Table of Contents

Mnesia

Mnesia A Distributed Robust DBMS for Telecommunications Applications

Håkan Mattsson, Hans Nilsson and Claes Wikström

Computer Science Laboratory
Ericsson Telecom AB
S-125 26 Stockholm, Sweden
{hakan,hans,klacke}@erix.ericsson.se

Abstract. The Mnesia DBMS runs in the same adress space as the application owning the data, yet the application cannot destroy the contents of the data base. This provides for both fast accesses and efficient fault tolerance, normally conflicting requirements. The implementation is based on features in the Erlang programming language, in which Mnesia is embedded.

1 Introduction

The management of data in telecommunications system has many aspects whereof some, but not all, are addressed by traditional commercial DBMSs (Data Base Management Systems). In particular the very high level of fault tolerance which is required in many nonstop systems, combined with requirements on the DBMS to run in the same address space as the application, have led us to implement a brand new DBMS. This paper describes the motivation for, as well as the design of this new DBMS, called Mnesia. Mnesia is implemented in, and very tightly connected to, the programming language Erlang and it provides the functionality that is necessary for the implementation of fault tolerant telecommunications systems. Mnesia is a multiuser Distributed DBMS specially made for industrial telecommunications applications written in the symbolic programminglanguage Erlang [1] which is also the intended target language. Mnesia tries to address all of the data management issues required for typical telecommunications systems and it has a number of features that are not normally found in traditional databases.

In telecommunications applications there are needs different from the features provided by traditional DBMSs. The applications we now see implemented in the Erlang language need a mixture of a broad range of features which generally are not satisfied by traditional DBMSs. Mnesia is designed with requirements like the following in mind:

1. Fast realtime key/value lookup.
2. Complicated non realtime queries mainly for operation and maintenance.
3. Distributed data due to distributed applications.
4. High fault tolerance.

5. Dynamic re configuration.
6. Complex objects.

What sets Mnesia apart from most other DBMSs is that it is designed with the typical data management problems of telecommunications applications in mind. Hence Mnesia combines many concepts found in traditional databases such as transactions and queries with concepts found in data management systems for telecommunications applications such as very fast realtime operations, configurable degree of fault tolerance (by means of replication) and the ability to reconfigure the system without stopping or suspending it. Mnesia is also interesting due to its tight coupling to the programming language Erlang, thus almost turning Erlang into a database programming language. This has many benefits, the foremost being that the impedance mismatch [5] between data format used by the DBMS and data format used by the programming language which is being used to manipulate the data, completely disappears.

Mnesia is currently used in almost all Erlang based projects within Ericsson ranging from small scale prototype projects to major switching product projects.

The remainder of this paper is organized as follows. Section 2 is a brief overview of the DBMS. Section 3 is organized as a listing of typical DBMS functionalities, a discussion of some telecommunications aspect on the functionality and how the functionality is provided by Mnesia. Section 4 contains some performance measurements and finally section 5 provides some conclusions.

2 A brief overview of Mnesia

We briefly overview the features of Mnesia DBMS. Mnesia is both an extension of the programming language Erlang as well as an Erlang application. The DBMS components such as lock manager, transaction manager, replication manager, logger, primary and secondary memory storage, backup system, etc are all implemented as regular Erlang programs. The query language, however, is implemented as a part of the actual Erlang syntax, whereas the optimizing query compiler and evaluator are regular Erlang programs. The data model of Mnesia is of a hybrid type: data is organized as tables of records similar to relations, but the attributes of the records (including the key) can hold arbitrarily complex compound data structures such as trees, functions, closures, code etc. As such, Mnesia could be characterized as a so called object-relational DBMS. Let us assume, a definition for a person record.

```
-record(person, {name,       %% atomic, unique key
                 data,       %% compound unspecified structure
                 married_to, %% name of partner or undefined
                 children}). %% list of children
```

Given this record definition, it is possible to create instances of the person record with the following Erlang syntax where

```
X = #person{name = klacke,
            data = {male, 36, 971191},
            married_to = eva,
            children = [marten, maja, klara]}.
```

binds the variable `X` to a new person record. The data field is bound to a tuple `{male, 36, 971191}` with three parts. This is an example of a complex object, and Mnesia puts no constraints on the complexity which is allowed on attributes. We can even have function objects with variable closures as attribute values. The variable `X` is merely an Erlang term, and in order to insert it into the database, it must be written:

```
mnesia:write(X)
```

