Impact of Operating System on Real-Time Main-Memory Database System's Performance

Jan Lindström, Tiina Niklander, and Kimmo Raatikainen

University of Helsinki, Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki,Finland
{jan.lindstrom,tiina.niklander,kimmo.raatikainen}@cs.Helsinki.FI

Abstract. As long as there have been databases there has been a large interest to measure their performance. However, operating system impact on database performance has not been widely studied. Therefore, this paper presents experimental results on operating system impact on database performance. Two different operating systems are studied: Linux and Chorus. Linux operating system is tested with different kernel versions and different network speeds. Chorus is used as reference point because it is a real-time operating system. Our results clearly indicate that Linux can be used as a platform for real-time main-memory databases, but the newest kernel version 2.4 should be used. Our simple experiment also confirms that the UDP gives better response time than TCP. The work done in the Linux community to reduce the long latency in the kernel has been successful and with sufficiently long request deadlines it can be used as a platform for real-time databases.

1 Introduction

Database performance is an important aspect of the database's usability. The performance of a database system depends not only on the database architecture and algorithms, but also on the platform the database is running on.

Real-time databases are needed when the database requests must be served within respecified time limits. The database is then designed to support the timely execution on all levels of the database architecture. It provides transaction scheduling, which supports priorities, deadlines, or criticality of the transactions. Alternatively they can be run on a general purpose operating system which supports real-time processes. Such functionality can be found, for example, in Solaris.

Telecommunication is an example of an application area, which has database requirements that require a real-time database or at least time-cognizant database. A telecommunication database, especially one designed for IN services [1], must support access times less than 50 milliseconds. Most database requests are simple reads, which access few items and return some value based on the content in the database.

Real-time databases have been designed for running mainly on real-time operating systems, which can provide real-time scheduling and guaranteed maximum latencies in the kernel. Previous work on real-time databases in general

has been based on simulation. However, several prototypes of general-purpose real-time databases has been introduced. StarBase [7] is constructed on top of RT-Mach. RTSORAC is implemented over a thread-based POSIX-compliant operating system and is based on an open OODB with real-time extensions [9]. Another object-oriented architecture is M^2RTSS-architecture, which is main-memory database system with real-time transaction scheduling [2].

Linux has gained popularity as a platform for web-servers and other network services. We wanted to find out if it would be suitable as a platform for a real-time database. The newest (February 2002) kernel version 2.4 supports priority-based scheduling and the latencies in kernel code have also been partially reduced.

This paper is organized as follows. Section 2 presents am overview of the main-memory databases and their requirements. Additionally, the prototype real-time database system used in experiments is shortly presented. Section 3 presents evaluation environment. Section 4 presents experimentation results. Finally, Section 5 concludes this paper.

2 Database system

In main memory database systems data resides permanently in the main physical memory. In some real-time applications, the data must be memory resident to meet the real-time constraints [3].

A network database system must offer real-time access to data [5, 6]. This is due to the fact that most read requests are for logic programs that have exact time limits. If the database cannot give a response within a specific time limit, it is better not to waste resources and hence abort the request. As a result of this, the request management policy should favor predictable response times with the cost of less throughput. The best alternative is that the database can guarantee that all requests are replied to within a specific time interval. The average time limit for a read request is around 50ms. About 90% of all read requests must be served in that time. For updates, the time limits are not as strict. It is better to finish an update even at a later time than to abort the request.

Network database system services consist of two very different kinds of semantics: service provision services and service management services. *Service provision services* define possible extra services for customers [4]. Service provision transactions have quite strict deadlines and their arrival rate can be high (about 7000 transactions/second), but most service provision transactions have read-only semantics. In transaction scheduling, service provision transactions can be expressed as firm deadline transactions. *Service management services* defines possible management services for customer and network administration [4]. Service management transactions have opposite characteristics. They are long updates which write many objects. A strict consistency and atomicity is required for service management transactions. However, they do not have explicit deadline requirements. Thus, service management transactions can be expressed as soft real-time transactions.

