage. MO optical disks retain data much like magnetic disks (even exceeding the data retention rate of magnetic disks) which is a major advantage over magnetic tapes. Tapes and tape drive heads wear out surprisingly quickly. On the other hand, the cost of tapes is still much lower than the cost of optical disks—so much so that tapes can be frequently replaced with new ones before any data is lost. Therefore, in many archival-type applications that require sequential access to data, magnetic tapes continue to offer the best storage solution due to their very low cost and high capacity per media unit.

The purpose of our study is not to commit to any particular tape or disk technology but to show that as long as there exists a mismatch between the speeds of two device types, a buffering scheme that exploits the speed difference can reduce the overall response time of a join. For ease of presentation, however, we have chosen magnetic tapes to play the role of a 'cheap but slow' device, and magnetic disks for the role of an 'expensive but fast' device. Some tape drives are in fact faster than most disk drives but computing facilities that invest a large sum of money in a fast tape drive, such as a Metrum or an Ampex drive or library, are likely to also invest in high-performance disk arrays, and the cost/performance balance between tapes and disks is maintained.

The performance of a tape device is affected by tape wear, especially on low-end tape drives. In 8mm and 4mm tape drives, the life of a tape is roughly 1,500 passes over the read/write head. The read/write head also needs frequent cleaning, after roughly every 25 hours of activity. If a tape will be read a very large number of times by a DBMS operation, it might make more sense to read the data from a disk even if the tape were faster. The response time of the operation may be sub-optimal, but wear of the tape and the tape drive head is reduced.

Many low, mid, and high-end tape drives offer on-the-fly data compression which increases the capacity as well as the transfer rate of a tape drive. Data compression ratios are variable and

depend on the type of data stored. Data compression products are available for many types of storage, but compression is particularly applicable in tape drives since data is written sequentially and therefore changes in data block sizes do not cause problems for data layout.

4 A Simple System Model

In this section we present the system model that we used to identify and profile the key factors in tape-disk joins. Since our goal is to perform a qualitative analysis and to capture the trade-offs in storage requirement and I/O cost, we employ a simple model of the system components and their interaction.

In our model the computer system consists of main memory, one CPU, one disk device, and one tape device. A fixed amount of main memory is allocated for use in the join. The CPU and the tape device are reserved exclusively for the join. It is assumed that the disk and tape device can be accessed concurrently by the system so that a transfer to/from one device can overlap with a transfer to/from the other.

The disk and tape device are abstractions which need not physically correspond to a single disk drive or a single tape drive, respectively. The disk device may in fact be an array of disks, and the device may or may not be shared with other processes in the system. The transfer rate of the disk device is just the aggregate transfer rate of the disk subsystem seen by the join process. Similarly, the tape device may be an array of tape drives, and the transfer rate is the aggregate transfer rate of the array.

We assume that all disk accesses are multi-page I/O requests. The cost of a disk access is therefore derived by counting the number of blocks transferred. The cost of seek and latency delays is ignored. As shown in [5], disk seeks and latency play a relatively minor role compared to transfer cost when disk requests are at least moderately large. In our model, the size of all disk requests is assumed to be at least 30 blocks, making seek and latency costs negligible[3]. Such disk accesses incur a fixed per-block transfer cost.

Data is read from and written to tape sequentially. The tape drive is capable of reading and writing data in both directions. It has an internal read-ahead, speed-matching buffer which allows the system to transfer data in requests as small as one block without performance deterioration.

The tape drive tries to keep its read-ahead buffer full at all times by reading as many blocks from tape as possible [3, 9, 13]. When the buffer fills up or a high-water mark is hit, the tape drive slows down and stops (*ramp down*), and the tape is repositioned so that the read/write head is at the first tape block that did not fit into the buffer. A subsequent read request removes blocks from the buffer and releases buffer space. When a low-water mark is hit or the buffer becomes

[3] Disk caching would reduce seek and latency costs even if requests were smaller than 30 blocks.

empty, the tape drive restarts and accelerates to full speed (*ramp up*), and more data is read into the buffer.

The drive is said to be in a *streaming* mode if the read-ahead buffer never fills up and therefore the drive never stops. Otherwise the drive operates in a *stop/start* mode. When the drive is in streaming mode, a fixed per-block cost is assumed for tape transfers. A fixed stop/start delay is added to the cost of a transfer when the drive is not streaming.

Note that if the read-ahead buffer is big enough, it can shield the application from the cost of stop/start delays. In other words, as far as the application is concerned, the drive is always streaming. For simplicity, we assume that the tape drive has enough buffer memory to hide these delays. In Appendix A we present an analysis where smaller buffers are considered and the response time of a tape drive is therefore affected by the stop/start delays.

Tape media switch delays which occur in tape robot systems are not included in this cost model. Current low-end tape technology stores several gigabytes of data on a single tape cassette but transfers data at less than 1 MB/s. Reading such a tape end-to-end takes several hours while rewinding one cassette and switching to another typically takes less than a minute—only a very small fraction of the transfer time. Switch delays are an important factor when the number of tape relations accessed is greater than the number of tape drives. In our assumed scenario, however, only one relation is stored on tape while the other is on disk. This means that media switches would occur very infrequently—once for each time a full tape has been read. In our model the switch times are dwarfed by the data transfer time, and we therefore simplify our model by ignoring switch delays and assume that a relation fits on a single tape.

5 Description of the Algorithms

In this section we describe algorithms for joining a tape-resident and a disk-resident relation. In all cases we assume that the larger of the two relations, S, does not fit on disk and is therefore stored on tape. The smaller relation, R, is on disk. We consider only the case where R does not fit in main memory. If it does fit, the simplest approach is to read and hash R into memory, and then scan S. We refer to this case as Simple Hash Join [4] from tape.

In the disk-tape version of Nested Block Join, the tape-resident relation (which is also the larger one) is selected as the outer relation. In the conventional, disk-based Nested Block Join the outer relation is always the smaller one. A simple analysis presented in Appendix B shows that when the larger relation is stored on a slower device than the smaller one, a Nested Block Join with reversed roles of the relations will have a lower cost. In the description of the algorithms, it is assumed that the tape device is indeed slower than the disk device. In Section 8, we consider the case where the tape device is faster than the disk.

In this paper we use the following notation. The size of R is denoted by $|R|$ and the size of S by

Table 2: Summary of Notation

M	size of memory in blocks		D	disk space in blocks
R	smaller relation, on disk		S	larger relation, on tape
$\|R\|$	size of R in blocks		$\|S\|$	size of S in blocks
M_R	size of R input buffer		M_S	size of S input buffer
F	fudge factor			

$|S|$. The join is allocated M blocks of main memory and D blocks of free disk space. M_R blocks of main memory are assigned for R tuples, while M_S blocks are assigned to store tuples from S. The space overhead incurred when building a hash table for a set of data is denoted by factor F [4]. A summary of the notation appears in Table 2.

5.1 Nested Block Join

The conventional Nested Block Join (NB) [8] can be used for joining tape-resident S with disk-resident R as follows. An M_S-size chunk of S is read from tape to memory, hashed, and joined with the entire R. This process is then iterated until S has been exhausted.

The number of iterations required by this algorithm is $NC = \left\lceil \frac{|S|}{M_S} \right\rceil$. Define S_i as the chunk of S read in at the ith cycle where $i = 0 \ldots NC - 1$. R is read from disk in M_R-size chunks. R_j is the jth chunk of R where $j = 0 \ldots \left\lceil \frac{|R|}{M_R} \right\rceil - 1$. Once S_i is in memory, it is hashed and joined with R. Define $cycle_i$ to consist of reading S_i, hashing it, and then computing $S_i \bowtie R$, for $i = 0 \ldots NC - 1$.

5.2 Nested Block Tape Join

The cyclic nature of NB and the presence of two different device types allows for effective I/O concurrency which can reduce the overall response time. Nested Block Tape Join (NBT) takes advantage of this opportunity by continuing to read S from the tape while R is joined with a previously read chunk of S. Chunks of S can be buffered either in main memory or on disk. In the following sections, we describe both variations of this algorithm.

