A Modeling Study of the TPC-C Benchmark

Scott T. Leutenegger *
ICASE: Institute for Computer Applications
in Science and Engineering
Mail Stop 132c
NASA Langley Research Center
Hampton, VA 23681-0001
leut@icase.edu

Daniel Dias
IBM Research Division
T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
dias@watson.ibm.com

Abstract

The TPC-C benchmark is a new benchmark approved by the TPC council intended for comparing database platforms running a medium complexity transaction processing workload. Some key aspects in which this new benchmark differs from the TPC-A benchmark are in having several transaction types, some of which are more complex than that in TPC-A, and in having data access skew. In this paper we present results from a modelling study of the TPC-C benchmark for both single node and distributed database management systems. We simulate the TPC-C workload to determine expected buffer miss rates assuming an LRU buffer management policy. These miss rates are then used as inputs to a throughput model. From these models we show the following: (i) We quantify the data access skew as specified in the benchmark and show what fraction of the accesses go to what fraction of the data. (ii) We quantify the resulting buffer hit ratios for each relation as a function of buffer size. (iii) We show that close to linear scale-up (about 3% from the ideal) can be achieved in a distributed system, assuming replication of a read-only table. (iv) We examine the effect of packing hot tuples into pages and show that significant price/performance benefit can be thus achieved. (v) Finally, by coupling the buffer simulations with the throughput model, we examine typical disk/memory configurations that maximize the overall price/performance.

1 Introduction

The TPC Benchmark C (TPC-C) [7, 11] is intended to model a medium complexity online transaction processing (OLTP) workload. It is patterned after an order-entry workload, with multiple transaction types ranging from simple transactions that are comparable to the simple debit-credit workload in the TPC-A/B benchmarks [6], to medium complexity transactions that have two to fifty times the number of calls of the simple transactions.

An important aspect of the workload is that is specifies skewed (i.e. non-uniform) access within individual data types/relations. By contrast, the TPC-A benchmark assumes uniform access within each relation/data type. The skewed access, which is typical for many OLTP workloads [4] allows better use of the main memory database buffer by allowing it to capture the hot data items.

The benchmark specifies a non-uniform random number generation function to be used for generation of tuple-ids. We provide insight into the distribution of this skew by simulating this function as specified by the benchmark. The output of this simulation specifies the skew at the tuple level, yet most typical DBMS's access and store data in pages. Therefore, to estimate the skew at the page level we also simulate the function assuming tuples are packed sequentially into pages. These results provide insight into the workload and help explain the miss rate results obtained in our buffer simulations. In addition we use the distribution obtained from this simulation to guide us in packing tuples into pages so that all tuples of similar "hotness" will be in the same page.

We assume the use of the LRU buffer replacement policy for the database buffer and simulate the buffer pool to determine the expected miss rates for each relation. We use the miss rates obtained from our buffer simulations as inputs to a throughput model. Using this model, we explore optimal buffer sizes to minimize hardware costs. Finally, we consider the impact of running the benchmark on a clustered/distributed database system, examining the impact of replicating one of the read-only relations.

We focus only on the access patterns and processing requirements of the benchmark. We do not consider terminal emulation, ACID properties, or pricing. When we present price/performance curves we will only consider hypothetical costs of hardware and do not include considerations such as terminal emulation or software maintenance costs as outlined in the TPC-C specification [11]. We describe the benchmark transactions only in the level of detail required to model the workload, primarily in terms of the access patterns and the number of database calls per transaction. Readers interested in details such as which fields are retrieved and updated are referred to the benchmark specification.

The rest of the paper is organized as follows. In Section 2 we provide a synopsis of the TPC-C workload, so that the paper is reasonably self contained. In Section 3 we present simulation results for the non-uniform random number generation routines to determine the degree of access skew. A description of our buffer model simulation including model results is contained in Section 4. A throughput model and price/performance results for both a single and a distributed system are given in Section 5. Concluding remarks appear in Section 6.

2 TPC-C Workload Synopsis

This section gives a summary of the TPC-C workload. For a more thorough treatment see [8, 9], the TPC-C specification [11], and

*A significant portion of this work was done while Leutenegger was a Post-Doctoral Researcher at IBM T. J. Watson Research Center. Support for Leutenegger was also provided by the National Aeronautics and Space Administration under NASA Conract Nos. NAS1-18605 and NAS1-19480.

Table 1: Summary of Logical Database

Relation Name	Cardinality	Tuple Length	Tuples Per 4K Page
warehouse	W	89 bytes	46
district	W * 10	95 bytes	43
customer	W * 30K	655 bytes	6
stock	W * 100K	306 bytes	13
item	100K	82 bytes	49
order		24 bytes	170
new-order		8 bytes	512
order-line		54 bytes	75
history		46 bytes	89

TPC-C overview [7]. In this paper, we focus only on the access patterns and processing requirements of the benchmark. For concreteness, we will assume a relational database model, though most of the development is applicable to other data models. We first give an overview of all five transaction types in the benchmark and then give a more detailed account of each of the transactions in the following section.

2.1 TPC-C Overview

The TPC-C benchmark is intended to represent a generic wholesale supplier workload. The workload is primarily a transaction processing workload with multiple SQL calls per transaction, but also has two aggregates, one non-unique select, and a join. The workload specifies skew (i.e. non-uniform access) at the tuple level for three of the relations.

Figure 1 shows the Business Environment Hierarchy of the TPC-C workload. This figure is a reproduction of that found in the TPC-C benchmark specification [11]. The overall database consists of a number of warehouses. Each warehouse is composed of ten districts where each district has 3,000 (3K) customers. There are 100K items that are stocked by each warehouse. The stock level for each item at each warehouse is maintained in the Stock relation. Customers place orders that are maintained in three relations: in the Order relation a permanent record of each order is maintained; in the New-Order relation, pending orders are maintained and later deleted by a Delivery transaction; in the Order-Line relation, an entry is made for each item ordered. A history of the payment transaction is appended to the History relation.

The logical database design is composed of 9 relations as listed in Table 1 and shown in Figure 2. In the table, W represents the number of warehouses. We make the assumption that only integral units of tuples fit per page. The cardinality of the Warehouse, District, Customer, and Stock relations scale with the number of warehouses. This is similar to the TPC-A benchmark where the cardinality of the Branch, Teller, and Account relations scale with the number of branches. The Item relation does not scale with the number of warehouses. The Order, Order-Line, and History relations grow indefinitely as orders are processed.

There are five transaction types in TPC-C as listed in table 2. The New Order transaction places an order for on average 10 items from a warehouse, inserts the order, and for each item updates the corresponding stock level. The Payment transaction processes a payment for a customer and updates balances and other data in the Warehouse, District and Customer relations. The customer can be specified either by a unique customer-id, or by a name. In the latter case, on the average three customers qualify from which one is selected. The Order Status transaction returns the status of a customer's last order. As in the Payment transaction, the customer may be specified by the customer-id or by name. Each item in the last customer order is examined. The Delivery transaction processes orders corresponding to 10 pending orders, one for each district, with 10 items per order. The corresponding entry in the New-Order relation is deleted. Finally, the Stock Level transaction examines the quantity of stock for the items ordered by each of the last 20 orders in a district.

