University of Helsinki
Department of Computer Science
Series of Publications C, No. C-2000-6

Evaluation of RODAIN-2000 Prototype Real-Time Database System

Jan Lindström, Tiina Niklander, and Kimmo Raatikainen

Helsinki, February 29, 2000

Report C-2000-6

University of Helsinki
Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, FINLAND

Evaluation of RODAIN-2000 Prototype Real-Time Database System

Jan Lindström, Tiina Niklander, and Kimmo Raatikainen

Department of Computer Science, University of Helsinki

Report C-2000-6

February 29, 2000

20 pages

Abstract. Real-time databases will be an important part of the future telecommunications infrastructure. The requirements for databases in telecommunications include real-time data access, fault tolerance, object orientation, efficiency, flexibility, multiple interfaces, and compliance with telecommunications standards. An object-oriented real-time database in which the needs of telecommunications applications are favored over generality is the basic design principle of the RODAIN database system.

The Distributed RODAIN Database Architecture supports data partitioning using multiple RODAIN Database Nodes. Transactions on each node can access data in the whole distributed database, but may update only objects in the local node. This approach is enough for most of the telecommunication applications, where the updates can be directed to a specific database node.

The fraction of transactions accessing data in remote partitions is relatively small, if the partitioning is based on the data usage. The RODAIN Database can support up to 20 % of read transactions accessing remote data without reducing the database performance too much. The transactions needed in the intelligent network concept are relatively simple. Hence, the access time needed for reading the remote data dominates the transactions' execution time and reduces the number of transactions executed within any given time period.

Key Words: Real-Time Databases, Object-Oriented Databases, Concurrency Control, Real-Time Scheduling, Fault-Tolerance

CR Classification: C.3, H.2, H.3, J.7

During the lifetime of the RODAIN project this report is internal to the project.

Contents

1 Introduction

Databases will have a dominant role in future telecommunication services. Databases will hold the information needed in the operations and management of telecommunication services and networks. The performance, reliability, and availability requirements of data access operations are demanding. Thousands of retrievals must be executed in a second. The allowed down-time is only a few seconds in a year.

RODAIN[1] is an architecture for a *real-time*, *object-oriented*, *fault-tolerant*, and *distributed* database management system [Kiviniemi *et al.* 1997],[Niklander *et al.* 1997]. The RODAIN architecture is designed to fulfill the requirements derived from the most important telecommunications standards including *Intelligent Network* (IN), *Telecommunications Management Network* (TMN), and *Telecommunication Information Networking Architecture* (TINA). The requirements of the telecommunications database architectures originate in the following areas: real-time access to data, fault tolerance, distribution, object orientation, efficiency, flexibility, multiple interfaces, and compatibility with telecommunications standards [Raatikainen 1997],[Taina and Raatikainen 1996].

The RODAIN Database Architecture consists of multiple RODAIN Database Nodes. One RODAIN Database Node consists of Database Primary Node, Database Mirror Node, and a Secondary Storage Subsystem. The Secondary Storage Subsystem is assumed to be any fault-tolerant disk storage with a RODAIN specified interface. The *Database Primary Node* executes the application requests whereas the *Database Mirror Node* maintains a copy of the main memory database. The mirror node also takes care of storing the transaction log into the Secondary Storage Subsystem. The Database Primary Node and the Database Mirror Node have identical subsystems. However, the functionality of the subsystems are not always the same.

In the rest of the paper we concentrate on the distribution aspect of the RODAIN Database Achitecture, because the performance of one RODAIN Database Node has been reported in Kiviniemi *et al.* [1998]. The need for distribution simply arises from the telecommunication environment. Sometimes there is a need to bring parts of the database closer to the database users. Sometimes the amount of data is too large to fit into the memory of a single machine. Since the RODAIN Database is a main-memory database, the only way of using all the data is to use multiple machines, each with its own memory.

The current implementation provides a limited distribution, which is suitable for most of the needs in telecommunication. Most of the data accesses are reads. Thus we restrict the data distribution

[1]RODAIN is the acronym of the project name *Real-Time Object-Oriented Database Architecture for Intelligent Networks* funded by Nokia Telecommunications, Solid Information Technology Ltd., and the National Technology Agency of Finland.

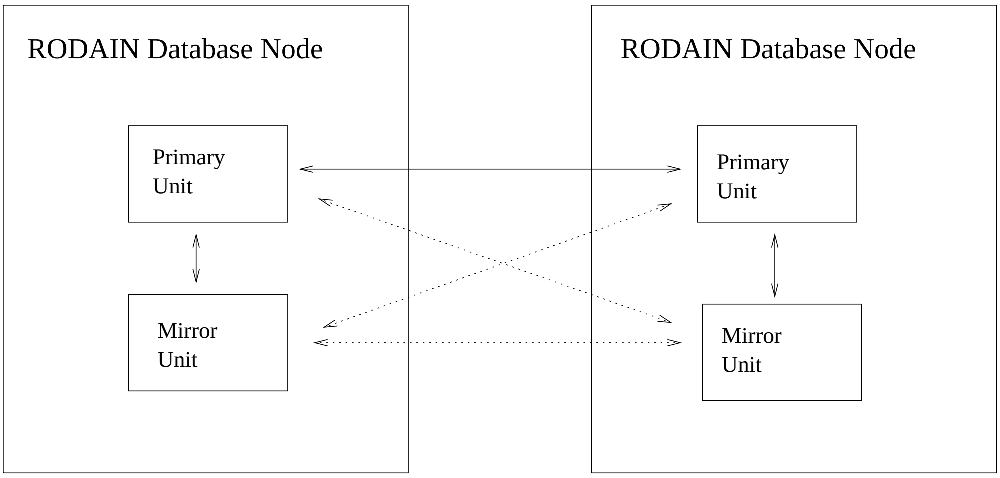


Figure 1: RODAIN 2000 Prototype

for only read operations. Any transaction may read remote data, but it can only update data that is local to its execution site. The remote data is fetched to a local cache for access. While the data is validly in the local cache, all other local transactions needing it can access it directly in the cache. A distributed transaction must be committed before the validity period of the cached data is over. If a transaction fails to do so, it must be aborted.