As in NB, S_i is the chunk read from tape in the ith iteration and R_j is the jth chunk read from disk. We define $cycle_i$ to consist of reading S_{i+1} while simultaneously hashing S_i and computing $S_i \bowtie R$, for $i = 0 \ldots NC - 2$. $cycle_0$ is preceded by $cycle_{-1}$ which reads S_0 from tape. $cycle_{NC-1}$ does not read data from tape but consists merely of hashing S_{NC-1} and computing $S_{NC-1} \bowtie R$. Except for $cycle_{-1}$, each cycle starts when all activities (tape and disk reads and hashing) of the previous cycle have barrier synchronized.

Each cycle has an associated time constraint: *Reading R from disk should be faster than reading S_i from tape.* If the constraint holds, the tape drive is streaming and the join is bound by the speed

of the tape drive.

NBT uses relation rocking [8, 9] to save on disk transfers. Relation R is read in alternating directions, and when changing from one direction to the other, the last M_R blocks of R are already in memory.

5.2.1 NBT with Memory Buffering

In NBT with memory buffering (NBT/MB), two memory buffers I_0 and I_1 are assigned to hold the tuples of S coming from tape. Both buffers are M_S blocks in size and are used in alternating order.

The algorithm for NBT/MB is shown as Algorithm 1 and the flow of data and allocation of memory are illustrated in Figure 1. The size of the buffers is $|I_0| = |I_1| = M_S = \frac{M-M_R}{1+F}$. The schedule in Figure 2 shows how NBT/MB works when the tape is streaming. Hashing S_i, reading R, and computing $S_i \bowtie R$ are finished exactly when reading S_{i+1} is complete. In other words, the time constraint holds and the tape drive is streaming.

Algorithm 1 *NBT with Memory Buffering*

```
k ← 0
i ← 0
read S_i from tape into I_k
while not (I_k is empty) do
    start reading S_{i+1} from tape into I_{1-k}
    hash S_i
    j ← 0
    while not (end of relation R) do
        read R_j from disk into memory
        compute S_i ⋈ R_j
        j ← j + 1
    end
    change direction of R scan
    wait until read of S_{i+1} is complete
    i ← i + 1
    k ← 1 − k
end
```

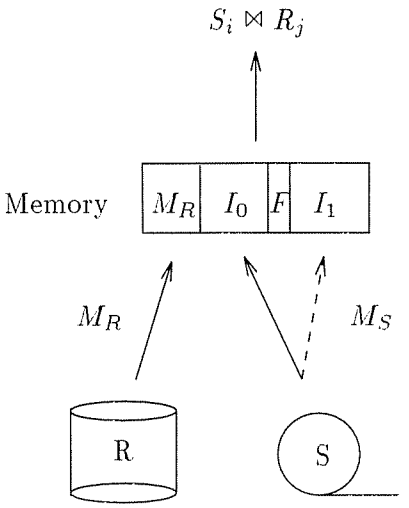

Figure 1: Data Flow in NBT with Memory Buffering

5.2.2 NBT with Disk Buffering

NBT can keep the tape drive streaming only if the time constraint is met. The amount of memory required to achieve that is proportional to $|R|$. The larger R is, the more time is consumed reading it from disk and computing $S_i \bowtie R$. Hence, more buffer space for S input is needed to "buy" additional tape time in each cycle.

In NBT with disk buffering (NBT/DB), two disk buffers I_0 and I_1 are used for storing tuples

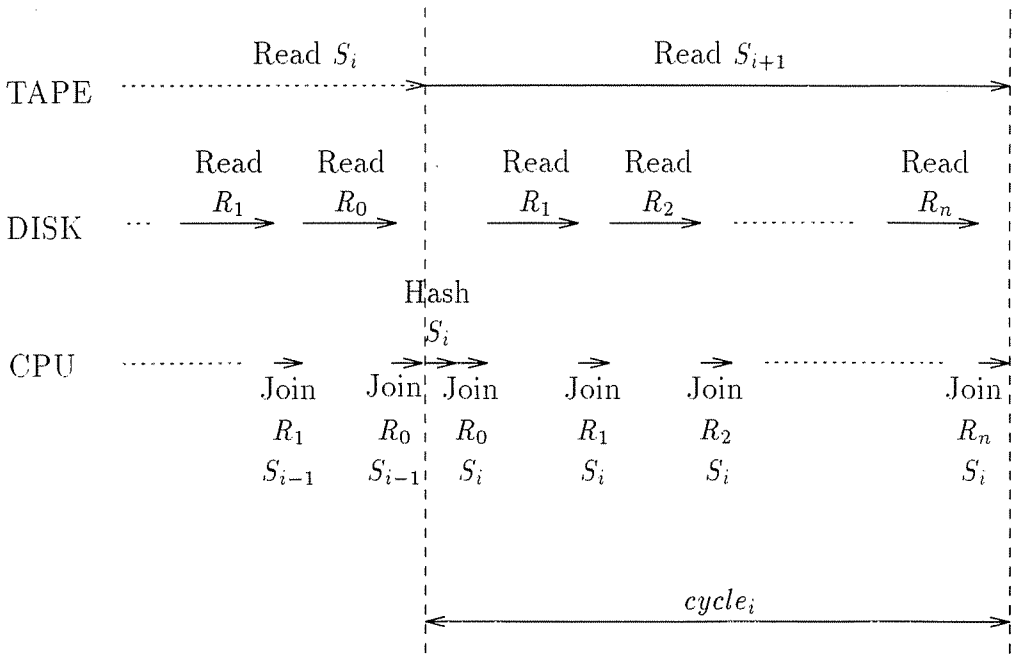

Figure 2: Optimal Schedule of Events in NBT with Memory Buffering

of S. A small memory buffer manages the data transfer to the disk buffers while a larger buffer in memory is used for reading data from disk. Define M_O as the amount of memory assigned to the transfer of data from tape to a disk buffer. It follows that $M_S = \frac{M-M_R-M_O}{F}$ blocks are left for the larger memory buffer. The size of the disk buffers I_0 and I_1 is also M_S.

When a chunk of S is retrieved from tape in the background, it is written to a disk buffer (I_0 or I_1), and later retrieved from disk in the next cycle. NBT/DB is shown as Algorithm 2 and the flow of data appears in Figure 3. The schedule of events in Figure 4 illustrates how NBT/DB works when the tape is streaming.

5.3 Hybrid Hash Join

Hybrid Hash Join (HH) on tapes operates exactly as the classic Hybrid Hash Join [4,11] except that Phase I where both relations are hash partitioned on disk is modified to read relation S from tape. The number of hash partitions stored on disk is $B = max(0, \frac{|R|F-M}{M-1})$. The hash partitions are then denoted by R_i and S_i where $i = 0 \ldots B$. First, hash partitions for relation R are created, retaining partition R_0 in memory[4] and writing other tuples to hash partitions on disk. Then relation S is hashed from tape to disk, joining tuples in S_0 with R_0 in memory and writing other tuples to partitions on disk. In Phase II each R_i is read into memory and joined with the corresponding S_i by scanning it. The difference $|R| - |R_0|$ represents the volume of hashed R data that is written to disk. The amount of disk space required for storing hashed S data is $|S| - |S_0|$.

[4]We assume R_0 is small enough to fit in memory and a 1-block output buffer is allocated for each other partition.