The prototype system used is based on the *Real-Time Object-Oriented Database Architecture for Intelligent Networks* (RODAIN) [8] specification. RODAIN Database Nodes that form one RODAIN Database Cluster are real-time, highly-available, main-memory database servers. They support concurrently running real-time transactions using an optimistic concurrency control protocol with deferred write policy. They can also execute non-real-time transactions at the same time on the database. Real-time transactions are scheduled based on their type, priority, mission criticality, or time criticality. All data in the database is stored in the main-memory database. Data modification operations are logged to the disk for persistence.

In order to increase the availability of the database each Rodain Database Node consists of two identical co-operative units. One of the units acts as the Database Primary Unit and the other one, Database Mirror Unit , is mirroring the Primary Unit. Whenever necessary, that is when a failure occurs, the Primary and the Mirror Units can switch their roles.

The client requests arrive via TCP/IP over a network directly to the database process, which contains threads to serve the clients. Each client may use the same connection for multiple transaction requests. The precoded real-time transactions get all their parameters in the requests and give their answers in the replies. No communication during the transaction execution is allowed between the transaction and the calling client.

The client and RODAIN server are originally designed to communicate over TCP. It provides a handy way for the client to recognize the failure of the server assuming that the network does not fail in between. The client can trust in the reliable communication, that the server has received each request and that it can expect to receive a reply also.

3 Evaluation environment

The database server was running on an Intel Pentium 450 MHz processor with 256 MB of main memory. A similar computer was used for the client. The computers were connected using a dedicated network, the speed of which was controlled by changing the hub connecting the computers. To avoid unnecessary collisions, there was no other network traffic while the measurements were performed.

Used database is based on a GSM model and transactions are simple one item reads to Home Location Register (HLR). Database size is 30000 items.

All time measurements were performed on the client computer using the gettimeofday function, which provides the time in microseconds. The client sends the requests following a given plan, which describes the request type and the time when the request is to be sent. When the request is about to be sent the current time is collected and when the reply arrives the time difference is calculated.

Linux provides static priorities for time-critical applications. These are always scheduled before the normal time-sharing applications. The scheduling policy chosen was Round-robin (SCHED_RR) using the scheduler function sched_setscheduler.

The database was also avoiding swapping by locking all the processes pages in the memory using mlockall function. The swap causes long unpredictable delays, because occasionally some pages are sent and retrieved from the disk. Because in our experiment environment our database system was the only application running no swapping occurred during the tests.

4 Experimentation Results

The measurements in the dedicated networks clearly show, that Linux can be used as platform for a real-time main-memory database, at least when the disk I/O is omitted and the request deadlines are suitably long (see Figure 1 for Linux and Figure 2 for Chorus).

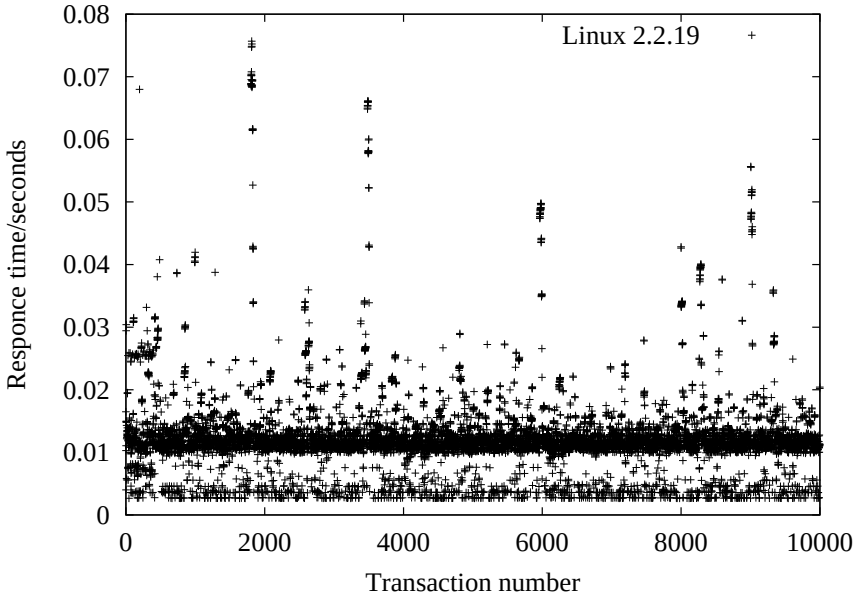

Fig. 1. The request responce times for each request using Linux kernel 2.2.19 with 10 Mb Ethernet. All the figures have as x the request sequence number over the test and as y the measured responce time in seconds.