The rest of the paper is organised as follows. In Section 2 we describe the architecture of RODAIN Database used in the experimentation and evaluation. Section 3 describes the parameters and metrics used in the evaluation. The rest of the sections describe the results of different experiments. In Section 4, the remote database node executes no transactions of its own, it just serves object access requests from other database node. In section 5, both database nodes execute equal transactional loads accessing both local and remote objects. The results of experimenting with the full system logging turned on, both nodes using both Primary and Mirror Node, are reported in Section 6. Finally, Section 7 presents general discussion of the results and some future development aspects.

2 Architecture of the RODAIN 2000 Prototype

The RODAIN 200 Prototype (see figure 1) used in these experiments consists of two RODAIN Database Nodes. These nodes contain separate parts of the whole distributed database. The database has one metadata, which the nodes share. One node may only read, but not update, objects located on the other node. Each RODAIN Database Node has two units. The Primary Unit is for database operations and the Mirror Unit for log processing and fault tolerance.

For most of the experiments, we omitted the logging of transaction operations. Thus, the Mirror

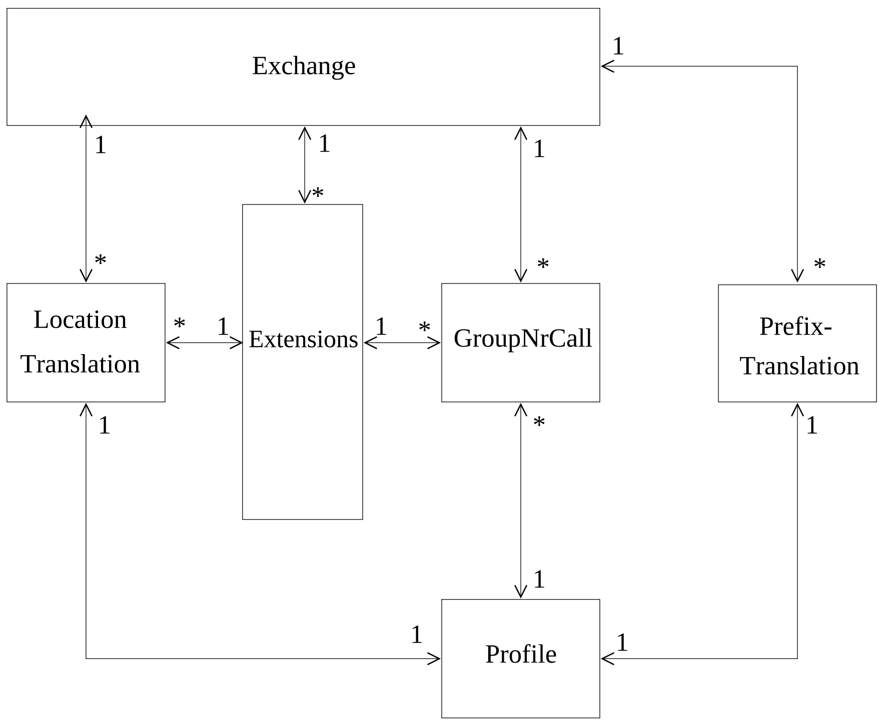


Figure 2: Object model schema of test database.

Units were not present on either Database Node. The logs were omitted simply to reduce the parameters in the test sets. The omission of logging only reduces the commit time of write transactions. Since our main focus is on the distribution aspects and the distributed transactions are read-only, this does not have large effect on the results. We ran the third and final experiment set with the logging turned on using also Mirror Units.

We measured the results only on one RODAIN Database Node. This node is later referred to as the local node. The other RODAIN node was used as the server of the remote data and, thus, called remote node. The test were executed in three sets. In the first set of experiments the remote node had no local load. It only served the incoming requests. In the second set of experiments the remote node and local node had identical loads. They both executed transactions and served requests of object reads arriving from the other node. The first set of experiments was similar to the second one, except that the Mirror Units were in use. The Primary Units generated logs to the Mirror Nodes, which in turn updated their database copies based on the arriving logs.

The objects in the database were designed according to the IN service 'Virtual Private Network'. The object classes and their relationships are presented in figure 2. A more detailed description of the classes can be found in [Lindström *et al.* 1999].

Transaction	Type
get_profile	r
translate_number	r
modify_profile	w
modify_virtual_network	w
search_group	r
update_group	w

Table 1: Test transactions

We used the profile class as the access point to the database. All transactions must first access the corresponding profile and then they may also access other objects, if needed.

The transactions used in these experiments (see Table 1) access the objects in different classes for different purposes. All transactions are fully implemented as part of the transaction execution environment (TRP). For test purposes, the transactions receive the OID of the object they need to access during their execution. This models the request from unexisting client. The transaction arrival order is generated off-line for test purposes.

The most simple transaction is a read-only transaction *get_profile* that reads customer profile information. It gives the OID of the profile needed and receives the profile information. We use OID instead of the true number because the test database does not yet contain an index structure for secondary keys.

Another read-only transaction, called *translate_number*, implements a redirection of the logical number. It first reads the Profile of the object. And then it accesses the PrefixTranslation. This way it can determine the correct destination.

In order to test and evaluate the conflict resolution mechanism we also need to update transactions. The simplest update transaction, called *modify_profile*, is an update that changes one customer profile information.

3 Test Metrics

All test sets used the same database and identical data access ratios for all transactions. The test parameters are in table 2. Each database was assumed to contain 500 objects in profile class. All data accesses were done on this data area. This number of objects seemed to be suitable for the tests, since it generated some conflicts for the concurrency controller, but the concurrency control aborts were still not the dominating reason in transaction aborts.

Dimension	Scale
Data access distribution	500 objects
Data locality in read ops	50 to 100
Write ratio	0 to 50
Transaction arrival rate	100 to 500

Table 2: Test parameters

For the tests we used only two transactions *translate_number* and *modify_profile*. Each test set consisted of 10000 transactions. The set was repeated 20 times. The ratio of write transactions was varied. Likewise, the ratio of read transactions accessing remote data instead of local data was varied.

The write transaction (*modify_profile*) always accessed only local objects. The read transaction (*translate_number*) may also read remote objects. We felt that, using only these two transactions, we were able to get a great deal of information out of the system behavior without increasing the overall complexity. We use three different groups of transactions $write$, $localRead$ and $remoteRead$.

