## AD-A264 793



#  

## A MODELING STUDY OF THE TPC-C BENCHMARK

## Scott T. Leutenegger <br> Daniel Dias



## This domment has been approved for public release and sole; its distribution is unlimited

NASA Contract Nos. NASI-19480 and NASI-IX605
March 1993
Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23681-0)(0)
Operated by the Universities Space Research Association


National Aeronautics and
Space Administration

# A Modeling Study of the TPC-C Benchmark 

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

Daniel Dias<br>IBM Research Division<br>T.J. Watson Research Center<br>P.O. Box 704<br>Yorktown Heights, NY 10598<br>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.




[^0]
## 1 Introduction

The TPC Benchmark C (TPC-C) $[7,9]$ is intended to model a medium complexity online transaction processing (OLTP) workload. It is patterned after an order-entry workload, with multiple transartion 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 assuring 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 [9]. 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 foction 3 we present simulation results
for the non-uniform random number generation routines to determine the degree of arcos shew. 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 dist ributed system are giren 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 the TPC S sperification [9] and 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 reprocuction of that found in the TPC-C benchmark specification [9]. The overall database consists of a number of warehouses. Each warehouse is composed of ten districts where each district has $3,000(3 \mathrm{~K})$ customers. There are 100 K items that are stocked by each warehouse. The stock level for cach 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 Warehonse, District, Customer, and Stock relations scale with the number of warehouses. This is similar to the TPC-A benchmark where the rardinality of the Branch, Teller, and Account relations scale with the number of branches. The Item relation

Table 1: Summary of Logical Database

| Relation <br> Name | Cardinality | Tuple <br> Length | Tuples Per <br> 4 K Page |
| :--- | :---: | :---: | :---: |
| warehouse | W | 89 bytes | 46 |
| district | $\mathrm{W}^{*} 10$ | 95 bytes | 43 |
| customer | $\mathrm{W}^{*} 30 \mathrm{~K}$ | 655 bytes | 6 |
| stock | $\mathrm{W}^{*} 100 \mathrm{~K}$ | 306 bytes | 13 |
| item | 100 K | 82 bytes | 49 |
| order |  | 24 bytes | 170 |
| new-order |  | 8 bytes | 512 |
| order-line |  | 54 bytes | 75 |
| history |  | 46 bytes | 89 |

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 |

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 TPCC as listed in table 2. Further details of the specific relations accessed and the access skew are given in Sections 2.2 and 3. The New Order transaction places an order for 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. When specified by customer-id, this transaction is of comparable complexity to the TPC-A transaction. 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 Levei 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-umique selects, and joins for a
relational model. There is a colume for minimum precout of workhad and a colnm for anmmed percent of workload. The benchmark specifies a mintmum perent for all the tranachion typerserph the New Order transaction. The benchmark metric is the number of New Order transations prowesent per minute, hence, it is desirable to set the percent New Order as high as posible ( $15 \%$ ) laking inll account that the size of the New-Order relation will grow without bound unkess the relative rate of Delivery transactions is sufficient to delete the entries in the New-Order relation at the same rat that the New-Order transaction inserts them. The third column in the table is the perornt of the workload mix that we have assumed for all studies in this paper. We have assumed the percent of Delivery transaction is $5 \%$ to ensure that the size of the New-Order relation remains small since our simulations must maintain the contents of the relation as the simution proceds. Note, the percout New-Order versus Delivery is a key parameter of this benchmark and should bo tuned carefully to achieve the maximum New-Order transactions per second. If the percent New-Order is $4 \%$ and the percent Delivery is $4 \%$ then the New-Order relation will grow without bound causing more misses on the New-Order relation $t n$ occur and a need for more storage. The join is an equi-join, where the two relations involved each have at most 200 tuples that meet the selection predicate. Further description of each of these transactions is found in section 2.2.

### 2.2 Transaction Access Patterns

In this section we summarize the access patterns for each database call of each transaction. For each transaction we first list how the random variables are generated, and then list the database operations made by that transaction in a simplified pseudocode. Although our pseudocode is not in SQL it succinctly conveys the function of each transaction. The TPC-C specification includes sample code [9] for each transaction. In the description of how the input data is generated many of the tuple-ids are generated from the NU() function. We define and simulate this function in section 3. for now just view this as a non-uniform distribution.

## New Order Transaction

This transactions places an order that consists of an average of 10 items. The input is generated as follows:

| whouse-id | uniform |
| :--- | :--- |
| dist-id | uniform |
| customer-id | $\mathrm{NU}(1023,1,3000)$ |
| number of items | uniform(5,10) |
| item-id | $\mathrm{NU}(8191,1,100000)$ |

The benchmark specifies that there are 10 districts per warehouse, and each district has one terminal. All transactions initiated by a terminal use that terminal's district and warehouse number. Since we are not explicitly modelling the terminals, we assume the whouse-id and dist-id are uniforml:
distributed. This assumption is reasonable since each terminal is submitting requests at the same rate.

