Clustered DBMS Scalability under Unified Ethernet Fabric

Krishna Kant
Intel Corporation

Amit Sahoo
Univ. of California, Davis

Abstract

In this paper, we study the performance of clustered DBMS running on-line transaction processing (OLTP) workload and using TCP/IP over Ethernet as a "unified fabric" for inter-process communication, iSCSI based storage access, and networking traffic. The study is primarily based on a comprehensive simulation model of such systems that we have built. In particular, we study scalability, impact of fabric latency and effect of cross traffic on the DBMS performance. We find that while the protocol overhead has a large impact on performance, the end to end latency has comparatively lesser impact. We also find that interfering high priority traffic from other applications can have a significant performance impact by delaying critical control messages.

Key words: Clustered database, TCP/IP, Quality of Service, Latency Impact, Thread switching

1 Introduction

In the e-business environment, mid-tier and backend applications have traditionally been implemented on SMPs (symmetric multiprocessors) because of their easier programming model and efficient inter-process communication (IPC). However, SMP implementations have some serious drawbacks including high cost, inability to grow the system gradually as the need arises and poor scalability. The clustering model provides an attractive alternative to address these issues provided that SMP applications can be ported without significant changes. The attractiveness of clustering model is further aided by the emergence of high bandwidth, low-latency cluster interconnect technologies such as Infiniband architecture (IBA) [13] and HW offloaded TCP/IP over Ethernet [7, 15]. On the software side, there are already solutions available for running applications without a painstaking manual partitioning in order to minimize inter-process communication (IPC). For example in the database space, clustered systems such as Oracle 9i/10g have claimed that a good concurrency control model coupled with a distributed caching service can avoid the need for database partitioning [9]. However, there isn't much information available in the open literature on the performance, scalability and stress behavior of such systems. In particular, although there are studies of clustered system scalability, they are mostly confined to either scientific computation (MPI based) or rather small clusters (8 way). In this paper we provide such an analysis based on a comprehensive simulation model and study the impact of fabric latency and network congestion on scalability.

The primary focus of our study is a cluster using TCP/IP over Ethernet as the "unified" clustering fabric, since we believe that specialized fabrics such as IBA, Myrinet, QSNet, etc, will remain niche due to huge installed based of TCP/IP/Ethernet systems. In particular, we consider the scenario of a single high speed "pipe" coming into a server as a *unified fabric* that carries all traffic types, which in this case includes inter-process communication (IPC), iSCSI based storage, and normal client-server traffic. Such an approach requires that the unified fabric work almost as well as isolated fabrics under stress conditions. Therefore, understanding the behavior of the clustered application under stress and other abnormal scenarios (e.g., under high latency) is of paramount importance. This paper attempts to contribute to this understanding via a very detailed simulation model of clustered OLTP server. Currently the model is still weak in validation against measurements, which we hope to rectify in coming months; however, we believe many of the trends are captured accurately.

2 Cluster Architecture, Workload and Modeling

In this section we briefly describe the modeled database clustering architecture, the workload driving it and how the simulation captures the essence of such systems.

The main reason to consider clustered databases as a target application is that such systems are already available commercially and represent an important class of applications from both storage and IPC traffic perspective. In contrast, front-end systems have no IPC traffic and low storage use. Most mid-tier systems have very little IPC traffic and their performance is not very sensitive to IPC latencies. From a IPC latency perspective, clustered implementation from the high-performance computing (HPC) is quite relevant, however, HPC workloads tend to be much more specialized and the IPC needs can vary substantially

from one implementation to another [3].

2.1 Database Clustering Architecture

Clustered DBMS implementations cover a wide range in terms of coupling of various nodes. On one extreme, there is the "shared nothing" approach, where each node has its own independent memory and IO subsystem. In this case, the database must necessarily be partitioned among nodes – either statically or dynamically. A more coupled approach is "shared IO" approach, where all nodes access a centralized IO subsystem which holds the database. The IO subsystem in this case is invariably a Fiber-channel based SAN (system area network). An even more coupled approach is essentially a NUMA approach where the node may also have access to a globally shared memory (in addition to individual memories). Such a model begins to look like a SMP and is not considered in this paper.

The shared IO model has been adopted in Oracle 9i/10g commercial DBMS and is attractive in that it does not require any partitioning effort in porting SMP applications to clusters. However, since we are interested in Ethernet as a unified fabric which carries storage traffic as well, we primarily consider the case of iSCSI (Internet SCSI) based storage available at each node. One attraction of such a "distributed storage" model is that it allows inexpensive IO system at each node which expands naturally with the cluster size. The partitioning scheme for this storage model will be discussed later. In either case, however, we assume a coordination mechanism similar to the *cache fusion* architecture used by Oracle 9i/10g [9].

From a very high level perspective, The *cache fusion* architecture essentially extends the cache-memory coordination in a SMP to the memory-disk coordination in a cluster. That is, each node has its private database memory (usually called *buffer cache*) and a shared secondary storage. Like the SMP, such a system needs a coherence protocol, which could also be MESI protocol used in SMPs (or some variant thereof) [14]. Unfortunately, such a protocol would require substantial inter-process communication (IPC) traffic to accomplish the necessary "snooping" and "invalidation" of copies held by various nodes. Oracle's mechanism, called RAC (real application cluster), attempts to avoid this overhead by exploiting *multi-version concurrency control* (MCC) [1].

The basic idea of MCC is to create a new *version* of the table row (or a larger unit, depending on the granularity) each time it is updated. MCC avoids any "read-locks" since a transaction can always find the appropriate version of the data to read. Write/update accesses still require locking, however, there is no need for a traditional "invalidation"; instead, the concurrency control needs to ensure that only the most recent version is written to. The price for MCC must be paid in terms of managing multiple versions, additional memory requirements, fatter IPC data messages, and more disk IO (due to less efficient memory use).

The basic value proposition of cache fusion is that retrieving data out of the buffer cache of a remote node is significantly cheaper than reading it from the disk (even in case of local disks). Thus, the overhead and end-to-end latency of IPC vs. that of disk IO are crucial parameters for the performance and scalability of cache fusion based clustered DBMS. It is well known that the traditional OS Kernel based TCP/IP implementations are quite inefficient [7]. Nevertheless, the corresponding IPC overheads and latencies are still considerably smaller than those for disk IO. Thus, one would expect small clusters to perform well even with the traditional "SW TCP" solutions. With HW TCP implementations, good scaling should be possible even for rather large clusters.