Series of Mnesia operations can be performed together as an atomic transaction. In order to have Mnesia execute a transaction, the programmer must construct a functional object and then present the Mnesia system with that functional object similar to [7]. We explain this by an example, assume we wish to write an Erlang function `divorce(Name)` which takes the name of a person, finds the person from the database, and sets the `married_to` field for the person and its partner back to the undefined value.

```
divorce(Name) ->
   F = fun() >
       case mnesia:read(Name) of
          [] >
             mnesia:abort(no_such_person);
          Pers >
             Partner = mnesia:read(Pers#person.married_to),
             mnesia:write(Pers#person{married_to = undefined}),
             mnesia:write(Partner#person{married_to = undefined})
          end
       end,
   mnesia:transaction(F).
```

The `divorce/1` function consists of two statements, the first `F = ....` statement creates a function object, it does not execute anything, it merely constructs an anonymous function. The second statement gives this function away to the Mnesia system which executes the function in the context of a transaction, adhering to traditional transaction semantics [2].

The actual function `F` first executes a read operation to find the person with the name `Name`, it then executes a second read to find the partner of the first person, and finally it writes two new records into the database with the `married_to` field set to undefined. These two write operations will effectively overwrite the old values. The function `divorce/1` returns the value of the transaction, which is either `{aborted, Reason}` or `{atomic, Value}` depending on whether the transaction was aborted or not.

Queries in Mnesia are expressed with a list comprehension syntax [15]. A query to find the names of all persons with more than X children is formulated as:

```
query [P.name || P < table(person),
                 length(P.children) > X]
end
```

This should be read as: construct the list of P.name's such that `P` is taken from the table of persons, and length of the children list is greater than `X`. It is indeed both possible and natural to mix a query with user defined predicates. Assume a function:

```
maturep({Sex, Age, Phone}) when Age > 30 >
   true;
maturep({Sex, Age, Phone}) >
   false.
```

Then the query:

```
query [P.name || P <- table(person),
                 maturep(P.data),
                 length(P.children) > X]
end
```

extracts all persons with more than `X` children and where the second element of the data field is larger than 30. It is also possible to define rules with an embedded logic language similar to Datalog [16]. If we define the rule:

```
oldies(Name) :-
   P <- table(person),
   maturep(P.data),
   Name = P.name.
```

This defines a rule, which then acts as a virtual table and application programs can access the virtual table oldies. The virtual oldies table contains a subset of the real person table. This is similar to, but more powerful than, the concept of views found in relational databases. Queries are compiled by an optimizing query compiler which has been integrated with the normal Erlang compiler.

Tables can be replicated to several sites (or nodes). The network of nodes can be a heterogeneous network. Replication is the mechanism whereby we can construct fault tolerant systems. Access to a table is location transparent, that is, programs do not have to have explicit knowledge of the location of data. A table has a unique name and certain properties, we have the following list of properties attached to each table.

- *type* controls whether the table has set or bag semantics. A set has unique keys, whereas a bag can have several objects with the same key.
- *ram_copies* a list of Mnesia nodes where replicas of the table are held in ram only.
- *disc_copies* a list of Mnesia nodes where replicas of the table are held entirely in ram, but all update operations on the table are logged to disc.
- *disc_only_copies* a list of Mnesia nodes where replicas of the table are held on disc only. These replicas are of course considerably slower than replicas held in ram.
- *index* a list specifying on which attributes of the record index information shall be maintained. All records are always automatically indexed on the primary key.
- *snmp* controls whether the table shall be possible to manipulate through the Simple Network Management Protocol protocol [4].

Descriptions of all tables are kept in a database schema and Mnesia has a multitude of functions to manipulate the schema dynamically. Tables can be created, moved, replicated, changed, destroyed etc. Furthermore all these system activities are performed in the background and thus allows the application to utilize the system as usual although the system itself is being changed.

Backups can be constructed of the entire distributed system, these backups can be installed as fallbacks. This means that if the system should crash, the database will automatically be recreated from the fallback.

3 DBMS feature discussion

Different DBMSs have different features and characteristics. This section is organized as a listing of different DBMS characteristics and a short discussion of the importance and necessity in our intended application domain.

3.1 Complex values

The ability to handle complex values, such as lists, sets, trees, etc efficiently and naturally in the DBMS is probably the single most important feature for a telecommunications DBMS. Telecommunications applications which handle traffic are usually driven by external stimuli which arrives at the system. When such a stimuli arrives in the form of a PDU (Protocol Data Unit) to the system, the PDU is decoded and some appropriate action must be taken. When the PDU has been decoded, the system usually has to retrieve some data object, possibly a subscriber record which is used to decide what action should be taken in response to the received PDU. In many telecommunications systems the single most important feature of the data management system is that this lookup operation is very efficient. It is also important that the DBMS allows the data to be structured and stored in such a way so that relevant data can be accessed in a single lookup operation. This fact makes it hard to model telecommunications

system with traditional relational databases. Organizing the telecommunications data in third (or even first) normal form is usually not possible.

