Performance Measurement and Tuning of Hot-Standby Databases

Antoni Wolski and Vilho Raatikka

Solid Information Technology, Itälahdenkatu 22B, 00210 Helsinki, Finland
{first_name.last_name}@solidtech.com

Abstract. General-purpose, high-availability database systems have lately proliferated to various network element platforms. In telecommunication, databases are expected to meet demanding availability levels while preserving the required throughput. However, so far, the effects of various high-availability configurations on overall database performance have not been analyzed. In this paper, the operation of a fully replicated, hot-standby database system is presented, together with some performance tuning possibilities. To study the effect of several database-tuning parameters, a telecom-oriented database benchmark, TM1, is used. The experiments involve varying of the read/write balance and various logging and replication parameters. It is shown that, by relaxing the reliability requirements, significant performance gains can be achieved. Also, it is demonstrated that a possibility to redirect the log writing from the local disk to the standby node is one of the most important benefits of a high-availability database system.

Introduction

The goal of highly available (HA) systems is to make system or component failures tolerable. The extent to which failures are tolerable is specified with the availability measure A that is equal to the percentage of the time a service is operational, as related to the total time the service is supposed to be operational. Availability can be derived from the maximum duration of an outage (equal to mean time to repair, MTTR) and the frequency of outages (represented with mean time between failures, MTBF), by using the following formula:

$$A = \frac{MTBF}{MTBF + MTTR} \bullet 100\% \tag{1}$$

To deal with failures, an HA system embodies redundancy both in hardware and software, typically managed by a framework such as AMF (Availability Management Framework) [1] of SA Forum[1].

In the simplest redundancy model, called 2N, the two units, active and standby, make up a mated pair, and the redundant application components are organized in pairs in the corresponding units. Should a failure occur, the failed active unit (hard-

[1] http://www.saforum.org

Third International Service Availability Symposium (ISAS 2006),
May 15-16, 2006, Helsinki, Finland.

ware or software) is quickly replaced with a corresponding standby unit. This operation is called a failover. The purpose of failover is to maintain the required service availability, in the presence of failures. The hot-standby technique described above allows achieving at least five nines (99.999%) availability required in the telecom systems.

The availability of the database services is maintained by using the very same approach. Various architectures of highly-available database management systems (HA-DBMS) have been proposed and implemented commercially [7]. In this paper, the focus is on the utility of a fully replicated hot-standby (HSB) HA DBMS. In such a system, a stream of transactions is continuously sent from the Active server to the Standby server, by way of a replication protocol.

A typical hot-standby database system can be configured in a variety of ways. Most important tuning parameters are the ones dealing with the synchrony of the log writing (the durability settings) and the synchrony of the database replication protocol (the safeness characteristics, first introduced in [2]). Typically, the goal of the HA-DBMS tuning is to achieve the best trade-off among the three characteristics: data durability, failover time, and performance.

The problem is that the usage of those parameters is often based on intuitive argumentation, without any experimental data to depend on. We are not aware of any published work shedding any light on the problem. In this paper, we report on experiments conducted using a real-life commercial HA-DBMS in a telecom setting. The purpose of the experiments was to find out what was the effect of the tuning parameters on the overall HA-DBMS performance. The product used in the experiments was Solid's HA-DBMS called Solid Database Engine with CarrierGrade Option [8]. For a case study on applying the product to a commercial telecom HA framework, see [11].

The obtained results support the general notion that increasing asynchrony improves performance. However, the comparison among the effects of different setting may not correspond to intuitive presumptions.

We summarize the HA-DBMS architecture in Section 2. Various configuration parameters are also introduced together with their intuitive purpose. Database benchmarking is discussed in Section 3, where also the TM1 benchmark is introduced. Test results are summarized in Section 4. We conclude with a summary of the results, and general guidelines for HA-DBMS users.

Tuning Highly Available Databases

An HA-DBMS based on the hot-standby principle is composed of the elements shown in Fig. 1.

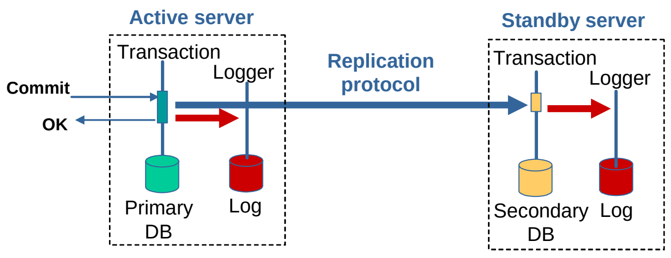

Fig. 1. Architecture of a hot-standby HA-DBMS.