Table 2 summarizes the transactions based on the percent of the workload each transaction comprises, and the number of selects, updates, inserts, deletes, non-unique selects, and joins for a relational model. There is a column for minimum percent of workload and a column for assumed percent of workload. The benchmark specifies a minimum percent for all the transaction types except the New Order transaction. The benchmark metric is the number of New Order transactions processed per minute, hence, it is desirable to set the percent New Order as high as possible (45%) taking into account that the size of the New-Order relation will grow without bound unless the relative rate of Delivery transactions is sufficient to delete the entries in the New-Order relation at the same rate that the New-Order transaction inserts them. The third column in the table is the percent of the workload mix that we have assumed for all studies in this paper. Note, the percent New-Order versus Delivery is a key parameter of this benchmark and should be tuned carefully to achieve the maximum New-Order transactions per second. The join is an equi-join, where the two relations involved each have on average just under 200 tuples that meet the selection predicate.

2.2 Transaction Access Patterns

In this section we summarize the access patterns of each transaction. For each transaction we list the database operations made by that transaction in a simplified pseudocode. Although our pseudocode is not SQL, it succinctly conveys the function of each transaction. A more detailed description is found in [8, 9] and the TPC-C specification [11] includes sample code for each transaction.

New Order Transaction

1. Select(whouse-id) from Warehouse
2. Select(dist-id, whouse-id) from District
3. Update(dist-id, whouse-id) in District
4. Select(customer-id, dist-id, whouse-id) from Customer
5. Insert into Order
6. Insert into New-Order
7. For each item (10 items):
   (a) Select(item-id) from Item
   (b) Select(item-id,whouse-id) from Stock
   (c) Update(item-id,whouse-id) in Stock
   (d) Insert into Order-Line
8. Commit

Payment Transaction

There are two cases. In the first case, which occurs 40% of the time, the customer is selected by customer-id. In the second case, which occurs 60% of the time, the customer is selected by last name. On average three customers will have the same last name, the actual customer chosen is determined by selecting all customers with that name, sorting on first name, and taking the middle one.

1. Select(whouse-id) from Warehouse
2. Select(dist-id,whouse-id) from District
3. (a) Case 1: Select(customer-id,dist-id,whouse-id) from Customer
   (b) Case 2: Non-Unique-Select(customer-name,dist-id,whouse-id) from Customer
4. Update(whouse-id) in Warehouse
5. Update(dist-id,whouse-id) in District

Table 2: Summary of Transactions

Transaction	Minimum %	Assumed %	Selects	Updates	Inserts	Deletes	Non-Unique Select	Join
New Order	$\star$	43	23	11	12	0	0	0
Payment	43	44	4.2	3	1	0	0.6	0
Order Status	4	4	11.4	0	0	0	0.6	0
Delivery	4	5	130	120	0	10	0	0
Stock Level	4	4	0	0	0	0	0	1

Table 3: Summary of Relation Accesses

Relation	New Order	Payment	Order Status	Delivery	Stock Level	Average
warehouse	U(1)	U(1)				0.87
district	U(1)	U(1)			P(1)	0.93
customer	NU(1)	NU(2.2)	NU(2.2)	P(10)		1.524
stock	NU(10)				P(200)	12.4
item	NU(10)					4.4
order	A(1)		P(1)	P(10)		0.53
new-order	A(1)			P(10)		0.49
order-line	A(10)		P(10)	P(100)	P(200)	13.3
history		A(1)				0.43

6. Update(customer-id,dist-id,whouse-id) in Customer
7. Insert into History
8. Commit

Order Status Transaction

1. (a) Case 1: Select(customer-id,dist-id,whouse-id) from Customer
   (b) Case 2: Non-Unique-Select(customer-name,dist-id,whouse-id) from Customer
2. Select(Max(order-id),customer-id) from Order
3. for each item in the order:
   (a) Select(order-id) from Order-Line
4. Commit

The database call "Select(Max(order-id),customer-id) from Order" selects the tuple in the Order relation that is the most recent order placed by the customer. This could be implemented as a max aggregate. Since the Order relation keeps on growing without bound this approach would be expensive. This could be implemented using an ordered multi-keyed index so that correct tuple can be fetched in just one index look up. Hence, in our studies we assume this requires the overhead of a single select.

Delivery Transaction

1. For each district within the warehouse (i.e. ten times):
   (a) Select(Min(order-id),whouse-id,dist-id) from New-Order
   (b) Delete(order-id) from New-Order
   (c) Select(order-id) from Order
   (d) Update(order-id) Order
   (e) For each item in the order (i.e. ten times):
      i. Select(order-id) from Order-Line
      ii. Update(order-id) Order-Line
   (f) Select(customer-id) from Customer
   (g) Update(customer-id) Customer
2. Commit

Similar to the "Max" operation in the Order-Status transaction, we assume the "Min" select is fetched in just one call.

Stock Level Transaction

Below we quote the sample SQL code directly from the TPC-C document [11] so that we do not confuse the query by oversimplification.

```
SELECT d_next_o_id INTO :o_id
  FROM District
  WHERE d_w_id = :w_id AND d_id = :d_id ;

SELECT COUNT(DISTINCT (s_i_id)) INTO :stock_count
  FROM Order-Line, Stock
  WHERE
    ol_w_id = :w_id AND
    ol_d_id = :d_id AND ol_o_id < :o_id AND
    ol_o_id ≥ (:o_id - 20) s_w_id = :w_id AND
    s_i_id = ol_i_id AND s_quantity < :threshold ;
```

Assuming an index on the order-id field of the Order-Line relation and a two keyed index on the whouse-id and item-id of the stock relation, the query results in an average of 200 Order-Line and Stock tuples each being fetched.

To summarize the access patterns of the five transaction we list the number of accesses to each relation for each transaction type and the average number of accesses per transaction in Table 3; the latter assumes the percentages for each transaction listed in Table 2. Within the table, the notation $U(x)$ signifies that x tuples are chosen **U**niformly from the relation, $NU(x)$ denotes **NonU**niform random selection of x tuples using the NU function, $A(x)$ denotes x tuples are **A**ppended to the relation, and $P(x)$ denotes x tuples are chosen where the tuples chosen were recently accesses by **P**ast behavior (in other words there is a form of temporal locality). Note that the tuples accessed by the Order-Status, Delivery, and Stock-Level transactions are more likely to be buffer pool hits since they are for tuples that have been recently put in the buffer pool by the New-Order transaction. Many of the tuple-ids are generated from the NU() function. We define and simulate this function in the next section.