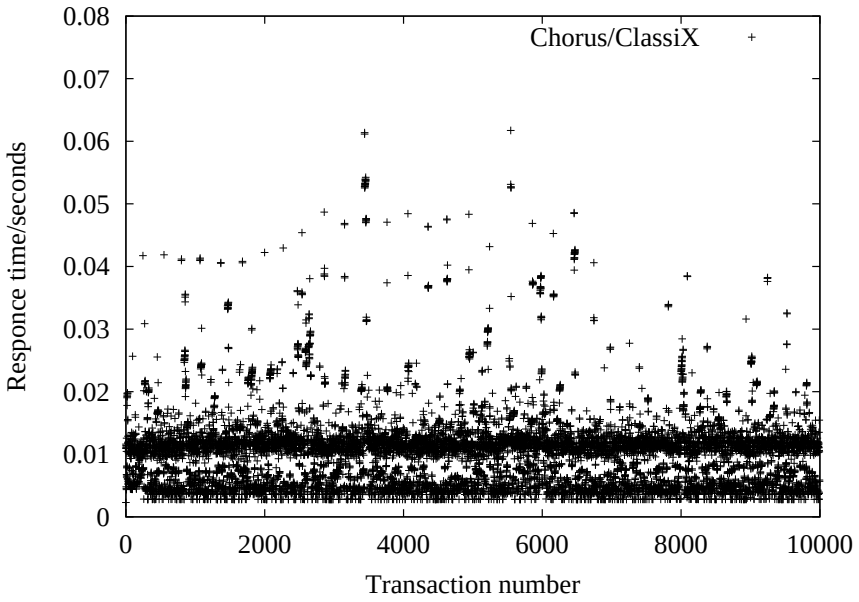

Fig. 2. The request responce times for each request using Linux kernel 2.2.19 as a client and Chorus as a server with 10 Mb Ethernet.

Our initial tests with Linux-kernel 2.2.19 were not very encouraging (see Figure 1). The occasional delays were over 50 milliseconds. Also the long delays were grouped so, that when one went over the others were also more likely to go over the assumed deadline of 50 milliseconds. The database server itself (see figure 3) used only a fraction of the observed time as seen on the client. This difference made us look more closely to the network behavior and the kernel itself. Our experience from other projects involving Linux and time requirements indicated that the change of the kernel to 2.4.x should be the first step.

Linux kernel 2.4 includes features that are designed to reduce the long time periods within kernel with the interrupts disabled. This change did the trick. The longest observed response time was slightly lower than 35 milliseconds, but most of the response times were still gathered around the 10 milliseconds line as in the 2.2 kernel experiment (see figure 4). Since 10 milliseconds is less than the required 50 milliseconds, it can be used as long as no swap to the disk is required.