A database (Active or Standby) server is a component that offers a database access service to applications, mostly by way of over-the-network connections. Transaction is an abstraction of a transaction processing thread. The DB (database file) is an abstraction of persistent storage of the data. In reality, one or more system-level files can be used for the purpose. Primary DB is a "live" database updated by the transactions running on the Active server. Secondary DB is kept up-to-date by way of a replication protocol. Secondary DB may be subjected to read-only load, if necessary. Logger is a thread that writes the Log. The Log represents one or more persistent files to store the effects of transactions as they are executed in the server. The Log is instrumental in making it possible to perform a startup database recovery. In the recovery process, the Log is scanned to ensure that the database is in a consistent state, by:

- removing the effects of uncommitted transactions, and
- re-executing committed transactions that have not been checkpointed to the database.

If we deal with a standalone database system (not hot standby), the recovery process preserves the Atomicity and Durability characteristics of the database over system failures and terminations [2].

The standard level of durability support is called strict durability. It requires that the commit record of a transaction is written synchronously to the persistent medium (disk) before the commit call is returned to the application. The technique is often referred to as WAL—Write-Ahead Logging. WAL processing is very resource-consuming and often becomes a bottleneck in the system. Therefore, whenever the durability requirement can be relaxed, it is done. Especially, in the telecom environment, in some applications like call setup, session initiation, etc., a service request is occasionally allowed to fail (and be lost) if the probability is not very high. In such a case, relaxed durability may be applied whereby the log is written asynchronously. This means that the commit call may be returned without the need to wait for the disk. This results in significant improvement in both the system throughput and response time. In this paper, an effort is made to quantify the effect of switching from strict to relaxed durability on total throughput.

In an HSB database, transactions are also sent to the standby server by way of a replication protocol. In order to preserve the database consistency in the presence of failovers, the replication protocol is built very much on the same principles as physical log writing: the transaction order is preserved, and commit records demarcate committed transactions. If a failover happens, the Standby server performs a similar database recovery as if a transaction log was used: the uncommitted transactions are removed and the committed ones are queued for execution.

Similarly to log writing, the replication protocol may be asynchronous or synchronous. To picture that, we use the concept of safeness level where 1-Safe denotes an asynchronous protocol and 2-Safe denotes a synchronous one [2]. The two safeness levels are illustrated in Fig. 2.

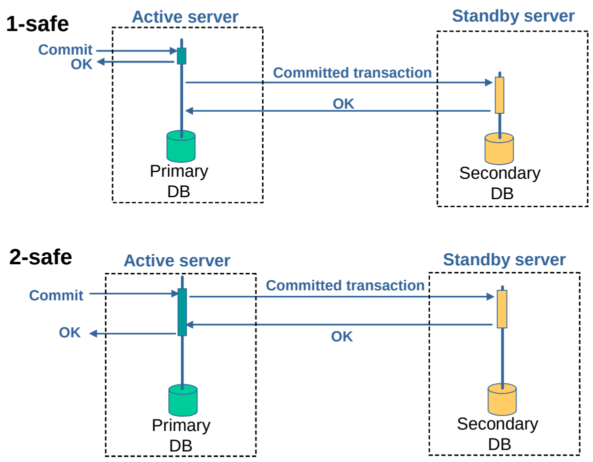

Fig. 2. Illustration of safeness levels.

One may see that the benefit of 1-Safe replication is similar to that of relaxed durability: the transaction response time is improved, and the throughput may be expected to be higher, too. On the other hand, with 2-Safe replication, no committed transactions are lost upon failover. You might call this transaction characteristic standby-based strict durability, as opposed to log-based strict durability of a traditional DBMS. One immediate observation is that the log-based durability level has no effect on actual durability of transactions in the presence of failover. It is the standby-based durability that counts. The traditional log writing is relegated to the role of facilitating the database system recovery in the case of a total system failure. All other (more typical) failures are supposed to be taken care of by failovers. If a total system failure is unlikely (as builders of HA systems want to believe), a natural choice is to replace strict log-based durability with strict standby-based durability, that is, the 2-Safe protocol. Here, the gain is a faster log processing without really loosing strict durability (if only single failures are considered). To take the full advantage of the possibility, Solid's HA DBMS has an automated feature called adaptive durability. With adaptive durability, the Active server's log writing is automatically switched to strict if a node starts to operate without a Standby. Otherwise, the Active server operates with relaxed durability.