Algorithm 2 *NBT with Disk Buffering*

```
k — 0
i — 0
copy S_i from tape to I_k
while not (I_k is empty) do
    start copying S_{i+1} from tape to I_{1-k}
    read S_i from I_k into memory
    hash S_i
    j — 0
    while not (end of relation R) do
        read R_j from disk into memory
        compute S_i ⋈ R_j
        j — j + 1
    end
    change direction of R scan
    wait until S_{i+1} copied to I_{1-k}
    i — i + 1
    k — 1 − k
end
```

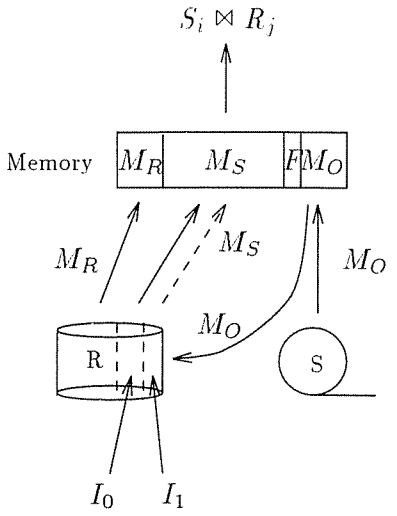

Figure 3: Data Flow in NBT with Disk Buffering

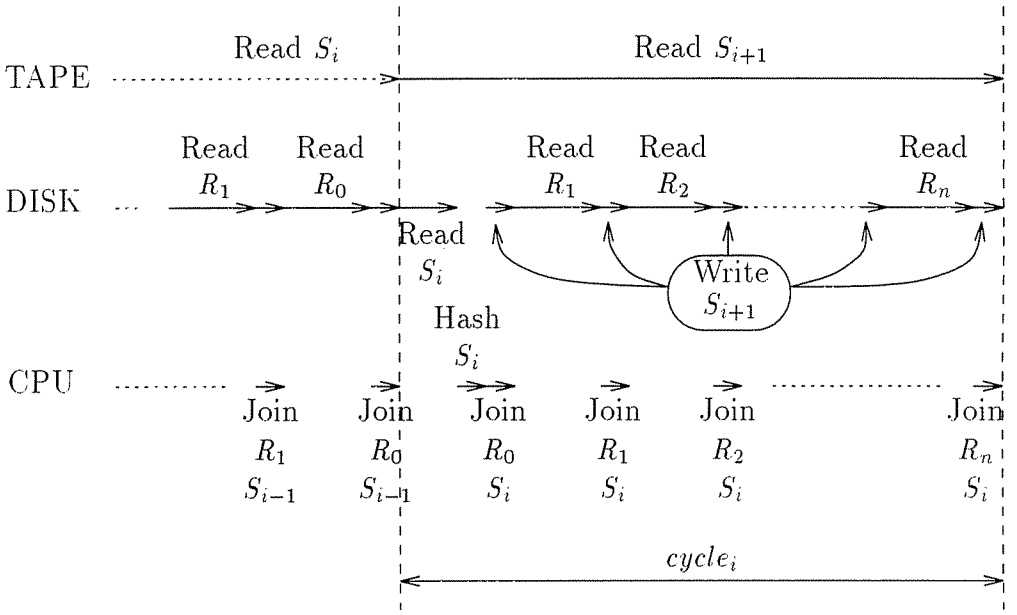

Figure 4: Optimal Schedule of Events in NBT with Disk Buffering

6 Analysis of the Algorithms

In this section we present an analysis of the resource requirements and the cost of the join methods. In the analysis we use four parameters that can succinctly describe the cost of each join method analyzed (Table 3). We denote the ratio of disk transfer rate to tape transfer rate by r. For example, $r = 4$ means that reading a block from disk is four times as fast as reading it from tape.

Table 3: Key Parameters Used in the Analysis

r	ratio of disk and tape transfer rates
α	ratio of M_S to $\|R\|$
β	ratio of $\|S\|$ to $\|R\|$
T_R	transfer time of R from disk

The ratio of tape transfer size to disk transfer size in each cycle of NB and NBT is defined as $\alpha = \frac{M_S}{|R|}$. The value of α depends on the total memory available to the join and on the join method used. As will be shown later, the value of $\alpha \cdot r$ is critical to the viability of NBT since it determines whether streaming tape access is possible or not.

Parameter β denotes the ratio of $|S|$ to $|R|$. The number of iterations required in NB and NBT can be expressed as $\left\lceil \frac{\beta}{\alpha} \right\rceil = \left\lceil \frac{|S|}{M_S} \right\rceil$. With T_R we denote the transfer time of R from disk. Note that T_R is the only parameter with an absolute value whereas the other parameters are ratios and have relative values. The transfer time of M_S blocks of S from tape is given by $T_R \cdot \alpha \cdot r$.

6.1 Assumptions

We make a number of simplifying assumptions in order to derive concise cost formulas and to present their derivation more clearly.

We assume that I/O times are the dominant factor in the join and the CPU cost can be ignored. Therefore, the total cost of the join (response time) is roughly the same as the I/O cost, like in most other cost models for joins. Our analysis does not include the I/O cost and memory requirement of the final output stream from the join. Output costs are typically not considered in join method analyses since they are the same for all methods [1,4,5,7,11].

Before the join operation begins, the tape which holds S has already been inserted and loaded into the tape drive. Since S fits on a single tape, no media changes are required.

In Nested Block Tape Join, M_R is allocated 10% of M, leaving 90% to buffer S[5]. The effect of relation rocking is assumed to be minimal[6] and is ignored in the analysis.

In Nested Block Tape Join with disk buffering, the memory space allocated for tape to disk buffer transfer, M_O, is assumed very small compared to M and is ignored in the analysis. The disk buffers I_0 and I_1 are assumed to be on one disk device, but in practice one would use two disks because the buffer I/O can be effectively parallelized. In that case, the I/O cost of this method

[5] Our analysis of a more detailed cost function (which takes disk seeks into account) suggests that the optimal memory allocation is $M_R = (\sqrt{S_{DISK}[S_{DISK} + M(X_{DISK} + probe)]} - S_{DISK})/(X_{DISK} + probe)$ where S_{DISK}, X_{DISK}, and $probe$ are the seek, transfer, and probe cost per block, respectively. It is very similar to the equation derived for the conventional, disk-based Nested Block Join in [5] although the scenarios are different.

[6] For example, if $|R| = 10 \cdot M$ and $M_R = 0.1 \cdot M$, relation rocking would save $\frac{M_R}{|R|} = 1\%$ in disk I/O.

Table 4: Resource Requirements and I/O Cost

algorithm	required disk space	required memory	I/O cost	bounding device
Nested Block Join	none	$M_R + M_S F$ $M_R \geq 30, M_S \geq 1$	$\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R(\alpha \cdot r + 1)$	T, D
NBT Memory Buffering	none	$\lvert R \rvert \cdot \frac{c}{r}$ if $r > c/F$ $M_R + M_S(1+F)$	$\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R \cdot \alpha \cdot r$ $\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R$	T D
NBT Disk Buffering	$M \cdot 1.8/F$	$\lvert R \rvert \cdot d$ if $r > 2(1+F)$ $M_R + M_S F$	$\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R \cdot \alpha \cdot r$ $\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R(1 + 2\alpha)$	T D
Hybrid Hash Join	$(\lvert R \rvert + \lvert S \rvert)(1-q)$ $q = \lvert R_0 \rvert / \lvert R \rvert$	$\sqrt{\lvert R \rvert F}$	$\frac{\beta}{\alpha} \cdot T_R \cdot \alpha \cdot r$ $+T_R \cdot [1 + (2+\beta)(1-q)]$	T, D

Coefficient $c = \frac{1+F}{0.9}$. Coefficient $d = \frac{F}{0.9(r-2)}$. Bounding device: tape (T) or disk (D) read.

would be reduced and the join would require less main memory to achieve tape-boundness.

With the notation and assumptions described, we can now move to the analysis of the algorithms. In Table 4 we present a summary of the resource requirements and I/O cost functions.

6.2 Nested Block Join

In the NB algorithm all I/O operations are sequential. For each chunk of S, R is scanned through and the join is computed. The I/O cost of each iteration is $T_R \cdot \alpha \cdot r + T_R$ where the first term is the transfer time of a chunk of S from tape and the second term is the cost of reading R from disk. Multiplying by the number of iterations $\left\lceil \frac{\beta}{\alpha} \right\rceil$, the total I/O cost is $\left\lceil \frac{\beta}{\alpha} \right\rceil \cdot T_R(\alpha \cdot r + 1)$. The memory requirement is $30 + 1 \cdot F$ since $M_R \geq 30$ and $M_S \geq 1$. No disk space is required by NB.

6.3 NBT with Memory Buffering

The speed or *join rate* of HH and NB is determined by the absolute transfer rates of disk and tape but does not depend on their relative values. Each step of HH and NB requires either disk I/Os or tape I/Os. Correspondingly, each step is either disk-bound or tape-bound and its cost is dictated by the disk or tape transfer rate individually.