(1)
$$\text{Total} = write + localRead + remoteRead$$

(2)
$$\text{Write ratio} = \frac{write}{Total}$$

(3)
$$\text{Ratio of local reads} = \frac{localRead}{localRead + remoteRead}$$

In the results we always report only write ratio and the ratio of local reads. Thus, the $localReads$ does not give the ratio of local transactions over all transactions. This was due to the fact that the current system does not support remote write operations. Therefore, we omit the local writes from the distribution.

For all test sets, we calculated the number of transactions sent for execution ($Total$), number of actually executed transactions ($Entered$), and number of aborted transactions ($Aborted$). From these numbers we calculated three values: overload fraction, transaction miss ratio and fraction of abortions compared with executed transactions. The definitions for the fractions are:

(4)
$$\text{Overload fraction} = \frac{Total - Entered}{Total}$$

Writes	Reads	Rem Reads	Miss ratio	Overload	aborted/entered	deadline aborts
0	100	20	0.0463	0.00914	0.0375	0.001357
10	90	18	0.03561	0.005995	0.02979	0.001162
20	80	16	0.0282	0.004055	0.02424	0.0008786
30	70	14	0.02441	0.00293	0.02155	0.0008926
40	60	12	0.01327	0.00092	0.01236	0.0006006

Table 3: Effect of write ratio using remote node with static load, when arrival rate is 350 trans/s and remote read ratio is 20 %

$$\text{Miss ratio} = \frac{Aborted}{Total} \tag{5}$$

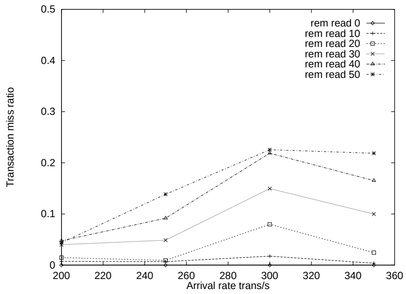
$$\text{Aborted/Entered} = \frac{Aborted}{Entered} \tag{6}$$

We also grouped the abortions based on their causes. Thus, we sometimes report values that instead of using the total amount of aborts uses only a portion of the aborts, like aborts due time limit ($deadline\ aborts$) or due concurrency conflict resolution ($cc - aborts$).

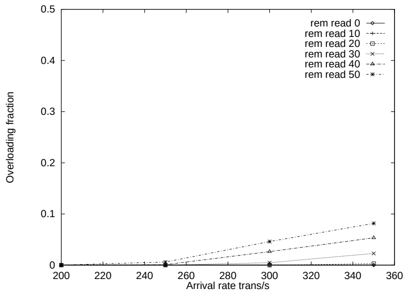
Based on the assumed load to the system the writes and remote reads are quite rare. If their ratio stays below 10 to 20 % they do not have much effect to the overall system performance. In this report we thus mainly describe the results of such experiments, where either one is fixed to 20 %.

To visualize the effect of the write and read ratios we collected a sample of test metrics to Table 3. The system behavior seems peculiar if we only look at the write ratio. The increase in the write ratio actually increases the system throughput, which cannot be. The explanation is in the actual number of remote read transactions. Although we keep the ratio of remote read over all reads constant, the ratio of remote reads over all transactions decreases when the write ratio increases. Since in this experimentation, we do not use logging, the execution time of remote read transaction is much longer than the execution time of either local read or local write transactions. The network communication is the dominating force in the execution time of the remote read transaction.

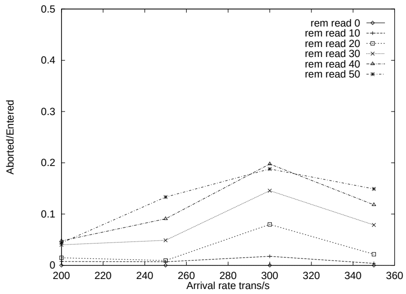
To visualize the results of the test experiments we present an example of each measurement in figure 3. The results are collected from a test set, where we used a single unit, no Mirror, on both local and remote nodes. The measurements are collected from the local node, where the arrival rate and the ratio of locally vs. remotely accessed data is varied. The remote node was running a fixed load of 100 transactions per seconds. The transactions accessed local data on remote note sequentially using the oids as the ordering mechanism. Thus, the object access was equally distributed over the whole local database. There was no randomness in the behavior of the remote node.



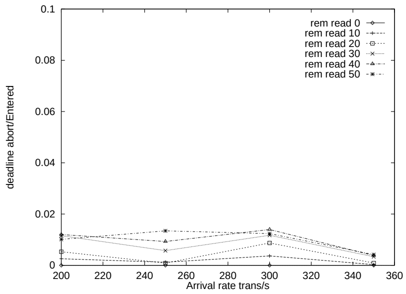
(a) Transaction miss ratio



(b) Overloading fraction



(c) Aborted transactions / Entered transactions

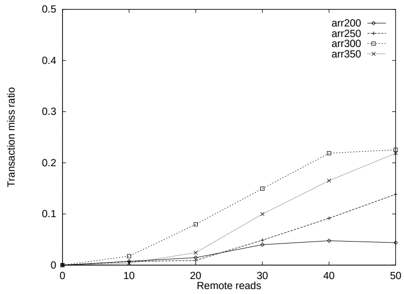


(d) Deadline aborts / Entered transactions

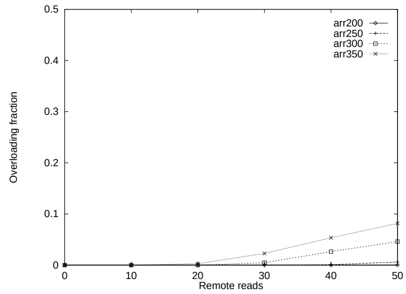
Figure 3: Results of experiments with no mirrors using static load on remote node, where the write ratio on the local node was fixed at 30 %, the arrival rate varied between 200 and 350 arriving transactions per second, and the ratio of locally accessing read transactions varied between 50 % to 100 % using the arrival rate as X scale.

Most of the trends are visible in Figure 3. The transaction miss ratio increases with the increase in the transaction arrival rate. It is expected since more concurrency means more conflicts leading to more aborts. The concurrency controller is really the reason for the aborts as can be seen in the subfigure 3 (c). It is not the only reason. The system overload also increases the miss ratio. A turning point in the system behavior is 250 transactions per second. When the arrival rate increases more, the system starts discarding transactions due to overload and this increases the transaction miss ratio.