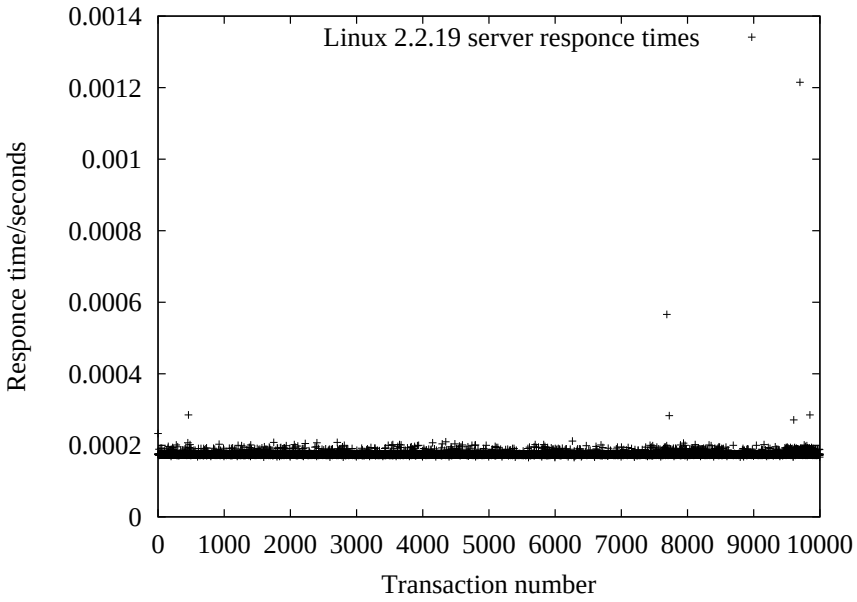

Fig. 3. The request responce times inside the database.

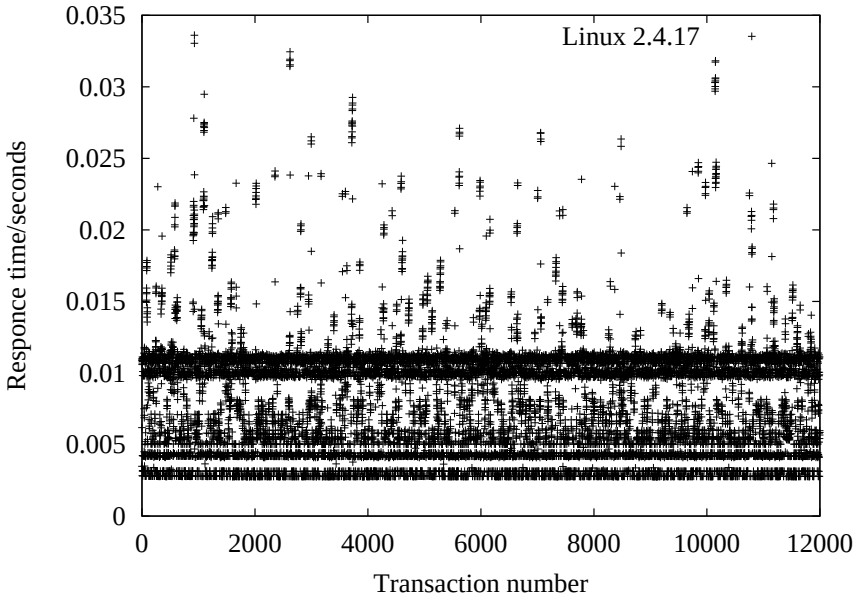

Fig. 4. The request performance time for each request using Linux kernel 2.4.17 with 10 Mb Ethernet

However, there still is the same gathering around the 10 milliseconds. Adding the low latency patch (see http://www.zip.com.au/ akpm/linux/schedlat.html) did not remove that either, but it made the response times more deterministic. All the values are on some particular level meaning some particular time value (see figure 5). This is exactly what the patch is trying to do, it tries to reduce the kernel latencies within some time limits. The distribution of the response times clearly is no longer even over all time values.