Cache fusion uses a directory based coherence scheme that proceeds as follows. Suppose that a node A experiences a miss on DB block X in its local buffer cache. Node A then determines (via a local table lookup) that some node, say B, holds the directory information for this block. The sequence of actions is then as follows:

1. A requests the block X from node B. B looks up the directory and returns positive or negative response to A.

2. In case of a negative response, A obtains block X from the disk (local or remote).

3. In case of a positive response, A waits to receive the block from some node C which is determined by B as the data supplier. B sends a message to C, and C responds to A directly with the block. (The last one is IPC data message, all others are control messages).

4. A eventually informs B of successful retrieval so that B can update the directory indicating A also as the data holder. (If A had to evict a block from its buffer to accommodate the new one, it informs B of that too.)

Note that it is possible that $A = B$, or $B = C$; in these cases some operations become local and the corresponding messaging is not needed.

The IPC data transfer is not limited to a single DB page – the transfer size could range anywhere from 4KB to 64KB. The optimal transfer size depends on a number of factors, and we do not attempt to adjust it for different runs of the model. Instead, we assume a basic IPC transfer size of 8 KB (same as disk block size).

Other than the block transfer and directory management related IPC traffic, the scheme involves a number of other IPC messages for such things as write lock acquisition/release, transaction commits/aborts, notification

of block caching/evictions to the directory node, checkpointing, directory migration, etc. These operations may result in a significant number of additional IPC messages between nodes.

2.2 Database Workload

Given its popularity and availability of detailed characterization data, the TPC-C benchmark (`http://www.tpc.org/tpcc/default.asp`) is a natural choice for an OLTP database workload. TPC-C models operations of a wholesale parts supplier operating out of a number of warehouses and their associated sales districts. Each warehouse supplies 10 sales districts, and each district serves 3000 customers. The database manages 100K parts in terms of orders, prices, stock level, etc. The workload has 5 transactions, namely new-order (enter a new order which requests 10 parts on the average), payment, order status, delivery (process a batch of ten orders for delivery), and stock level (level of stock of the items ordered by last 20 orders). The nominal fractions of these transactions are 43%, 43%, 5%, 5% and 4% respectively.[1] The performance metric reported by TPC-C is the number of "new-orders" processed *per minute* and is expressed as tpm-C.

The benchmark involves 9 tables: *warehouse, district, customer, stock, item, new-order, order, order-line and history*. Of these, the first 5 tables are fixed, but others are variable. New-order table grows & shrinks as new orders come in and are retired. The last 4 tables keep a permanent record of transactional operations and thus only grow. The benchmark is designed such that *the database size increases linearly with the throughput*. In particular, the number of configured warehouses is approximately tpm-C/12.5. Sizes of all tables, except item, are multiples of warehouses. The item table stays constant at 100K rows. The largest tables are typically customer and stock and may require significant space for their indices. Although the variable tables like order, order-line and history are also quite large, access to them is quite localized.

A notable characteristic of TPC-C transactions is that they all refer to a single warehouse. In fact, according to the specification, a given "terminal" always generates transactions with the same warehouse-id. This, coupled with the fact that most tables have #warehouses as a multiplier, makes TPC-C database trivially partitionable: assign equal blocks of warehouses to each server and direct queries based on the warehouse. For this reason, TPC-C is usually considered an inappropriate workload for clustering studies. We address this weakness by tweaking the workload behavior according to our needs. In particular, we still partition the database in blocks of warehouses, but do not necessarily direct queries to the right sever. Instead, we introduce the notion of *affinity*. An affinity of 1.0 corresponds to the case where a query always goes to the server that hosts the referenced warehouse. An affinity of $\alpha < 1$ means that the query goes to the right server with probability α and to a random server with probability $1 - \alpha$.

[1] Actually, TPC-C allows new-orders to go up to 44% at the cost of delivery, but this may have some undesirable consequences.

2.3 Overview of Cluster simulation Model

The simulation model called distributed cluster emulator (DCLUE) was developed using the simulation package OPNET (`www.opnet.com`). OPNET provides a fairly complete emulation of the network infrastructure including TCP, IP, and Ethernet MAC layers, QoS support, commercial switches and routers, etc. DCLUE was built on top of the OPNET provided TPAL (transport adaptation layer) which can support multiple transports underneath.

The model implements cache fusion, multiversion concurrency control, row/page locking, logging, disk IO, database tables, table operations, buffering, IPC handling, application processing, scheduling, thread switching, processor-memory data transfers, etc., often in painstaking detail. As a result, it requires a rather fine-grain model calibration. This is a problem in spite of a wealth of available measurement data on TPC-C, TCP processing, iSCSI processing, etc. On the positive side however, *the model isn't dependent on high level results that would be easily invalidated by a change in system parameters*. For example, the hit ratio in the buffer cache is not an input parameter; instead, it is a result of the actual buffer cache management done by the simulation. This allows us complete freedom in choosing the cached fractions of various tables and their indices. Similarly, the number of locks acquired per transaction, IPC messages sent/received per transaction, log blocks written to the disk, blocks read from the disk, data versions created per block, context switches per transaction, etc. all fall out of the actual functioning of the simulation rather than being artificially provided as some inconsistent set of values.

In spite of the detail, DCLUE obviously could not mimic a real system at a fine grain level; the purpose of DCLUE is to merely implement the most important functionality from a performance perspective and thereby allow sensitivity studies. Some of the high-level functionality missing from DCLUE are failure recovery and checkpointing since these are not essential for our purposes. Nevertheless, given the model calibration based on actual measurement data, the results provide valuable insights into the performance of OLTP workloads on a cluster. In fact, the lack of idiosyncrasies of specific implementations allow us to study true scaling characteristics of the cluster instead of being limited by the physical bottlenecks that invariably pop up in measurement based studies.

