UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

Eventual Consistency in Mnesia

Shuntian Liu

Magdalene College

Sat 17th Jun, 2023

Submitted in partial fulfilment of the requirements for the
Computer Science Tripos, Part III

Total page count: 84
Main chapters (excluding front-matter, references and appendix): 39 pages (pp 1–39)
Main chapters word count: 11876
Methodology used to generate that word count:
[

```
$ texcount -inc -alphabets=Latin -unicode -sum -1 report.tex
```

]

Declaration

I, Shuntian Liu of Magdalene College, being a candidate for the Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed: Shuntian Liu

Date: 01/06/2023

Abstract

Mnesia is a soft real-time embedded Database Management System written for Erlang, a programming language that powers the infrastructures of various organisations like Cisco, Ericsson and the NHS. Due to Mnesia's tight integration with Erlang, it is also impactful in open source projects such as RabbitMQ and ejabberd.

However, the development of Mnesia has remained stagnant for years, resulting in the lack of features such as automatic conflict resolution: Mnesia leaves the handling of conflicts after network partitions entirely to the developer. Moreover, as a distributed database, Mnesia only provides two extreme forms of consistency guarantee: transactions and weak consistency. Existing solutions to this problem are either external libraries or commercial standalone products, none of which is integrated into Mnesia natively. This means Erlang developers often have to introduce new dependencies into their codebase or resort to less ideal alternative databases.

To address this issue, we propose a new consistency guarantee for Mnesia: eventual consistency (EC). The benefit of this is twofold: first, EC introduces an intermediate consistency guarantee between transactions and weak consistency, offering more choices to developers; second, this implementation of EC with Conflict-free Replicated Data Types (CRDTs) enables automatic conflict resolution after a network partition.

We have implemented EC as an extension to Mnesia named *Hypermnesia* and evaluated its correctness, efficiency and usability. Evaluation results show that Hypermnesia's EC operations can produce consistent results in the presence of partitions and perform more than 10 times faster than Mnesia's default transactions. Moreover, Hypermnesia's API enables minimum code refactoring for adoption in real-world systems. We hope Hypermnesia can be integrated into Mnesia and used by Erlang developers in the future.

Acknowledgements

I am grateful to:

- **Evangelia Kalyvianaki** and **Andreas Grammenos** for their patient guidance and encouragement throughout this project.
- **Natalia Chechina** for her expert knowledge in Erlang and Mnesia ecosystem, and the rest of the Erlang Solutions[*] for their professional industrial insights.
- **John Fawcett** for his support during my Part III study and beyond.
- **Martin Kleppmann** for his advice on CRDTs.
- **Han** for his feedback on this report.

[*]`https://www.erlang-solutions.com`

Contents

List of Figures

List of Listings

Chapter 1

Introduction

This project report presents *Hypermnesia*, an extension to the Mnesia Database Management System (DBMS) [18]. Hypermnesia provides eventual consistency (EC) guarantee on top of Mnesia's transactional semantics. In this chapter, I first discuss the motivation for building Hypermnesia (§1.1), then list the challenges and contributions in building Hypermnesia (§§1.2 and 1.3). After that, an outline of the report structure is given (§1.4).

1.1 Motivation

Mnesia is an embedded distributed Database Management System (DBMS) designed for industrial-grade communication systems written in Erlang [18]. It is influential in its usage across companies and organisations like Cisco, Goldman Sachs and Mastercard [11]. It is also used in open source highly scalable message brokers and XMPP services, such as RabbitMQ [56], MongooseIM [23] and ejabberd [44]. Furthermore, Mnesia is part of the Linux, Yaws, Mnesia, Erlang (LYME) software bundle for building dynamic web pages that can handle heavy traffic [62].

Despite its usage in heavy-load applications such as WhatsApp [58] that handle tens of thousands of messages per second [35], the development of Mnesia has stayed relatively stagnant for many years[*]. Due to Mnesia's highly specialised nature and complex codebase, extending it often requires a combination of research knowledge and language expertise. This has deterred many developers from contributing to Mnesia, which means Mnesia lacks features that many modern distributed databases have. Examples include the lack of non-transactional guarantees and automatic recovery after network partitions.

Mnesia provides two ways to access the database: transactions and dirty operations. Transactions provide ACID guarantees [27], while dirty operations only give weak consistency [57]. Developers have to choose either the strong transactional API while sacrificing performance and availability during network partitions, or use dirty operations for fast performance, but risk their data being inconsistent. Moreover, Mnesia leaves the handling of potential data inconsistency after a partition entirely up to the developer [18], which means a manual restart of the cluster is often needed. Indeed, developers have reported that "*the experience has convinced us that we need to prioritize up our network partition recovery strategy*" [41] while handling a partitioned Mnesia cluster.

There are plenty of consistency models that are weaker than transactions but stronger than dirty

[*]Much of the core code modified in this project are over 13 years old, dating back to the time when Erlang was published on Github.

operations' weak consistency [19]. One such model is eventual consistency [57], which guarantees convergence eventually when no updates are made to the data. This is an attractive model for Mnesia since it often has better performance than transactions, while also maintaining consistency eventually. Modelling data this way can serve as a viable approach for achieving data consistency after a network partition.

To this end, I propose the following research questions (RQs):

RQ1. *Can we introduce a non-transactional consistency guarantee, for example, eventual consistency, into Mnesia?* Mnesia was developed and designed before many of these consistency models became popular [38], and therefore retrofitting them into Mnesia can be a non-trivial task.

RQ2. *How much overhead, in terms of time and space, will this new guarantee bring, compared to Mnesia's existing operations?* Mnesia's tight integration with Erlang gives it outstanding performance compared to other standalone databases. Therefore, it is important to evaluate the overhead of the new operations to ensure they are still competitive.

RQ3. *Can eventual consistency help us achieve automatic conflict resolution after a partition in Mnesia?* The fault tolerance model of Mnesia (§4.5) may present extra challenges in this otherwise straightforward extension.

RQ4. *How much code refactoring is needed for the consistency guarantee to function?* Mnesia is used in many large-scale (legacy) codebases, which might be too expensive to do refactoring. Therefore it is important to understand the cost of adopting a new consistency model API in Mnesia.

1.2 Challenges

There are several technical challenges (TCs) that need to be overcome to design, implement and evaluate Hypermnesia:

TC1. Mnesia is a rather old database system that has not received much attention for years. This means the codebase contains legacy code and is not well documented.