The possibility to transfer the log writing responsibility from the disk to the network is very tempting because, by a common perception, a message round trip travel over a high-speed network may be almost an order of magnitude faster than writing synchronously to the disk.

In addition to the choice between 1-Safe and 2-Safe replication, 2-Safe protocols may be implemented with different levels of involvement of the Standby server in the processing of the commit message. In [7], two levels were proposed: 2-Safe Received and 2-Safe Committed. In this paper, the following 2-Safe policy levels are defined:

- 2-Safe Received: the Standby server sends the response immediately upon receipt (as in [7]).
- 2-Safe Visible: the Standby server processes the transaction to the point that the results are externally visible (in-memory commit).
- 2-Safe Durable: the Standby process processes the transaction to the point that it is written to a persistent log (strictly durable commit).

The three policy levels are illustrated in **Fig. 3**.

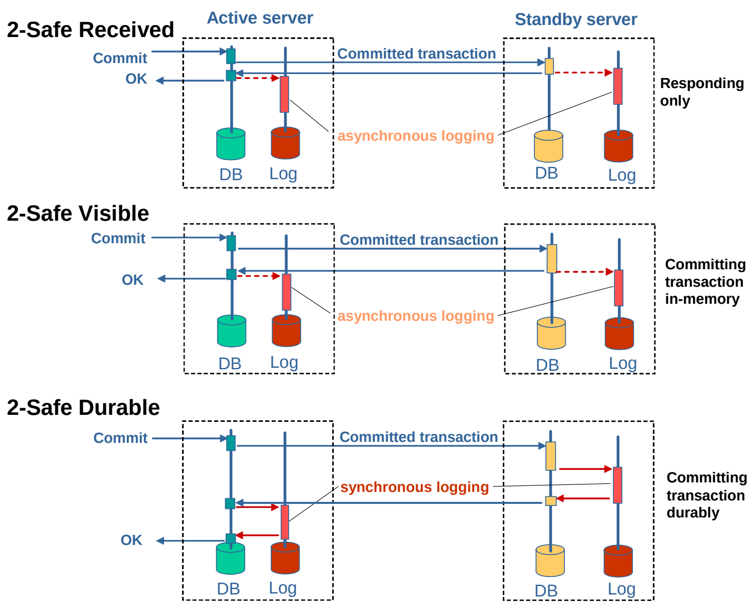

Fig. 3. 2-Safe policy levels.

Of the three 2-Safe policy levels, 2-Safe Received is intuitively the fastest and 2-Safe Durable the most reliable. In a system with 2-Safe Durable replication, the database may survive a total system crash and, additionally, a media failure on one of the nodes. This comes, however, at a cost of multiple synchrony in the system.

The 2-Safe Visible level is meant to increase the system utility by maintaining the same externally visible state at both the Active and Standby servers. Thus, if the transactions are run at both the Active and Standby servers (read-only transactions at Standby), one-copy serializability [2] may be maintained over global transaction histories. The cost of maintaining this consistency level involves waiting for the transaction execution in the Standby server, before acknowledging the commit message.

To summarize, the intuitive rules for choosing the best trade-off between performance and reliability are the following:

1. To protect against single failures while allowing for some transactions to be lost on failover ➔ use 1-Safe replication with relaxed log-based durability.
2. To protect against single failures, with no transactions lost on failover ➔ use 2-Safe Received replication with relaxed log-based durability.
3. To protect against single failures, with no transactions lost on failover and a possibility to use the Primary and Secondary databases concurrently ➔ use 2-Safe Visible replication with relaxed log-based durability.
4. To protect against total system failure (in addition to single-point failures) ➔ use any 2-Safe protocol and strict log-based durability in the Active server.
5. To protect against total system failure and a media failure ➔ use 2-Safe Durable replication with strict log-based durability in both the Active and Standby servers.

It is also worth noting that a third dimension in assessing different replication protocols is the failover time. The further transactions are processed in the Standby server at the time of a failover, the faster the failover is. The protocols may be ordered by the failover time, from the shortest to the longest, in the following way: 2-Safe Durable, 2-Safe Visible, 2-Safe Received and 1-Safe.