In NBT the relative values of the transfer rates are important. NBT has two distinct rates at which it can join relations, corresponding to whether the time constraint is met or not (cf. Section 5.2). In a *tape-bound* mode of NBT the time constraint holds and the join rate is determined by the tape transfer rate alone. The cost of a tape-bound cycle is $T_R \cdot \alpha \cdot r$. Reading M_S blocks from

tape is slower than reading $|R|$ blocks from disk, that is, $\alpha \cdot r > 1$. The join is *disk-bound* when the time constraint is not met, and then the join rate depends on disk transfer rate only. The cost of a disk-bound cycle is simply T_R. The total cost of the join is derived by multiplying the cycle cost by the number of iterations $\left\lceil \frac{\beta}{\alpha} \right\rceil$.

The requirement $\alpha \cdot r > 1$ can be expanded by using the definition $\alpha = \frac{M_S}{|R|}$. Recall that 10% of M is allocated to M_R and the remaining space is split evenly between the two memory buffers I_0 and I_1. Taking into account the space overhead of building a hash table, we get $M_S = \frac{0.9M}{1+F}$. Combining $\alpha \cdot r > 1$ with the value of M_S yields the memory requirement $M > |R|\frac{1+F}{0.9 \cdot r}$. This equation shows that the amount of memory required to guarantee streaming tape access is proportional to the size of R and depends on the ratio of disk and tape transfer rates.

Since we consider only the case where $M < |R|F$, coefficient $\frac{1+F}{0.9 \cdot r}$ (c/r in Table 4) must be less than F. The condition for the ratio of transfer rates is thus $r > \frac{1+F}{0.9 \cdot F}$.

The memory requirement for a disk-bound join is $M_R + M_S(1+F)$ but, like in NB, there is no minimal value for r. NBT/MB does not require disk space.

6.4 NBT with Disk Buffering

For tape-bound joins, the I/O cost of NBT/DB is the same as that of NBT/MB but the memory size and device speed constraints are different. The time constraint for streaming tape access can be expressed as $T_R(1+2\alpha) \leq T_R \cdot \alpha \cdot r$. This condition states that reading R from disk plus reading the *current* M_S-size chunk of S (S_i) from a disk buffer plus writing the *next* chunk of S (S_{i+1}) to disk must take less time than reading S_{i+1} from tape. M_S is substituted with $\frac{0.9M}{F}$ which means that after 10% of M has been allocated to R input, a hash table for a chunk of S can be built in the remaining memory space. Combining the time constraint with the value of M_S yields the memory requirement $M > |R|\frac{F}{0.9 \cdot (r-2)}$. As with memory buffering, it is linear in $|R|$ and depends on the ratio of disk and tape transfer rates.

If the time constraint fails, the join is disk-bound and the memory requirement is $M_R + M_S F$.

Disk buffering should only be used if there is not enough memory to perform a tape-bound join using memory buffering. It is therefore required that coefficient d be less than $c/r = \frac{1+F}{0.9r}$ (Table 4). This results in condition $r > 2(1+F)$ which means that the disk transfer rate must be at least 4.4 times the rate of tape transfer if hashing incurs a 20% space overhead ($F = 1.2$).

When disk buffering is used, the disk space requirement is $2M_S$ because the two disk buffers I_0 and I_1 are M_S blocks each. Taking into account $M_S = 0.9M$ and the fudge factor F, the disk space requirement is $M \cdot 1.8/F$.

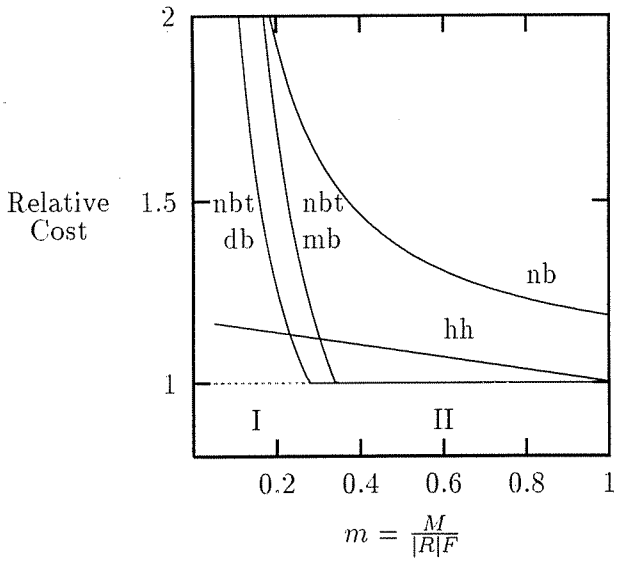

Figure 5: Total I/O cost ($r = 6$)

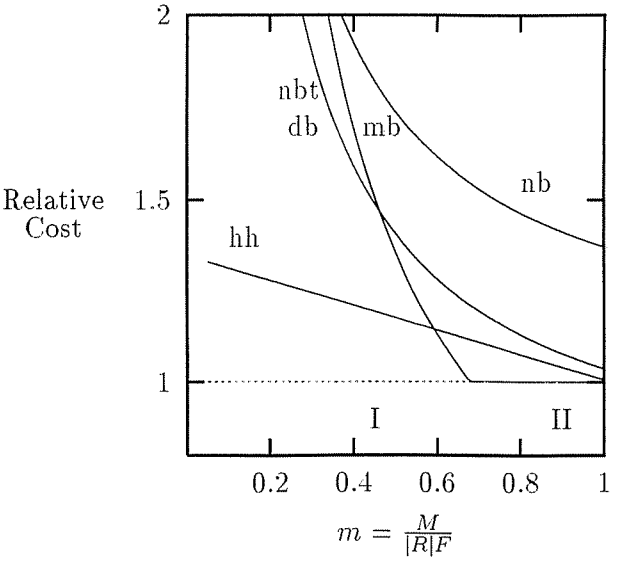

Figure 6: Total I/O cost ($r = 3$)

6.5 Hybrid Hash Join

In [11], it is shown that HH requires at least $\sqrt{|R|F}$ blocks of memory if recursive partitioning is to be avoided. The disk space requirement of HH is $(|R| + |S|)(1 - q)$ where $q = \frac{|R_0|}{|R|}$ represents the I/O savings because R_0 and S_0 are not written to disk. We assume a uniform distribution of join attribute values so the size of S_0 is estimated to be a fraction q of $|S|$.

Using the parameters in Table 3, the disk transfer cost of R is T_R and the tape transfer cost of S can be expressed as $T_R \cdot \beta \cdot r$. Reading S from tape is overlapped with hashing it and writing hash partitions to disk. Since the tape transfer rate is lower than that of disk, only the tape transfer cost (which is larger) is incurred.

The cost of writing R partitions to disk plus reading partitions of both relations requires $T_R[(2+\beta)(1-q)]$. Grouping all common terms together, the total I/O cost of a cycle can be expressed as $T_R[1 + \beta r + (2+\beta)(1-q)]$. In Table 4, the term representing S transfer time has been extracted to allow for easier comparison with other cost functions.

7 Discussion

In this section we compare the cost and resource requirements of the join methods. The parameter values assumed are $\beta = 100$ (that is, S is 100 times as large as R) and $F = 1.2$. In HH, R input, S input, and the output of each hash partition are allocated 1% of M each.

The relative cost of the four join methods is shown in Figures 5 and 6. The minimum cost of a join is the cost of reading S from tape and is depicted with value 1. The figures show the cost of the join methods relative to this minimum when the amount of memory is varied.

In the charts we observe that the cost of HH decreases as memory increases since fewer tuples

need to be written to disk and joined in Phase II of HH. If little memory is available, the overhead of HH compared to the tape transfer cost (the 'join cost' of HH) is roughly 15%.

Corresponding to the disk-bound and tape-bound modes of operation in NBT/MB and NBT/DB, the costs of these methods have two regions. In Region I, the joins are disk-bound and the associated cost decreases as memory increases since more memory means larger tape requests and hence fewer iterations over R. Also, tape requests take a longer time and result in less penalty because of slow disks. With enough memory, the disk cost in each cycle becomes less than the cost of tape access. In Region II, the total cost of the join is then just the transfer cost of S from tape.