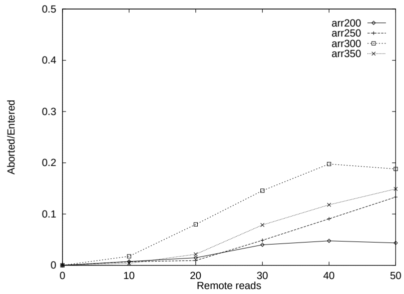
The aborts due to the transaction missing its deadline are not affected by the system load. Hence, the overload mechanism seems to work quite well. In this experiment it keeps the deadline aborts around one out of a hundred.



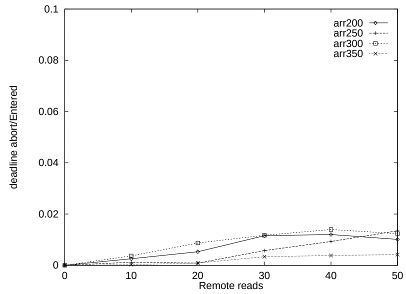
(a) Transaction miss ratio



(b) Overloading fraction



(c) Aborted transactions / Entered transactions



(d) Deadline aborts / Entered transactions

Figure 4: Results of experiments, where the write ratio was fixed at 30 %, the arrival rate varied between 200 and 350 arriving transactions per second, and the ratio of locally accessing transactions varied between 50 % to 100 % using the access ratio as X scale.

The results in Figure 3 can be plotted using the ratio of locally accessing transactions as the X-scale and plotting each arrival rate on a separate curve. The figure 4 presents the same experiment but gives a completely different impression of the results. It emphasizes the effect of data locality. With local data, the system can serve almost all arriving transactions successfully. When the data locality decreases, the system aborts more transactions.

Most of the transactions aborts (figure 4 (c)) performed by the concurrency controller are not due to direct data conflicts, but due to remote data being too old. The concurrency controller uses the data validity period in its decisions. If the validity period of the read data has expired, the committing transaction is aborted. This abortion is counted as done by the concurrency controller. The reasons for the concurrency control aborts are not categorized more specifically.

The *deadline aborts/entered transactions* stays quite small. This simply indicates that the number of deadline aborts is not the primary source for the system behavior. It only indicates that the overload prevention used in the RODAIN database system works by keeping the ratio of transactions that miss their deadlines quite stable. Since this value does not describe the system behavior based on different parameters or configurations, we do not report it in the later tests.

4 No load on remote node

In this test set we wanted to find out the behavior of the system, when the remote node has no local transactions. All it has to do is to serve incoming read requests from the other node. The measurements were made on the local node only. We used the largest variance of parameters in these tests.

For the general overview of the system performance we present two different tables. Table 4 gives the results of experiments where the arrival rate of transactions was fixed to 350 trans/s and the ratio of write transactions was 20 %. We can see that the increase in the ratio of remote read transactions increases the overall miss ratio of transactions. Some of these are the remote reads themselves, since they do not always have enough slack time for the remote object fetch. Mainly the increase results from the longer execution periods and the restriction in the number of concurrently allowed transactions.

Table 5 gives similar results based on the variance in the write ratio, when the ratio of remote reads is fixed to be 20 % of all read transactions. The effect of increase in the write ratio decreases the miss ratio, which seems quite an odd behavior. Although quite an alarming result, it has quite a simple explanation. This is mainly due to the fact that the increase in the ratio of write transactions decreases the overall ratio of the remote read transactions, since the ratio of read transaction decreases and the remote reads are the ratio of all read transactions. Since the execution time of the local write transaction is shorter that the execution time of remote read, this result is quite natural.

The effect of the arrival rate is presented in Table 6. As can clearly be seen, the system can easily tolerate 300 transactions per second or less. The overload abort is manageable also with 350 transactions per second. We could expect to see an increase in the transaction miss ratio when the arrival rate increases. Thus, the decrease in 250 transaction per second may need some explanation. The most probable cause are the cache hits. With a lower arrival rate the transactions accessing the same remote objects arrive too far apart to be able to benefit from the objects already in the cache.

In Figure 5 is printed all the measurements of the same situation where the transaction arrival rate is fixed to 350 transactions per second. We used different graphs presenting different write ratios on

Remote reads	Miss ratio	Overload fraction	aborted/entered
0	0	0	0
10	0.008595	0	0.008595
20	0.03887	0.00437	0.03466
30	0.1025	0.02089	0.08339
40	0.1992	0.0528	0.1545
50	0.2879	0.09036	0.2171

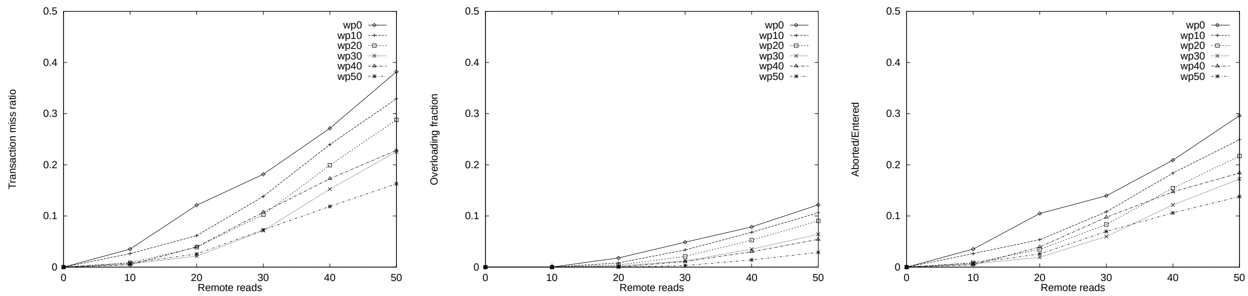
Table 4: The effect of the remote read ratio using an empty remote node, when the arrival rate is 350 trans/s and the write ratio is 20 %