TC2. The previous point implies that there are potentially legacy projects using Mnesia. The new feature, therefore, needs to be backwards compatible and ideally easy to be adopted by developers.

TC3. Another consequence of Item TC1 is that its test suite does not include network partition tests, which are essential in testing the correctness of eventual consistency. Moreover, its benchmark suite is only written for transactions. Both of these need to be extended to evaluate Hypermnesia.

TC4. There are multiple ways to achieve eventual consistency. Therefore an informed choice needs to be made based on Mnesia's existing architecture. A survey of existing methods and a study of the relatively undocumented architecture of Mnesia is essential.

1.3 Contributions

- **A new eventual consistency model for the Mnesia DBMS.** A new set of APIs is designed for this new model to minimise the amount of code refactoring for existing codebases. Moreover, with this new API, automatic conflict resolution is possible after a network partition, relieving developers from the burden of manually restarting the database.

- **An extension of the test suite for network partitions.** These are used to test the correctness of Hypermnesia. It might be useful for testing existing functionalities such as transactions as well.

- **An extension of the existing benchmarking library for dirty operations and EC operations.** These are built on top of the original benchmarks for transactions.

- **An evaluation of the new eventual consistency API**, in terms of the following: (a) fault tolerance by testing its correctness under network partitions and ability to reconcile after a partition; (b) performance by analysing time and space overhead and comparing against existing operations. Results show that Hypermnesia's EC operations can often achieve 10x higher throughput and lower latency than Mnesia's transactions; (c) usability by analysing the amount of code refactoring needed to adopt the new API in real codebases.

1.4 Outline

The rest of the report is organised as follows: Chapter 2 introduces the reader to the background knowledge, including the architecture of Mnesia and different ways of achieving eventual consistency. Chapter 3 describes recent research on eventually consistent databases and CRDTs: a data structure for implementing eventually consistent systems. Chapter 4 goes into detail about designing an API for Hypermnesia and implementing it with practical optimisations. Chapter 5 evaluates Hypermnesia in terms of its correctness, performance and usability. Chapter 6 concludes the report and outlines potential future work.

Chapter 2

Background

This chapter introduces consistency models in a distributed system relevant to this work (§2.1). It then discusses the architecture of Mnesia (§2.2) before giving an overview of CRDTs.

2.1 Consistency models

A distributed system often involves a set of replicated state machines [34]. Coordinating these clusters of machines is a challenging task, and tradeoffs can be made between consistency and performance. Many consistency models are formalised for both transactional and non-transactional systems [55]. This section gives a brief introduction to three of them in detail: distributed transactions (provided by Mnesia's transaction API), weak consistency (provided by Mnesia's dirty operations) and eventual consistency (the aim of this project).

Distributed transactions is sometimes also called distributed atomic commits. It is achieved if all nodes commit or all nodes abort [48]. This is part of the properties provided by ACID [27] and is often implemented with the distributed two-phase commit protocol [9]. Mnesia provides such a guarantee via its transaction APIs.

Weak consistency is defined as follows: "The system does not guarantee that subsequent accesses will return the updated value, and several conditions need to be met before the value will be returned" [8, 55, 57]. It is (deliberately) vaguely defined to incorporate systems whose replicas "might become consistent by chance" [8]. Mnesia's dirty operations fall into this category.

Eventual consistency is defined as follows: "If no new updates are made to the object, eventually all accesses will return the last updated value" [57]. There are often two steps in achieving eventual consistency [63]: (i) anti-entropy [13] disseminates data to all the nodes in a cluster; (ii) conflict resolution addresses potential conflicts when handling received data.

Anti-entropy typically involves some form of broadcasting messages, while conflict resolution tends to use (one of) the following three techniques: [30]: *Conflict-free Replicated Data Types (CRDTs)* [42, 50] *Mergable persistent data structures* [24] and *Operational transformation* [14].

Among these three ideas, CRDTs is one of the database community's most used and well studied approaches. It has been successfully deployed in many NoSQL databases, such as Riak [33], Redis [47] and Microsoft Azure Cosmos DB [52]. Operational transformation requires a central coordinator, which is unsuitable for Mnesia as it uses a leaderless replication strategy (§2.2.2). Mergable persistent data structures are based on version control ideas, but multi-version support is not available in Mnesia, meaning this method is not easily implementable in Mnesia. In conclusion, CRDTs is chosen as the basis of conflict resolution and discussed in more detail in §2.3.

2.2 Mnesia

Mnesia is an embedded [45], specialised distributed DBMS for Erlang/OTP applications (§§ 2.2.1 and 2.2.2). It is similar to relational databases where each table stores tuples (§ 2.2.3), but also different since it uses Erlang as its query language instead of SQL. It has dual mode support for accessing data (§2.2.4), discussed in more detail below.

2.2.1 Design goals

The original design of Mnesia attempts to meet several requirements [18]:

1. Fast real-time key-value lookup
2. Complex non-real-time queries (mainly for operation and maintenance tasks)
3. Distributed data (due to the distributed nature of the applications)
4. High fault tolerance
5. Dynamic reconfiguration

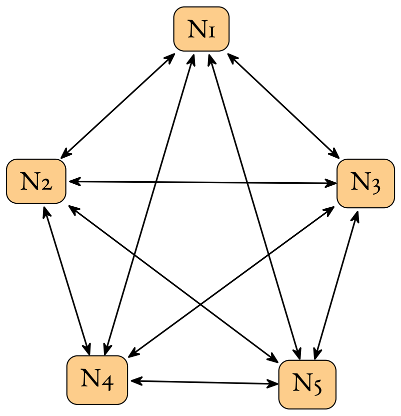
Based on these, Mnesia was built as part of the Erlang/OTP distribution. Erlang was originally designed for building fault-tolerant telecommunication systems, and Mnesia helps it to better achieve that goal by acting as an embedded database: it is tightly coupled to Erlang, giving it two distinct features: (a) The database runs in the same address space as the application itself, offering minimum overhead while accessing data. (b) The database uses native Erlang records to store its data, removing the impedance mismatch between different data formats.