Below we list the simplified format of the New-Orde: 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

In the benchmark, a district is associated with a specific warehouse, hence, the key used to uniquely identify a district tuple is composed of two fields: (dist-id, whouse-id). Similarly, the key used to uniquely identify a customer tuple is composed of three fields: (customer-id, dist-id, whouseid). In the benchmark the number of items ordered is uniformly distributed between 5 and 15 . We assume all transaction have a fixed number of items ordered equal to 10 . This assumption also has no effect on our results since we only report mean miss rates and throughputs. For each of the 10 items ordered, the supplying warehouse is the local warehouse $99 \%$ of the time and uniformly distributed among all the other warehouses $1 \%$ of the $t$ me. The implication of a having a remote warehouse involved is that the tuple retrieved from the stock relation may be on a different node if the warehouse is remote and the database is configured across a distributed system. We will assume that calls to remote warehouses located on the same node incur the same overhead as a call to the local warehouse. To uniquely identify a stock tuple the key has two fields: (item-id, whouse-id). A specific Stock tuple contains the number of that particular item in stock at that particular warehouse. In addition, the benchmark specifies that $1 \%$ of the transactions should be rolled back to simulate entry errors. We ignore this aspect.

## Payment Transaction

This transaction processes a payment by one of the customers. 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 mame. Due to the methoul periferd by the benchmark for the population of the database (eath district has 3000 customers but only 1000 names), on average three customers will have the same last name. the actual enstomer chemen is determined by selecting all customers with that name, sorting on the first mame and taking the middle one. To define the accesses to the relation we will assume that this non-umique select has the same overhead as 3 selects.

Regardless of the method used for selecting the customer, $15 \%$ of the transactions assume the customer is paying through a warehouse other than the customer's home warehouse. The input is generated as follows:

|  | whouse-id <br> dist-id | uniform |
| :--- | :--- | :--- |
| uniform |  |  |

Note that, in case two, the customer name is drawn from the NU funtion from lbound to ubound. We assume one of three (lbound, ubound) pairs are chosen with equal probability as (1,1000), (1001,2000), (2001,3000). In actuality there are 1000 unique names per district and the remaining 2000 names are uniformly drawn from these 1000 names. Hence, when a customer is specified by name on average three tuples satisfy the predicate and are distributed across the 3000 tuples in some manner similar to above. We have chosen the distribution above to keep the simulations simple. Below we list the SQL calls made by the transaction in a simplified format.

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

## Order Status Transaction

This transaction determines the status of a customer's last order, returning information about the customer, and a summary of the order. The customer is determined as in the Payment transaction, i.e. $60 \%$ of the time by name and $40 \%$ by customer id.

|  | whouse-id | uniform |
| :--- | :--- | :--- |
| dist-id | uniform |  |
| case 1: | customer-id | NU(1023,1,3000) |
| case 2 | customer-name | $\mathrm{NU}(255$, lbound,ubound) |

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" is the selection of the tuple in the Order relation that is the most recent order placed by the customer. This could be implemented as a max aggregate, or an order by descending order-id and return only the first tuple. Since the Order relation keeps on growing without bound, both of these approaches could 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

This transaction processes a delivery. The transaction assumes that during a delivery the oldest order not yet delivered for each district within a warehouse is processed. Hence, there are really 10 deliveries per delivery transaction. The benchmark specifies that this transaction has less stringent rasponse time constraints and can be executed in batch mode, i.e. deferred execution. The only input to the transaction is the whouse-id which is uniformly distributed. The transaction proceeds as follows:

1. For each district within the warchouse (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
 the tuple in the New. Order relation that is the oldent order for that diathet and wathon wh the New-Order relation. As in the Max seder in the Order Statustranaction, thin whd be implament using a multi-keyed index so that the correct thple can be fetched in just mo call. The cuthome in used in the Select from Customer is obtained from the tuple in the Order retation

## Stock Level Transaction

This transaction determines the number of tems sold by orders from the bant 20 order of a specific district that bave a stock level below a rertain threshold. The inf ats ase the dith. whim is uniformly distributed, and the threshold. Below we quote the sample SQL code directy from the tpec document [9] so that we do not confuse the query by oversimplification.

```
SELECT d_next_oid INTO :o.id FROM District WHERE d_wid \(=:\) wid AND d_id \(=:\) did :
```

SELECT COUNT(DISTINCT (s.idd)) INTO :stock_count

## FROM Order-Line, Stock

WHERE
olwid $=$.w_id AND
olddid $=$ :did AND oLoid $<$ :oid AND
oLo_id $\geq$ (:oid - 20) s_w.id $=$ :wid AND
s_ieid $=$ oLi_id AND s-quantity $<$ :threshold ;
In the above query, oldid specifies the dist-id attribute of a tuple in the orderline relation, wid is the order-id attribute, i_id is the item attribute, and wid is the warehouse attributr. The first select acquires the current order number for the district and places it in the variable :oid. which ctands for order-id. Having obtained the current order-id for that district, the query computes a juin of the Order-Line and Stock relations to find the number of distinct items ordered in the distriet: last 20 orders which have a stock quantity below the specified threshold.