Write ratio	Miss ratio	Overload fraction	aborted/entered
0	0.121	0.01805	0.1048
10	0.06161	0.008335	0.05372
20	0.03887	0.00437	0.03466
30	0.0215	0.002305	0.01924
40	0.03966	0.00032	0.03935
50	0.02582	0	0.02582

Table 5: The effect of the write ratio using an empty remote node, when the arrival rate is 350 trans/s and the remote read ratio is 20 %

Arrival rate	Miss ratio	Overload fraction	aborted/entered
100	0.02589	0	0.02589
200	0.04138	0	0.04138
250	0.01363	0	0.01363
300	0.04727	0.000225	0.04705
350	0.03887	0.00437	0.03466
400	0.1852	0.07007	0.1238
450	0.3106	0.1937	0.145
500	0.3857	0.2794	0.1475

Table 6: The effect of the arrival rate using an empty remote node, when the write ratio is 20 % and the remote read ratio is 20 %



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

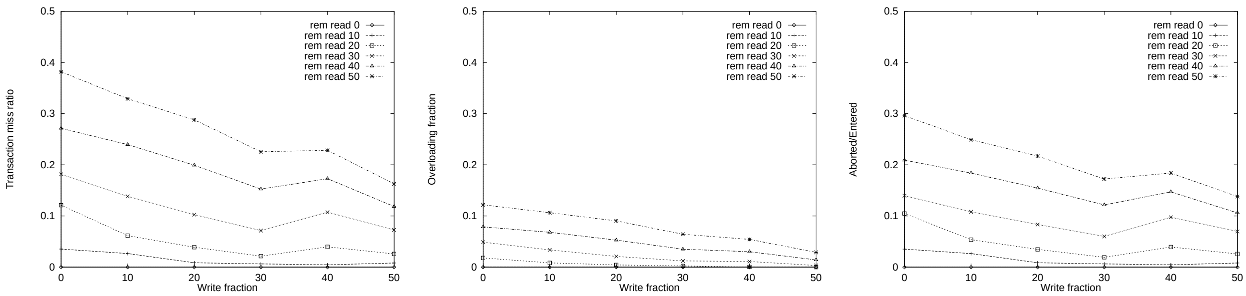
Figure 5: No load on remote node and fixed arrival rate 350 tr/s

the same figure using the local data access ratio as the X scale. From this figure, we can see that the increase in the write transactions increases the transaction miss ratio. This is mainly due to the fact that the execution time needed for write transactions is longer than for read transactions. The reduction in the locality of data has the same effect.

The transaction miss ratio figure is quite interesting as it seems to indicate that the miss ratio decreases with the increase of writes. This is true in the sense that the increase in writes increases the total number of local transactions. Since our write transactions are quite simple, their execution time is much smaller that the execution time of remote read operation. The increase in the number of transactions performing remote reads is the actual cause behind this quite unusual behavior.

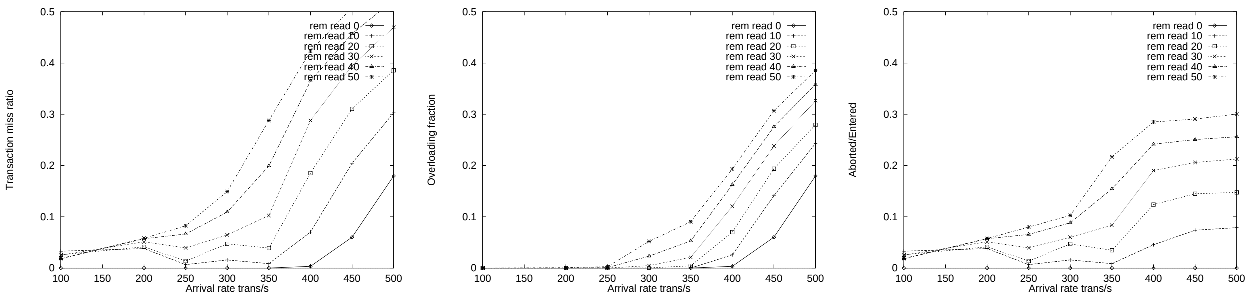
Most of the aborts are also due to the remote data access. The objects fetched from the remote node are stored in the local cache for their validity period. In our tests, most of the transaction aborts were done by the concurrency controller, when it encountered a committing transaction, which has accessed remotely owned data, whose local copy is not valid any longer. The concurrency controller aborts all these transactions, since it cannot be certain of the database consistency anymore. The data validity is simply based on the validity window given to the object when it is fetched.

The information of the system behavior in Figure 5 can be presented in a different set of figures (Figure 6), where the graphs are based on the local read ratio and the X scale presents the write ratio. The same trends are also visible in these figures. The distances between different curves are larger than in the figure where the X scale and curve indicator were switched. This indicates that the ratio of locally vs. remotely read data is more important in the overall system performance than the difference between locally read vs locally written data. This was the result expected, since the access of remote data takes so much longer compared to the local access. The remote access requires two



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 6: No load on remote and fixed arrival rate 350 tr/s



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 7: No load on remote and fixed arrival rate 350 tr/s

messages between the nodes: one for request and the other for reply. In addition to these messages, the data must be accessed locally in the remote node and then again in the requesting node.

The effect of the arrival rate is visible in Figure 7, where the write ratio is fixed at 20%. From this figure we can see that the 250 transactions per second is a turning point in system overload. Below it the overload is quite small. Above it more and more transactions are not accepted for execution due the current system load.