In the performance testing experiments, we study the effect of all the above parameters on the system performance. We take advantage of the fact that the Solid's HA DBMS product has all the necessary controls, both in the form of configuration parameters and dynamic administrative commands.

Database Benchmarks

Benchmarking serves multiple needs. Results are often used to help product evaluation and they offer valuable information for the developers. Moreover, if the results happen to be good, manufacturers use them in product marketing to show the superior performance of a product over the competitors.

Evaluating different database products requires reliable information of their performance under domain-specific workload. In order to be useful in performance comparisons, the workload generated by the benchmark should imitate the reality as much as possible. Therefore, there is an inevitable need for domain-specific benchmarks. Furthermore, a benchmark is more likely to be widely accepted if its provided by a consortium consisting of wide range of independent participants than if its published and owned by a single organization. It is also assumed that a domain-specific benchmark must generate typical operations in that domain and measure them trustworthy. It must also be easily portable to different platforms. It should scale to systems of different sizes, too. Finally, it must be easily understandable to create credibility [4].

Wisconsin benchmark [3] measures the performance of a single-user database software and the hardware it runs on. It is generic, not domain-specific, but it is intuitive and the database scales in size. To get reliable results, the test must be conducted under certain conditions. For example, the database size must be at least five times the size of the buffer pool (page cache). Moreover, the effect of the buffer pool, which is one of the main performance makers in read-intensive operations, is intentionally eliminated by varying the queries of query sets.

AS3AP [10], which fills the deficiencies of the Wisconsin benchmark, is also general-purpose in its nature, but it provides both single-user and multi-user tests to measure the performance of a database system under "typical" OLTP[2], IR[3], mixed, and single-user workloads.

Nowadays, enterprise-oriented benchmarks come from the Transaction Processing Performance Council[4]. They deal with enterprise applications: order entry, decision support and web server applications.

The telecom field has been long in the need of a domain-oriented benchmark. The TM1 Benchmark started life as a special purpose benchmark used internally by a telecom equipment manufacturer. It was used to estimate the speed of various database management systems on different hardware platforms. In April 2004, the benchmark specification was published as part of a Master's Thesis at the University of Helsinki [9]. In November 2004, Solid published the entire benchmark description on their web site and made a benchmark kit available for free download[5]. The kit enabled any interested party to set up a test environment and run the TM1 benchmark. In February 2006, the corresponding Nokia's Network Database Benchmark was published[6].

Unlike the TPC benchmarks, the TM1 benchmark is based on a telco scenario, the Home Location Register (HLR). The HLR holds subscriber's identification information as well as other details of service provisioning. TM1 measures performance only. Maintenance or purchasing costs are not considered.

The benchmark uses four tables and a set of seven transactions that can be combined in different mixes (see the Appendix for more details on TM1). The most typical mix is denoted as "R80W20" meaning 80% of read transactions and 20% of modification transactions. In the experiments reported here, we used both the R80W20 and R20W80 mixes.

Testing Results

Test System

The system under test (SUT) consisted of two database servers both running on Linux Fedora Core 2. One server played the role of an Active database server while the other was a Standby database server. The configuration of both the Active and Standby servers remained the same during the tests excluding the logging modes. That is, the durability level and the safeness level of the servers varied. The workload was generated by TM1 benchmark, which ran simultaneously on two separate Windows 2000 computers. All four computers shared an isolated one gigabit network. More specific description of the test environment is presented in the table below:

[2] OLTP = On-Line Transaction Processing
[3] IR = Information Retrieval
[4] http://www.tpc.org
[5] http://www.solidtech.com/tm1
[6] https://hoslab.cs.helsinki.fi/savane/projects/ndbbenchmark/

Table 1. Test system configuration data.

Role	CPU/MHz	Memory/B	disks	OS
Active server	1800	4096	2xSCSI	Fedora Core 2
Standby server	1800	4096	2xSCSI	Fedora Core 2
TM1 client host	2x1666	1024	IDE	Win2000 SP4
TM1 client host	2x1666	2048	2xSCSI	Win2000 SP4

The results of the tests were stored into the Test Input and Result Database (TIRDB), which was located on its own computer.