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

To summarize the access patterns of the five transaction we list the number of accenses to wath relation for each transaction type and the average number of accesses per transartion in Tablle $3:$ the latter assumes the percentages for each transaction listed in Table 2 . Within the tahle. the butation $U(x)$ signifies that $x$ tuples are chosen Uniformly from the relation, NV(r) dmoto NonUuiform random selection of $x$ tuples using the NE function. A(x) demotes $x$ tuples are Appetuded the the relation, and $P(x)$ denotes $x$ tuples are chosen where the tuples chosen were recouly accomen by Past behavior (in other words there is a form of temporal locality). Note that the tuplen acressed

Table 3: Summary of Relation Accesses

| Relation | $\begin{aligned} & \text { New } \\ & \text { Crder } \end{aligned}$ | Payment | Order <br> Status | Delivery | Stork <br> Level | Average |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| warehous* | U(1) | U(1) |  |  |  | 0.87 |
| district | (1) | (1) |  |  | $P(1)$ | 0.93 |
| rustomer | N(1) | NU(2.2) | NE(2.2) | $\mathrm{P}(10)$ |  | 1.524 |
| stock | NU(10) |  |  |  | $P(200)$ | 12.4 |
| item | NU(10) |  |  |  |  | 4.4 |
| order | A(1) |  | P (1) | $\mathrm{P}(10)$ |  | 0.53 |
| new-order | A(1) |  |  | $\mathrm{P}(10)$ |  | 0.49 |
| order-line | A(10) |  | $\mathrm{P}^{\prime}(10)$ | $\mathrm{P}(100)$ | $\mathrm{P}(200)$ | 13.3 |
| history |  | A(1) |  |  |  | 0.43 |

by the Orler-Status, Delivery, and Stock-Level transactions are more likely to be buffre 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 Data Access Skew in TPC-C

The TPC-(' 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 TPCC documents, used for the generation of tuple id’s. The non-uniform random number generating function, NU(), which we paraphrase from the benchmark specification [9]. is defined as follows:

$$
\begin{equation*}
N(1(A, x, y)=(((\operatorname{rand}(0, A) \mid \operatorname{rand}(x, y))+(\cdot) \%(y-x))+x \tag{1}
\end{equation*}
$$

whera:


- ( C is a ronstant within [0..A].
- A is a constant chosen according to the size of the range [ $x . . y$ ],
- (N M M) stands for N modulo M.
- and $(N \mid M)$ stands for the bitwise logical OR of $N$ and $M$.

For the remainder of this paper we assume (' equat zero (the TP'C sabdard dewmemt allow an arbitrary cheice of ( within $[0 \ldots 1])$. We choose $A$ and $\because$ according the the serification for the ruple id being generated.

First we comsider acresses to the stock and item relations. All tuple id's for acceming then relations are drawn from the $\mathrm{NC}(8191,1,100000)$ distribution. In Figure 3 we plot the probability mass function (PAF) for this distribution as ohtained from simulating our billion samples. The plot shows the mon-uniformity in access and the periodicity of the arcess probability in the first parameter ( $\times 191$ ) of the NL function above. The number of rycles equals the (floor of the) third paraneter divided by the first parameter of the NU function, or 12 cycles for this case. In the dppendix we show that if the third parameter of the NU function is a power of two, then these cyrles are exact. and we derive a closed form expression for the resulting PMF. Figure 3 is hard to interpret because of the large number ( $100, \hat{v} \hat{0}$ ) 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. Lat ( $x_{i}$ be the probability of accessing tuple $i$. Let $\beta_{i}$ be the fraction of the relation represented by that tuple. Note $b_{1}=\beta_{j} \forall i, j$ for stock tuples. In Figure 5 we order the tuples by increasing order of $\alpha$ (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 Irvel. We first assume tuples are parked 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 $4 \mathrm{~K}(8 \mathrm{~K})$ page.

Again, we order the pages by frequency of access and plot the cumulative probability of acress versus the cumulative fraction of the database in Figure 5 (top two curves). The top curve is tor an \$KByte page size and the second curve is for a 4 KByte page size. For a 4 KByte 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 gis $1020 \%$ of the data. Again, there is a more skew in the tail of the distribution and about $59 \%$ of the arcesses go to about $10 \%$ of the hottest pages, and abont $28 \%$ 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 chane 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 cat he obtained at the page level. Packing tuples into pages in sequential order spreads out her tuplo. among all the pages of the relation. A simple optimization is to first sort the tuples from hothon th coldest and then pack them into pages in that order. Since the distribution parameters for TPC © are know a priori and are static in time, this could be done. (In this context we note that the TP( $C$ 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 databare. and where the distribution does not vary with time. (We note, however, that m many real workmods. 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 parking 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 age 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 $4 \mathrm{~K}(8 \mathrm{~K})$ 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 arcess patterns which are superimposed upon the relation. If the customer-id is used as the selection key. one tuple is selected from the $\operatorname{NU}(1023,1,3000)$ distribution. If the customer-name is used, we make the simplifying assumption that the customer name is selected fron one of the $\operatorname{NU}(255,1,1000)$, $\mathrm{NU}(255,1001,2000)$ and $\mathrm{NU}(255,2001,3000)$ 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 10 the customer relation use the $\mathrm{NU}(1023,1,3000)$ distribution and $58.14 \%$ are divided equally among $\mathrm{NU}(255,1,1000), \mathrm{NU}(255,1001,2000)$, and $\mathrm{NU}(2001,3000)$ 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 considerauly 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 simulatod the buffer nool for the TP(C-C benchmark assuming an LRU replacement poliry. We hypothesize that more sophisticated replarement poliries conld result in an even larger differmen between optimized parking 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 confidener intervals nsing bateh 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 $90 \%$ ronfidence level.
In the bufler model, we simulate transactions entering the system sequentially and do but onsider the case where multiple transactions may be in the system at the same time. The prewence of concurrent transactions does not change the buffer hit ratio significantly becanse the fractinn of page accessed by any transaction is small compared to the buffer size. We include concurront tianactions: in the throughput model in Section 5.1. When a transaction enters it is chosen ats ohe of the five typer according to the distribution for each type. Each transaction generates tuple requests and inserts as specified in Secion 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 verd by the the Order-Status, Delivery, and Stock-Level transactions. The ontput 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 lowe. 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 exhibits a larger miss rate than the Stock relation even though the ( $u$ umomer relation is the smaller of the two. The first is that the customer relation has loss skow as show 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 mumber of warelonmes.