3 Analysis of TPC-C Data Access Skew

The TPC-C benchmark assumes access to the tuples are skewed, i.e. within a relation some tuples are referenced more frequently than others. In this section we define and simulate the non uniform random number function, as specified by the TPC-C documents, used for the generation of tuple id's. The non-uniform random number generating function, NU(), which we paraphrase from the benchmark specification [11], is defined as follows:

$$NU(A,x,y) = (((rand(0,A) \mid rand(x,y)) + C)\%(y-x)) + x \quad (1)$$

where:

- rand(x,y) denotes a uniformly distributed integer random number in the closed interval [x..y],
- C is a constant within [0..A],
- A is a constant chosen according to the size of the range [x..y],
- (N % M) stands for N modulo M,
- and (N | M) stands for the bitwise logical OR of N and M.

For the remainder of this paper we assume C equals zero (the TPC-C standard document allows an arbitrary choice of C within [0..A]). We choose A and y according to the specifications for the tuple id being generated.

First we consider accesses to the stock and item relations. All tuple id's for accessing these relations are drawn from the NU(8191,1,100000) distribution. In Figure 3 we plot the probability mass function (PMF) for this distribution as obtained from simulating one billion samples. The plot shows the non-uniformity in access and the periodicity of the access probability in the first parameter (8191) of the NU function above. The number of cycles equals the (floor of the) third parameter divided by the first parameter of the NU function, or 12 cycles for this case. In [8, 9] we derive a closed form expression for the resulting PMF assuming the third parameter is a power of two for which the cycles are exact. Figure 3 is hard to interpret because of the large number (100,000) of points; hence, we plot the same distribution for tuples 1 to 10,000 in Figure 4. In this figure, the non-uniformity within a cycle (8191 points) is clear.

While the non-uniformity of access is apparent in Figure 4, the degree of skew is not clear. Let α_i be the probability of accessing tuple i. Let β_i be the fraction of the relation represented by that tuple. Note $\beta_i = \beta_j \quad \forall\, i,j$ for stock tuples. In Figure 5 we order the tuples by increasing order of α (increasing order of hotness) and plot $\sum \alpha_i$ versus $\sum \beta_i$, i.e. the cumulative probability of access versus the cumulative fraction of the relation. If a relation has no skew the curve would be linear; hence the more convex the curve is, the more skew there is. For the moment ignore the top two curves, and focus on the lower curve which represents the access skew at the tuple level. The graph shows that 16% of the accesses go to about 80% of the tuples, or alternatively, 84% of the accesses go to about 20% of the tuples. There is even more skew in the tail of the distribution, so that about 71% of the accesses go to about 10% of the (hottest) tuples and about 39% of the accesses go to about 2% of the (hottest) tuples.

In most typical databases data is stored in pages; hence we need to determine the skew at the page level. We first assume tuples are packed into pages in sequential order with the maximum number of whole tuples that fit per page. We assume the remainder of the page is wasted. For the stock relation 13 (26) tuples fit in each 4K (8K) page. Again, we order the pages by frequency of access and plot the cumulative probability of access versus the cumulative fraction of the database in Figure 5 (top two curves). The top (bottom) curve is for an 8KByte (4K) page size. For a 4KByte page size, we see that 25% of the access go to 80% of the data, or viewed the other way 75% of the accesses go to 20% of the data. This is similar to the so called "80-20" rule where 80% of the accesses go to 20% of the data. Again, there is a more skew in the tail of the distribution and about 59% of the accesses go to about 10% of the hottest pages, and about 28% of the accesses go to about 2% of the pages. The smaller page size results in more skew than the larger page size since there is less of a chance to spread out the hot tuples among the pages.

The milder skew at the page level leads to the question of whether the tuple level skew can be obtained at the page level. Packing tuples into pages in sequential order spreads out hot tuples among all the pages of the relation. A simple optimization is to first sort the tuples from hottest to coldest and then pack them into pages in that order. Since the distribution parameters for TPC-C are know a priori and are static in time, this can be done. (In this context we note that the TPC-C standard (Clause 1.4.1) allows clustering of tuples within pages.) This technique would also work for any workload where we know the distribution of accessing tuples within the relations of the database, and where the distribution does not vary with time. (We note, however, that in many real workloads, while there is considerable skew in data access, the access distribution is often not static in time.) The bottom curve in Figure 5 is the resultant skew when this optimized packing of tuples is used, and is virtually indistinguishable from the tuple level skew. Hence, the optimized packing results in more skew at the page level which should result in lower miss rates in the buffer pool. As a further note, this optimized tuple to page packing approach was insensitive to page size.

Accesses to the item relation exhibits a similar skew except there is less skew for the non-optimized packing approach since 49 (99) tuples fit per 4K (8K) page.

Access to the customer relation is less skewed than the stock and item relations since tuples are accessed by both tuple-id and customer-name. Hence, there are two different access patterns which are superimposed upon the relation. If the customer-id is used as the selection key, one tuple is selected from the NU(1023,1,3000) distribution. If the customer-name is used, we make the simplifying assumption that the customer name is selected from one of the NU(255,1,1000), NU(255,1001,2000) and NU(255,2001,3000) distributions with equal probability. Hence, as can be derived from the transaction access patterns as specified in Section 2.2, 41.86% of the accesses to the customer relation use the NU(1023,1,3000) distribution and 58.14% are divided equally among NU(255,1,1000), NU(255,1001,2000), and NU(2001,3000) distributions. In Figure 6 we plot the PMF for the customer relation and in Figure 7 we plot the $\sum \alpha_i$ versus $\sum \beta_i$. We note that there is considerably less skew for the customer relation than for the Stock relation.

4 LRU Buffer Simulation

In this section we outline our buffer simulation model and present miss rates obtained from our model. We simulated the buffer pool for the TPC-C benchmark assuming an LRU replacement policy. We hypothesize that more sophisticated replacement policies could result in an even larger difference between optimized packing of tuples and non-optimized packing of tuples since they should be able to capitalize more on the access skew. In our simulations we collected confidence intervals using batch means with 30 batches per simulation and a batchsize of 100,000 samples. All results (i.e. the miss rates of each relation) have confidence intervals of 5% of less at a 90% confidence level.

In the buffer model, we simulate transactions entering the system sequentially, and do not consider the case where multiple transactions may be in the system at the same time. The presence of concurrent transactions does not change the buffer hit ratio significantly because the fraction of pages accessed by any transaction is small compared to the buffer size. We include concurrent transactions in the throughput model in Section 5.1. When a transaction enters it is chosen as one of the five types according to the distribution for each type. Each transaction generates tuple requests and inserts as specified in Section 2.2. The simulation keeps track of the last order placed by each customer, the last 20 orders for each district, and which tuples are in the New-Order relation. This information is used by the the Order-Status, Delivery, and Stock-Level transactions. The output from the simulation is the miss rates for each relation summed over all transaction types, and also the miss rates for the accesses by the Order-Status, Delivery, and Stock-Level transactions in isolation to be used as inputs for the throughput model.