Tests started by copying a fresh database file to the working directory of the Active database server, disabling the write-ahead caches of both SCSI devices, and starting the server. In those tests where the Standby database server was used, it was started in the similar way, and the database of the Active server was replicated to it. According to the TM1 specifications, each test run was divided to ramp-up time and measured run time. We used 10 minutes ramp-up time followed by 20 minutes run time.

TM1 measures mean qualified throughput (MQTh) and response times for each transaction type. MQTh is a sum of successfully executed transactions from all clients divided by the duration (in seconds) of test run. The result is the average transaction rate per second.

We used 10 database client instances in each TM1 client host to create the workload. In TM1, clients are represented by separate processes and each client retains its own data structure for result data

The workload consisted of write and read intensive transaction mixes. In write intensive workload 80% of transactions included writes and 20% included reads only. In read intensive load the share between write and read transactions was the opposite.

The distribution of different transactions is presented in the following table:

Table 2. Transaction mixes for two kinds of load.

write-intensive load	*read-intensive load*
GET_SUBSCRIBER_DATA 9	GET_SUBSCRIBER_DATA 35
GET_NEW_DESTINATION 2	GET_NEW_DESTINATION 10
GET_ACCESS_DATA 9	GET_ACCESS_DATA 35
UPDATE_SUBSCRIBER_DATA 8	UPDATE_SUBSCRIBER_DATA 2
UPDATE_LOCATION 56	UPDATE_LOCATION 14
INSERT_CALL_FORWARDING 8	INSERT_CALL_FORWARDING 2
DELETE_CALL_FORWARDING 8	DELETE_CALL_FORWARDING 2

The TM1 HLR database size is determined by the number of rows in the SUBSCRIBER table. The population used consisted of 500 000 subscribers, 1.25 million rows in ACCESS_INFO, 1.25 million rows in SPECIAL_FACILITY and 1.9 million rows in CALL_FORWARDING. The database file size was about 840MB.

Result Summary

We ran three kind of tests in which emphasis was on the log write mode (log-based durability) of the Active server, the log write mode of the Standby server and the safeness level of the replication protocol. Every test was run with both read-intensive and write-intensive workload. A set of tests was run on a standalone server first, to quantify the effect of durability settings on the general performance. The results are shown in Fig. 4.

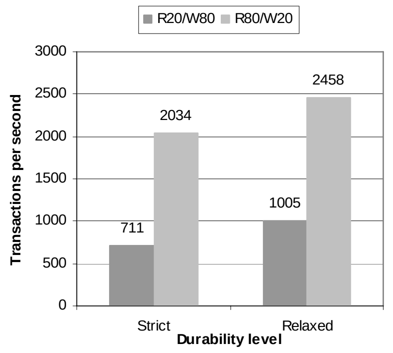

Fig. 4. Effect of the durability level on performance of a standalone database server.

The comparison above shows that using asynchronous log writing in a standalone server increases the throughput of the system with read and write intensive workload by 20-40%, respectively. In the hot-standby configuration, a question may arise whether to use strict log-based durability in the Active server or not. The difference is illustrated in Fig. 5.

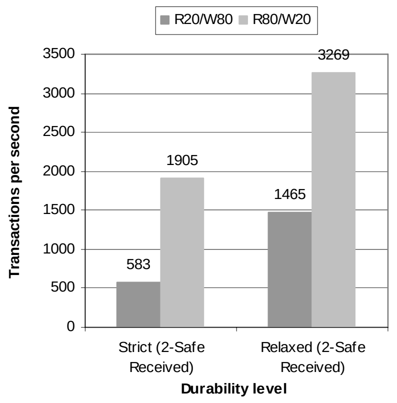

Fig. 5. Effect of the durability setting on performance of an HSB database using 2-Safe Received protocol.

The results shown suggest that, by using relaxed durability in the Active server (that is, "delegating" the log writing to the Standby), a significant (70-250%) increase in the system throughput can be gained. In the case of the presented 2-Safe Received replication protocol, the Standby node runs with relaxed durability, in any case. Since logging affects the write transactions only, the benefit especially materializes with write intensive workload.

When looking at the comparison of all the replication protocols (all of which run with relaxed durability in the Active node, except for 2-Safe Durable), one can see that the earlier intuitive conjecture is verified: the more asynchrony there is in the system, the more throughput can be achieved (Fig. 6).