5 Equal Load

In this section, we have tested a system where both local and remote database nodes behave symmetrically. This was achieved by using identical parameters in both local and remote node load

Arrival rate	Miss ratio	Overload fraction	aborted/entered
100	0.08417	0	0.08417
200	0.03739	0	0.03739
250	0.09232	0	0.09232
300	0.0937	0.000065	0.09364
350	0.1252	0.02037	0.107
400	0.2294	0.1191	0.1252
450	0.3316	0.2246	0.138
500	0.4248	0.3305	0.1409

Table 7: The effect of the arrival rate using an equal load on both nodes, when the write ratio is 20 % and the remote read ratio is 20 %

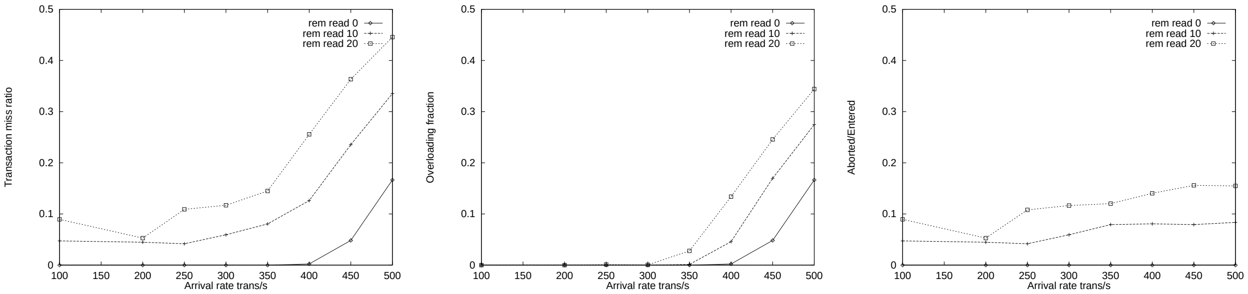
Write ratio	Miss ratio	Overload fraction	aborted/entered
0	0.1725	0.03943	0.1385
10	0.1449	0.02789	0.1203
20	0.1252	0.02037	0.107
30	0.1053	0.01357	0.09297
40	0.08907	0.005795	0.05583
50	0.05982	0.002985	0.05701

Table 8: The effect of the write ratio using an equal load on both nodes, when the arrival rate is 350 trans/s and the remote read ratio is 20 %

descriptions. Both nodes had the same arrival rate of transactions. The transactions were characterized using the same read/write ratio and local/remote ratio in both nodes. The object access was equally divided on all objects in the global database.

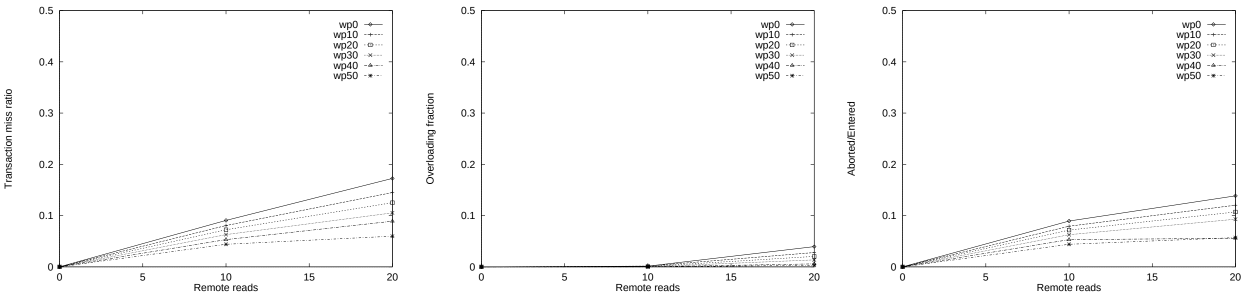
We chose to use values that were more within the range of the system performance without generating too heavy an overload, and that also described the assumed behavior of an IN database. Thus, most of the accesses were assumed to be local and only few of those were assumed to be writes. Therefore, the local access rate varies between 80 % and 100 % and the write ratio varies between 0 % and 50 %.

For comparison with the results in the previous chapter we present the identical information in Tables 7 and 8 using 350 transactions per second as the arrival rate and 20 % as the ratio of both write transactions and remote read transactions.



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 8: Equal load on local and remote nodes using write ratio 10 %.

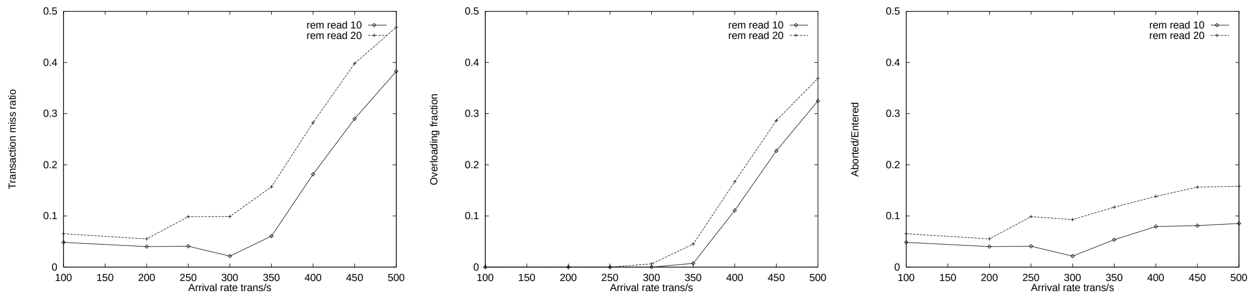


(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 9: The equal load on local and remote nodes using write ratio 10 %.

In Figure 8 we have varied the arrival rate of transactions in both nodes from 100 to 500 transactions per second. It presents the situation in a local node. From the subfigures we can see that the system misses very few deadlines up to 350 transactions per second. The turning point is clearly dependent on the fraction of the remote reads. When the all reads are local, the turning point is at 400 transactions per second. When fraction of local reads is 80 %, the turning point is at 200 transactions per second. After that point, the system becomes overloaded. Again, these results show that the fraction of remote reads should not be more than 10 %.

In Figure 9 we have fixed the arrival rate of the transactions to 350 transactions per second and we have varied the fraction of write transactions from 0 % to 50 %. From the subfigures we can see that the write fraction does not effect system performance much. The fraction of remote reads effects the system performance more than write fraction.



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 10: The full system with the write ratio fixed at 10%.