In Figure 8 we plot the miss rates versus the buffer size for the Stock, Customer, and Item relations. The other relations all have significantly lower miss rates. We include curves for both the sequential packing of tuples into pages and the optimized packing of tuples. The curves are, from top to bottom, the Customer relation, Stock relation, and Item relation. For each of the relations, the optimized packing of tuples results in significantly lower miss rates. There are two reasons why the Customer relation

Table 4: Throughput Model Summary : Single Node

resource	parameter	n	overhead	NewOrder V_1	Payment V_2	Status V_3	Delivery V_4	Stock V_5
CPU	select	1	20K	23	4.2	13.2	130	1
CPU	update	2	20K	11	3	0	120	0
CPU	insert	3	20K	12	1	0	0	0
CPU	delete	4	20K	0	0	0	10	0
CPU	commit	5	40K [1]	1	1	1	1	1
CPU	initIO	6	5K	$1 + mc$ $+10(mi + ms)$	1+2.2(mc)	2.2(mc) +mo+10(ml)	1+10(mc+mo+mn) +130(ml)	200(ms+ml)
CPU	application	7	0.1K	47	8	13	261	3
CPU	send/receive	8	15K	0	0	0	0	0
CPU	prepCommit	9	40K	0	0	0	0	0
CPU	initTransaction	10	50K	1	1	1	1	1
CPU	releaseLocks	11	35K	1	1	1	1	1
CPU	non-unique-select	12	50K	0	0.6	0.6	0	0
CPU	join	13	2000K	0	0	0	0	1
disk	IO	14	25ms	$mc + 10(mi + ms)$	2.2(mc)	2.2(mc) +mo+10(ml)	10(mc+mo+mn) +130(ml)	200(ms+ml)

exhibits a larger miss rate than the Stock relation even though the Customer relation is the smaller of the two. The first is that the customer relation has less skew as shown in Section 3. The second is that the stock relation is accessed more frequently as show in table 3. The item relation has a much lower miss rate since the relation is much smaller than the stock and customer relations due to the fact that the item relation does not scale with the number of warehouses.

The optimal packing approach results in significantly lower miss rates than the sequential packing approach. For example, the miss rate for the stock relation for a buffer size of 52M is 30% lower in absolute terms for the optimized packing approach than for the sequential approach. The miss rate for the stock relation averaged over all buffer sizes considered is 13% lower in absolute terms for the optimized packing approach than for the sequential approach. This significantly lower miss rate translates directly to a lower I/O rate, and hence better performance. Similar improvements are seen for the Customer relation miss rates and to a lesser extent for the Item relation.

We assume 20 Warehouses at a node. The reason for choosing the case of 20 Warehouses relates to the throughput model in Section 6, where it is estimated that about 20 Warehouses could be supported by a 10 MIPS processor. Beyond a sufficiently large number of warehouses, the buffer hit characteristics approximately scale with the number of Warehouses. The reason the scaling is not exact is that the item relation does not scale with the number of Warehouses, but it's effect diminishes with an increase in the number of Warehouses. The Warehouse and District relations are sufficiently small that they fit in the buffer (miss rate 0%) for all simulations considered.

5 System Model and Performance Estimates

5.1 Throughput Model Description

In this section we describe our throughput model. The parameter values used in the model are similar to those in [3, 5]; they do not reflect any particular system, but are intended to be somewhat representative. The objective is to identify trends rather than providing specific throughput or price-performance estimates. Our model incorporates both the CPU and the data disks. We assume that the system is configured with a sufficient number of disk arms to ensure disk arm utilization remains below 50% and hence the CPU is the bottleneck. To calculate CPU utilization the model sums the average CPU demand per transaction, divides by the MIPS rating of the processor, and then multiplies by the throughput. Our primary metric is maximum throughput which we obtain by fixing the CPU utilization and calculating the throughput. To calculate the disk utilization we sum the average disk demand per transaction in milliseconds, divide by the number of disk arms, and then multiply by the system throughput. We assume that there is a separate log disk.

In table 4 we summarize the assumed parameter values and visit counts for each transaction type for a single node system. The column label n is the subscript of the parameter. In the equations below we will use o_n to denote the overhead for a parameter n call. We define visit count as the number of times a transaction requires a certain operation per transaction type. The visit counts are in the columns heading $V_1 \ldots V_5$. We define $V_{i,j}$ to be the visit count for transaction i to operation j.

Most of the parameters in the table are self evident from the names with the following possible exceptions. The parameter *application* is for application code between SQL calls, the parameter *send/receive* is for the CPU overhead at one node to send and receive a message across the network, the parameter *releaseLocks* is for the release lock portion of the commit phase, *prepCommit* is for the prepare to commit portion of a 2 phase commit, and *initIO* is the CPU overhead for initiating an I/O. The overhead for releasing locks is obtained by summing the overhead to release read-locks and write-locks times the number of locks held by each transaction type weighted by the percent of the workload comprised by each transaction type. We assume an overhead of 1K instructions for releasing each lock.

The parameters mc, mi, ms, mo, and ml found in $V_{i,6}$ and $V_{i,14}$, $i \in 1, \ldots 5$, are the miss rates for the Customer, Item, Stock, Order, and OrderLine relations respectively. These miss rates are obtained from the buffer model. Note that for completeness we could have also included the miss rates for the Warehouse, District and New-Order relations in the performance estimates, but these miss rates are always negligibly small and hence are omitted from the table.

The overhead for the non-unique select is based on the fact that on average three values are returned and need to be sorted. The overhead for the join is estimated as follows. On average there are 200 items ordered by the last 20 order transactions and hence a range scan returning an average of 200 items is invoked to create a temporary table for the outer relation. Each one of these tuples will join with exactly one tuple from the inner relation. Assuming that appropriate indexes exists on the inner relation, each outer relation tuple requires an indexed select on the inner relation. Finally, the result must be sorted to eliminate duplicate

Table 5: Definition of Notation

symbol	meaning
RC_{stock}	expected number of calls for obtaining and updating stock tuples
RC_{cust}	expected number of calls for obtaining and updating customer tuples
RC_{item}	expected number of calls for obtaining and updating item tuples
U_{stock}	expected number of unique remote sites that supply stock tuples
U_{cust}	expected number of unique remote sites that supply customer tuples
U_{item}	expected number of unique remote sites that supply item tuples
$U_{item+stock}$	expected number of unique remote sites that supply item or stock tuples
L_{stock}	probability that all stock tuples are supplied from the local warehouse

Table 6: Throughput Model Summary : Multi Node with Replication

resource	parameter	n	overhead	NewOrder V_1	Payment V_2
CPU	commit	5	30K	$1 + U_{stock}$	$1 + U_{cust}$
CPU	initIO	6	5K	$1 + mc$ $+10(mi + ms)$ $+U_{stock}$	$1 + 2.2\ mc$ $+U_{cust}$ $+U_{cust}$
CPU	send/receive	8	10K	$4\ U_{stock}$ $+2\ RC_{stock}$	$2\ RC_{cust}$ $+4\ U_{cust}$
CPU	prepCommit	9	15K	$U_{stock} + 1$ $-L_{stock}$	U_{cust}

items. We assume the overhead for the range scan is 5K per tuple, the overhead for the indexed select is 5K instructions per tuple, and the overhead for the final sort is 40K resulting in a total CPU overhead of 2040K instructions.