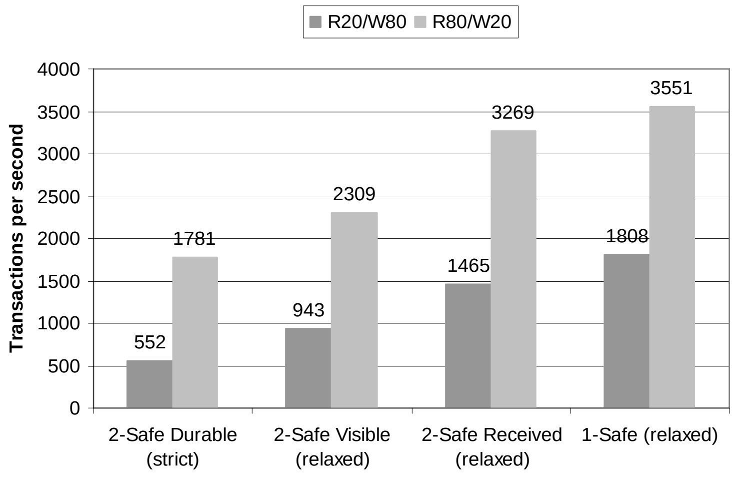

Fig. 6. Comparison of all HSB database replication protocols.

The difference in performance is most significant for the write-intensive loads: there can be more than 300% of improvement between the extreme ends. On the other hand, the modest benefit of the 1-Safe protocol comes as a surprise. The explanation may be that, in the presence of high concurrent load, other optimization mechanisms (like group commit [11]) compensate for the lack of asynchrony, with 2-Safe protocols. Another explanation may be that another bottleneck appears in the system with the speed-up of the protocol.

Conclusions

We have conducted performance testing of a hot-standby highly available database, using a telecom-oriented benchmark and different load mixes. The obtained results provide quantitive guidance to developers of HA systems, needing to configure the databases properly. In seeking a right trade-off among the required reliability, failover time and throughput, the effect of the durability and replication settings on the performance is the most important factor in making the decision.

The results support the intuitive conjecture that increasing asynchrony leads to more throughput. The reported significant gains in throughput should encourage the developers to select asynchronous processing modes as much as they are not restricted by other requirements.

References

1. Application Interface Specification, SAI-AIS-B.02.02, December 2005. Service Availability Forum, available at http://www.saforum.org.
2. Bernstein, Ph. A.; Hadzilacos, V.; Goodman, N.: Concurrency control and recovery in database systems. Addison-Wesley Publishing Company, 1987, ISBN 0-201-10715-5.
3. Bitton, D., DeWitt, D.J., Turbyfill, C.: Benchmarking Database Systems A Systematic Approach. VLDB 1983: 8-19.
4. Gray, J. (ed.): The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann Publishers, 1993, ISBN 1-55860-292-5.
5. Gray, J. and Reuter, A.: Transaction Processing Systems, Concepts and Techniques. Morgan Kaufmann Publishers, 1992, ISBN 1-55860-190-2.
6. Brossier, S., Herrmann, F., Shokri, E.: On the Use of the SA Forum Checkpoint and AMF Services. Proc. ISAS 2004, May 13-14, 2004 Munich, Germany. Springer-Verlag Lecture Notes in Computer Science, Vol. 3335, ISBN: 3-540-24420-4.
7. Drake, S., Hu, W., McInnis, D.M., Sköld, M., Srivastava, A., Thalmann, L., Tikkanen, M., Torbjørnsen, Ø.,Wolski. A.: Architecture of Highly Available Databases. Proc. ISAS 2004, May 13-14, 2004 Munich, Germany. Springer-Verlag Lecture Notes in Computer Science, Vol. 3335, ISBN: 3-540-24420-4.
8. Solid High Availability User Guide, Version 4.5, Solid Information Technology, June 2005, available at http://www.solidtech.com.
9. Strandell, T.: Open Source Database Systems: Systems study, Performance and Scalability. Master's Thesis, University of Helsinki, Department of Computer Science, May 2003, 54 p. (http://www.cs.helsinki.fi/u/tpstrand/thesis/)
10. Turbyfill, C., Orji, C.U., Bitton. D.: AS^{3}AP - An ANSI SQL Standard Scaleable and Portable Benchmark for Relational Database Systems. In [4].
11. Wolski, A. and Hofhauser, B.: A Self-Managing High-Availability Database: Industrial Case Study. Proc. Workshop on Self-Managing Database Systems (SMDB2005), April 8-9, 2005, Tokyo, Japan. http://research.solidtech.com/publ/wolhof-smdb05-ha-case.pdf.