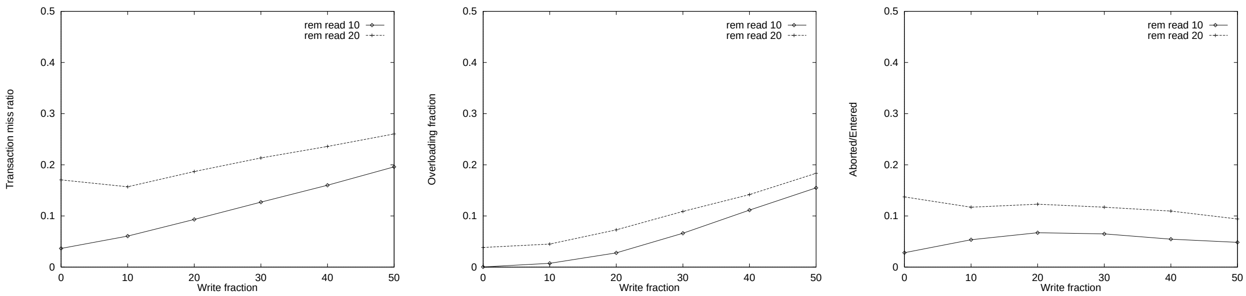
6 Equal Load on Database Nodes using Mirror Units

As the final experimentation we executed some tests on the whole architecture, where both nodes used both Primary and Mirror Units. We concentrated on the overall behavior of the system. Once again, we measured the performance only on one Primary Unit, called local node. This Primary Unit stored its logs to the accompanying Mirror Unit, which used the logs to maintain its copy of the database in its memory. The remote RODAIN Node, Primary and Mirror Units, behaved similarly.

For test purposes, we ran equal loads on both nodes. This makes the results of this experiment comparable with results of the tests in the previous chapter.

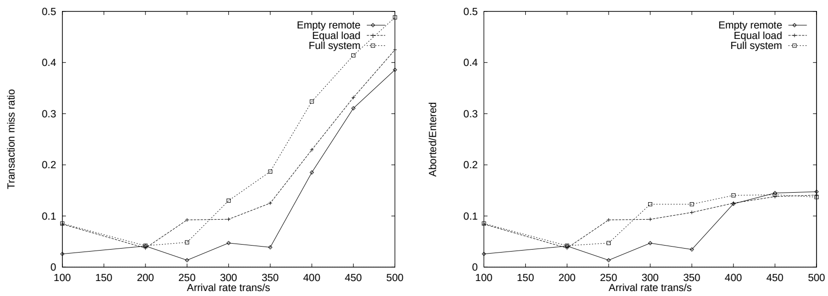
In Figure 10, we have varied the arrival rate of transactions in both nodes from 100 to 500 transactions per second. From the subfigures we can see that the system misses very few deadlines, up to 300 transactions per second. The turning point is clearly dependent of the fraction of the remote reads. When the fraction of remote reads is 10 %, the turning point is in 350 transactions per second. When the fraction of remote reads is increased to 20 %, the turning point goes to 200 transactions per second. After that point, the system becomes overloaded. Again, these results show that the fraction of remote reads should not be more than 10 %. If fraction of remote reads is more than 20 %, the system misses more and more of the transaction deadlines. This is because reading a remote object takes too long.

In Figure 11 we have fixed the arrival rate of the transactions to 350 transactions per second and we have varied the fraction of write transactions from 0 % to 50 %. From the subfigures we can see that the write fraction does have more effect on the system performance than it did in the previous experiments. This is because logs generated by write transactions are now sent to the Mirror Unit. Delivering logs from Primary to Mirror takes some time, which can be seen as increasing miss ratio.



(a) Transaction miss ratio (b) Overloading fraction (c) Aborted transactions / Entered transactions

Figure 11: The full system with arrival rate 350 tr/s.



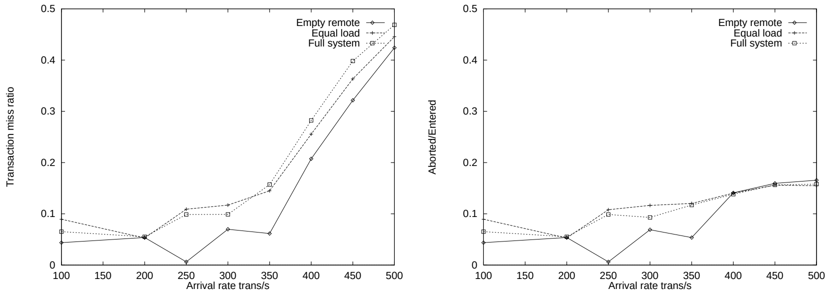
(a) Transaction miss ratio (b) Aborted / Entered trans

Figure 12: A comparison of different test sets with 20 % write ratio and 20 % remote read ratio.

The increase in the transaction miss ratio is mainly based on the system overload. The overload management policy allows the accepted transactions to be finished succesfully. Instead an increasing number of transactions is rejected due to maximum number of allowed concurrent transactions. When each transaction takes longer to commit either due waiting for remote data object or waiting for log writing it occupies a transaction position and hinders some other transactions from arriving for execution.

7 Discussion and Future Development

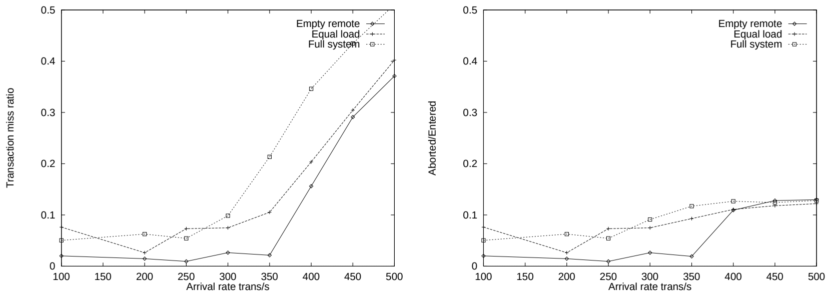
The system performance (see figures 12, 13, 14, and 15) in the three different test configurations is quite natural. The highest overall performance, the lowest miss ratio, is reached with the system, where the remote node is ready to serve all access requests immediately. This was assumed since



(a) Transaction miss ratio

(b) Aborted / Entered trans

Figure 13: A comparison of different test sets with 10 % write ratio and 20 % remote read ratio



(a) Transaction miss ratio

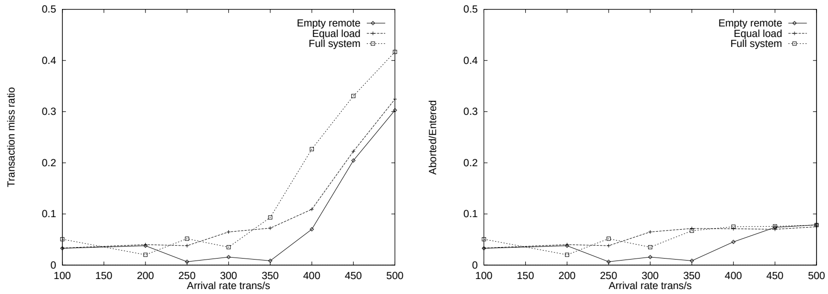
(b) Aborted / Entered transactions

Figure 14: A comparison of different test sets with 30 % write ratio and 20 % remote read ratio

the overall system load is lowest when all transactions are executed in one node, because the node can tolerate only a fixed amount of load.