We now look at the cost of NB. We observe that the cost decreases as memory increases because the number of iterations over R on disk is reduced. The overall cost is higher than in NBT since NB does not use concurrent I/Os but operates sequentially. It is interesting to note that with little memory, it would actually outperform NBT/MB. The reason is that NBT/MB reads S in chunks half the size that of NB, thus doubling the number of iterations over R. If r is not large enough as to compensate for the added disk I/O, NBT/MB will perform worse than NB!

In Figure 6 the value of r is smaller than in Figure 5, that is, tape speed is higher or disk speed is lower. Reducing r has little impact on HH but shifts the others horizontally. Except for NBT/MB and NBT/DB, the relative position of the methods has not changed much. In Figure 5, NBT/DB dominated NBT/MB across the memory range but in Figure 6 the costs of the two intersect. If the device speed constraint for tape-boundness in NBT/DB fails ($r < 2(1+F)$), NBT can perform better with memory buffering than with disk buffering. With such low values of r, NBT with disk buffering cannot keep the tape drive streaming with any amount of memory (less than $|R|$), but with memory buffering it still can. The range where NBT/MB or NBT/DB are able to keep the tape drive streaming and beat other methods gets smaller as r decreases. With $r = 6$, the range is $m = 0.3$ to 1.0 whereas if $r = 3$, the range shrinks to $m = 0.7$ to 1.0.

Next we will compare the disk I/O consumption of each method as memory size changes. Figures 7 and 8 show the disk time consumed by the methods, relative to the transfer time of R from disk (T_R). Figure 7 displays the case where S is 100 times as large as R ($\beta = 100$). In Figure 8 the size of S is 10 times that of R.

We observe that the disk I/O consumed by HH is inversely proportional to the amount of memory since more memory means that less data need be written to disk. NB follows a curve like a/x since increasing memory means fewer iterations. The curves of NBT/DB and NBT/MB are similar to NB in shape but are higher than NB across the memory range. The reason is that NBT/DB buffers all tape I/O through the disk whereas NBT/MB uses almost twice the number of iterations over R to compute the join.

In NB and NBT/MB, disk I/O consumption is directly proportional to β since β determines the number of times R must be read from disk. NBT/DB incurs additional disk I/O because of its disk buffer traffic. HH requires disk I/O that is roughly proportional to β but $|R|$ has a minor

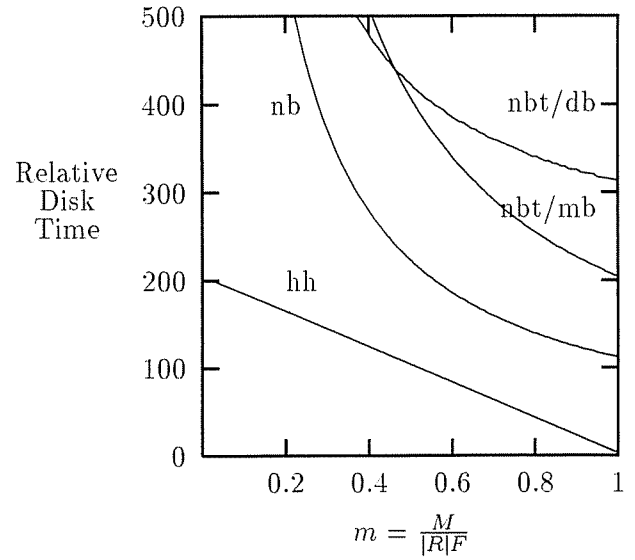

Figure 7: Disk Time ($\beta = 100$)

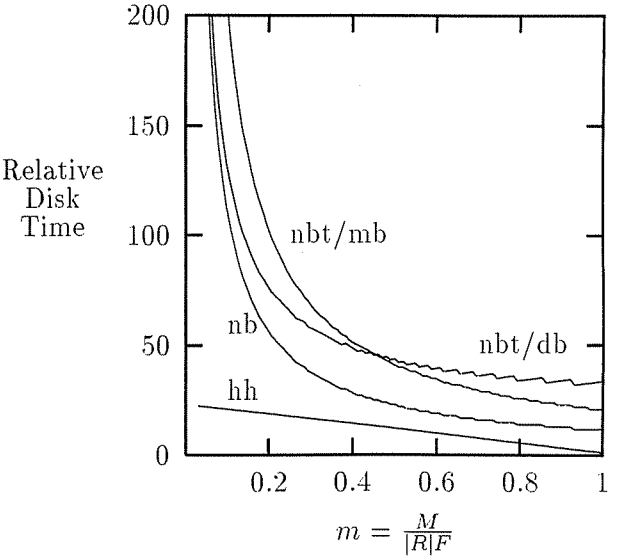

Figure 8: Disk Time ($\beta = 10$)

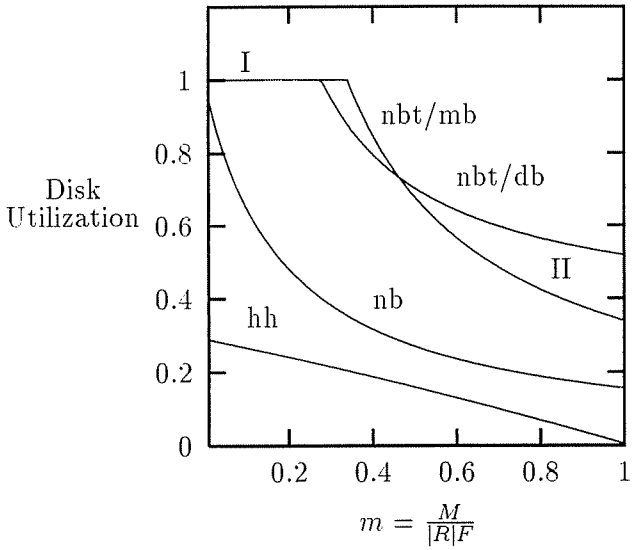

Figure 9: Disk Utilization ($r = 6$)

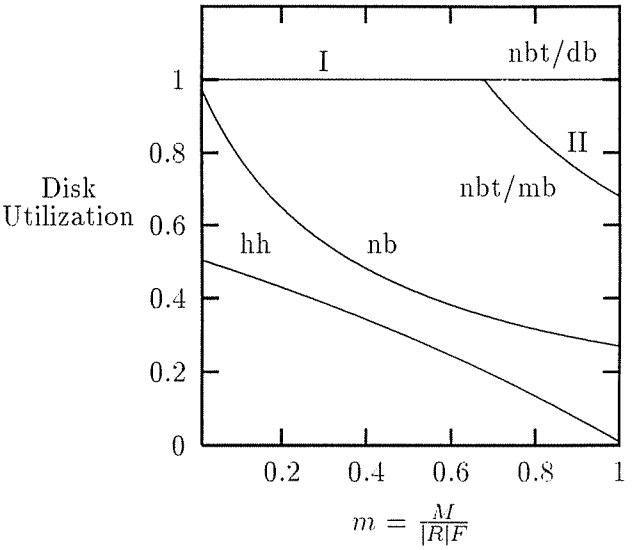

Figure 10: Disk Utilization ($r = 3$)

Table 5: Major Trade-offs in Join Methods

method	weak spot	reason	trade-off
NB	$\|R\| \gg M$	# iterations	trades in disk I/O for reduced memory requirement
HH	$\|S\| \gg D$	disk space	trades in disk space for reduced disk I/O
NBT/MB	$\|R\| \gg M$, small r	buffer space, tape not streaming	trades in memory space and disk bandwidth for concurrent I/O and reduced elapsed time
NBT/DB	$\|R\| \gg M$, small r	buffer space, tape not streaming	trades in disk bandwidth for concurrent I/O and reduced elapsed time

impact as well ($|R| - |R_0|$ blocks are hashed to disk).

We shift our focus now to disk utilization. Disk utilization is computed as the fraction of total join time that the join spends accessing the disk. Figures 9 and 10 show disk utilization as a function of memory size. We notice that in HH, disk utilization is inversely proportional to the amount of memory. With more memory, disk time decreases but tape transfer time does not change. The ratio of disk time to total join time decreases and therefore disk utilization decreases as well. Similarly, disk utilization in NB decreases as memory increases since the number of iterations and the total disk time are reduced but tape time is unchanged.