In table 6 we summarize the visit counts which differ from the single node case for a distributed environment when the Item relation is replicated across all nodes, i.e. we include remote calls and distributed commits. In table 7 we summarize the visit counts assuming the Item relation is not replicated. The visit counts for the Payment transaction are the same for both replication and no replication since the Payment transaction does not access the Item relation. Note that only the New-Order and Payment transactions differ from the single node case since the other transaction only access local warehouses as specified by the benchmark. The notation found in tables 6 and 7 is defined in table 5. For brevity reasons we omit the derivation of these terms and detailed explanation of tables 6 and 7. The details can be found in [8, 9].

Let $V_{i,n}$ equal the visit count of a type i transaction to the CPU as a type n request. The values of $V_{i,n}$ are obtained from tables 4, 6, or 7 depending on whether the system being modeled is a single node system, distributed system with the Item relation replicated, or a distributed system without replication of the Item relation. Let λ equal the system throughput and α_i denote the fraction of the workload from transactions of type i. The utilization of the CPU is calculated as:

Table 7: Throughput Model Summary : Multi Node No Replication

resource	parameter	n	overhead	NewOrder V_1
CPU	commit	5	30K	$1 + U_{stock+item}$
CPU	initIO	6	5K	$1 + mc + 10(mi + ms)$ $+U_{stock}$
CPU	send/receive	8	10K	$2RC_{stock} + 2RC_{item}$ $+ 4U_{stock} + 2U_{item}$
CPU	prepCommit	9	15K	$U_{stock} + 1 - L_{stock}$

$$Util_{CPU} = \frac{\lambda \left(\sum_{i=1}^{i=5} \sum_{n=1}^{n=13} \alpha_i \cdot V_{i,n} \cdot o_n \right)}{MIPS} \quad (2)$$

Let DA = the number of disk arms. The utilization of the disk is calculated as:

$$Util_{disk} = \lambda \left(\frac{\sum_{i=1}^{i=5} \alpha_i \cdot V_{i,14} \cdot o_{14}}{DA} \right) \quad (3)$$

5.2 Single Node Performance Estimates

In this section we present our results for a single node system running the TPC-C benchmark, for the parameter values and assumptions given above. We assume the MIPS rating of the processor is 10 MIPS. We obtain the maximum throughput by fixing the maximum CPU utilization at 80% and calculating the throughput using the throughput model outlined above. We then obtain the number of disks needed by fixing the maximum disk utilization at 50% and finding the minimum number of disks such that disk utilization is less than or equal to 50%. Note that typical configurations are designed so that the average disk utilization is lower than the 50% we assume, so as to take into account variance in the disk load (for example see [10]). However, in a benchmark environment a higher disk utilization may be permissible because of a smaller variance in the disk load. All experiments assume a 4K page size.

In Figure 9 we plot the maximum throughput in new-order transactions per minute versus buffer size. The curves from top to bottom are for optimized packing of tuples into pages and non-optimized packing of tuples into pages.

The maximum percentage difference between the methods occurs at a buffer size of 44 megabytes where the optimized workload results in a 2.5% higher throughput relative to the non-optimized workload. The average throughput improvement (averaged over all 64 buffer sizes plotted in Figure 9 is 1.0% relative to the non-optimized workload. Hence, based on maximum throughput there is little incentive to pack all the hot tuples into separate pages versus just loading the database in sequential order.

In Figure 10 we plot the cost per transaction/minute versus buffer size, where we define cost as the cost of the memory, disks (including sufficient storage space for all relations), and the processor. *We emphasize that this is not the cost as specified by the TPC-C benchmark since it does not include software cost, maintenance cost, terminal cost, etc. The intent is to estimate the optimal database memory buffer size in the trade-off between memory and disks.* The storage cost is computed by summing the storage needs for the Warehouse, District, Customer, Stock, and Item relations as specified in table 1. Assuming 20 warehouses per node (which leads to about 80% CPU utilization), the space required is 1.1 Gbytes. In addition, we must include sufficient storage for running the benchmark for 180 8 hour days as specified by the benchmark. Each NewOrder transaction inserts 1 order tuple, and 10 order-line tuples. In addition each Payment transaction inserts one History tuple. By multiplying the transaction rate times the number of bytes needed for these inserts we arrive at approximately 11 Gbytes of disk space per node needed for storing these three relations. This space requirement scales linearly with the throughput. We assume each 3 Gbyte disk costs $5000, the processor costs $10000, and memory costs $100 per megabyte. Although these hardware costs are debatable and will quickly be out of date, they enable us to present a methodology which can be used for determining the optimal price/performance point. This method is beneficial in determining how much memory versus disk arms the system should be configured with.

We first focus on the bottom two curves in Figure 10. These two curves do not include the storage capacity needed for maintaining the Order, Order-Line, and History relations. The top curve of these two is for a workload with sequential packing of tuples into pages, while the bottom curve is for the case of optimal packing of tuples into pages. The jagged shape of the curves results from the adding of memory until the disk utilization drops sufficiently to configure the system with one less disk and still have a utilization of less than 50%. The lowest point on the y axis for each curve corresponds to the optimal cost/performance point and shows the corresponding amount of database buffer memory. (Note again that this is not the entire system cost.) The lowest points occurs for a 154 Mbyte buffer with a value of about $139/tpm for sequential packing, and at 84 Mbyte with a value of about $107/tpm for the optimal packing case. Thus, the optimized packing of tuples results in about a 30% improvement of price performance relative to sequential packing.

The top two curves in Figure 10 include the the storage capacity needed for maintaining the Order, Order-Line, and History relations. In this case, adding memory causes the disk utilization to drop sufficiently to configure the system with less disks, but the required storage capacity precludes removal of additional disks. A minimum of 4 disks are required for storage capacity requirements. The lowest points occurs at a 52 Mbyte buffer with a value of about $167/tpm for sequential packing, and at 26 Mbyte with a value of about $154/tpm for the optimal packing case. Thus, the optimized packing of tuples results in about an 8% improvement of price performance relative to sequential packing. Put another way, the system is disk bandwidth bound for memory sizes less than 26 megabytes (52) for the optimized (non-optimized) case, and storage capacity bound for larger memory sizes. Hence, there is no benefit obtained from adding additional memory beyond these points. Note, given the rate at which disk size is currently increasing the system will become disk bandwidth bound in the near future rather than storage capacity bound, in which case the cost/performance difference will become closer to the 30% predicted when storage costs are not included. For example, when a $5000 6 Gbyte disk is assumed the cost/performance improvement resulting form optimal packing is 20%. If a 12 G byte disk is assumed the entire database fits on one disk and the cost/performance improvement is 30%.

From this simple model, we conclude that depending on the disk bandwidth to storage capacity ratio, the (hardware cost)/performance ratio may be improved by up to 30% by careful loading of the database, i.e. packing all hot tuples into the same set of pages. Note, this does not take into consideration the cost of the software or software maintenance which when all lumped together will reduce the percent difference significantly.