The optimal packing approach results in significantly lower miss rates than the sequental parking approach. For example, the misis rate for the stork relation for a buffer size of $52 . \pm$ is $30 \%$ lower in absolute terms for the optimized packing approach than for the sequential approth. The miss rate for the steck relation averaged over all buffer sizes considered is $13 \%$ lower in aboblute torms for the optimized packing approarh than for the sequential approach. This significantly hour mise ratre
 for the c'ustomer relation miss rates and to a lesser extent for the Item relation.

Wr assume 20 Warehouses at a node. The reason for choosing the case of 20 Warehouson ratres to the throughput model in Section G, where it is estimated that about 20 Warehomes combl be supported by a 10 MIPS processor. Begond a sufficiently large mumber of warthoures the bufor hit chararteristies approximately scale with the number of Warehouses. The reason that the acaling is not exart is that the item relation does not scale with the umber of Warohouses, but it affor diminishes with an increase in the number of Warehomses. The Warehouse and Distriet relations are sufficiently small that they fit in the buffer (miss rate 0\%) for all simations comsidered.

Table 4: Throughput Model Summary : Single Node

| resource | parameter | n | overhear | $\begin{gathered} \text { NewOrder } \\ V_{1} \end{gathered}$ | $\begin{gathered} \text { Payment } \\ V_{2} \end{gathered}$ | $\begin{gathered} \text { Status } \\ b_{3}^{\prime} \end{gathered}$ | Delivery 14 | $\begin{gathered} \operatorname{sink} \\ 1 \% \end{gathered}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| CPU | select | 1 | 20 K | 23 | 4.2 | 13.2 | 1.31 | 1 |
| CPU | update | 2 | 20 k | 11 | 3 | 0 | 120 | 0 |
| CP! | insert | 3 | 20 K | 12 | 1 | 0 | 0 | 1 |
| CPU | delete | 4 | 20 K | 0 | 0 | 0 | 10 | 0 |
| CPT | commit | 5 | $40 \mathrm{~K}^{1}$ | 1 | 1 | 1 | 1 | 1 |
| (PL | inito | 6 | 5 K | $\begin{gathered} 1+m i \\ +10(m i+m s) \end{gathered}$ | 1+2.2(me) | $\begin{gathered} 2.2(\mathrm{mc}) \\ +\mathrm{mot}+10(\mathrm{ml}) \end{gathered}$ | $\begin{gathered} 1+10(m 14+m c+m n) \\ +330(m) \end{gathered}$ | $200(90+m i)$ |
| CIU | application | 7 | 0.1 k | 47 | 8 | 13 | 261 | 3 |
| CPU | send/receive | 8 | 15 K | 0 | 0 | 0 | 0 | 0 |
| CPU | prepConmit | 9 | 40K | 0 | 0 | 0 | 0 | $1)$ |
| CP! | initTransaction | 10 | 50K | 1 | 1 | 1 | 1 | 1 |
| CPU | releaseLocks | 11 | 35 K | 1 | 1 | 1 | 1 | 1 |
| CPU | non-unique-select | 12 | 50 K | 0 | 0.6 | 0.6 | 0 | 9 |
| CPU | join | 13 | 2000K | 0 | 0 | 0 | 0 | 1 |
| disk | 10 | 14 | 25 ms | $m c+10(m i+m s)$ | $2.2(\mathrm{mc})$ | $\begin{gathered} 2.2(\mathrm{mc}) \\ +\mathrm{mo}+10(\mathrm{md}) \end{gathered}$ | $\begin{gathered} 10(m \mathrm{~m}+\mathrm{mot}+\mathrm{mn}) \\ +130(\mathrm{ml}) \end{gathered}$ | $200(m s+161)$ |

## 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 priceperformance 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 reftain operation per tratsaction type. The visit counts are in the columns heading $V_{1} \ldots V_{s}$. 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 exeptions. The parameter application is for application code between SQl, calls, the parameter send/ureve is for the CPU overhead at one node to send and receive a mesage actoss the network.
the parameter release Locks is for the release lock portion of the commit phase. prepe 'ommut in for the prepare to commit portion of a 2 phase commit. and inill) is the ('ll werhead fur initiating an $1 / O$. The overhead for releasing locks is obtained by smming the overhead wreane read lockand 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 lk instrutions far releasing each lock.

The parameters $m c, m i, m s, m o$, and $m b$ 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 conld 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 items. We assume the overhead for the range scan is 5 K per tuple, the overhead for the indexed select is 5 K instructions per tuple, and the overhead for the final sort is 40 K resulting in a total CPU overhead of 2040 K 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 NewOrder 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. The values for these terms are derived in Appendix 1.