Figure 1 shows the DCLUE network model. The net-

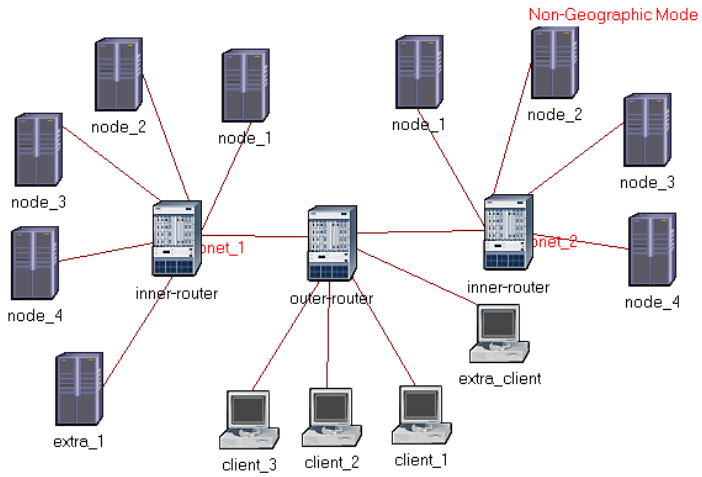

Fig 1: A sample DCLUE model w/ 2 latas & 4 nodes per lata

work is organized as one or more "subclusters" which we call LATAs (borrowed from telecom). The subclusters are connected via an "outer-router" (or an "outer-switch" if we only want layer 2 switching), at which the clients also home in. The used router model is a standard OPNET provided model and represents 3M Gigabit routers fairly well. Each subcluster has its own router (or switch). Each server has internal disk subsystems for normal IO and logging, but not all of them may be used. In the distributed storage configuration, the disks are accessed remotely via iSCSI protocol and via SCSI protocol locally. In the centralized (or SAN based) storage configuration, the set of all IO subsystems forms a virtual SAN which is accessed via some unmodeled SAN fabric.

One of the objectives for the model is to study potential ill effects of running IPC and storage traffic on the normal Ethernet network that carries miscellaneous other types of traffic. For this, the model allows some extra clients and servers to be added to the cluster (distinguished in the model by a different address range). These clients/servers can run some additional applications and cause that traffic to interfere with DBMS traffic on various links and routers. For example, Fig 1 shows the nodes marked "extra_client" and "extra_server" whose traffic interferes with regular DBMS traffic on inter-lata links.

During initialization, each server establishes 2 TCP connections to every other server: one for IPC messages (data & control) and the other for iSCSI related traffic (command, status, data, etc.). The reason for separate connection is to allow QoS studies that treat IPC and storage separately. The TCP flavor used is Reno, SACK is enabled, and so is ECN. TCP timer values are reduced by 100X to make them suitable for data center operation. The receive buffers are set at 64KB. Various router parameters are also adjusted to make them suitable for the data center. The client-server TCP connections are established dynamically on a per "business transaction" basis. A business transaction consists of the sequence of TPC-C transactions starting with the new-order in the proportions specified in section 2.2.

The complexity of the simulation model (about 12,000 lines of C-code) precludes a detailed discussion here; a somewhat detailed description of many of its crucial aspects may be found in [6]. Here we provide only a very high level overview of some of these features.

DCLUE basically builds the entire TPC-C database in the memory and initializes it according to TPC-C rules. However, the information retained in the tables is only what is essential to interpret and execute queries, which means that DCLUE can use the storage much more efficiently while still retaining the precise row sizes, rows per block, etc. DCLUE also explicitly maintains B^+-tree indices for each table. Since the entire database is sitting in the main memory, buffer cache operations merely change status of the pages in question. Disk IO operations are still simulated in terms of their latency and *path-length*, i.e., number of instructions required to execute an operation. Normal disk IO optimizations such as elevator algorithm are implemented on a per table basis. Although the disk writes are lazy and could finish after the transaction is done, the transaction does not commit without writing a log. The logging is done on disks separate from those for normal IO.

DCLUE implements fine-grain locking but dividing pages into subpages. We found that we had to "tune" the size of subpage for each table separately. In particular, the district table is accessed very frequently and need a small subpage size. The locking mechanism itself in-

volves 2 phases, where phase 1 performs "intention locking" (or *latching*) and brings in any missing data into the buffer cache [12]. The phase 2 then actually attempts to convert the latches into actual locks. Also, if a lock cannot be acquired, a lock wait is performed on the first lock in the sequence, and later failures result in lock release followed by a delayed retry. The scheme appears to work well even under high contention.

The multiversion concurrency control was implemented using time-stamping mechanism and keeps track of minimum version no, maximum version number and current version no. Space for versions is allocated from an overflow memory area. If this overflow area runs low, unpinned pages from the buffer cache are stolen to replenish it.

The application processing is implemented in detail for each operation (e.g., transaction initiation, table operation, etc.) and so are message sends and receives. In particular, application processing is interrupted to handle message receives. The calibration of various *path-lengths* was done based on the NASA report on TPC-C [10] and current TPC-C measurements. The data related to platform characteristics is taken from a long list of comprehensive measurements available internally in Intel. Similarly, data related to IO operations (e.g., accelerated and non-accelerated TCP/IP, RDMA and iSCSI stacks) is taken from internal prototypes and measurements [7, 15, 8, 4].

The most crucial aspect of the model is the modeling of threads, thread switching, and its impact on processor caches. Basically, in a transactional workload, latency can be hidden by simply having more concurrent threads. However, given the processor cache size and working set of each thread, only so many threads can be accommodated conveniently. With larger number of threads, the context switch penalty rises very sharply and the cache begins to thrash. Capturing this behavior was essential to properly model the impact of latency on performance. Fortunately, we had available to us a very detailed characterization of this and other OS aspects under Redhat Linux 7.3 OS [2]. This along with internal studies on TPC-C working set size provided us with the requisite modeling of the threads.

The final aspect modeled in detail was the load on processor bus and memory channels and corresponding impact on CPU stalls. This again is essential for accurate modeling of platform level performance. Fortunately, this is one area that is routinely studied in connection with performance projections for various platform configurations (e.g., see [5]). Also, a lot of information exists based on both measurements and cycle-accurate workload simulation of TPC-C. Yet, an accurate projection of MPI as a function of affinity is challenging and is currently based on some heuristics. Address bus, data bus and memory channels are modeled as queuing systems and the resulting memory latency determines CPU stalls via the concept of *blocking factor* (the fraction of latency visible to HW threads). We exploited available data on Intel Pentium 4 to calibrate this aspect of the model.

3 Cluster Performance Studies

In this section we use the DCLUE model to obtain a number of interesting results on scalability, latency sensitivity and impact of cross traffic. As stated earlier, although standard TPC-C specification is exploited heavily in the implementation and model calibration, we are interested in scenarios beyond basic TPC-C particularly in terms of the role of IPC in clustered databases.