This is one of the reasons why the telecommunications industry have payed so much attention to object-oriented database systems which allows the data to be organized in a more flexible way than relational database systems. Thus Mnesia allows the user to use arbitrarily complex objects both as attribute values but also as key values in the database.

3.2 Data format and address space

Many databases use an internal, language independent format to store data. This is most unfortunate in telecommunications systems due to the previously mentioned fast lookup requirement. Many object oriented DBMSs are tightly coupled to a programming language like C++ or Smalltalk. The ability to manipulate regular programming language objects in the database, makes the impedance mismatch disappear. This not only eases the manipulation of the DBMS but it also provides an opportunity to implement very efficient lookup operations since a lookup can immediately return a pointer to the object by means of the programming language being used. If we for example want to implement a routing table as a database table, it is not realistic to transform the routing data back and forth from an an external DBMS format for each packet we need to route. Furthermore it is not realistic to perform any context switches and search the relevant data for each packet in a process executing in another address space. This effectively rules out all DBMSs that cannot be directly linked into the address space of the application, as well as all DBMSs that are linked into the application but use a language independent format to store data.

The major disadvantage of letting the application run in the same address space as the DBMS is that if the application should crash due to a programming error, the DBMS may not be given the opportunity to store vital data to secondary storage before terminating. This means that the entire DBMS must be recovered before starting up again. This is usually a time consuming process and in telecommunications system, the downtime must be short. We avoid this problem since the applications as well as the DBMS are implemented in the Erlang language. An Erlang application cannot crash in such away that it effects the DBMS. The application run in the same address space as the DBMS, but Erlang itself makes it impossible for the crash of one application to effect an other application. Erlang processes have the efficiency advantage of running in the same address space but they do not have the possibility to explicitly read or write each others memory.

3.3 Fault tolerance

Many telecommunications applications are nonstop systems. The system must be able to continue to provide its services even if a number of hardware or software errors occur. This adds requirements not only to the DBMS, but also to the

telecommunication application itself. It influences the design of the entire application and the DBMS must provide the application designers with mechanisms whereby a sufficiently fault tolerant system can be designed. The mechanism provided by Mnesia is the ability to replicate a table to several nodes. All replicas of a Mnesia table are equal and there is thus no concept of primary and standby copies at the DBMS level. If a table is replicated all write operations are applied to all replicas for each transaction. If some replicas are not available the write operations will succeed anyway and the missing replicas will be later updated when they are recovered. This mechanism makes it possible to design systems where several geographically distinct systems cooperate to provide a continuously running nonstop system. Many other highly fault tolerant systems like ClustRa [11] also provide fault tolerance through means of replication. However, they do not have the ability to execute in the same address space as the application.

Mnesia can also recover partially from complete disasters. All objects that are written to disc, are coded in such away that it is possible to safely distinguish data from garbage. This makes it possible to scan a damaged or crashed disc or filesystem and retrieve data from the crashed disc.

3.4 Distribution and location transparency

Mnesia is a truly distributed DBMS where data can be replicated or simply reside remotely. In such an environment it is important that the DBMS programmer can access data without explicit knowledge about the location of data. That is, location transparency of data is important. On the other hand, since it is most certainly more expensive to access data remotely, it must also be possible for the application programmer to explicitly find this location information in order to execute the program where the data is. Hence, we want to provide location transparency as well as the ability to explicitly locate data. Different applications have different requirements.

Mnesia applications which access tables by using the name of the table only, work regardless of the location of the table. The system keeps track of where data is replicated. However, it is also possible for the Mnesia programmer to query the system for the location of a table, and then execute the code remotely instead. This can be done by sending the code to the remote site or by ensuring that the code is preloaded there.

3.5 Transactions and ACID

DBMSs have ACID properties, Atomicity, Consistency, Isolation and Durability. These properties are implemented by means of transactions, writeahead logging and recovery in Mnesia. Most Mnesia transactions consists of series of operations on ram only (possibly replicated) tables. These transactions do not interact with the disc storage system at all, hence the Durability property is not fulfilled for these transactions. An example where transaction semantics are required in telecommunication systems is when we add a new subscriber to the system.

When we do this, several resources are allocated in the system, and several data objects are written into the memory of the system. It is vital that all of these operations are performed as one single atomic action. Otherwise the system could end up in an inconsistent state with possibly unreleased resources.