We first exp'ain the terms when the Item relation is replicated, i.e. table 6 . In this case all accesses to the Item relation are local because the relation is accessed read only. We assume that the distributed Concurrency Control (CC) protocol allows retention of read locks across transactions. and only requires a broadcast/semicast when acquiring an exclusive lock. ${ }^{2}$

[^1]Table 5: Definition of Notation

| symbol | meaning |
| :--- | :--- |
| $R C_{\text {stock }}$ | expected number of calls for obtaining and updating stock tuples |
| $R C_{c u s t}$ | expected number of calls for obtaining and updating customer tuples |
| $R C_{\text {item }}$ | expected number of calls for obtaining and updating item tuples |
| $U_{\text {stock }}$ | expected number of unique remote sites that supply stock tuples |
| $U_{\text {cust }}$ | expected number of unique remote sites that supply customer tuples |
| $U_{\text {item }}$ | expected number of unique remote sites that supply item tuples |
| $U_{\text {item }+ \text { stock }}$ | expected number of unique remote sites that supply item or stock tuples |
| $L_{\text {stock }}$ | probability that all stock tuples are supplied from the local warehouse |

The visit counts for four parameters change: commit, send/receive, prepCommit, and initIO. Although portions of these overheads actually occur at the other nodes. all the other nodes will be using the modeled node for remote calls, so by symmetry we can sum the overhead at the modeled node.

We first consider the NewOrder transaction. The only remote calls are for retrieving and updating stock tuples. The number of remote nodes involved in a 2 phase commit is $U_{\text {stock }}$. The visit counts for commit and initIO are each increased by $U_{\text {stock }}$ since a commit must be done at each node involved. The count for prepCommit is changed from zero to $U_{\text {stock }}+1-L_{\text {stock }}$ since the prepare portion of the two phase commit must be done at every site plus the coordinator minus the probability that the transaction is purely local. The count for send/receive is change from zero to $4 U_{\text {stock }}+2 R C_{\text {stock }}$ since we assume 2 round trip messages must be sent to each unique remote node involved in the 2 phase commit, and one round trip message for each remote call for retrieving or updating a stock tuple. Note the multiplier is 4 for $U_{\text {stock }}$ ( 2 for $R C_{\text {stock }}$ ) not 2 (1) since we model the overhead at all nodes involved on the coordinator by symmetry arguments.

For the Payment transaction the only remote calls are for obtaining and updating customer tuples. The number of unique remote sites involved in a two phase commit for the Payment transaction is $U_{\text {cust }}$. The new visit counts for the payment transaction are found in table 6 and are expressed in terms of the expectations expressed above.

We now explain the terms when the Itrm relation is not replicated, i.e. table 7. The visit counts for the Payment transaction are the same as for the replicated case since the Payment transaction does not access the Item relation. The visit counts for the NewOrder transaction differ since the 10 retrievals of the item tuples may require a remote call in addition to the remote calls for stock tuples. The item tuples are accessed read only, hence a 2 phase commit is need only for those nodes supplying a stock tuple. The number of remote nodes involved in the 2 phase commit is $U_{\text {stock }}$. Thus, the visit counts for initIO and prepCommit are the same as when the item relation is replicated. A 1 phase commit is necessary at each node that supplies an item tuple but no stock tuples. Hence, the number of nodes involved in a 1 phase commit is $U_{\text {item }}=U_{\text {stock+item }}-U_{\text {stork }}$. Relative to the replicated case.

[^2]Table 6: Throughput Model Summary : Multi Node with Replication

| resource | parameter | n | overhead | NewOrder <br> $V_{1}$ | Payment <br> $V_{2}$ |
| :---: | :---: | :---: | :---: | :---: | :---: |
| CPU | commit | 5 | 30 K | $1+U_{\text {stock }}$ | $1+U_{\text {sust }}$ |
| CPU | initlO | 6 | 5 K | $1+m \mathrm{~m}$ <br> $+10(m i+m s)$ <br> $+U_{\text {stock }}$ | $1+2.2 \mathrm{mc}$ <br> $+U_{\text {cust }}$ <br> $+U_{\text {cust }}$ |
| CPU | send/receive | 8 | 10 K | $4 U_{\text {stock }}$ <br> $+2 R C_{s t o c k}$ | $2 R C_{\text {cus: }}$ <br> $+4 U_{\text {cust }}$ |
| CPU | prepCommit | 9 | 15 K | $U_{\text {stock }}+1$ <br> $-L_{s t o c k}$ | $U_{\text {cust }}$ |

the visit count for send/receive is increased by $R C_{\text {item }}$ for obtaining item tuples and by $2 U_{i t e m}$ for the round trip message necessary for each node that participates in a one phase commit. The visit count for commit is changed to include commit overhead at all remote nodes $U_{\text {stock }+i t e n}$, whether they be involved in a 1 phase or 2 phase commit.