3.1 Configurations and database scaling

The configurations that we considered are clusters of Intel Pentium IV class dual-processor (DP) servers. For these systems, unclustered TPC-C measurements and validated platform performance models were readily available and thus allowed detailed result validation at affinity of 1.0. In particular, our baseline server configuration is a 3.2GHz P4 DP system with 1 MB second level cache, 133 MHz bus and 16GB of DDR-266 memory. One such node delivers about 50K (unclustered) tpm-C performance, which amounts to about 4K warehouse database. For the network infrastructure, we stuck with the current 1 Gb/s ethernet links and routers primarily because the current processors are unable to drive 10 Gb/s bandwidth except in large clusters. However, in a few cases, 10 Gb/s inter-lata links had to be used since 1 Gb/s links were becoming a bottleneck. The router models used are OPNET supplied 3M Gigabit routers. Unless stated otherwise, we assume that both TCP and iSCSI have been implemented in hardware in the following.

Unfortunately, a direct simulation of even a small cluster will require long simulation times and huge amounts of memory. The need for > 4GB memory would require the complexity of reworking the simulation to use PSE/AWE on a 32-bit machine. To avoid these problems, we consistently scaled all relevant parameters by a factor of 100x. This means, for example, (1) Ethernet network model is 10baseT instead of 1000baseAE, (2) disk parameters (seek, rotation, data transfer) are slowed down by a factor of 100, (3) CPU, bus and memory channel frequencies are cut down to 32MHz, 1.33MHz, and 1.33MHz respectively, and (4) Various other delays such as chipset, IP packet forwarding, context switch, interrupt handling, etc. are also increased by a factor of 100x. In order to allow for a convenient scaling of all processing overheads, all input parameters are expressed as "path-lengths" (i.e., number of instructions required to accomplish the operation) or as or path-length equivalents. This ensures that a speed cut of CPU by 100x automatically scales everything

by 100x. Finally, as for the database itself, a slow-down in all platform and OS parameters will automatically reduce the throughput (and hence the number of warehouses) by 100x – the only scaling required is for the item table, which does not depend on number of warehouses. This is done by reducing the number of items from 100K to 1000.

With the above scaling, it is possible to simulate reasonable sized clusters. The results must be scaled back to correspond to the original system.

3.2 Performance Scaling vs. Cluster Size

Before launching into latency and traffic impacts, it is important to first see how the cluster performance scales with number of nodes. However, since scaling is the net result of many complex activities in the system, we start by examining the latter first. One of most important aspects in this regard is the growth of IPC messages as a function of number of nodes. This is shown in Figs. 2 and 3 for 0.8 and 0 affinity respectively. Each figure shows the IPC control and data messages per transaction separately. The IPC messages are much smaller than data messages (about 250 bytes vs. more than 8KB) but significantly more numerous. The interesting point to note is that the IPC message count rises sharply at first but then "saturates" rather quickly. As a result, number of IPC messages very quickly cease to have any impact on scalability. In other words, no nonlinearly in performance is expected due number of IPC messages beyond small cluster sizes.

Note that the figures do show some variability and inconsistency in the results. This is not so much a simulation error but more because of wide variations in transaction characteristics as discussed more fully in [6].

Figures 4 and 5 show lock waits per transaction and lock wait time as a function of number of nodes. The variability in the results is particularly pronounced here since both of these parameters are very much a function of prevailing conditions. Nevertheless, the trend is clear: Both lock waits per transaction and average lock wait time increase steadily with cluster size. The same holds for number of lock failures per transaction (not shown). In the absence of other effects, this aspect will limit the cluster scalability.

Let us now examine the scalability. Figure 6 shows this wrt cluster size and affinity as a parameter. The affinity 1.0 case is shown just as a reference and corresponds to the case of perfect scaling. As expected, the scaling gets progressively poorer as the affinity rises. However, the interesting part is an almost linear scaling from 2 or 3 nodes to 10 nodes. For larger clusters, locking related issues start to come into effect. Also, topological issues also come into play. For very small (i.e., 2 or 3 node clusters), the behavior can also be different, and becomes more pronounced with lower affinity.

The slope of the scalability line strongly depends on the affinity. This is more clearly shown in Fig. 7. The scaling goes down rapidly with decrease in affinity. Another important point to note is that the performance sensitivity is high at high affinity values and decreases with the affinity. In other words, tuning the system for small decreases in affinity is more beneficial when the database is already well partitioned.

In our experiments we considered 14-port routers/switches, which would be typical for a bladed system. Therefore, for cluster larger than 12 nodes (i.e., 16 and 24 node cases shown here), we had to move to a 2-lata scenario. This brings in the latency and queuing impacts of IPC traffic going across latas (through 2 extra links and 2 extra routers). Consequently, there is a change in slope around 12 nodes. However, it is important to note that the increasing lock failures and lock waits also play a substantial role in flattening out the scalability curve. In fact, with affinity of 0.5 or less, the network effectively stops scaling beyond 12 nodes. At 0.8 affinity, the scaling is decent and perhaps could be improved with faster links and routers.

At high affinities, the reason for continued scaling is the lack of any shared bottlenecks in the system. In fact, most resources increase linearly with the cluster size. For example, each new node adds not just CPUs, but also many others. These are: memory, memory channels, processor bus, normal and logging disks, and router links. If the network grows by adding more subnets, the stress on each inner-router also remains unchanged. Even the lock contention per page stays the same since TPC-C mandates that the database size increase linearly with the throughput. At low affinity values, although the MPI grows significantly, the low realized throughput in this case prevents bus from becoming a bottleneck for moderate cluster sizes.

Poorer scaling properties can be observed if the linear growth in resources is broken. As an example, Fig 8 shows a case where the forwarding rate of the routers is reduced from the normal 10000 packets/sec to 4000 packets/sec. The scenario shown is for a single lata cluster. The rate reduction causes the inner router to saturate beyond 8 connected servers and it limits the scaling too.

Fig 9 shows another scenario, one where a single node is responsible for all logging operations. Normally, each node performs its own local logging operation. While this yields good performance, it may make rollback very complex since the recovery procedure would have to obtain logs from all nodes, sort them by timestamp and then do the rollback. Centralized logging makes recovery easier but at the cost of potential bottleneck during normal operation. It is seen that the performance in this case is consistently lower. Eventually as the node and local IO subsystem capacity is reached, the cluster will stop scaling.