These special features make Mnesia only suitable for specific purposes. Mnesia is typically used when there is a need to replicate a relatively small amount of data: compared with standard SQL databases that can handle terabytes of data, Mnesia is instead built for (tens of) gigabytes of data [28]. For example, user login details are often stored as session data, and to scale the application out, these data need to be replicated across nodes and accessed quickly to provide a good user experience. Mnesia can be a suitable candidate in such a case due to its compelling performance.

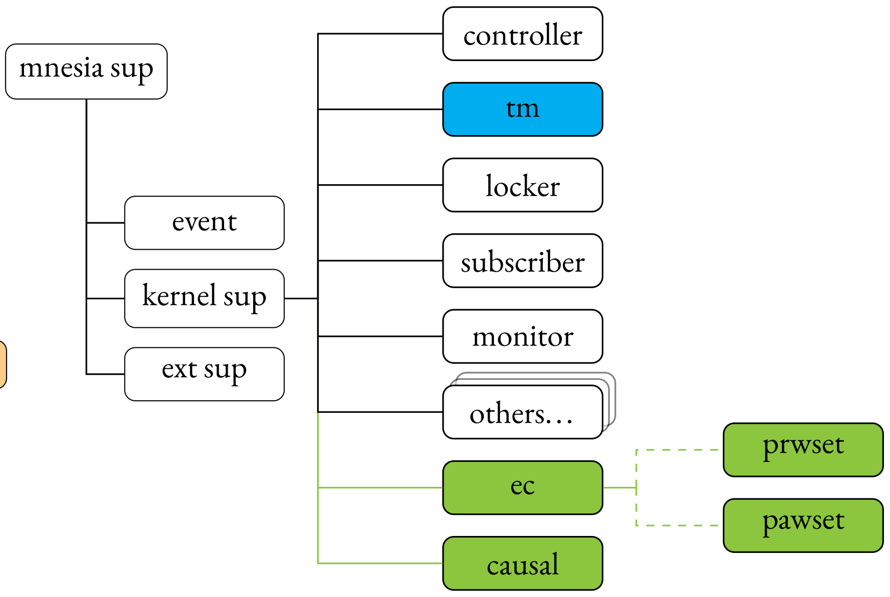
2.2.2 Architecture

Mnesia is built on top of Erlang's built-in memory and disk term storage `ets` and `dets` [22]. These term storage can be thought of as primitive storage engines that provide constant (or logarithmic) access time for large amounts of data [28]. They support different data structures for storing data, such as set, bag, etc. Internally, these are implemented as hash tables or balanced binary trees. Mnesia also provides additional functionalities such as transactions and distribution on top of `ets` and `dets`.

A Mnesia cluster generally has a leaderless architecture where every replica can handle client requests. A cluster of Mnesia nodes are connected via the Erlang distribution protocol, which uses TCP/IP as its carrier by default, providing reliable in-order delivery. Moreover, the connection is transitive, like Figure 2.1a which forms a cluster of fully connected nodes (or a mesh).



(a) Example Mnesia cluster of five nodes. Note they always form a fully connected network.



(b) Relationships between various Mnesia processes, in the form of a supervision tree. Most of the logic for replication is inside `mnesia_tm` (highlighted in blue). Modules added by Hypermnesia are highlighted in green. More details on the new modules are discussed in §4.2. Diagram partly based on [39].

Figure 2.1: Mnesia cluster architecture and code structure.

Figure 2.1b shows a brief overview of the modules in Mnesia according to the supervision relation. Erlang's fault tolerance model has the concept of a supervision tree [20], a hierarchy of processes in charge of monitoring/supervising child processes for killing and restarting "misbehaving" processes.

2.2.3 Data representation

Mnesia stores data in Erlang records [22]. Figure 2.2a is an example of a student record, and this is how each row of a Mnesia table is represented, with Figure 2.2b as the corresponding definition in Erlang.

2.2.4 Access contexts

A central API provided by Mnesia for table manipulation is given in Listing 2.1. A user calls the `activity` function which takes in:

1. A `Kind` of activity, also called a *access context*. Currently supported contexts include transactions and dirty operations;

student	id (key)	name	college	age
	bb123	Bruce Banner	Avengers	54
	ts233	Tony Stark	Avengers	50
	sg333	Steve Rogers	Avengers	100

(a) An example Mnesia table for student information.

```
-record(student,
  {id :: integer(),
  name:: string(),
  college :: string(),
  age:: integer()}).
```

(b) Erlang record definition for a student.

Figure 2.2: Example Mnesia table and its definition for students.

2. A function (`Fun`);

3. Arguments to be passed to the function (`Args`);

4. The module in which the function is defined (`Mod`).

There are two almost equivalent[*] ways of using this API, and Listing 2.2 provides an example of writing a tuple `{k,v}` into the table `tab_name` and then reading it out. I use the convention of `function_name/arity` to represent Erlang functions in these code examples and the rest of this report.

```
mnesia:activity(Kind, Fun, Args :: [Arg :: term()], Mod) -> t_result(Res) | Res
  where Kind = activity()
  Fun = fun((...) -> Res)
  Mod = atom()
```

Listing 2.1: Mnesia table manipulation API.

```
mnesia:activity(transaction, fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(a) transaction with `activity/2`

```
mnesia:activity(async_dirty, fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(b) asynchronous dirty with `activity/2`

```
mnesia:transaction(fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(c) transaction with `transaction/1`

```
mnesia:async_dirty(fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(d) asynchronous dirty with `async_dirty/1`

Listing 2.2: Comparing transaction and asynchronous dirty operations API.

[*]Subtle differences between these two APIs can be found in the reference guide [19]

Transactions in Mnesia provide ACID properties [18]. While this is attractive, it also incurs large performance overhead and hence a full-fledged transaction might not be suitable for all tasks. For example, in a datagram routing application, it can be too slow to initiate a transaction each time a packet is received [18]. The underlying implementation of Mnesia uses two-phase commit for distributed transactional atomicity, write-ahead logging for durability and consistency, and two-phase locking for isolation.

Dirty operations in Mnesia bypass most transactional processing and operate directly on the data. Therefore they are much faster (at least 10x [19]) than transactions. As a consequence, they lose the ACID properties. However, dirty operations still provide some level of consistency such as no garbled records for individual dirty reads. The underlying implementation of dirty operations follows this pattern: when the function is called, the operation is first performed on the local table. Then relevant information is collected including the Erlang record, the target table, and so on. These are then sent to other replicas in the cluster. Mnesia does not examine the content in the table during this process. This property is useful in helping us choose a CRDT, as we will see later that some CRDTs require examinations of the state before broadcasting the message (§4.1.2). On the receiving side, the transaction manager monitors and applies the received message accordingly.