5.3 Multiple Node System Estimates

In this section we present our results for a multiple node distributed system running the TPC-C benchmark. We assume each node contains 20 warehouses and all data pertaining to the node (except the item relation in the non-replicated case) is located on that node. We consider two cases. The first case is when the item relation is replicated across all sites. Since the item relation is read-only, replication protocols could be optimized for this case resulting in little/no overhead for replica management. Note that in a real database this would not be a trivial task if the Item relation can be changed. The second case assumes that the Item relation is not replicated, but rather partitioned equally among the nodes. In this case, all accesses to the item relation will incur a remote call with probability $\frac{N-1}{N}$, where N is the number of nodes in the system. In addition a one-phase commit involving each node that supplies an item tuple is necessary.

In Figure 11 we plot the maximum throughput versus the number of nodes for a buffer size of 102 Mbytes. We only plot results for the optimized packing model; results for the non-optimized model are similar. The top curve is for comparison purposes only, and represents a perfectly linear growth in maximum throughput with the number of nodes. The second curve is for the case where the Item relation is replicated, and the third curve is for the case where the Item relation is not replicated.

The benchmark scales almost linearly when the Item relation is replicated. This excellent scaleup occurs because only 10% of the New-Order transactions and 15% of the Payment transactions involve a remote warehouse. When the Item relation is not replicated the benchmark does not scale as well since each New-Order transaction must make $10\left(\frac{N-1}{N}\right)$ remoted calls, one for each item ordered. The replicated case has a 10, 30, and 39% higher throughput than the non-replicated case for 2, 10, and 30 nodes respectively. Hence, if the benchmark is to be run on a distributed system, replication of the Item relation will greatly improve system performance. We should emphasize that this assumes the use of a concurrency protocol (CC) which only requires remote access only when acquiring exclusive locks, i.e. the concurrency control (CC) protocol is optimized for read-only sharing so that no remote calls are made for CC for the replicated item relation. If a protocol optimized for write sharing were used, the performance would drop considerably. For instance if the primary copy protocol [2] were used for replication, there would be little performance gain over the non-replicated system since locks would have to be acquired remotely for each access.

The TPC-C benchmark specifies that for each item ordered in the New-Order transaction only 1% are stocked by a remote warehouse. In addition, the benchmark specifies that 15% of customers making payment via the Payment transaction are making the payment through a remote warehouse. These specifications result in a very low percentage of remote calls and hence the good scale-ups shown for the replicated case shown in Figure 11. We now examine the sensitivity of the results to this assumption. In Figure 12 we plot the maximum throughput versus the number of nodes for different probabilities of ordering items stocked by a remote warehouse in the new order transaction. We see that if the probability of remotely stocked items increases to 1.0, the scale-up decreases by about 44%. Note that even at a probability of remotely stocked items of 1.0, most of the accesses are still local since only 43% of the transactions are New-Order transactions, and of these only the ten stock tuples selected are remote; the warehouse, customer, district, and 10 item tuples selections are all local. The TPC-C benchmark favors distributed systems by having a very small percentage of remote calls.

6 Summary and Conclusion

In this paper we modelled the TPC-C benchmark for single node and multiple node distributed database systems. One key difference of the TPC-C benchmark, from the debit-credit benchmark of TPC-A, is that it includes significant skew (i.e., non-uniform access) within several key relations. By contrast, the TPC-A benchmark has uniform access within each relation, and in particular, each account in the large account relation is accessed with equal probability. As a consequence, in TPC-A each account tuple is accessed infrequently and it is not beneficial to hold them in a memory buffer. Therefore, one focus of this paper was to quantify the access skew in the TPC-C benchmark, and to examine it's impact on the optimal system configuration, price-performance and scalability.

To this end, we first quantified the tuple data access skew as specified in the benchmark. Consider the stock relation as an example for quantifying the access skew. At the tuple level we found that about 84% of the accesses go to about 20% of the hottest stock tuples. There is even more skew in the tail of the distribution, so that about 39% of the accesses go to about 2% of the (hottest) tuples. Since the database buffer is typically organized as pages, we next examined the skew at the page level. If tuples are inserted sequentially by key (or randomly) then hot tuples are scattered among the pages in the database. As a consequence, the skew at the page level is milder than that at the tuple level. Specifically, about 75% of the accesses go the hottest 20% of the pages. Again, there is a more skew in the tail of the distribution and and about 28% of the accesses go to about 2% of the pages. We then considered clustering the hot tuples into the same pages in an optimal manner. This is possible for the TPC-C benchmark because the access probabilities are static in time and known a-priori. If this were done, the resulting skew at the page level is about the same as that at the tuple level, in terms of the fraction of accesses that go to any specific fraction of data.

Having quantified the access skew, we examined the buffer hit ratio versus buffer size characteristic, assuming an LRU buffer replacement policy. We quantified this for each relation, both for the case of sequential assignment of tuples to pages and for that with hot tuples clustered within pages. Significant differences in the buffer hit ratio was found for these two cases. The specific hit ratios and the difference for the two cases differs for different relations. In absolute terms it is largest for the customer relation, but the higher frequency of access to the stock relation makes this relation dominant.

The results of the buffer model were fed to a throughput model to examine the overall throughput and optimal memory and disk configuration. The access skew makes the results rather different from that for the TPC-A benchmark where, as outlined above, buffering any of the account tuples is of little value. For the TPC-C case, almost all the item tuples, the hotter stock tuples, and some of the customer tuples are buffered in the estimated optimal configurations. The optimal configurations depend on the specific costs of disks and memory, specific estimates are given in Section 5.2.

We also found that depending on the disk bandwidth to disk storage capacity ratio, packing hot tuples into pages may result in significant benefits in terms of price-performance. We note, however, that this observation applies only to a workload where the access probabilities do not vary with time, and where they are known a-priori. In this sense, the TPC-C benchmark is not quite representative of many real workloads, where often neither of these conditions apply.

Finally, we examined the scalability of the TPC-C workload in terms of how the throughput can be expected to grow with the number of nodes in a distributed database system. Like the TPC-A benchmark, the TPC-C benchmark is largely partitionable, and close to linear scale-up in the number of nodes can be obtained. This assumes that the read-only item relation is replicated across all nodes, and that no remote communication is needed for concurrency control for access to this read-only relation. Specifically, if the Item relation is replicated, there are few remote calls in the workload. In the New-Order transaction on average 0.1 stock tuples accessed and updated are from a remote warehouse. Since the New-Order transaction selects 23 tuples these 0.1 remote calls comprise only 0.4% of the New-Order transaction workload. In the Payment transaction 0.33 (0.15×2.2) customer tuples accessed are from and updated are from a remote warehouse. Since the Payment transaction selects 4.2 tuples these 0.33 remote calls comprise only 7.9% of the Payment workload. The Order-Status, Delivery, and Stock-Level transactions access 11.4, 130, and 401 tuples respectively. Hence, once weighted by the percentage of the workload only 0.54% of the accesses are to remote data. This low fraction of remote access should be carefully considered when using the TPC-C benchmark to assess the performance of a distributed or clustered database system.