Fig 10 shows the impact of slower growth of DB size as a function of throughput. For this we assumed that for up

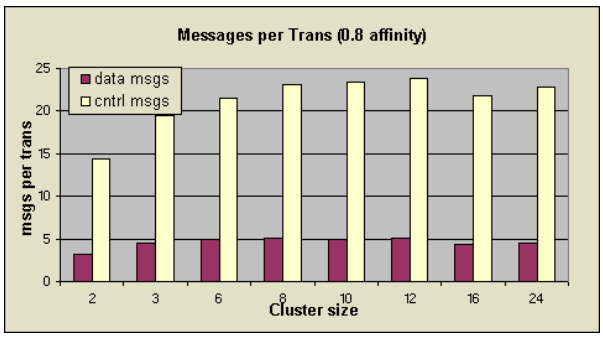

Fig 2: IPC messages per trans for 0.8 affinity

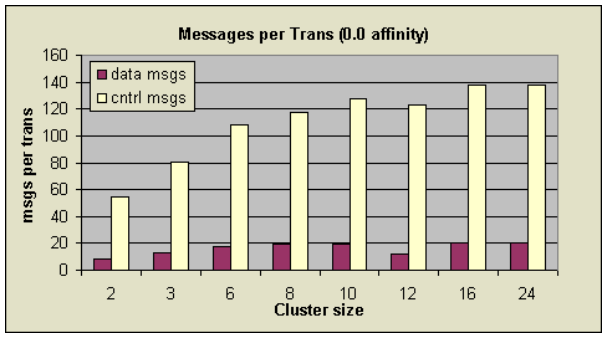

Fig 3: IPC messages per trans for 0 affinity

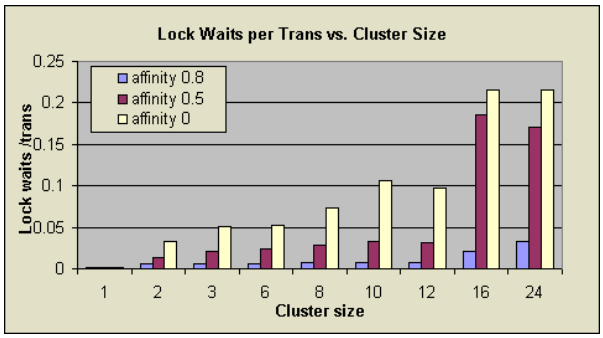

Fig 4: Lock waits/trans vs. #nodes and affinities

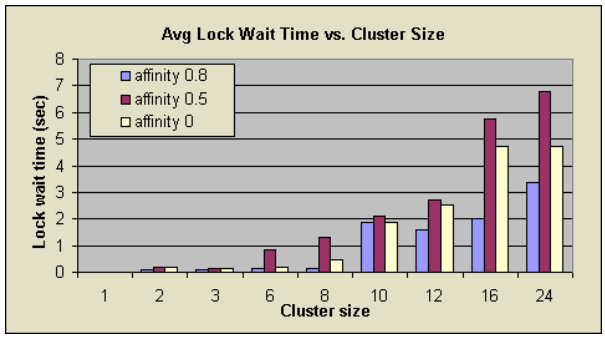

Fig 5: Lock wait time vs. #nodes and affinities

to 90K tpm-C, the database sizing is according to TPC-C rules (No of warehouses calculated assuming 12.5 tpm-C per warehouse). However, beyond this, the growth rate of warehouses goes a square root of the additional throughput, rather than linearly. With this, the contention for the data increases as the cluster size increases. Consequently, the throughput no longer goes up linearly with the cluster size.

3.3 Protocol Overhead vs. Latency

Compared with specialized fabrics, the traditional SW based TCP/IP suffers from two drawbacks: (a) significant overhead of code execution (and associated OS bottlenecks), and (b) significantly higher latency. It is important not to confuse the two. For example, a significantly better performance achieved with specialized fabrics could well be due to much lower overhead rather than the ultra-low latency. Fig. 11 compares the performance of the following 3 cases for various affinities:

1. Both TCP fast path and iSCSI implemented in HW. This is the normal case considered for most of our experiments. For this, detailed TCP and iSCSI parameters were obtained from current offload prototypes.

2. TCP fast path in HW but iSCSI implemented in SW.

3. Both TCP and iSCSI implemented entirely in SW. The SW TCP assumes single copy for sends and 2 copies for receives.

With affinity 1.0, there is no appreciable difference between the 3 cases. This is because in this case there almost no IPC traffic (except for occasional access to item table pages). Also, all disk accesses are local, so iSCSI doesn't come into play at all. The only traffic that benefits from TCP acceleration is client-server. Consequently, the HW TCP performance is slightly better than SW TCP, but not by much.

With affinity 0.8, HW TCP provides almost twice as much throughput as SW TCP. This is because the lower overhead and latency of TCP substantially reduces both the workload path-length and stall cycles. However, the difference between SW iSCSI and HW iSCSI is marginal. Partly this is due to the fact that disk IO rate is small (since most data comes from other buffer caches). Also, iSCSI implementation path-lengths are small except for the rather large overhead of CRC calculations [4].

Finally, with affinity 0.5, the difference between HW and SW TCP is even wider, but not by much. This result may be surprising since the number of messages per transaction does increase significantly from 0.8 to 0.5 (from 21 control messages per transaction to about 54). However, with 0.5 affinity the major expense in completing a transaction is