2.2.5 Consistency and availability

The CAP theorem [25] forces a system to choose between consistency and availability during a network partition. Mnesia responds to this problem by taking *no* stance with respect to the CAP theorem [2], but leaves the handling of the recovery process entirely to the developer [18], often resulting in a manual restart of the Mnesia nodes after partitions. Recent improvements have incorporated the feature to allow developers to set the `majority` option which disallows any non-dirty operation to a table not in a majority quorum. This option is useful for preventing conflicts for critical data, but sacrifices the availability of the database further and only works for transactions. There is also a `set_master_nodes/2` function which unconditionally loads data from the master node after a partition. It can be useful in some cases, but is a rather "brutal" way of handling partitions since it might lead to data loss.

Erlang's built-in distribution protocol has a failure detector that periodically sends heartbeats. If a node does not receive a heartbeat from another node for some time, then the failure detector considers that node dead and disconnects itself. Mnesia relies on Erlang's failure detector to function, and two possible situations can occur during a network partition (NP):

NP1. The partition is a *transient failure* in which the failure detector does not consider a temporarily unresponsive node dead. In this case, operations can carry on as usual, but transactions will stall since the two-phase commit protocol requires responses from all participating nodes. Dirty operations can be performed as usual. When the partition heals, buffered messages will then be delivered. But dirty operations risk the database being in an inconsistent state when the network heals, as we shall see in §4.5.1.

NP2. The partition lasts long enough that the failure detector detects the *communication failure* and disconnects the nodes it thinks have failed. In this case, Mnesia would reconfigure itself and perform future operations without the failed nodes, i.e. Mnesia considers itself as having a new cluster with only the members still alive. When the partition heals, Mnesia logs an error message saying that the database might be inconsistent and ask the developer to fix this issue manually.

2.3 CRDTs

Conflict-free Replicated Data Types (CRDTs) [42, 50] are a family of replicated data types with a common set of properties that enable operations to be performed locally on each node while always converging to a final state among replicas if they receive the same set of updates. There are two types of CRDTs: state based (§2.3.1) and operation based (§2.3.2). Each of them is discussed in detail with examples below.

2.3.1 State-based CRDTs

Here I provide a summary of the properties of state-based CRDTs. Refer to Appendix C and [1, 42, 50, 54] for mathematical details and examples.

A state-based CRDT communicates by propagating its state of the data structure with other parties, and performs the merge operation with a merge (sometimes called join) operator to converge towards the Least Upper Bound (LUB) of two states. The merge operator must be commutative, idempotent and associative to guarantee the convergence property (Proposition 2.3.1), which is key to eventual consistency.

Proposition 2.3.1 ([50]). *Any two object replicas of a state-based CRDT eventually converge, assuming the system transmits payload infinitely often between pairs of replicas over eventually-reliable point-to-point channels.*

Propagating the entire states of a (big) data structure can often be expensive. Therefore Delta-State Conflict-Free Replicated Data Types (δ-CRDTs) [1] is proposed. It disseminates only the changing parts of the state as a δ-state, reducing communication costs. Due to their simple requirements on the channel, δ-CRDTs often have complex logic in managing and disseminating states.

2.3.2 Operation-based CRDTs

Operation-based (op-based) CRDTs send the operation performed on the CRDT object (as opposed to the state) along with some metadata. An op-based CRDT typically requires two *concurrent* operations f and g to be commutative under the causal delivery order $<_d$: $f \parallel_d g \iff f \not<_d g \wedge g \not<_d f$.

For this reason, a causal delivery channel is often required for op-based CRDTs to work. Given such a channel, the following property (Proposition 2.3.2) holds:

Proposition 2.3.2 ([50]). *Any two replicas of an op-based CRDT eventually converge under reliable broadcast channels that deliver operations in delivery order $<_d$.*

One advantage of op-based CRDTs is their communication efficiency. Using a set as an example data structure, instead of sending the entire set with all its elements, op-based set can just send operations like `add` or `delete`. This could significantly reduce the communication cost, especially when the set is large.

Given the stronger assumption on the channel, designers of op-based CRDTs only need to choose how they want to order concurrent operations, an example of a particular type (which is also the one implemented in Hypermnesia) of op-based CRDTs is given below.

Pure op-based CRDTs

The definition of the original op-based CRDTs makes the separation between state-based and op-based CRDTs less clear in that an op-based CRDTs can often include state information while sending operations. For example, before sending operation `add(1)` on a set, the state of the set can be inspected and the entire set can be added as the "operation". It also has a relatively high implementation complexity [6]. To address this, pure op-based CRDTs were developed [4, 5], which forbids inspecting the state σ while generating the message m.

I now illustrate pure op-based CRDTs with an example add-wins set (AW-set). Listing 2.3 is the conceptual view of implementing it. Typically there are three operations on an op-based CRDT: a generator `prepare`, which turns the operation o_i on the current state σ_i into a message m_i that is sent via reliable causal broadcast; an effector `effect`, which receives the message m and applies it onto the current state σ_j and produces a new state σ_j'; and `eval`, which takes an operation, such as lookup an element in a set and the current state σ_i, and returns the result of the operation. For pure op-based CRDTs, data and timestamps need to be stored in a *Partially Ordered Log (PO-Log)* [4, 5], partially ordered by the timestamp attached to each operation. This aims to identify causal relationships between different elements, which is important for the convergence property (Proposition 2.3.2).

$$\begin{aligned}
\Sigma &: T \hookrightarrow O \\
\mathrm{prepare}_i(o, s) &= o \\
\mathrm{effect}_i(o, t, s) &= s \cup \{(t, o)\} \\
\mathrm{eval}_i(\mathrm{elems}, s) &= \{v \mid (t, [\mathrm{add}, v] \in s \wedge \nexists (t', [\mathrm{rmv}, v] \in s \cdot t < t'))\}
\end{aligned}$$

Listing 2.3: A pure op-based AW-set implementation, adapted from [5].