In a real environment, the item relations would be updated albeit infrequently, and provision would have to be made for this. If a general concurrency control protocol was used for this, e.g. the primary copy approach, or if the item relation is not replicated, then the scale-up as a function of the number of nodes is significantly lower, as we have quantified. Even so, the fraction of remote calls is rather small. While we have focussed on examining the TPC-C benchmark, the methodology we have used has more general applicability.

References

[1] Bernstein, P.A., and Goodman, N., "Concurrency Control in Distributed Database Systems," Computing Surveys, Vol. 13, No. 2, pp. 185-221, June 1981.

[2] Bernstein, P.A., and Goodman, N., "A Sophisticates Introduction to Distributed Database Concurrency Control," in Proc. 8th VLDB Conf., Sept. 1982, pp.62-76.

[3] Ciciani, B., Dias, D.M., and Yu, P.S., "Analysis of Replication in Distributed Database Systems," IEEE Trans. Knowledge and Data Engrg., Vol. 2, No. 2, June 1990, pp. 247-261.

[4] Dan, A., Yu, P.S, and Chung, J.Y., "Characterization of Database Access Skew of a Transaction Processing Environment,", IBM Research Report RC 17436, 1991.

[5] Dias, D.M., Iyer, B.R., Robinson, J.T. and Yu, P.S., "Integrated Concrurrency-Coherency Controls for Multisystem Data Sharing", IEEE Trans. Software Engrg., Vol. 15, No. 4, April 1989.

[6] Gray, J., (Editor), *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, 1991, isbn 1-55860-159-7.

[7] Kohler, W., Shah, A., Raab, F., "Overview of TPC Benchmark C: The Order-Entry Benchmark," technical report, Transaction Processing Performance Council, December 23, 1991.

[8] Leutenegger, S., Dias, D., "A Modeling Study of the TPC-C Benchmark," ICASE Report, number 93-12.

[9] Leutenneger, S., and Dias, D., "A Modeling Study of the TPC-C Benchmark," IBM Technical Report (in preparation).

[10] McNutt, B., "DASD Configuration Planning: Three Simple Checks", CMG Conference Proceedings, 1988.

[11] Transaction Processing Performance Council, "TPC Benchmark C, Standard Specification, Revision 1.0", Edited by Francois Raab, August 13, 1992.

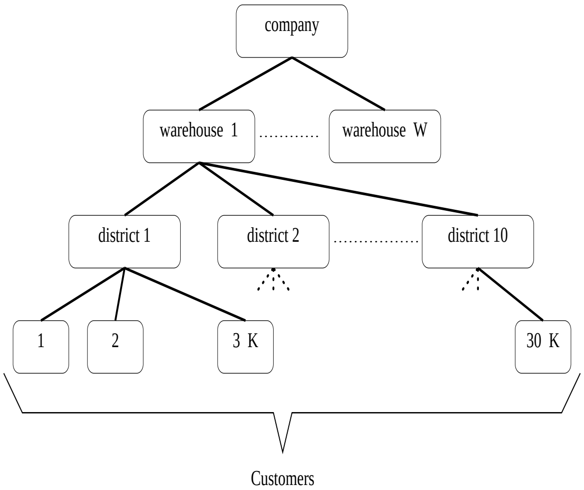

Figure 1: TPC-C Business Enviornment.
Reproduced with permission from the TPC

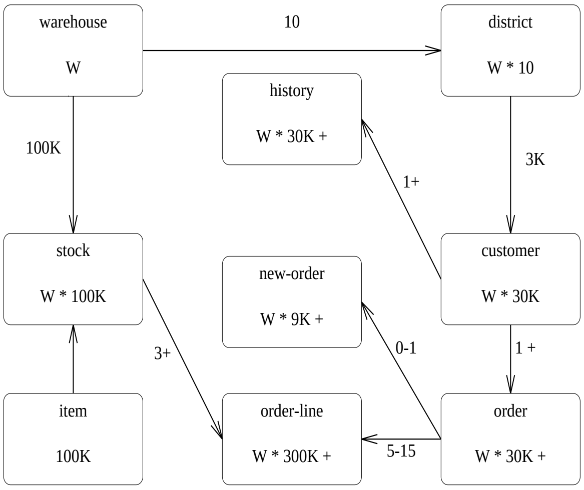

Figure 2: TPC-C Entity/Relationship Diagram.
Reproduced with permission from the TPC