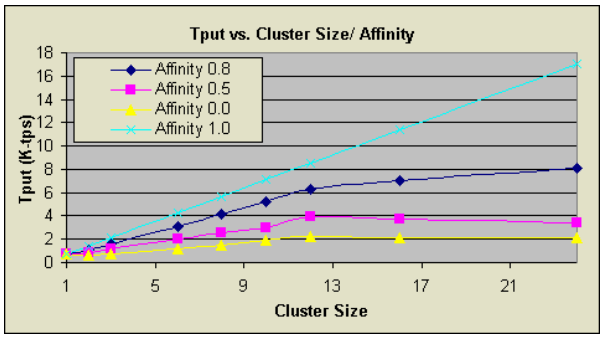

Fig 6: Scaling vs. nodes and affinity

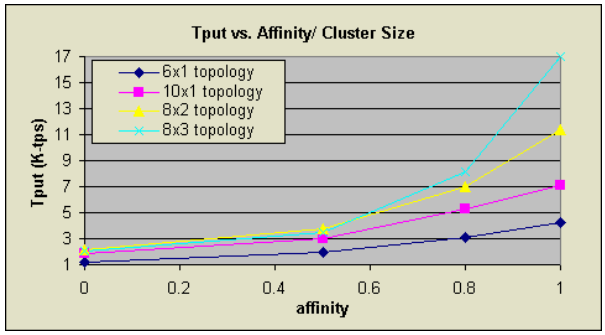

Fig 7: Scaling vs affinity and nodes

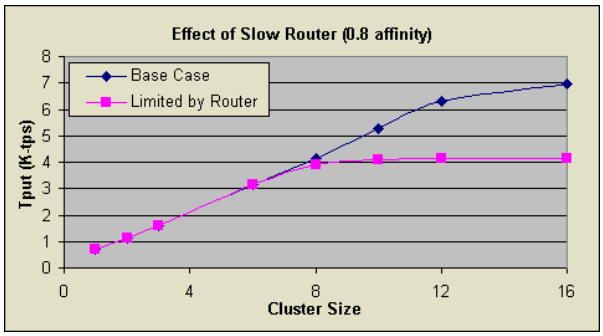

Fig 8: Impact of router forwarding rate on scalability

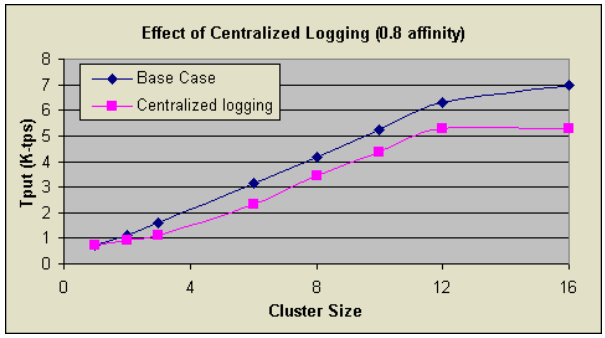

Fig 9: Impact of single node logging on scalability

due to lock failures and corresponding path-length increase and CPU stalls. The TCP/IP overhead thus has proportionately smaller impact.

Next we consider the latency impact. High latency tolerance would allow a less expensive implementation and even geographically distributed clusters. We study the impact of latency by simply adjusting link lengths to achieve the desired extra latency. It is important to note that this type of latency introduction is quite different from latencies within the platform (e.g., greater memory access latency or context switch latency) which cause direct CPU stalls. In a transactional workload, the true impact of latency is felt only when the latency cannot be hidden by employing additional threads; therefore, we do not place any bound on the number of threads used. Figure 12 shows performance of a 2 lata system where each of the two inter-lata links includes one-half of the additional latency shown. The two curves are for 0.8 and 0.5 affinity respectively. It is seen that in both cases, a 1 ms additional delay results in about 3.4% performance drop, whereas a 2 ms delays causes a 6% performance drop. These latencies should be viewed in the context of 1 Gb/sec link. If we were to consider systems capable of driving 10 Gb/sec bandwidth, we might expect similar drops with 1/10th as much latency. That is, we could expect a 100 μu latency to drop the performance by a few percentage points. This is a rather low sensitivity considering the fact that the normal end-to-end delay with HW TCP can easily be brought down to 10-20 us.

We ran the experiment for 0.5 affinity (in addition to 0.8) hoping to see higher latency sensitivity for this case because of much higher IPC messages per transaction. Surprisingly, however, the sensitivity is the same in both cases. This result is a result of worse threading behavior for 0.5 affinity and is discussed more fully in the next subsection.

One reason for low sensitivity of TPC-C to latency is its huge computational component as indicated by a path-length of 1.5M for the unclustered case, of which only about 15% is related to disk IO. Other OLTP workloads are significantly lighter on computation and thus could have higher sensitivity to latency. To investigate this, we simply reduced all computational path lengths by a factor of 4 and the resulting situation is designated as *low computation'*. (A more realistic method would be to actually make the queries more light-weight, but that requires lot more effort.) Figure 13 shows that the change indeed makes the workload lot more latency sensitive. In particular, 1 ms additional latency now results in 10.4% drop in performance.

The results above show that the latency sensitivity is low enough that there is no need for designing ultra-low latency TCP/IP offload or router and switches. Furthermore, if the database characteristics are like TPC-C, it should be possible to geographically separate the subclusters (or lata's) on the scale of MAN distances. For example, if we

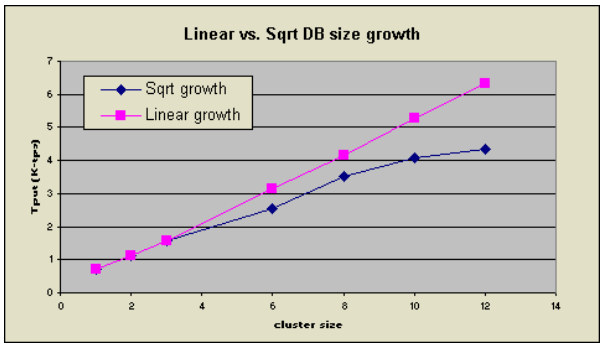

Fig 10: Impact of slower growth in DB size

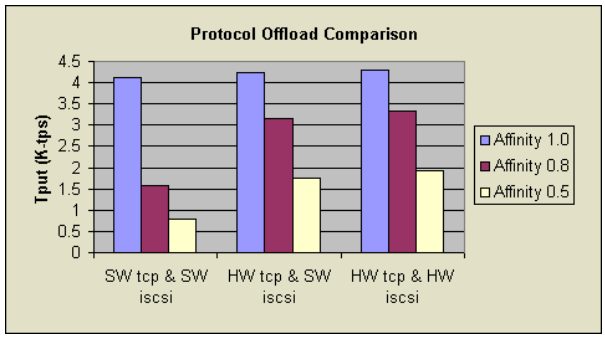

Fig 11: Impact of TCP and iSCSI offload

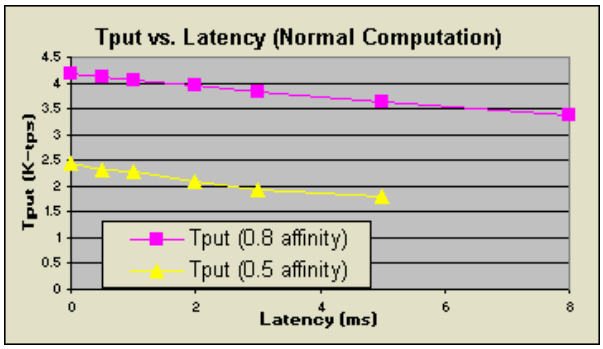

Fig 12: Latency impact: normal comp, 0.5 & 0.8 affinity

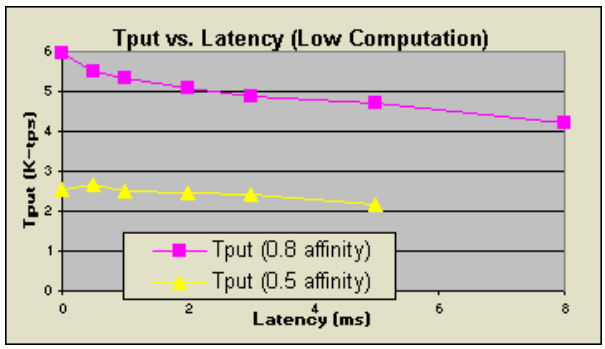

Fig 13: Latency impact: low comp, 0.5 & 0.8 affinity

have two subclusters with one of them located 50 miles away, the additional 1 ms RTT increase will lower the performance by only a few percent on systems driving 1 Gb/sec bandwidth.