The regions in NBT's disk utilization correspond to the regions in its total I/O cost (Figures 5 and 6). In Region I, the join is disk-bound and hence disk utilization is 1.0. When memory is increased, the join becomes tape-bound and simultaneously the number of iterations goes down. This results in decreasing disk utilization because tape time remains unchanged.

Disk space requirement is the next item we compare. The amount of disk space is computed relative to $|R|$. Figure 11 shows the disk space requirement of HH and NBT/DB only. NB and NBT/MB do not require disk space and are not shown on the chart.

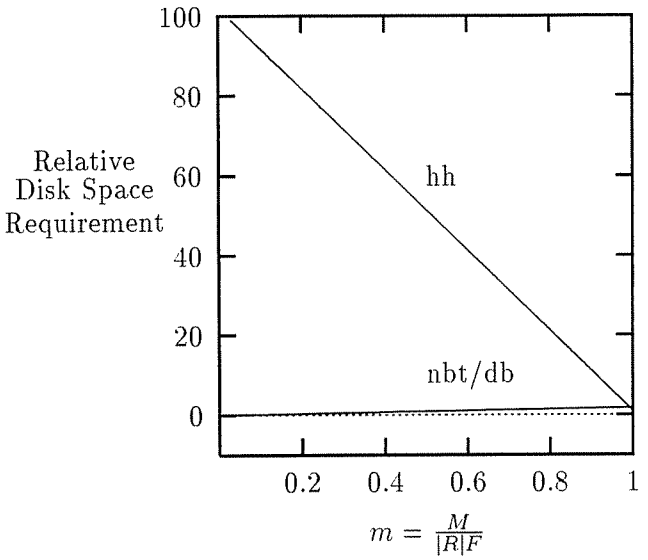

Figure 11: Comparison of Relative Disk Space

When very little memory is available, HH is forced to store most of R and S on disk. Its disk space consumption is at most $\beta + 1$ in which case S and R are written to disk in their entirety. Disk space requirement decreases as memory increases since less data need be written to disk (q increases). In NBT/DB, the disk space requirement is $M \cdot 1.8/F$ which is proportional to the amount of main memory but minimal compared to HH.

Table 5 highlights the major trade-offs in each join method. For NB the major drawback is the amount of I/O that it consumes. The advantage of HH is that its memory requirement is much lower than that of the other methods. HH reduces the amount of I/O by storing a hashed version

of both relations on disk but will be out of luck if not enough disk space is available. For NBT with either buffering scheme, the key factor is the relationship of device speeds and buffer space. With slow disks the amount of buffer space required to keep the tape drive streaming may be prohibitive. It also consumes more disk I/O than HH and at higher utilization levels. On the other hand, the response time will be similar to that of HH, without HH's disk space requirement.

8 Considerations when Tape is Faster Than Disk

As described in the analysis in Appendix B, it would make sense for Nested Block Join to use S as the inner and R as the outer relation when S is stored on the faster device. If M is a moderate fraction of $|R|$, then S will be read a small number of times $\left\lceil \frac{|R|}{M} \right\rceil$ and tape wear will not be a problem. The join would be computed as a regular Nested Block Join. However, if M is a small fraction of $|R|$, then the number of iterations over S may become excessive, and due to tape wear concerns one might prefer to reverse the roles of R and S.

If the tape device is faster than the disk, it would not be worthwhile to use NBT with either of the buffering schemes we have presented. If NBT with memory buffering were used, the number of iterations over S would be doubled since memory buffering halves the R chunk size. This would also double the tape wear. Using NBT with disk buffering, the tape transfer would only be slowed down by the disk.

Table 6 summarizes the selection of a join method under various scenarios.

Table 6: Choice of Join Method

faster device	memory, $\frac{M}{\|R\|}$	disk space required	method chosen
disk	high	none	NBT/MB, R as inner
disk	moderate	little	NBT/DB, R as inner
tape	moderate	none	NB, S as inner
tape	low	none	NB, R as inner
either	very low	large amount	HH

9 Implementation and Measurement

9.1 System Configuration

For performance measurement and comparison, we implemented the NBT/MB and NBT/DB algorithms as well as a tape version of NB. Our system has an 8 mm Aviv TFS-8200 tape drive which is attached to a DECstation 5000/125. The capacity of the drive is 2.3 GB and its maximum

sustained transfer rate is 0.26 MB/s. With a 4 kB block size, we reached a sustained rate of 0.23 MB/s.

Our implementation allows data read from tape to be distributed to n hosts, all running the 'join part' of an algorithm in parallel. In the experiment configuration, however, all tape data was shipped over the local area network to one host, a DEC Alpha 3000/400 workstation with a 3 MB/s disk.

From the disk and tape transfer rates one can derive $r = \frac{3}{0.23} = 13.0$ for our configuration. Using our formulas from Table 4, the minimum amount of memory to achieve a tape-bound join in NBT/MB is $M = \frac{1+F}{0.9 \cdot 13.0} \cdot |R| = 0.19|R|$. In NBT/DB the memory requirement for tape-boundness is $M = \frac{F}{0.9 \cdot (13.0-2)} \cdot |R| = 0.12|R|$.

9.2 Experiment Results

The tape relation S was loaded with random data in 1 million records of 128 bytes each (totaling 122.1 MB). Each record had an integer number, a floating-point number, and a string field. The integer field was used as the join attribute. On the disk, relation R with the same schema and similar random data was created but with 200,000 records (24.4 MB total).

Figure 12 shows the response time of $R \bowtie S$ as a function of the amount of memory available. In all experiments, the time to read relation S (122.1 MB) from tape was roughly 600 seconds, or 10 minutes, and is shown as a dotted line in both figures. The data transfer time includes 40-50 seconds to 'open' the tape device for I/O. For a given amount of memory, each join was executed three times and the lowest response time was plotted. Small variations in response time were due to competing CPU, disk, and network activity in the system.

The relative performance of NB and NBT/MB in Figure 12 demonstrates the potential of reducing response time when a sufficient amount of memory is assigned to tape I/O buffering. NBT/MB reached a tape-bound state when approximately 19 percent of R fit in memory, almost exactly the predicted requirement. The response time for tape-bound joins was roughly 5 percent higher than the tape transfer time. This is partly due to the time consumed in the last cycle of NBT/MB where tape transfer has already finished but the last chunk of S still needs to be joined with R.

When little memory is available, NBT/DB performed the best of all three. Its response time settled at the tape-bound level when roughly 13 percent of R fits in memory. It is interesting to note, however, that the tape-bound response time was about 10 percent higher than the minimum possible. Again, this is partly, but not completely, explained by the time consumed in the last cycle which takes place after the tape transfer has finished.

Obviously, increasing memory beyond the tape-bound requirement did not reduce the response time of NBT/MB or NBT/DB. However, allocating more than 30 percent of $|R|$ in main memory

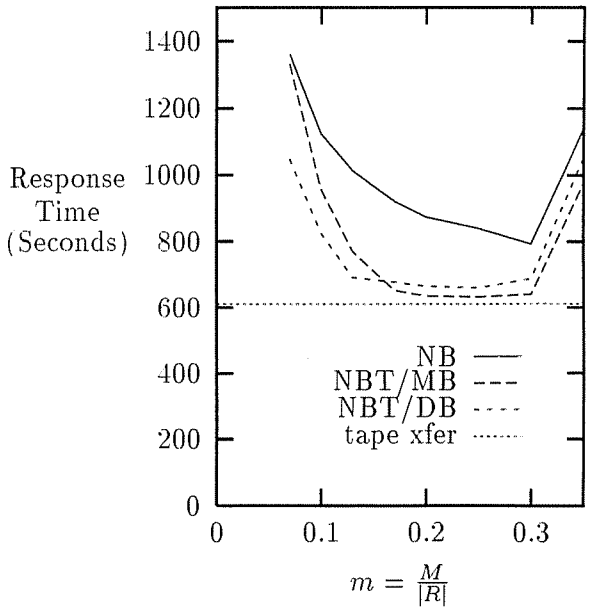

Figure 12: Response Time vs. Memory Size ($|R|$ = 24.4 MB, $|S|$ = 122.1 MB)

resulted in page faulting in our system and this is observed as an abrupt increase in the response time of all three join algorithms.