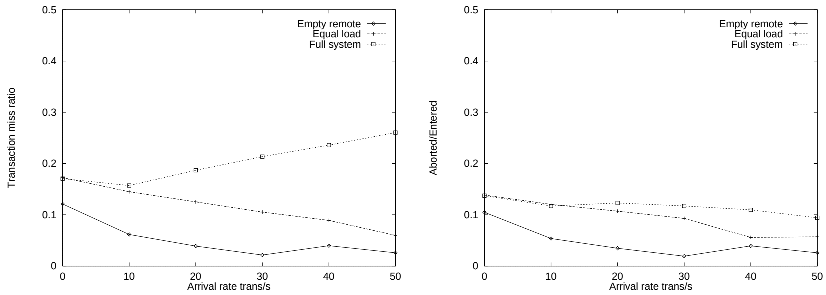
The difference between a system without logging and a system with logging is noticeable, but not significant, when the write ratio is in the assumed load area lower than 20 %. Most of the difference comes from the longer commit time needed for write transactions. This increases the overall execution time and reduces the number of transactions that can be executed within a given time frame.

When the ratio of write transactions increases as in Figure 16 the database system with logging clearly performs worse than the other system configurations. The overload management reduces to accept more transactions than the overall system can handle creating a slight increase in the transaction miss ratio. The increase is reduced by the fact that the increase in the number of write transactions decreases the overall number of read transactions, especially the ones accessing



(a) Transaction miss ratio (b) Aborted transactions / Entered transactions

Figure 15: A comparison of different test sets with 20 % write ratio and 10 % remote read ratio



(a) Transaction miss ratio (b) Aborted / Entered transactions

Figure 16: A comparison of different test sets with 350 tr/s arrival rate and 20 % remote read ratio

remote data objects. This is very visible in the behavior of the test systems not using logging. The average execution times of a read transaction making a remote data read and a write transaction, whose updates are logged to a remote node, are quite close, since the network communication time dominates the transaction execution.

The dominating force in the system performance seems to be the remote data access rate. Although anticipated, its effect is very clear. The limited distribution with read-only access to remote data is not enough for all situations. The possibility of remote writes should be added. It will probably reduce the system performance even more, but it is necessary and cannot be avoided in all situations.

The current tests were run using fixed validity duration for all objects in the cache. It would be necessary also to evaluate the effect of this duration: how does it behave with the other parameters.

In this version of RODAIN implementation there are several points which should be considered for change in future development. The STL map data structure found in OCC_ORD Transaction_Table and Object_Table, HOT Object_Table, and Cache Object_Table should be changed to hash data structure (extensible or not). The list data structure used in RTC should also be changed to some more efficient data structure.

Mutual exclusion is made in interfaces. This can be changed to data structure mutual exclusion if needed.

TRP should be considered for implementation with one process and several threads.

8 Conclusions

Based on these experiments, we feel that the overall architecture of RODAIN Database Prototype is well suited to a system, where most of the transactions are local reads and only a small portion of the transactions are either remote reads or local writes. Hence, the architecture can be used in telecommunication where these types of system exist.

We also noticed that the succesfull execution of the transactions depends on the tightness of the deadline. In the tests the system was often fully loaded and some of the transactions were aborted due to missing their deadlines. This indicates that if the deadline is fairly tight and the system cannot serve it immediately, then the transaction may not be able to finish on time. This happened quite often for the read transactions that needed to access remote objects. If we want to keep the distribution hidden from the database user, then all possibly distributed transactions must have slack time enough for the distributed data access. The system's overall performance is quite sensitive to this parameter. If the deadline is too tight, then most of the transactions that really need the remote data are aborted due to missing their deadlines. This is something of which the database designer must be aware.

The increase in transaction execution times due to the remote object access is the main cause for the increase in the transaction miss ratio. The overload management by limiting concurrency seems to work well also with distributed transactions. This is seen in the stability of the abort / entered ratio. It remains quite stable although the miss ratio increases as more and more transactions are not accepted for execution.

The experiments with the full system show that the chosen logging mechanism does not reduce the overall performance too much. This level of reduction cannot be avoided since the logs must be stored in the remote node and the guaranteed storing requires one message round trip. The reduction in the system performance is mainly due to the increase in transaction execution time forcing the overload manager to reject transactions.

References

KIVINIEMI, J., LINDSTRÖM, J., NIKLANDER, T., PORKKA, P., AND RAATIKAINEN, K. 1998. Evaluation of RODAIN Prototype Real-Time Database System. Tech. Report C-1998-2, University of Helsinki, Department of Computer Science, Finland.

KIVINIEMI, J., NIKLANDER, T., PORKKA, P., AND RAATIKAINEN, K. 1997. Transaction Processing in the RODAIN Real-Time Database System. In *Real-Time Database and Information Systems*. Kluwer Academic Publishers, London, 355–375.

LINDSTRÖM, J., NIKLANDER, T., PORKKA, P., AND RAATIKAINEN, K. 1999. Design of RODAIN 2000 Experimentation. Rodain internal report, University of Helsinki, Department of Computer Science.

NIKLANDER, T., KIVINIEMI, J., AND RAATIKAINEN, K. 1997. A Real-Time Database for Future Telecommunication Services. In *Intelligent Networks and Intelligence in Networks*. Chapman & Hall, Paris, France, 413–430.

RAATIKAINEN, K. 1997. Real-Time Databases in Telecommunications. In *Real-Time Database Systems: Issues and Applications*, A. Bestavros, K.-J. Lin, and S. H. Son, Editors. Kluwer, 93–98.

TAINA, J. AND RAATIKAINEN, K. 1996. Experimental real-time object-oriented database architecture for intelligent networks. *Engineering Intelligent Systems 4*, 3 (Sept.), 57–63.