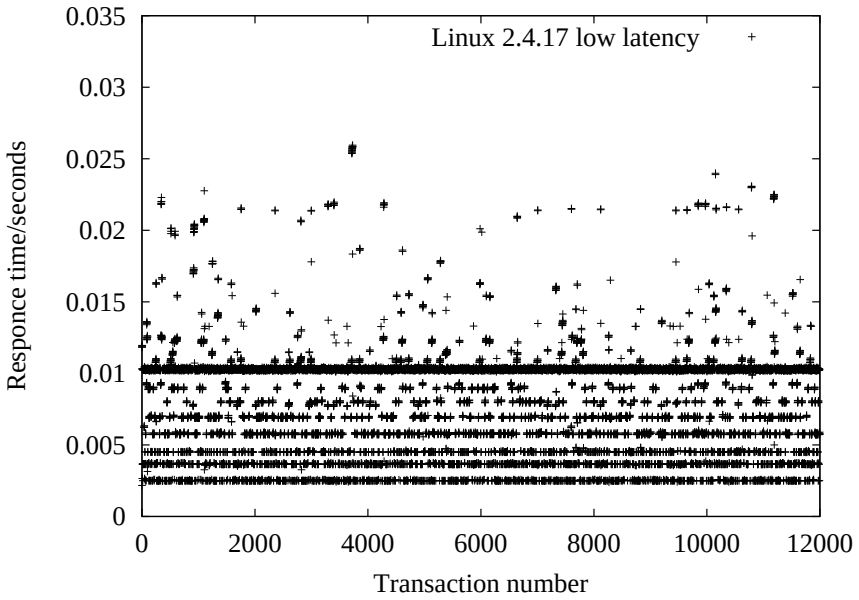

Fig. 5. The request performance time for each request using Linux kernel 2.4.17 low latency patch with 10 Mb Ethernet

The most surprising result came, when the speed of the network connection was increased (see Figure 6). We were hoping to see some reduction in the overall response times, since the network capacity was increased to almost 10 times. Some response times really dropped, but they reduced only to the nearest low 10 milliseconds. Because we used TCP as the communication mechanism between the client and the server the 20 milliseconds line is most probably due to retransmissions of messages. This gathering is mainly due to the 10 milliseconds scheduling delay usually visible on Linux.

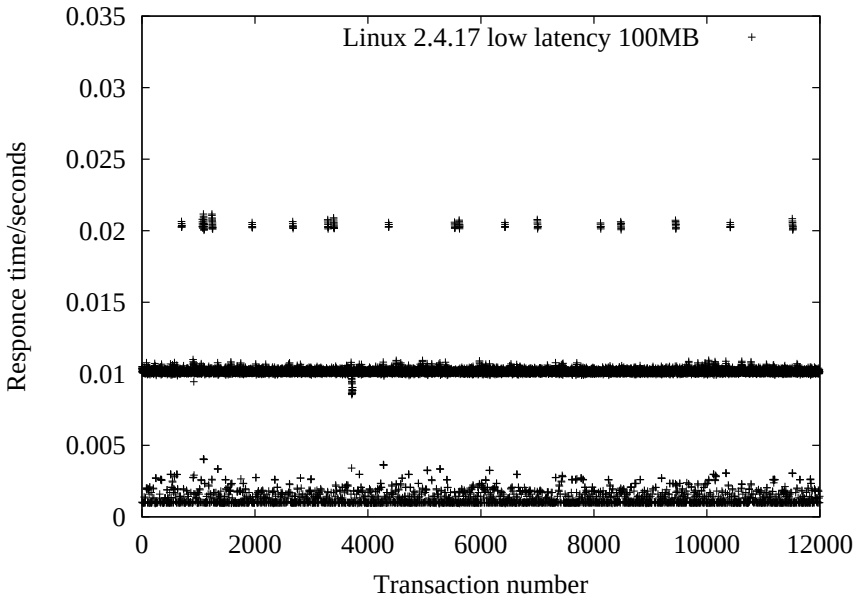

Fig. 6. The request performance time for each request using Linux kernel 2.4.17 low latency patch with 100 Mb Ethernet

We knew that the network communication was the dominating force, but the switch from using connectionless UDP communication instead of the connected TCP communication reduces all reply durations below 10 milliseconds (see Figure 7).

Of course, the number of messages drops to a third, since the TCP acks are not sent. This gives the most reduction, but it does not explain the concentration visible in the TCP measurement.