The PO-Log (Σ) for the AW-set is a partial function from timestamps to operations (with appropriate payload). Note that the effector takes three arguments, the operation o, timestamp t and state s, and it simply unions the operation and timestamp with the current state (i.e. appending it to the log), leaving the complexity to the lookup function. This is a fairly space-expensive operation, since all operations need to be stored in the log including deletions. Optimisations are discussed later in §4.4.

2.4 Summary

In this chapter, we discussed transactions, weak and eventual consistency (§2.1), the first two are offered by Mnesia (§2.2), and this project aims to provide the last. A family of data structures, (CRDTs) are also described (§2.3) to show how they help achieve eventual consistency.

Chapter 3

Related work

The field of database research is a dynamic and broad domain that encompasses a variety of topics. This chapter focuses on the architecture and design of eventually consistent databases (§3.1). Similarly, for CRDTs (§3.2), this chapter surveys them in the context of how they impact the design and implementation of Hypermnesia.

3.1 Databases and eventual consistency

3.1.1 Mnesia and eventual consistency

Mnesia's lack of automatic conflict resolution has been problematic for developers using it [41, 61]. Unsplit [60], an external library for Mnesia, allows for user-defined merge logic but requires developers to define the custom logic. ForgETS [58] is WhatsApp's Mnesia "drop-in" replacement database, aiming to provide Mnesia's missing features such as auto reconciliation and auto reconnection. ForgETS uses the last-write-wins strategy for conflict resolution. However, ForgETS is a proprietary database influenced by WhatsApp's operational experience, and the details of its effectiveness are unknown. Moreover, ForgETS is a standalone database built on top of `ets`, while Hypermnesia aims to extend Mnesia to support automatic resolution natively.

3.1.2 Key-value stores

Key-value stores like Redis[*] serve as a caching layer between servers and databases, offering an Active-Active architecture that allows replicated database instances to distribute over different (possibly distant) locations. Redis is an advanced system with many more features such as expiring keys, and uses op-based CRDTs for conflict resolution between different instances [47]. While Mnesia has a similar use case, it is primitive and embedded into the Erlang ecosystem. Mnesia can provide a faster response time than Redis when used on a smaller scale, thanks to its shared address space with the application. This research aims to enhance Mnesia with stronger non-transactional consistency models found in a general purpose in-memory database (like Redis), reducing the constraints when developers opt for Mnesia as the choice of their Erlang applications.

[*]`https://redis.com`

3.1.3 Multi-version concurrency control

SwiftCloud [43] is a fault-tolerant geo-replicated transactional system. It provides causally consistent snapshots for each transaction with object versioning. As a geo-replicated system, SwiftCloud allows local operations to be performed on the client without going through the server but only when they operate on mergeable data types (e.g. CRDTs) during a transaction. Antidote database [51] goes one step further, integrating TCC with stronger consistency models and allowing developers to choose between them.

Multi-version Concurrency Control (MVCC) is a popular technique used in conjunction with CRDTs to provide eventual or sometimes even transactional consistency. Mnesia does not support object versioning by default, so adding MVCC requires extensive work. In this project, we focus on adding eventual consistency and leave the integration of MVCC and CRDTs as future work (§6.2).

3.1.4 Augmenting existing embedded databases

SQLite[†] is one of the most widely used embedded SQL database engines in the world [29]. It does not have built-in support for replication, but there has been work that extend it for local-first software [53] with Conflict-free replicated relations (CRRs): CRDTs applied to relational databases [65]. They use a two-layer architecture: an application layer for handling client requests, and a CRR layer for conflict resolution and anti-entropy protocols. However, their work still remains in progress and it is not clear how effective their approach is in terms of overheads. ElectricSQL[‡] is another attempt to augment SQLite with local-first behaviours. It uses RichCRDT [3], which are CRDTs extended with database guarantees, based on ideas from Antidote.

These techniques for enhancing SQLite are similar in spirit to Hypermnesia. However, SQLite is a rather general purpose single-node database receiving constant developments. In contrast, Mnesia is integrated with the Erlang ecosystem and designed as a distributed database, hence they have different use cases. Moreover, to the best of our knowledge, there is no performance evaluation on these extensions to SQLite yet.

3.2 CRDTs research

3.2.1 Systems using CRDTs

CRDTs are often used in collaboration software such as shared text editors [59] to provide local-first behaviours [32], i.e. users of the editors can keep typing into the document despite unstable networks. It has then found its use in areas such as synchronisation on the edge and opportunistic networks [26, 64]. These systems typically have constrained network or computing resources and CRDTs can be used to delay synchronisation and respond to the user first.

[†]https://sqlite.org/index.html
[‡]https://electric-sql.com

Researchers have been investigating new areas where CRDT techniques can be applied. Hypermnesia does not aim to apply CRDTs in a novel way but rather to experiment with their suitability in an embedded database like Mnesia.

3.2.2 Time and space improvements

CRDTs are constantly invented and improved to reduce their resource consumption. For example, the proposal of δ-CRDTs [1] is partly due to the communication overhead of propagating the entire state in normal state-based CRDTs. Pure op-based CRDTs [4] were also proposed to reduce the space overhead. Bauwens and Boix [6] further improve the reactivity and storage requirements of pure op-based CRDTs by exposing more information from the causal broadcast layer. van der Linde et al. [54] suggest a new way to adapt δ-CRDTs for more unstable networks. Furthermore, Bauwens and Gonzalez Boix [7] show a more eager way to remove metadata using acknowledgements from replicas.

One research question of this project is to understand the overhead of eventual consistency in Mnesia. The space and time optimisations proposed are useful for improving Hypermnesia's efficiency, therefore they are investigated in detail, and some of their ideas are used in Hypermnesia's implementation (§4.4).

Chapter 4

Design and implementation

This chapter provides an account of Hypermnesia's design and implementation. Hypermnesia's API and the selection of a CRDT are discussed in § 4.1, with alternative options also considered. Subsequently, the implementation is outlined in §4.2 at a high level before diving into three key components: causal broadcast, Set CRDT and fault tolerance from §§4.3 to 4.5.

4.1 Design

4.1.1 API design

Hypermnesia's API is designed with the research question Item RQ4 on refactoring in mind. We consider the SOLID principle [37] with the following design goals (DGs):

DG1. **Backwards compatibility and code refactoring.** Hypermnesia should be backwards compatible with the existing codebase and should run normally without modification if developers choose not to use the new feature. Moreover, if they opt for the new feature, the API should minimise the amount of refactoring needed for an existing codebase.