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 ltem relation replicated, or a distributed system without replication of the Item relation. Let $\lambda$ equal the system throughput and $\alpha_{i}$ denote the fraction of the workload from transactions of type $i$. The utilization of the CPU is calculated as:

$$
\begin{equation*}
U t i l_{C P U}=\frac{\lambda\left(\sum_{i=1}^{i=5} \sum_{n=1}^{n=13} \alpha_{i} \cdot V_{i, n} \cdot o_{n}\right)}{M I P S} \tag{2}
\end{equation*}
$$

Let $D A=$ the number of disk arms. The utilization of the disk is calculated as:

$$
\begin{equation*}
U t i l_{d i s k}=\lambda\left(\frac{\sum_{i=1}^{i=5} \alpha_{i} \cdot V_{i, 14} \cdot o_{14}}{D A}\right) \tag{3}
\end{equation*}
$$

Table 7: Throughput Model Summary : Multi Node No Replication

| resource | parameter | $n$ | overhead | NewOrder <br> $V_{1}$ |
| :---: | :---: | :---: | :---: | :---: |
| CPU | commit | 5 | 30 K | $1+U_{s t o c k+i t e m}$ |
| CPU | initIO | 6 | 5 K | $1+m c+10(m i+m s)$ <br> $+U_{s t o c k}$ |
| CPU | send/receive | 8 | 10 K | $2 R C_{\text {stock }}+2 R C_{\text {item }}$ <br> $+4 U_{\text {stock }}+2 U_{\text {item }}$ |
| CPU | prepCommit | 9 | 15 K | $U_{\text {stock }}+1-L_{\text {stock }}$ |

### 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 [8]). 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 4 K 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 procosor .
 software cost. maintenance cost. terminal cost, the. The intent is to stimate the aptimal dutabuw, memory buffer size in the trade-off betwecn memory and disks. The storage cost is conseated b: summing the storage needs for the Warehousn, District, Customer, Stock, and Item relations as specified in table 1. Assuming 20 warehouses per node (which leads to about No\% (Pl utilization). the space required is 1.1 Gbytes. In addition, we must include sufficient storage for running the benchmark for 1808 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 arriw 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 (we will refer to this as optimal packing). 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 $\$ 139 / \mathrm{tpm}$ for sequential packing, and at 84 Mbyte with a value of about $8107 / \mathrm{tpm}$ for the optimal parking case. Thus, the optimized packing of tuples results in about a $30 \%$ improvement of price performance relative to sequential parking.

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 $\$ 167 / \mathrm{t}$ pm for sequential packing, and at 26 Mbytes with a value of about $\$ 154 /$ tpm for the optimal packing case. Thus, the optimized parking of tuples results in about an $8 \%$ improvement of price performance relative to segiential 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 $\$ 50006$ Gbyte disk is assumed the cost/performance improvement resulting form optimal packing is $20 \%$. If a 12 Gbyte 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 tugether will reduce the percent difference significantly.

### 5.3 Multiple Node System Estimates

In this section we prescut our resuits 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 c that node. We consider two cases. The first case is when the item relation is replicated across all stes. Since the item relation is readonly, 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 vstem performance. We shonld emphasize that this assumes the use of a concurrency protocol (CC) which only requires remote access only when acquiring exclusim 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-np, 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 stecled $\mathrm{b}_{j}$ a remote warehouse in the new order transartion. We see that if the probability of remotely stocked items increases to 1.0 the scale-up decreasmb by about $44 \%$. Note that even at a probability of remotely stocked items of 1.0 , most of the accosse. 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 selection. 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 distribnted 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 TP(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 arcess skew as specified in the benchmark. Consider the stock relation as an example for quantifying the access skew. At the tuple level we fouth 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 areesses go to about $2 \%$ of the (hottent) tupkes. 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 scatiered among
the pages in the database. As a conscquence, 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 ho uples 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 term ${ }^{c}$ 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 .!.is, 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 replirated across all nodes, and that no remote communcation is neded for concurency control for acress 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 aressed and uphated are from a remote warehouse. Since the New-Order transaction selects 2.3 tuples these 0.1 remote ralls comprise only $0.4 \%$ of the New-Order transaction workload. In the Payment tranaction 0.33 $(0.15 \times 2.2)$ customer tuples accessed are from and updated are from a remote warehonse. Since the Payment transaction selects 4.2 tuples these 0.33 remote calls comprise only $7.3 \%$ of the Payment workload. The Order-Status, Delivery, and Stock-Level transartions access 11.4, 130, and 101 inples
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 nsing the TPC-C benchmaik to assess the performance of a distributed or clustered database systm.

In a real environment, the item relations would be updated albeit infrequently, and prowinion 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 Systoms." 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 System. 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] McNutt, B., "DASD Configuration Planning: Three Simple Checks", CMC Confernuce Proceed ings, 1988.
[9] Transaction Processing Performance Council, "TPC Benchmark C, Standard Specification. Revision 1.0", Edited by Francois Raab, August 13, 1992.

## Appendix A Derivation of probabilities for throughput model

In this appendix we derive the expected number of remote requests and unique sites involved for a distributed system. These terms are used in Section 5.

We first derive the probabilities assuming the Item relation is replicated and then derise the probabilities assuming no rephication.

## Appendix A. 1 Item relation is replicated

When the item relation is replicated requests for item tuples are always local. The only remote accesses possibly needed are for stock tuples by the NewOrder transaction and for customer tuples by the Payment transaction. We first consider the NewOrder transaction. The NewOrder transaction requests 10 stock tuples, each tuple belonging to a remote warehouse with probability 0.01 as specified by the benchmark. Assume there are $N$ nodes in the system. Let $P\left[S_{j}\right]$ be the probability that $j$ of the 10 stock tuples accessed are remote.

$$
\begin{equation*}
P\left[S_{J}\right]=\binom{10}{j}\left(P_{S}\right)^{j}\left(1-P_{S}\right)^{10-j} \tag{4}
\end{equation*}
$$

where $P_{S}=0.01\left(\frac{N-1}{N}\right)$, and $N$ is the number of nodes in the system. The term 0.01 is the probability that an individual stock tuple is from a remote warehouse, and $\frac{N-1}{N}$ is the probability that the remote warehouse is located on a remote node. We make the simplifying assumption that requests to remote warehouses located on the same node require the same overhead as a local request.