3.4 QoS Impact

In this section, we examine how the IPC and storage traffic is affected by other interfering traffic on the network. The results presented here are for the case where the extra clients/servers run FTP traffic with 50% GETs and 50% PUTs. As usual, the FTP application sets up new TCP connection for each transfer. This makes the interfering traffic more "stubborn" than the IPC traffic which uses a static connection. In particular, under overload conditions IPC connection may get reset and may have to be re-established. Since connection re-establishment involves a lot of overhead and lost traffic, we have avoided this situation by artificially bumping up the maximum retransmission count to rather high values. While this may not be realistic, we were interested primarily in the effect of cross-traffic as opposed to abnormal conditions created by it. Clearly, some admission control scheme needs to be in place to ensure that unlimited amount of traffic doesn't get into the network and cause connection resets.

We note here that we used FTP traffic here as a generic interfering traffic, and weren't overly concerned with maintaining a real-life file size distribution. In fact, as might be expected, setting the the file sizes too large would punish the FTP connection severely in case of congestion, and thus the traffic will not be able to wound the DBMS traffic that much. On the other hand, with very small transfers, FTP spends most of its effort in setting up/tearing down connections and therefore isn't a good interference candidate. Consequently, we decided to make FTP file sizes similar to DBMS transfer sizes. DBMS control messages are in 250 byte range and data messages are 8 KB or larger (the larger part comes because of additional versioning data).

The scenario studied consisted of two latas, each with 4 nodes and an affinity of 0.8. We considered both normal and low computation cases (see last subsection) for this. In the normal computation case, the combined DBMS traffic on the inter-lata links was about 650 Mb/sec and for low computation case, the traffic is about 920 Mb/sec. The cross traffic (FTP) was varied from 0 Mb/sec to 600 Mb/sec in both cases. It important to note here that the *carried* traffic from both DBMS and FTP domains will depend on the interference and QoS setup – the numbers here merely refer to *offered* traffic in isolation.

With respect to QoS setup, we were primarily interested in diff-serv, since int-serv may not be implemented or even

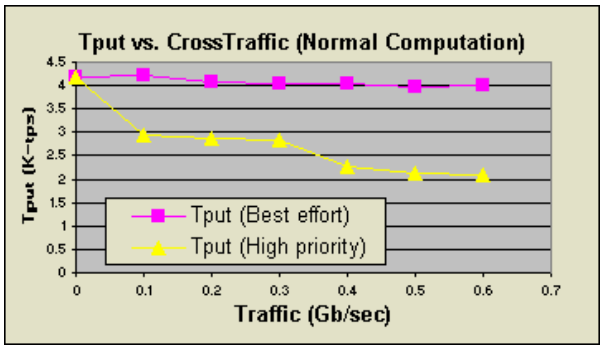

Fig 14: Impact of cross traffic w/ normal computation

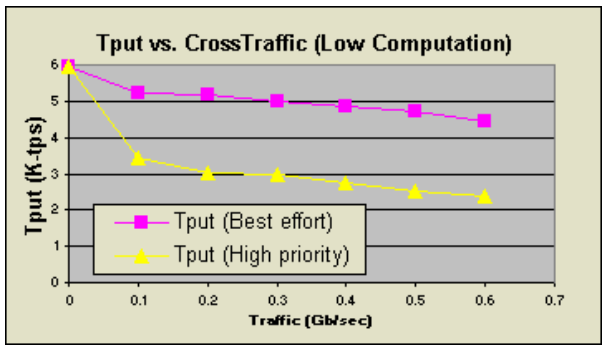

Fig 15: Impact of cross traffic w/ low computation

necessary. The diff-serv space itself is huge and include 4 different mechanisms:

1. Queuing schemes (priority, WFQ, ...)
2. Packet drop schemes (tail drop, WRED, FRED, ...)
3. Traffic Policing/shaping (e.g., leaky bucket)
4. Connection admission control (CAC)

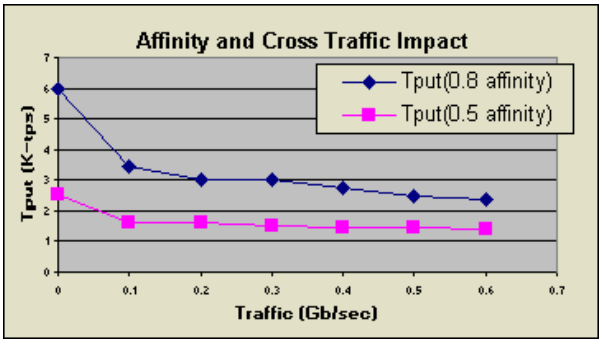

Fig 16: Impact of cross traffic w/ low computation

Each of these mechanisms involves numerous algorithms and tuning parameters thereby making QoS setup a nightmare. In view of this, we concentrate only on rather simplistic scenarios because they are most likely to be found within the data center. In particular, we report results on only the following two cases:

1. Both traffic types are of *best effort* type. This can be described as the "lazy" approach, where the administrator makes no effort to exploit diff-serv mechanisms.
2. DBMS traffic is best effort but the FTP traffic is assigned the DSCP AF21. This scenario was motivated by the usual situation where most of the traffic runs as best effort, but certain limited types of traffic use diff-serv for business reasons or because they demand good treatment. Admittedly, FTP isn't a good example of such a traffic, but for the purposes interference study, it probably doesn't matter.