DG2. **Single responsibility.** The new feature should be contained in module(s) isolated from existing Mnesia code, while respecting Mnesia's code structure. This makes the implementation self-contained and easier to maintain.

DG3. **Extensibility.** Hypermnesia should be extensible to new implementations. This is beneficial since there are various possible CRDTs, e.g. operation-based and state-based. They often achieve similar goals but rely on different assumptions. Hypermnesia should be extensible to new CRDT variants for different tradeoffs.

Based on the existing APIs provided by Mnesia (§2.2.4), one way to extend it with new APIs for eventual consistency would be to add a new access context called `async_ec` so that database operations performed within this context are eventually consistent. Using the same example, we can change Listing 2.2 (the original API for transactions and dirty operations) to Listing 4.1 (the new EC API). We argue that this is a fairly natural extension to the existing Mnesia access contexts, and requires little refactoring of existing code (`dirty` or `transaction` changed to `ec`). It does not break existing features either if no changes are made to the existing code, thus fulfilling Item DG1.

```
mnesia:activity(async_ec, fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(a) `activity/2` with new access context `async_ec`

```
mnesia:async_ec( fun () ->
  mnesia:write({tab_name, k, v}),
  mnesia:read({tab_name, k})
end).
```

(b) `async_ec` with new function `async_ec/1`

Listing 4.1: New eventual consistency (EC) API based on existing Mnesia APIs, using the same example as Listing 2.2.

We also wish to allow programmers to declare which type of CRDTs they want to use. This can be done while declaring the table type as `pawset` at creation time. Mnesia asks users to enter the type of each table at creation time (Listing 4.2a). The default values are `set`, `bag`, etc. This can be extended to support pure AW-set (`pawset`) and RW-set (`prwset`) as well (Listing 4.2b). Note that having multiple Set CRDT implementations allow developers to choose the most appropriate one for their applications, demonstrating the extensibility of Hypermnesia (Item DG3).

```
mnesia:create_table(project,
                    [{type, set},
                    {ram_copies,
↪   all_nodes()},
                    {attributes,
↪   record_info(fields, student)}]).
```

(a) Creating a table of default type `set` and storing student record data Figure 2.2b.

```
mnesia:create_table(project,
                    [{type, pawset},
                    {ram_copies,
↪   all_nodes()},
                    {attributes,
↪   record_info(fields, student)}]);
```

(b) Creating a table of type `pawset` (pure add-wins set) and storing student record data. A `prwset` (pure remove-wins set) can be used as well.

Listing 4.2: New eventual consistency (EC) API based on existing Mnesia APIs.

Finally, different CRDT logic is implemented inside separate modules, aiming to achieve the single responsibility requirement (Item DG2). The new `mnesia_ec` module is built for handling data replication and interfacing with the underlying CRDTs, `mnesia_causal` module for causal delivery, and `mnesia_pawset/mnesia_prwset` module(s) for data storage and conflict resolution (Figure 2.1b).

4.1.2 Choosing a CRDT

We talked about different kinds of CRDTs in §2.3. One type of Set CRDT needs to be chosen for Hypermnesia's implementation. We consider two main factors (Fac):

Fac1. How efficient is this CRDT in terms of space and time?

Fac2. How easy does it fit into the existing Mnesia codebase and how many breaking changes need to be made?

On the one hand, state-based CRDTs (§2.3.1) have an important drawback in their communication overhead [1]. This might not be acceptable for data types with large state such as sets. δ-CRDTs is a more suitable candidate for our purpose but even with δ-CRDTs, a fair amount of data needs to be broadcast, especially as the number of operations increases (see Appendix C for details). Moreover, it might be difficult to examine the state before broadcasting, since Mnesia does not do this by default (§2.2.4). State-based CRDTs do have the advantage of low demand on the channel. Therefore reliable broadcasting in Mnesia would be sufficient.