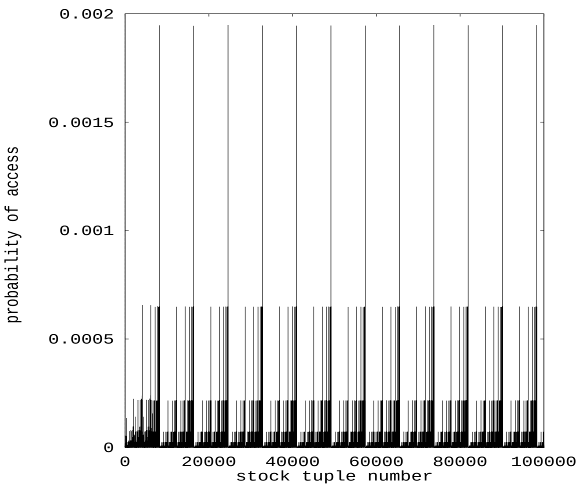

Figure 3: Stock Relation PMF

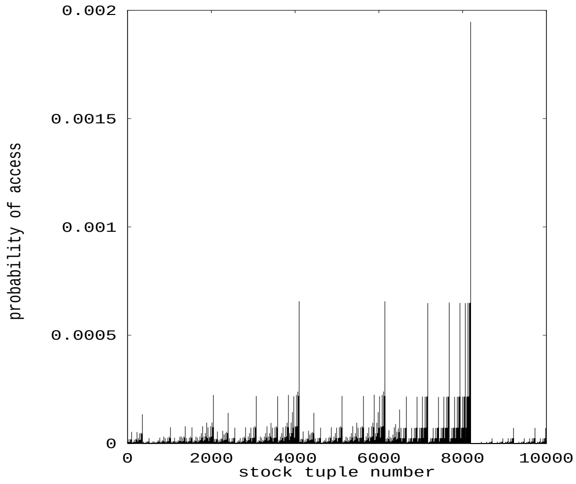

Figure 4: Stock Relation PMF: 10,000 tuples

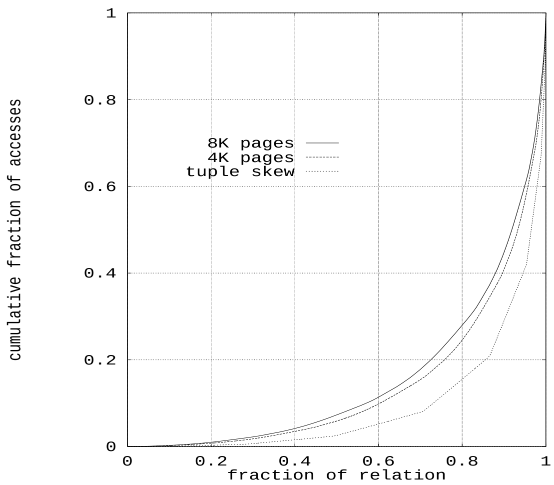

Figure 5: Stock Relation CDF

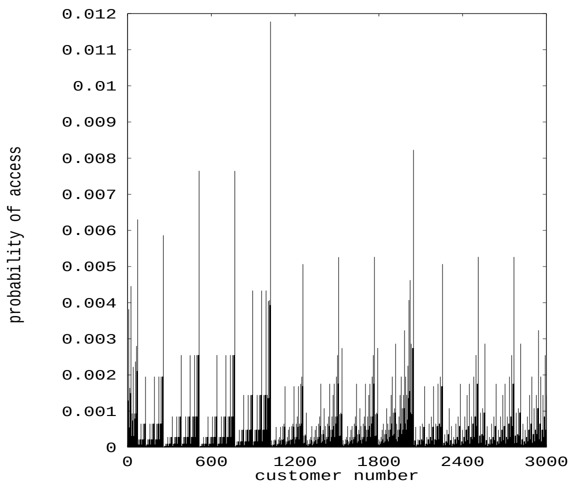

Figure 6: Customer Relation PMF

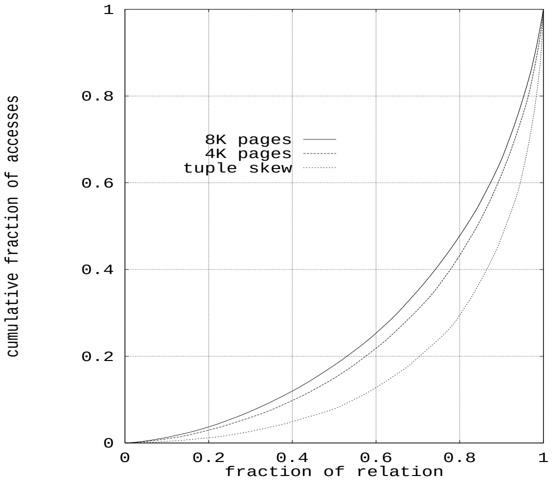

Figure 7: Customer Relation CDF

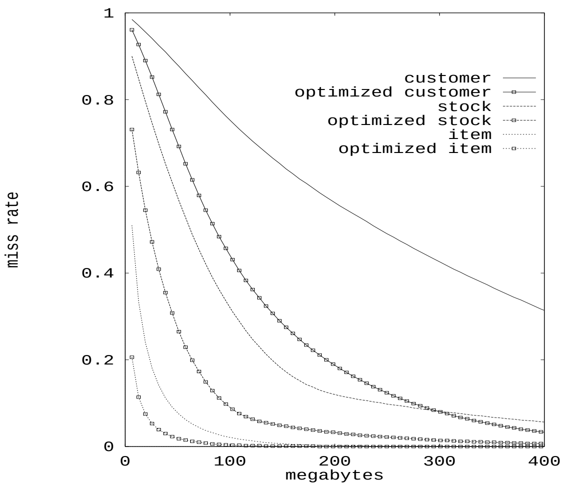

Figure 8: Significant Miss Rates

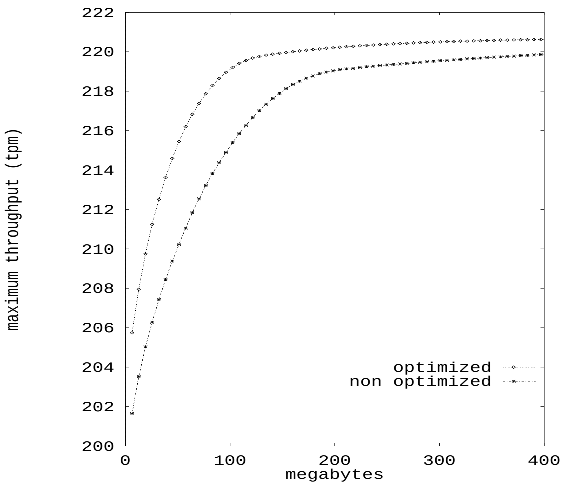

Figure 9: Maximum Throughput

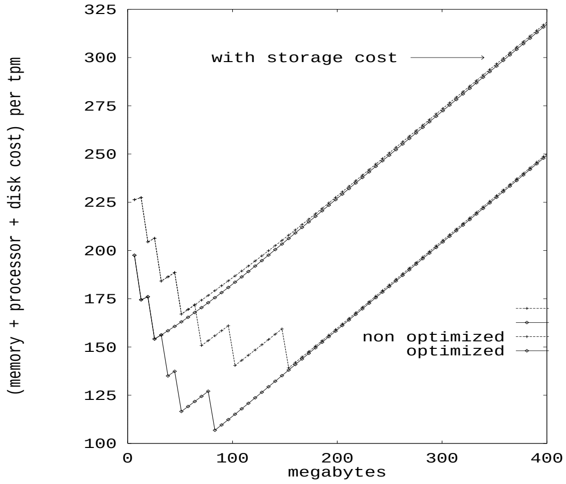

Figure 10: Price Performance

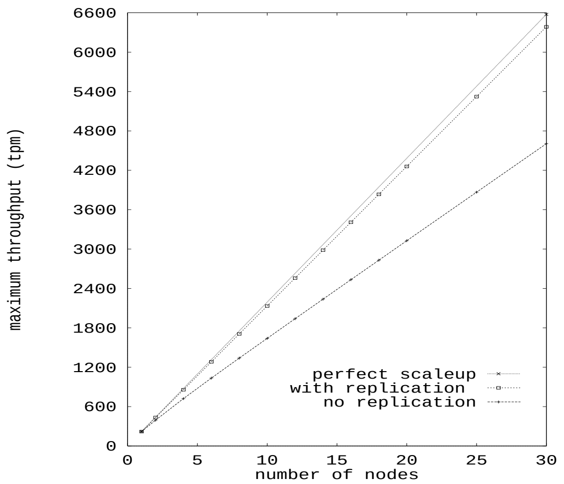

Figure 11: Scaleup of TPC-C

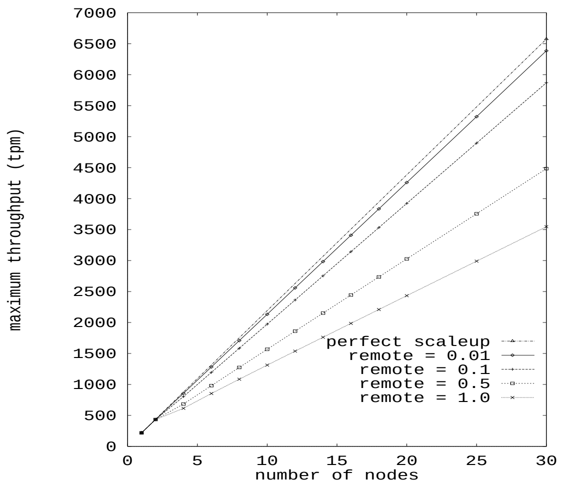

Figure 12: Sensitivity to Percent Remote