Let $E\left[R_{s}\right]$ be the expected number of remote stock tuples retrieved made by the New-Order transartion.

$$
\begin{equation*}
E\left[R_{s}\right]=\sum_{j=0}^{10} j P\left[S_{j}\right] \tag{5}
\end{equation*}
$$

Each tuple retrieved is also update, hence the expected total number of remote calls by the New Order transaction for reading and writing stock tuples is

$$
\begin{equation*}
R C_{\text {stock }}=2 \times E\left[R_{s}\right] \tag{6}
\end{equation*}
$$

Let $L_{\text {stock }}$ be the probability that all stock tuples are referenerd locally.

$$
\begin{equation*}
L_{\text {stock }}=\left(1-P_{S}\right)^{10} \tag{i}
\end{equation*}
$$

The number of remote sites involved in the transaction is the number of unique sites from which stock tuples are obtained. We derive this expectation, $l$ stork in the following theorem.

## Theorem:

$$
U_{\text {stock }}=\sum_{j=0}^{10} P\left[S_{j}\right](N-1)\left[1-\left(\frac{N-2}{N-1}\right)^{j}\right]
$$

## Proof:

Assume the system has $N$ rodes, and that a site generates $j$ remote requests. With out loss of generality, assume the originating site to be node 1 .

Let $J_{i}, i \in(2 \ldots N)$ be an indicator variable for the event that node $i$ supplies at least one tuple. A remote request is satisfied by one of the $N-1$ nodes with equal probability, hence the probability that a node supplies at least one tuple (the probability that the indicatcr variable is 1 ) is

$$
1-\left(\frac{N-2}{N-1}\right)^{j}
$$

The expected number of unique sites supplying tuples is

$$
\sum_{i=2}^{N} I_{i}=(N-1)\left[1-\left(\frac{N-2}{N-1}\right)^{j}\right]
$$

Unconditioning on the number of remote requests, $j$, results in the expected number of unique sites:

$$
U_{\text {stock }}=\sum_{j=0}^{10} P\left[S_{j}\right](N-1)\left[1-\left(\frac{N-2}{N-1}\right)^{j}\right]
$$

We now derive the expectations for the Payment transaction. The only remote accesses are for tuples from the customer relation. The customer is from a remote warehouse with probability 0,15 . The customer is selected based on customer-id $40 \%$ of the time (hence one tuple is selected), at d based on customer-name $60 \%$ of the time (hence three tuples are selected). In addition. once the tuple has been selected the update must be written back to the remote node. Hence, the expected number of remote calls for obtaining and updating customer tuples, $R C_{\text {cust }}$, is:

$$
R C_{c u s t}=0.15\left(\frac{N-1}{N}\right)[(0.4)(1)+(0.6)(3)+1]
$$

At most one remote site may be involved and hence the experted mumber of unique remote sites from which customer tuples are obtained, $U_{\text {cust }}$, is:

$$
\begin{equation*}
U_{c o s t}=0.15\left(\frac{N-1}{N}\right) \tag{9}
\end{equation*}
$$

## Appendix A. 2 Item relation not replicated

We now derive the expectations assuming the item relation is not replicated. The expectations for the Payment transaction are the same as for the replicated case siner the Payment transaction does not access the Item relation.

For the NewOrder transaction the number of remote calls for stock tuples, $R C$ 'suck expected number of unique sites supplyin, stock tuples, $U_{\text {stock }}$, and the probability that all stock tuples are supplied locally are the same as when the item relation is replicated. The differnene from the replicated case is that the 10 item tuples retrieved may be remote since we assume the item relation is uniformly distributed among the $N$ nodes.

Let $f\left[I_{\mu}\right]$ be the probability that $j$ of the 10 item tuples accesses are remote.

$$
\begin{equation*}
P\left[I_{j}\right]=\binom{10}{j}\left(P_{I}\right)^{j}\left(1-P_{I}\right)^{10-3} \tag{10}
\end{equation*}
$$

where $P_{I}=\frac{N-1}{N}$ is the probability that an item iuple is located on a remote node, and $N$ is the number of nodes in the system. Let $E\left[R_{I}\right]$ be the expected number of remote item tuples retrieved made by the New-Order transaction.

$$
\begin{equation*}
E\left[R_{l}\right]=\sum_{j=0}^{10} j P\left[I_{j}\right] \tag{11}
\end{equation*}
$$

The number of remote calls for item tuples, $R C_{\text {item }}$, is equal to $E\left[R_{I}\right]$ since the tuples are not updated. Let $U_{\text {item }}$ be the expected number of unique remote sites involved for fetching the remote item tuples. This expectation is derived as in theorem 1.

$$
\begin{equation*}
U_{i t e m}=\sum_{j=0}^{10} P\left[I_{j}\right](N-1)\left[1-\left(\frac{N-2}{N-1}\right)^{J}\right] \tag{12}
\end{equation*}
$$