On the other hand, operation-based CRDTs tend to require the channel to provide causal broadcast, which has to be implemented in Mnesia from scratch. Dynamic membership, i.e. nodes leaving and joining the cluster, is also tricker with operation-based CRDTs as buffering and replaying of operations are needed. Nevertheless, pure operation-based CRDTs largely resemble Mnesia's anti-entropy strategy, with no examination of the current content and immediate synchronisation for each operation [40]. State-based CRDTs are less suitable for these requirements [42]. Therefore pure op-based CRDTs better meets the second requirement (Item Fac2).

In terms of efficiency, it is challenging to characterise the performance metrics of each CRDT without practical benchmarking: δ-CRDTs uses periodic synchronisation of delta states, while pure op-based CRDTs go through an extra causal broadcast layer. Moreover, Almeida et al. [1] and Baquero et al. [5] proposed optimisations for these CRDTs. One could also argue that these two CRDTs are fundamentally the same as they are both doing the necessary work for conflict resolution, but at different layers of the system.

In conclusion, pure op-based CRDTs is chosen for its operational similarity with Mnesia's dirty operations. We leave experimenting of δ-CRDTs as future work (§6.2). Table 4.1 highlights the differences between these two CRDTs.

	State-based	Operation-based
Variant of interest	Delta-state	Pure op-based
Understandability	Complex	Medium
Efficiency	Good	Good
Complexity	Periodic anti-entropy different from Mnesia's protocol	Causal broadcast not provided by Mnesia
Advantage	Little requirement on the channel	Similarity to current Mnesia's anti-entropy strategy

Table 4.1: Comparison of state and operation based Set CRDTs.

4.2 Implementation overview

Figure 4.1 shows an overview of Hypermnesia's architecture. A client sends a request to one of the database nodes connected in the cluster, and this request is then processed by the `mnesia_ec` module (§§2.2.2 and 4.1.1), responsible for calling the underlying pure op-based Set CRDT implementation. In Hypermnesia, two op-based Set CRDTs are implemented: add-wins set (`mnesia_pawset`) and

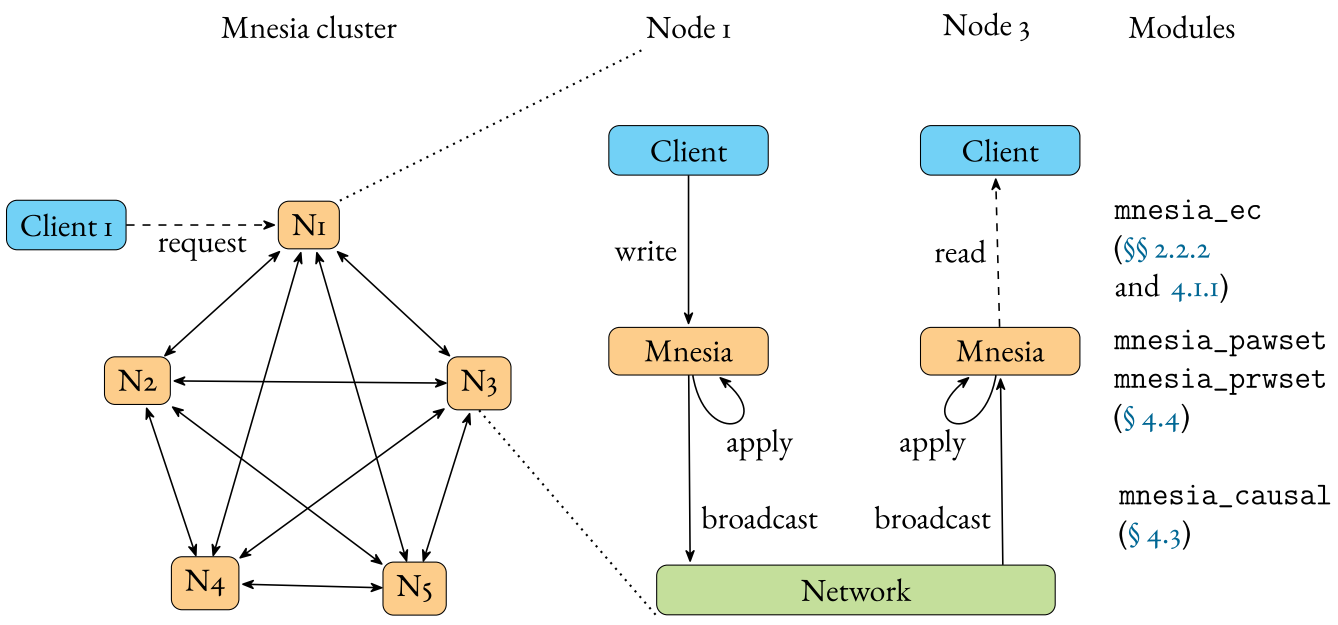


Figure 4.1: Hypermnesia architecture overview.

remove-wins set (`mnesia_prwset`) (§4.4), which are called by `mnesia_ec` based on the appropriate table type (Listing 4.2). `mnesia_ec` also calls the `mnesia_causal` module that handles the causal broadcast (§4.3). Each of these modules is explained in more detail in the following sections.

4.3 Causal Broadcast

Causal broadcast is a mature algorithm with standard implementation strategies [10, 49]. This section focuses on various modifications of the causal broadcast algorithm and pure op-based set to the Erlang ecosystem.

4.3.1 Causal broadcast server

Causal broadcast is often treated as a middleware between the network and the application. It buffers messages until they are causally ready to be delivered to the application. The `gen_server` behaviour is a suitable abstraction, as it provides an interface from which developers can write custom implementations of request handlers to obtain a generic server (similar to interface in object-oriented languages) [20].

More concretely, we define a collection of public functions for the causal broadcast server, which can be called by `mnesia_ec` while sending and receiving messages. Two most basic ones are `send_msg/1` and `rcv_msg/1`, as shown in Listing 4.3. The `send_msg/1` function returns the current timestamp as a vector clock to be attached to the message. The `rcv_msg/1` function takes a received message and returns a list of messages ready to be delivered to Mnesia for further processing, such as database writing. The `mnesia_causal` server also keeps track of a list of buffered messages. Each time a message is received, it is added to the buffer, and the buffer is searched for deliverable messages.