In OPNET's default implementation, a higher AF number really translates into a larger queue (and hence a lower drop probability) and priority treatment. Note that since the two traffics don't share a client or server, priority treatment is confined only to the router. Thus, unless the router itself is congested, the priority queuing will not shut out the lower priority traffic under overload. The routers use simple tail-drop (instead of RED, WRED, etc.) and no connection admission control, policing or shaping policies are employed.

Fig 14 shows the results for normal computation for the two QoS arrangements. It is seen that with both traffics as best-effort, interfering traffic does not make any significant impact on performance. Instead, the performance goes down marginally and at a slower rate as the traffic goes up (until the link really saturates). The explanation is that both DBMS and FTP traffics suffer due to competition and back off. For the DBMS traffic, this simply means a longer wait for the threads and hence more active threads. So long as the thread wait is small enough for the request/response to get through, the performance is not adversely affected.

With FTP traffic given a higher priority, the impact is much more pronounced — a 30% drop in DBMS throughput with only 100 Mb/sec of FTP traffic. The large drop was found to be a result of increased queuing delay not only at the links but also at the router, whose impact is enhanced due to priority handling of FTP traffic. In particular, critical IPC control messages such as lock acquire and release are delayed substantially while the FTP traffic is establishing/tearing down connections or is transmitting its data. It was found that not only the message delays almost doubled, the lock wait time also went up substantially from about 2 ms to 10 ms.

Surprisingly however, most of the drop happens initially only; with higher FTP traffic, the performance still goes

down by much more slowly. The reason for this phenomenon has to do with caching and context switch behavior of the DBMS traffic. The 100 Mb/sec interfering traffic doubles delays for DBMS IPC traffic, which in turn requires more active threads to keep the CPU busy. In fact, the *average* number of active threads jumps from about 20 to 75. More threads, however, result in more competition for the processor cache since each time a thread is scheduled it needs to bring in its working set. Consequently, the average context switching cost skyrockets – from 17.7K CPU cycles to 69.7K CPU cycles. The result is a significant increase in CPU stalls, which increases the CPI (cycles per instruction) from 11.5 to 16.9 although the path-length does not change much. A larger cross traffic does increase the number of active threads further; however, with cache already almost thrashing, it is harder to afflict significant additional damage.

Fig 15 shows the results for low computation for the two QoS arrangements. As expected, the effect of cross traffic is more pronounced in this case. In particular, with both traffic as best effort, the throughput drops from 6 K-tps to 5.2 K-tps due to 100 Mb/sec cross traffic, a 13% drop. With high priority traffic, the drop is very severe – down to 3.4 K-tps, or a 43% drop. The greater sensitivity is obviously due to greater dependence of transactions on getting their IPCs completed in a timely manner.

Fig 16 shows the impact of affinity on performance in the presence of cross traffic. For this, we have chosen the low computation case. Since lower affinity leads to more IPC messages per transaction, we might have expected the sensitivity to *increase* as the affinity goes down. In fact, the result is just the opposite. The reason for the apparent anomaly is that lower affinity already requires more threads to keep the CPU busy (because of more communication). Thus further delays due to interference do not degrade the cache performance quite as much. Also, once the cache is on the verge of thrashing, further delays have little chance of degrading the performance further.

4 Conclusions and Future Work

In this paper, we studied the performance of clustered OLTP workloads as a function of a variety of parameters with TCP/IP over Ethernet as a unified fabric. The main result of the modeling appears to be that OLTP workloads are more sensitive to protocol overhead rather than pure end to end latency. The latency sensitivity obviously depends on computation vs. communication, and may be higher for OLTP workloads that are less computationally intensive than TPC-C; however, the sensitivity appears low enough that they may not benefit from *loaded* end-to-end latencies under a few tens of microseconds at 10 Gb/sec.

In terms of QoS issues, interfering traffic does not adversely affect the DBMS performance so long as all traffic is defaulted to be best effort or the interfering traffic is at a lower priority. However, when competing with higher priority traffic, a substantial increase in queuing delays of crucial IPC messages such as lock acquisition/release could result in significant performance loss. Thus it is important to examine QoS schemes that can minimize inter-application interference and yet provide a good performance for all. Moreover, to be really useful this should be done almost autonomically without the data center administrator doing manual setups based on detailed workload knowledge.

References

[1] P.A. Bernstein and N. Goodman, “Multiversion concurrency control — theory and algorithms”, ACM Trans on Database Systems ., 8(4):465–483, December 1983.

[2] P. Deng, “Telecom Linux performance evaluation”, Intel measurement and evaluation report, Aug 2002.

[3] R. Dimitrov and A. Skjellum, “Impact of latency on application performance”, Proc. of 4th MPI developer & user conference, Ithaca, NY, March 2000.

[4] A. Joglekar, “iSCSI Technology Investigation”, Intel measurement and evaluation report, Nov 2004.

[5] K. Kant, “An Evaluation of Memory Compression Alternatives”, Proc. of CAECW (Computer Architecture Evaluation using Commercial Workloads), Feb 2003, Anaheim, CA.

[6] K. Kant, A. Sahoo and N. Jani, “DCLUE: A Distributed Cluster Emulator”, available at http://www.ccwebhost.com/DCLUE.

[7] K. Kant, “TCP offload performance for front-end servers”, Proc. of GLOBECOM 2003, Dec 2003, San Francisco, CA.

[8] S.R. King and F.L. Berry, ”Software RDMA over TCP/IP on a general purpose CPU”, submitted for publication.

[9] T. Lahiri, V. Srihari, et. al., “Cach Fusion: Extending shared disk clusters with shared caches”, Proc. 27th VLDB conference, Rome, Italy 2001.

[10] S. Leutenegger and D. Dias, “A modeling study of the TPC-C benchmark”, ACM SIGMOD Record archive, Volume 22 , Issue 2 (June 1993), pp22 - 31

[11] J. Pinkerton, “The case for RDMA”, available at `www.rdmaconsortium.org`

[12] E. Rahm, “Concurrency and coherency control in database sharing systems”, Technical Report 1993, Institut fr Informatik, Leipzig Germany

[13] T. Shanley, *Infiniband Network Architecture*, Mindshare Inc., 2002.

[14] T. Shanley, *The unabridged Pentium 4*, Mindshare Inc., 2004.

[15] G. Regnier, S. Makineni, et. al., ”TCP onloading for data center servers”, Special issue of IEEE Computer on Internet data centers, Nov 2004 (Eds. K. Kant & P. Mohapatra).

Acknowledgements: Authors are grateful to Nrupal Jani for running the simulation and generating data for many of the cases included in this paper.