In addition to $U_{\text {stock }}$ and $U_{\text {item }}$ we need the expected total number of unigur ardes referenced the NewOrder transaction, $U_{\text {stocktitem }}$. The expected number of unique sites given $j$ stork tuple requests and $k$ item tuple requests is equal to

$$
(N-1)\left[1-\left(\frac{\dot{-}-\frac{2}{N}}{-1}\right)^{1+k}\right] .
$$

Hence, upon unconditioning on $j$ and $k$,

$$
\begin{equation*}
U_{\text {stock+item }}=\sum_{j=0}^{10} \sum_{k=0}^{10} P\left[I_{j}\right] P\left[S_{k}\right](N-1)\left[1-\left(\frac{N-2}{N-1}\right)^{1+k}\right] \tag{1:3}
\end{equation*}
$$

## Appendix A. 3 Proof of Perodicity of the NURand Function

In this appendix we show that the NURand $(x, 0, y)$ function is peridoc if both $x$ and $y$ are a power of $2, x \leq y$. Although the function is not exactly peridoc when $y$ is not a power of 2 . we have observed it to be close to periodic.

$$
\begin{equation*}
N U \operatorname{Rand}(x, 0, y)=((\operatorname{random}(0, x) \mid \operatorname{random}(0, y)) \notin y) \tag{1.4}
\end{equation*}
$$

where
random( $\mathrm{x}, \mathrm{y}$ ) denotes a uniformly distributed integer random number in the closed interval [ $\mathrm{x} . . \mathrm{y}$ ]
( $\mathrm{N} \% \mathrm{M}$ ) stands for N modulo M
( $N \mid M$ ) stands for the bitwise logical $O R$ of $N$ and $M$
Let $x=2^{a}-1$ and $y=2^{b}-1, b \geq a$
Let $z=b-x$,
Let $A=A_{b-1} A_{b-2} \ldots A_{0}$ be the binary representation of the number drawn from random( $0, \mathrm{x}$ ).
Let $B=B_{l-1} B_{b-2} \ldots B_{0}$ be the binary representation of the number drawn from random( $0 . y$ ).
Note that if $X>0$, then the top $z$ bits of $A$ will all be zero.
Let $P\left[A_{i}\right]$ denote the probability that bit $A_{i}$ is set to one. Then,

$$
\begin{aligned}
& P\left[A_{i}\right]=\frac{1}{2}, i \in(0,1, \ldots(a-1)) \\
& P\left[A_{i}\right]=0, i \geq a \\
& P\left[B_{i}\right]=\frac{1}{2}, i \in(0,1, \ldots(b-1))
\end{aligned}
$$

Let $C=A \mid B=C_{b-1} C_{b-2} \ldots C_{0}$. Sinee $A_{i}$ and $B_{2}$ are independent for all $i$, bit $C_{2}$ is set if either $A_{1}$ or $B_{t}$ or both are set. Hence,

$$
\begin{aligned}
& P\left[C_{i}\right]=\left(P\left[A_{i}\right] * P\left[B_{i}\right]\right)+\left(\left(1-P\left[A_{i}\right]\right) * P\left[B_{i}\right]\right)+\left(P\left[A_{i}\right] *\left(1-P\left[B_{i}\right]\right)\right) \\
& P\left[C_{i}^{\prime}\right]=\frac{3}{4}, i \in(0 \ldots(a-1)) P\left[C_{i}\right]=\frac{1}{2}, i \in(a \ldots(b-1))
\end{aligned}
$$

Thus, the the probability of accessing a specific tuple-id generated from the NVRand (x. $0 . y$ ) function is $\left(\frac{3}{4}\right)^{i}\left(\frac{1}{4}\right)^{j}\left(\frac{1}{2}\right)^{z}$ where $i$ equal the number of non-zero bits in the low a bit. j is the number of zero bits in the low a bits, and $z$ is as defined above. Hence, the probability mass function is periodic where the size of the period equals $x$, and the number of periods equals $\left[\begin{array}{l}\frac{y}{r} \\ r\end{array}\right\rfloor$.


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


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


Figure 3: Stock Relation PMF


Figure 4: Stock Relation PMF: only 10,000 tuples


Figure 5: Stock Relation CDF


Figure 6: Customer Relation PMF


Figure 7: Customer Relation CDF


Figure 8: Customer, Stock, and Item Miss Rates


Figure 9: Maximum Throughput


Figure 10: Price Performance


Figure 11: Scaleup of TPC-C


Figure 12: Sensitivity to Percent Remote



[^0]:    - A significant portion of this work was done while Lentenegger was a Post-Ductoral Researcher at IBM T. J. Watson Research Center. Support for Leutenegger was also provided by the National Aeronantics and Space Administration under NASA Contract Nos. NAS1-18605 and NAS1-10480 while he was in residence at the Institute for Computer Applications in Science and Ensineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-000).

[^1]:    ${ }^{2}$ Such a distributed CC; protocol is optimized for read-only sharing of replicated data, and fares poorly when there is significant write sharing. Many distributed CC protocols with replication are optimized for significant write sharing. and consequently are worse for read-only sharing. See $[1,2]$ for a good summary of distributed (: $($ protocols and $[3]$ for

[^2]:    an analytical comparison of distributed $C$ (: with data replication.