Appendix: TM1 specifications

Because of lack of space, only general descriptions are included here. For a full benchmark description, see http://www.solidtech.com/tm1.

The purpose of the benchmark is to derive a maximum performance that can be achieved in a database system under a certain load. The SUT (system under test) is composed of target equipment and a database system being tested. In the case of HSB tests, two target computers are used. The load is generated in a separate computer (TM1 client host) where distinct processes emulate user applications. The detailed description includes specifications for the test duration, the database scaling and population rules, transactions mixes and user loads.

TM1 Database schema

The TM1schema models an HLR (Home Location Register) structure found in all mobile telephone systems. The schema is composed of the four tables shown below.

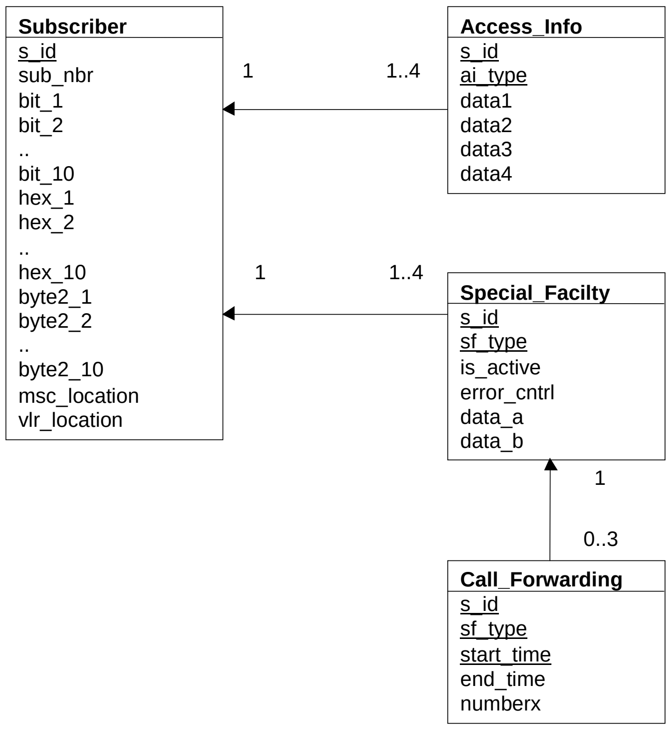

Fig. A-1. Schema of TM1.

TM1 Transactions

The purpose of the transactions is to emulate typical HLR activities in call set-up and provisioning. The transactions are listed below.

```
GET_SUBSCRIBER_DATA {
        SELECT s_id, sub_nbr, bit_1, bit_2, bit_3, bit_4, bit_5, bit_6, bit_7, bit_8, bit_9, bit_10, hex_1, hex_2,
            hex_3, hex_4, hex_5, hex_6, hex_7, hex_8, hex_9, hex_10, byte2_1, byte2_2, byte2_3, byte2_4,
            byte2_5, byte2_6, byte2_7, byte2_8, byte2_9, byte2_10, msc_location, vlr_location
        FROM subscriber
        WHERE s_id = <s_id rnd>;
}
GET_NEW_DESTINATION {
        SELECT cf.numberx FROM special_facility AS sf, call_forwarding AS cf
        WHERE (sf.s_id = <s_id rnd>
                AND sf.sf_type = <sf_type rnd>
                AND sf.is_active = 1)
                AND (cf.s_id = sf.s_id
                AND cf.sf_type = sf.sf_type)
                AND (cf.start_time \<= <start_time rnd>
                AND <end_time rnd> \< cf.end_time);
}
GET_ACCESS_DATA {
        SELECT data1, data2, data3, data4 FROM access_info
        WHERE s_id = <s_id rnd>
        AND ai_type = <ai_type rnd>;
}
UPDATE_SUBSCRIBER_DATA {
        UPDATE subscriber
        SET bit_1 = <bit rnd>
        WHERE s_id = <s_id rnd subid>;

        UPDATE special_facility
        SET data_a = <data_a rnd>
        WHERE s_id = <s_id value subid>
                AND sf_type = <sf_type rnd>;
}
UPDATE_LOCATION {
        UPDATE subscriber
        SET vlr_location = <vlr_location rnd>
        WHERE sub_nbr = <sub_nbr rndstr>;
}
```