10 Future Work

We are analyzing a hash join method which hashes S from tape to another tape instead of disk. This approach uses tape space and requires that S be read from tape twice. On the other hand, tape space is much cheaper than disk space. Reading from tape and disk concurrently also introduces opportunities for reducing overall response time, as demonstrated by Nested Block Tape Join.

The join of two relations is typically accompanied by a selection or a projection on one or both of the relations (select-project-join queries). Since selection and projection are performed before the join, the reduction in data volume has a direct effect on the Nested Block Tape Join synchronization issues discussed earlier. The conditions for guaranteeing streaming tape access are relaxed in many ways. The result is that relation R can be larger than before if memory size is unchanged, or that the memory requirement for a given $|R|$ is lower than previously. In our future work, we plan to study in detail the effect of select and project operations on disk and tape synchronization.

11 Conclusion

Joining two relations is one of the most common and expensive operations in a relational DBMS. The question of which join method a query optimizer should choose is not a matter of minimizing response time only. Allocating system resources, such as disk space, disk bandwidth, or memory

space, among competing processes is a challenging task for a query optimizer. The resource requirements of two join methods can vary significantly even if the predicted response times were identical. For instance, the choice between Nested Block Join and Hybrid Hash Join basically involves choosing to consume more disk bandwidth (transfer capacity) or more disk space (storage capacity).

Extensive research has been devoted to the analysis of joining relations on disk. To our knowledge, there are few studies on joins that consider tape devices as a storage medium. It is typically assumed that data on tapes is copied to disk, followed by the execution of a chosen disk-based join method. Such an approach requires that disk space is made available for storing tape-resident data. In this paper we have explored how the conventional, disk-based Nested Block Join and Hybrid Hash Join can be modified to operate directly on tapes. In particular, we have studied how the disk space requirements of these methods can be reduced or eliminated while at the same time reducing the overall I/O cost.

We have presented a modification to Nested Block Join which exploits I/O concurrency when one of the relations is stored on tape and the other is on disk. Using double buffering, streaming tape access throughout the join operation can be achieved and maintained. If relation R is stored on disk and S is on tape, Nested Block Tape Join can compute $S \bowtie R$ in only the transfer time of S from tape. In other words, the response time of the join depends only on the speed at which data can be retrieved from tape. We have identified the key factors that determine which method to use to join a disk relation with a tape relation, and what the trade-offs are in each method.

In this paper we have shown that a DBMS can effectively reduce the disk space requirement and response time of tertiary join operations by having direct control over data on tertiary devices. Disk space will be saved if secondary storage in a storage hierarchy can be bypassed and tape-resident data can be read directly into memory. Large time savings can be realized if the DBMS is aware of the differences in the operating characteristics of the devices on which data is stored. In particular, overlapping tape and disk I/O can be very effective when there is a large difference in tape and disk transfer rates.

References

[1] M. Blasgen and K. Eswaran. Storage and access in relational data bases. *IBM Syst. J.*, 16(4):363–377, 1977.

[2] K. Bratbergsengen. Hashing methods and relational algebra operations. In *Proc. Conf. Very Large Databases*, pages 323–333, Singapore, Aug. 1984.

[3] B. W. Culp, D. R. Domel, W. T. Gregory, J. J. Kato, G. C. Melton, K. A. Proehl, D. W. Ruska, V. K. Russon, and P. Way. Streaming tape drive control electronics. *Hewlett-Packard J.*, 39(3):43–54, June 1988.

[4] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. Wood. Implementation techniques for main memory database systems. In *Proc. ACM SIGMOD*, pages 1–8, Boston, MA, June 1984.

[5] L. M. Haas, M. J. Carey, and M. Livny. SEEKing the truth about *ad hoc* join costs. Technical Report 1148, Univ. of Wisconsin at Madison, May 1993.

[6] L. M. Haas, M. J. Carey, and M. Livny. Tapes hold data too: Challenges of tuples on tertiary store. In *Proc. ACM SIGMOD*, pages 413–417, Washington, D.C., May 1993.

[7] R. B. Hagmann. An observation on database buffering performance metrics. In *Proc. Conf. Very Large Databases*, pages 289–293, Kyoto, Japan, Aug. 1986.

[8] W. Kim. A new way to compute the product and join of relation. In *Proc. ACM SIGMOD*, pages 179–187, Santa Monica, CA, May 1980.

[9] D. Knuth. *The Art of Computer Programming, Vol. III: Sorting and Searching.* Addison-Wesley Publishing Co., Redwood City, CA, 1973.

[10] S. Sarawagi and M. Stonebraker. Single query optimization for tertiary memory. Technical Report 45, Univ. of California at Berkeley, Mar. 1994.

[11] L. Shapiro. Join processing in database systems with large main memories. *ACM Trans. Database Syst.*, 11(3):239–264, Sept. 1986.

[12] M. Stonebraker. An overview of the Sequioa 2000 project. Technical Report 5, Univ. of California at Berkeley, Dec. 1991.

[13] R. Thomas. Cache memory splits computer and tape operations. *Computer Design*, 24(13):89–93, Oct. 1985.

A Analysis of Disk-Tape Joins with Stop-Start Tape Drives

In this section we describe an analysis of Nested Block Tape Join under the assumption that each stop and start of the tape drive is visible to the DBMS and incurs a *stop/start delay.* The performance of Hybrid Hash Join is unaffected by the stop/start delay because it reads the tape relation in one, uninterrupted transfer.

A.1 Cycle Time Constraints

In addition to the first time constraint introduced in Section 5.2, each cycle will now have a second time constraint: *Reading R takes less time than reading S_i plus slowing down the tape drive to a full stop.* As stated in Section 5.2, the tape is streaming if time constraint 1 holds. If the first constraint fails but the second one holds, an extra stop/start delay will be incurred.

As before, the join is tape-bound if the first constraint holds. The join is *stop-bound* when constraint 1 fails but constraint 2 holds. This means that the effective join rate is determined by the combination of tape transfer rate and the stop/start delay. If constraint 2 fails, the join is disk-bound.

Table 7: Resource Requirements and I/O Cost

algorithm	required disk space	required memory		I/O cost	bounding factor
NBT Memory	none	$\|R\| \cdot \frac{c}{r}$	if $r > c/F$	$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R \cdot \alpha \cdot r$	T
Buffering		$\|R\| \cdot \frac{c}{r}(1-s)$	if $r > c(1-s)/F$	$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R(\alpha \cdot r + s)$	T, S
		$M_R + M_S(1+F)$		$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R$	D
NBT Disk	$M \cdot 1.8/F$	$\|R\| \cdot d$	if $r > 2(1+F)$	$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R \cdot \alpha \cdot r$	T
Buffering		$\|R\| \cdot d(1-s)$	if $r > 2(1+F)$	$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R(\alpha \cdot r + s)$	T, S
		$M_R + M_S F$		$\lceil \frac{\beta}{\alpha} \rceil \cdot T_R(1+2\alpha)$	D

Coefficient $c = \frac{1+F}{0.9}$. Coefficient $d = \frac{F}{0.9(r-2)}$.
Bounding factor: tape read (T), tape stop (S), or disk read (D).

A.2 Cost Formulas

Define s as the ratio of tape stop/start time to T_R (transfer time of R from disk). Then $T_R \cdot s$ represents the cost of one interruption in a streaming tape access, in seconds. The cost of a stop-bound cycle is $T_R(\alpha \cdot r + s)$ where $T_R \cdot \alpha \cdot r$ represents the transfer cost of a chunk of S from tape, and $T_R \cdot s$ is the added stop/start delay.

The memory requirement for stop-bound NBT/MB and NBT/DB is derived by substituting $r/(1-s)$ for r in the tape-bound analysis (cf. Sections 6.3 and 6.4). $r/(1-s)$ is the ratio of disk transfer rate to tape transfer rate adjusted with the stop/start delay. The adjustment reduces the effective tape transfer rate, and therefore the memory requirement of a stop-bound join is lower than that of a tape-bound join. Table 7 shows the revised cost formulas for both NBT/MB and NBT/DB.