3.6 The ability to bypass the transaction manager

The overhead for a transaction is quite high and for certain parts of traffic handling applications, it is simply not feasible to use a transaction system to access the data. Consequently, a DBMS that is suitable for telecommunications must be able to support both atomic transactions consisting of series of database operations as well as very light weight locking on the same data. The traffic system consists of a number of tables. Many of these tables are seldom written but very often read. For example it is more common to process a single call than it is to add a subscriber and it is more common to route a PDU packet than it is to change the routing tables.

When we execute performace critical code, we do not want to impose the overhead of an entire transaction in order to read data. On the contrary, if for example packet routing code reads routing information from a routing table while the routing table is being updated, it is acceptable that some packets get lost due to this access conflict. What is needed here, is very lightweight locking protection so that application processes can access the data tables and be certain that each data object that is read, is not garbled due to concurrent writers. This is supported by Mnesia through a so called dirty interface. It is possible to read, write and search Mnesia tables without protecting the operation inside a transaction. These dirty operations are true realtime DBMS operations: they take the same predictable amount of time regardless of the size of the database.

3.7 Queries

Apart from traffic processing, telecommunications systems contain substantial amounts of operational maintenance (O & M) code. For example, when we delete a subscriber from a switching system, we need to search several tables for data which is associated with this subscriber, hence the need for a query language. Operational and maintenance code is characterized by the following properties:

1. It has none or very low realtime requirements.
2. It reads, searches and manipulates large parts of the traffic data.
3. It constitutes a large part of the code volume of the system.
4. It is seldom (if ever) executed, thus subject to software rot and consequently inherently buggy.

Thus, a powerful query language which executes on the target system and has complete access to all traffic tables, can remedy (3) by making the O & M code smaller and (4) by being declarative and by being able to automatically adapt to changes in table layout or network topology. Since an optimizing compiler is

used to decide the execution order of the query, O & M code can also become more efficient.

The Mnesia query language is based on list comprehensions. This idea has been exploited in several other functional DBMSs such as [15]. The syntax of list comprehensions blend perfectly with the Erlang programming language.

3.8 Schema alteration

The Erlang programming language has extensive support to change the code of executing processes without stopping the process. It is possible to change the layout or organization of Erlang data without stopping the system. Thus, it is also possible to change the Mnesia schema at runtime without stopping the system. Since Mnesia is intended for nonstop applications, all system activities such as performing a backup, changing the schema, dumping tables to secondary storage and copying replicas have to be performed in the background while still allowing the applications to access and modify tables as usual. We believe that this is a requirement which is satisfied by few, if any, of the commercial DBMSs.

4 Some implementation aspects

Mnesia is entirely implemented in Erlang. The Erlang programming environment has turned out to be the ideal vehicle for the implementation of a distributed DBMS and the entire implementation of Mnesia including all aspects of the system from low level storage management to the optimizing query compiler is small and consists of not more than approximately 20000 lines of Erlang code.

Persistent storage is implemented on top of the underlying operating system file system. The disadvantage of this is performance of disc operations and the major advantage is portability. Since Mnesia is primarily intended to work as primary memory DBMS, we feel that the portability aspect is the more important. Tables and indexes in primary memory are implemented as linear hash lists [13] and secondary storage tables are implemented as named files. Each file is organized as a linear hash list with a medium chain length of the hash bucket set to a small value. The linear hash list is very efficient for lookup operations and reasonably efficient for insert operations. Files and tables can grow and shrink dynamically. Space management on each file is performed through a buddy algorithm.

The Mnesia lock manager uses a multitude of traditional techniques. Locking is dynamic, and each lock is acquired by a transaction when needed. Regular twophase locking [6] is used and deadlock prevention is traditional waitdie [14]. The time stamps for the waitdie algorithm are acquired by Lamport clocks [12] maintained by the transaction manager on each node. When a transaction is restarted, its Lamport clock is maintained, thus making Mnesia live lock free as well. The lock manager also implements multi granularity locking [10]. Traditional twophase commit [9] is used by the transaction manager when a transaction is commits.

Simple queries are evaluated by the relational DBMS technique operators [8], whereas recursive queries are evaluated by SLG [3] resolution. Since Mnesia is running on top of distributed Erlang the implementation is greatly simplified. In a distributed application there are separate Erlang nodes running on (usually) different machines. Erlang takes care of the communication between processes possibly on separate nodes transparently. Distributed Erlang works also transparently across different computers with different endianism, thus a Mnesia system can consist of a set of heterogenous computer systems. Processes and nodes can easily be started, supervised and stopped by processes on other nodes. This makes much of communication implementation difficulties disappear for Mnesia as well as for applications.