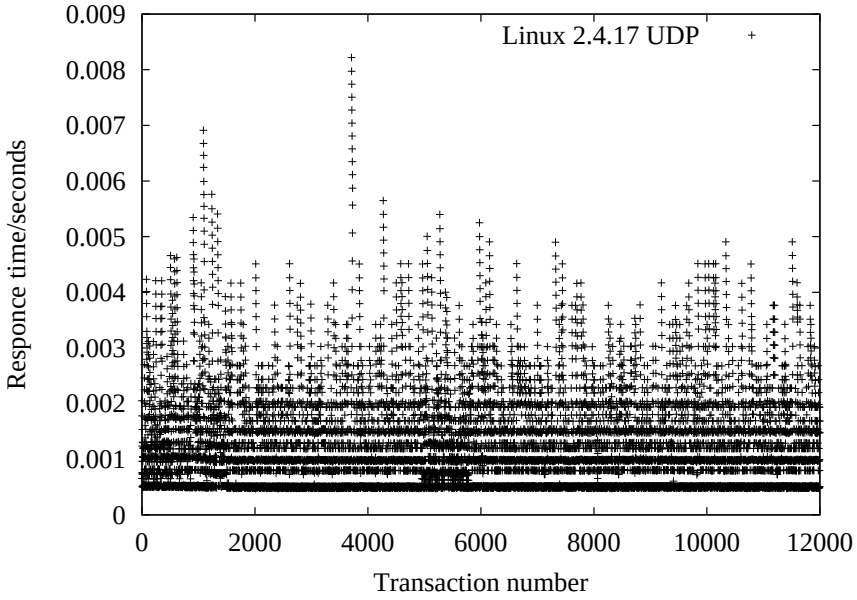

Fig. 7. The request performance time for each request using Linux kernel 2.4.17 with 100 Mb Ethernet, when the communication is connectionless

Finally, Table 1 shows minimun, maximum, median, and average responce times with different tests. Table shows also stardard deviation and variance of the responce times.

Table 1. Experimental statistics.

Test	Min	Max	Median	Average	Deviation	Variance
Linux 2.2.19 Figure 1	0.0027	0.0757	0.0116	0.0123	0.0065	4.2319-e-05
Chorus Figure 2	0.0023	0.0617	0.0109	0.0105	0.0063	3.9338e-05
Linux 2.4.17 Figure 4	0.0028	0.0336	0.0087	0.0081	0.0040	1.6035e-05
Linux 2.4.17 Figure 5	0.0022	0.0260	0.0092	0.0080	0.0034	1.1822e-05
Linux 2.4.17 Figure 6	0.000921	0.0212	0.0101	0.0086	0.0037	1.3739e-05
Linux 2.4.17 UDP Figure 7	0.00047	0.0082	0.000979	0.0012	0.00085	7.2258e-7

5 Conclusion

Linux can be used as a platform a for real-time main-memory database if the deadlines for the requests are feasible. The request must allow at least 10 milliseconds simply for the network and operating system overhead. Although the

10 milliseconds in this experimentation also covered the costs of our database, that was mainly due to the fact that there were no conflicts because all requests were simple reads.

References

1. I. Ahn. Database issues in telecommunications network management. *ACM SIGMOD Record*, 23(2):37–43, June 1994.
2. S. Cha, B. Park, S. Lee, S. Song, J. Park, J. Lee, S. Park, D. Hur, and G. Kim. Object-oriented design of main-memory dbms for real-time applications. In *2nd Int. Workshop on Real-Time Computing Systems and Applications*, pages 109–115, Tokyo, Japan, October 1995.
3. H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, December 1992.
4. ITU. *Introduction to Intelligent Network Capability Set 1. Recommendation Q.1211.* ITU, International Telecommunications Union, Geneva, Switzerland, 1993.
5. ITU. *Distributed Functional Plane for Intelligent Network CS-1. Recommendation Q.1214.* ITU, International Telecommunications Union, Geneva, Switzerland, 1994.
6. ITU. *Draft Q.1224 Recommendation IN CS-2 DFP Architecture.* ITU, International Telecommunications Union, Geneva, Switzerland, 1996.
7. Young-Kuk Kim and Sang H. Son. Developing a real-time database: The StarBase experience. In A. Bestavros, K. Lin, and S. Son, editors, *Real-Time Database Systems: Issues and Applications*, pages 305–324, Boston, Mass., 1997. Kluwer.
8. J. Lindström, T. Niklander, P. Porkka, and K. Raatikainen. A distributed real-time main-memory database for telecommunication. In *Databases in Telecommunications*, Lecture Notes in Computer Science, vol 1819, pages 158–173, Edinburgh, UK, Co-located with VLDB-99, 1999.
9. V. Wolfe, L. DiPippo, J. Prichard, J. Peckham, and P. Fortier. The design of real-time extensions to the open object-oriented database system. Technical report TR-94-236, University of Rhode Island, Department of Computer Science and Statistics, February 1994.