A.3 Performance Charts

The relative cost of NBT/MB and NBT/DB as well as NB and HH is shown in Figures 13 and 14. Corresponding to the disk-bound, stop-bound, and tape-bound modes of operation in NBT/MB and NBT/DB, the costs of these methods have now three regions. As before, the joins are disk-bound in Region I. With enough memory, the disk cost in each cycle becomes less than the cost of tape access plus a tape stop-start. The joins become stop-bound. This forms Region II and shows up as a tilted plateau in the chart. Increasing memory still more allows tape stops and starts to be avoided, and hence the total cost of the join is then just the transfer cost of S from tape (Region III).

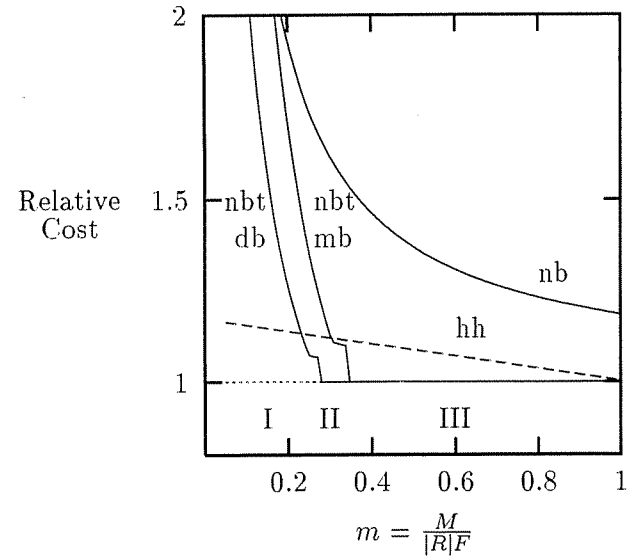

Figure 13: Total I/O cost ($r = 6$)

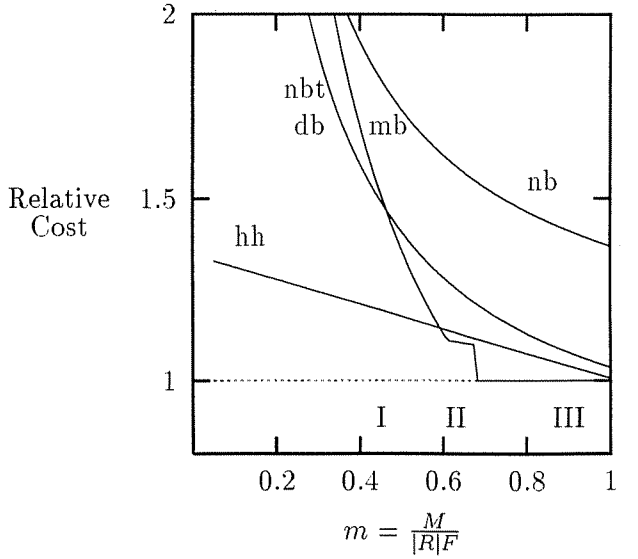

Figure 14: Total I/O cost ($r = 3$)

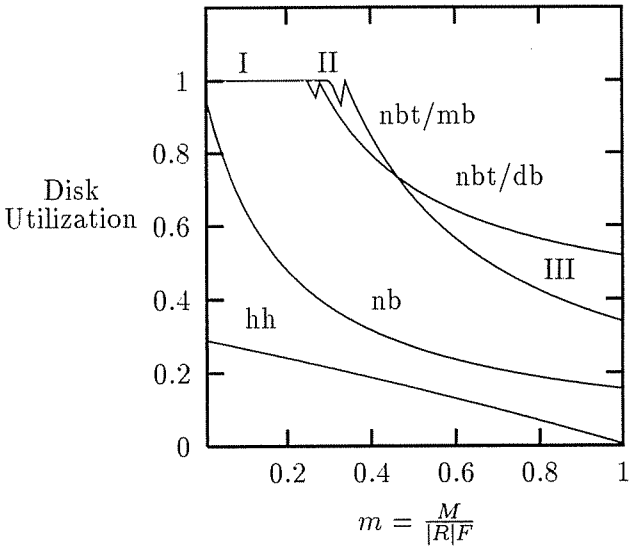

Figure 15: Disk Utilization ($r = 6$)

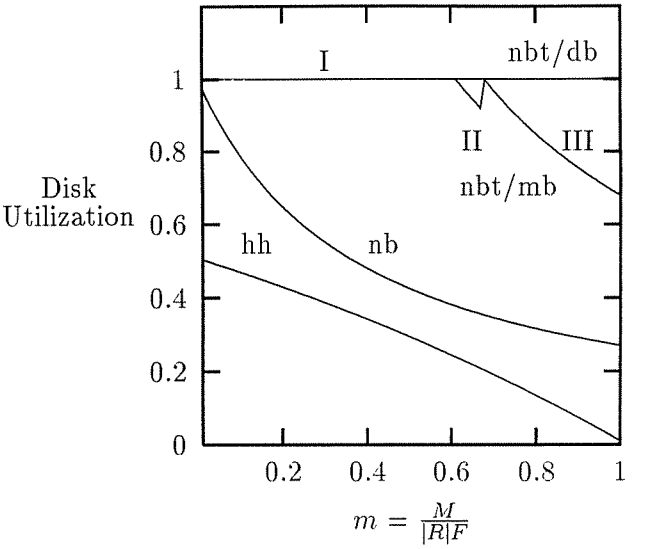

Figure 16: Disk Utilization ($r = 3$)

Figures 15 and 16 show disk utilization as a function of memory size. In Region I, the join is disk-bound and the disk utilization is 1.0. When memory is increased and the join becomes stop-bound, disk utilization drops because disks are not accessed during tape stop and start (Region II). Increasing memory a little more makes tape stops and starts unnecessary, and therefore disk utilization abruptly increases back to the 1.0 level. This is the point where disk time and tape time are exactly the same. In Region III, increasing memory reduces the total disk time (number of iterations goes down), which results in lower disk utilization because tape time remains unchanged.

B Selection of Inner vs. Outer Relation in Nested Block Join

In this section we discuss the implications of device transfer rates on the selection of inner vs. outer relation in Nested Block Join. Given that the larger relation, S, is stored on tape while the smaller, R, is on disk, we consider the transfer time of S and R as the prime criterion. Due to tape wear concerns, we also consider limiting the number of passes over S. Define the size of R and S as $|R|$ and $|S|$. Assume that M blocks are read from the outer relation in each iteration. The transfer rates of disk and tape are X_{DISK} and X_{TAPE}.

The cost of a Nested Block Join with S as the outer and R as the inner is approximated by the following, continuous formula:

$$\frac{|S|}{M} \cdot \frac{|R|}{X_{DISK}} + \frac{|S|}{X_{TAPE}}$$

When S is the inner and R is the outer, the formula is:

$$\frac{|R|}{M} \cdot \frac{|S|}{X_{TAPE}} + \frac{|R|}{X_{DISK}}$$

Since S is likely to be much larger than R, it is safe to assume that the transfer time of R from disk is less than the transfer time of S from tape:

$$\frac{|R|}{X_{DISK}} < \frac{|S|}{X_{TAPE}}$$

Combining the preceding three formulas, we find that it is cheaper to perform the join with S as the inner and R as the outer if the following holds:

$$\frac{|R|}{M} \cdot \frac{|S|}{X_{TAPE}} < \frac{|S|}{M} \cdot \frac{|R|}{X_{DISK}}$$

This yields the following, simple constraint for X_{DISK} and X_{TAPE}:

$$X_{DISK} < X_{TAPE}$$

If S is the inner relation, the number of passes over it is $\left\lceil \frac{|R|}{M} \right\rceil$. To reduce tape wear, in practice we would like to limit the number of passes over S to a relatively small number p_{max}, say, ten. The conditions for performing a Nested Block Join with S as the inner and R as the outer are therefore:

$$X_{DISK} < X_{TAPE} \qquad \left\lceil \frac{|R|}{M} \right\rceil < p_{max}$$