5 Performance discussion

In this section we provide some measurements on the Mnesia system. The figures clearly indicate that:

- The cost of using the transaction system, as opposed to using the dirty interface, is substantial. We believe that the correct interpretation of this is that the dirty interface is fast and not that the transaction system is slow.
- The cost of replication is fairly high. This is expected since the computers used in the test are interconnected by an ordinary shared media 10 Mbit/sec Ethernet.

The computers in the test are three Sun UltraSparcs running Solaris 2.5. All transactions are initated from the one UltraSparc at 167 Mhz and the other two machines run at 143 Mhz.

number of replicas	1	2	3
`divorce/1`	1877	5009	13372
`divorce/1` using wread	1225	4703	12185
dirty `divorce/1`	181	592	1121

Table 1. Wallclock microseconds to execute divorce/1 with different configura tions

In the first row, we run the function `divorce/1` from section 2. In the second row we run a version of divorce using a Mnesia function `wread/1` instead of `read/1`. This function reads the object but sets a write lock instead of a read lock. This is more efficient if we know that we are going to subsequently write the same object. This way the lock need not be upgraded from a read lock to a write lock. Finally in the last row we use the dirty functions to read and write the replicated tables, thus bypassing the transaction system and using the lightweight locks.

6 Conclusions

There exists today a very large number of DBMSs, a large number of commercially available systems as well as an uncountable number of research systems. It could appear as if it would be a better solution to use a commerical DBMS but if all the aspects from section 3 are taken in account, no suitable commercial DBMSs exists. We feel that our main contributions are the following.

- We have implemented an entire distributed DBMS by combining a large number of well known techniques. Many research groups chose to study some aspects of DBMSs only, we have implemented a full distributed DBMS. Few such implementations exist.
- We have showed that Erlang is very well suited for not only telecommunications system but also for the implementation of DBMS systems like Mnesia. To our knowledge this is the first time a distributed DBMS is implemented in a symbolic programming language.
- We have provided a DBMS solution that address all, or at least many, aspects of data management in telecommunications systems.

The Mnesia system is currently being used to build real products in Ericsson today, thus it is no longer a mere prototype system, it has matured enough to be labeled a product. The system is available at http://www.ericsson.se/erlang

References

1. Armstrong, J. L., Williams, M. C., Wikström, C. and Virding, S. R., Con current Programming in Erlang, 2:nd ed. Prentice Hall (1995)
2. Bernstein, P.A., Hadzilacos, V., Goodman, N. Concurrency Control and recovery in Database Systems Addison Wesley, 1987.
3. Chen, A.W., Warren, D.S. Query Evaluation under the WellFounded Se mantics Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys. Whashington, 1993.
4. Case, K. McCloghrie, M. Rose, S. Waldbusser. Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2), Jan, 1996.
5. Copeland, G., Maier, D. Making Smalltalk a database system Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. pp. 316325. Boston 1984.
6. Eswaran, K.P., Grey, J.N., Lorie, R.A. and Traiger, I.L. The Notions of Consistence and Predicate Locks in a Database system Communications of ACM, 19(11):624633, November 1976.
7. Faehndrich, M., Morrisett, G., Nettles, S., Wing, J. Extensions to Standard ML to Support Transactions ACM SIGPLAN Workshop on ML and its Applications, June 2021, 1992.
8. Goetz, G. Query Evaluation Techniques for Large Databases ACMCS 2(25):73170, June 1993.
9. Grey, J.N. Notes on Database operating system: An advanced course Lecture notes in Computer Science,Springer Verlag, Berlin. 1(60):393481, 1978.

10. Grey, J.N., Lorie, R.A., Putzolo, G.R. and Traiger, I.L. Granularity of Locks and degrees of consistency in a shared database IBM, Research report RJ1654, September 1975.
11. Hvasshovd, SO., Torbjornsen, O., Bratsberg, S.E., Holager, P. The ClustRa telecom database: High availability, high throughput, and realtime response Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland, pp. 469477, September 1995.
12. Lamport, L. Time, clocks and the ordering of events i a distributed system ACM Transactions on Programming Languages and Systems, 21(1):558565, July 1878.
13. Larsson, PÅ Larsson. Dynamic Hash tables Communications of the ACM, 31(4), 1988
14. Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. System Level Concurrency Control for Distributed Databases ACM Transactions on Database Systems, 3(2):178198, June 1978.
15. Trinder, P.W. and Wadler, P. List comprehensions and the Relational Cal culus Proceedings of the Glasgow 1988 Workshop on functional programming, Rothesay , August 1988, pp 115123.
16. Ullman, J. Principles of Database and KnowledgeBase Systems, vol 2. Computer Science press, 1989.

Subject Index