```
-record(state,
    {send_seq :: integer(),
     delivered :: vclock(),
     buffer :: [msg()] }).

-spec send_msg() -> vclock().
-spec rcv_msg(msg()) -> [msg()].
```

Listing 4.3: `mnesia_causal` server for causal broadcast.

4.3.2 Tagged causal stable broadcast

The algorithm discussed above is a standard causal broadcast algorithm [10, 31, 49]. It does not expose any ordering or timestamp information when delivering a message. A Set CRDT often relies on unique identifiers to ensure commutativity of causally concurrent operations. Baquero et al. [5] observe that the timestamp information from the causal broadcast layer can act as a unique identifier for the CRDT, and thus propose exposing the ordering information from the causal broadcast layer to the application, i.e. the receive function would return a list of messages along with their timestamps (Listing 4.4).

```
-spec send_msg() -> timestamp().
-spec rcv_msg(msg()) -> [{msg(), timestamp()}].
```

Listing 4.4: `mnesia_causal` server interface for Tagged Causal Stable Broadcast.

The extended Tagged Causal Stable Broadcast (TCSB) protocol works like this: when a message is ready to be sent, the function `send_msg/0` is called to obtain the timestamp which is attached to the message. And when a message is received, the `rcv_msg/1` is called to buffer the message and find all messages ready to be delivered. A message is ready when the following two conditions hold [10, 31, 49]:

1. The sender entry in the receiver's delivered map, which keeps track of how many messages are delivered for each sender, is exactly one less than the entry in the received message's timestamp;

2. The other entries in the received message's timestamp are all less than or equal to the corresponding entries in the receiver's delivered map.

The first condition ensures that all causally preceding messages from the sender have been delivered, and the second condition prevents us from delivering messages that causally depend on messages from other nodes. Note that the sender of the message can always deliver messages from itself immediately, a desirable property for availability during partitions, and causal consistency is the strongest consistency model that provides such always-available property [36].

4.4 Pure operation-based Set CRDT

We have discussed the idea of a pure op-based set (§ 2.3.2). This section builds on top of the basic ideas of op-based Set CRDT and highlights a few optimisations exploiting the causal broadcast layer (§§4.4.1 to 4.4.3). More subtle implementation challenges in interactions between Mnesia and the pure op-based set CRDT are also discussed (§4.4.4).

4.4.1 Causal Redundancy

Causal redundancy is proposed by Baquero et al. [5] for elements in the PO-Log that can be removed without affecting the output of queries. We define the redundancy relation in Listing 4.5. The first rule says that a clear or remove operation is redundant in all cases. The second rule makes causally older additions of the same elements redundant. The third rule says every addition of an element present in the PO-Log is in the set. Redundant elements can be safely removed from the PO-Log, which helps reduce the storage cost.

$$\begin{aligned}(t, o)\, R\, s &\iff o[0] = \text{clear} \vee \text{rmv} \\ (t, o)\, R\_\, (t', o') &\iff t < t' \wedge (o'[0] = \text{clear} \vee o[1] = o'[1]) \\ \text{eval}_i(\text{elems}, s) &= \{v \mid (\_, [\text{add}, v]) \in s\}\end{aligned}$$

Listing 4.5: The redundancy relation R for an AW-set. $o[0]$ is the operation, while $o[1]$ is the key. Taken from [5].

4.4.2 Causal stability

The number of timestamps associated with each element grows as elements are added to the set. Moreover, each of them is proportional to the number of nodes in the cluster. Baquero et al. [4] suggest removing the timestamps that are *causally stable* [5, 50]:

Definition 4.4.1 (Causal Stability [4, 5])**.** A timestamp τ and a corresponding message, is causally stable at node i when all messages subsequently delivered at i will have timestamp $t > \tau$. Or equivalently:

$$\text{tcstable}_i(\tau) \textbf{ if } \forall j \in I \setminus \{i\}.\exists t \in \text{deliv}_i() \cdot \text{src}(t) = j \wedge \tau < t.$$

Once a timestamp is stable, it can be removed. This is safe since there will be no more concurrent operations delivered in the future and the timestamp metadata is used to ensure commutativity for concurrent operations and is therefore no longer needed, reducing the storage cost of the PO-Log.

The actual implementation in Hypermnesia is done by asking the pure AW-set periodically scanning elements and removing causally stable timestamps.

4.4.3 Broadcast and CRDT algorithms

Algorithm 1 shows the broadcast algorithm obtained by putting these optimisations together (§§4.3.2 to 4.4.2). When broadcasting a message (Lines 6 to 10), the appropriate timestamp and sender id are attached to the message, which is then received and buffered (Lines 11 to 12). The receiver searches for messages that are causally ready to be delivered using the condition in §4.3.2 (Lines 13 to 20). The causal broadcast server also provides a function to check for the stability of a timestamp (Lines 22 to 26). This algorithm combines the standard causal broadcast algorithm [10, 31] and extra exposed APIs proposed by [4, 5], adapted to fit Mnesia's architecture (§2.2.2).

Algorithm 2 shows the corresponding algorithm for the pure AW-set, written according to the specification [4, 5]. The original specifications are written as a mathematical specification, while we take a programming perspective and present them in terms of set operations like `add`, `delete`, etc. The `remove_redundant` function is called each time a deletion or insertion happens, and checks for redundant elements using the `redundant` function (Lines 23 to 29), based on the idea of causal stability (Listing 4.5).

Apart from the add-wins set, a remove-wins set (§4.2) is also implemented, and the algorithm is mostly similar to that of an add-wins set. This can be found in Appendix D.

4.4.4 Practical concerns

This section discusses several practical issues in implementing the algorithm.

Faster access to the PO-Log As discussed in §2.2.2, Mnesia uses `ets` and `dets` as its storage system, which are implemented as hash tables and balanced binary trees. There is no direct support for log-like structures such as lists. This presents challenges as to how the PO-Log could be implemented. Typically a Mnesia table is a set indexed by the n-th element in a tuple, where n is customisable. We keep this structure but use a bag instead of a set to allow multiple elements with the same key but different (concurrent) timestamps. This representation gives us the advantage of accessing the element by key in $\mathcal{O}(1)$ time, which is useful in speeding up the `remove_redundant` function that needs a liner scan of the entire set for matching keys (Line 20 in Algorithm 2). However, this representation also loses the partial order of the PO-Log, making the stabilisation process (§4.4.2) much more difficult.

Payload conflicts Algorithm 2 deals with operations of a set in terms of addition, deletion, etc. Although the Mnesia documentation claims the data structure to be a set, it behaves more like a map, with keys and payload (recall from §2.2.3 we saw that each row in the student table has an id as the key, and other information like name as the payload). Therefore the Set CRDT falls short when there are concurrent additions of the same key but different payloads. This can be solved in general with a Map CRDT [50], which is a straightforward extension of a Set CRDT, but requires the payload to be CRDTs as well so that the conflict can be resolved recursively for the payload.

This puts constraints on the data a user can put in a Mnesia table, and is considered as breaking too

Algorithm 1: Tagged Causal Stable Broadcast (TCSB) protocol algorithm, ideas taken from [5, 6, 31].

```
1  on init:
2      sendSeq ⟵ $0$
3      buffer ⟵ $\emptyset$
4      delivered ⟵ $\{(s, 0) \mid \forall s \in \text{nodes}\}$
       /* a map from node to last the timestamp of the last delivered
          message                                                      */
5      ts_map ⟵ $\{(s, \perp \mid \forall s \in \text{nodes})\}$
6  on broadcast msg at replica $i$:
7      delivered$[i]$ ← sendSeq $+ 1$
8      sendSeq ← sendSeq $+1$
9      deps ← delivered
10     broadcast (msg, $i$, deps) to all nodes
11 on receive (msg,sender,deps) at replica $i$:
12     buffer ← buffer $\cup$ (msg, sender, deps)
13     repeat
           /* find all causally deliverable messages                   */
14         $D \leftarrow \{(\text{msg}, \text{sender}, \tau) \mid \exists(\text{msg}, \text{sender}, \tau) \in \text{buffer.deps}[\text{sender}] =$
             $\tau[\text{sender}] + 1 \wedge \forall s \in \text{dom}(\text{deps}) \setminus \{\text{sender}\}.\ \text{deps}[s] \geq \tau[s]\}$
           /* update delivered vector and timestamp map                 */
15         for (msg, sender, $\tau$) $\in D$ do
16             delivered[sender] ← delivered[sender] $+ 1$
17             ts_map[sender] ← $\tau$
18         buffer ← buffer $\setminus D$
19         deliverable ← deliverable $\cup D$
20     until $D = \emptyset$
21     deliver deliverable to application
22 on check stability of $\tau$:
23     if $\tau \leq \min(\{\text{ts\_map}[s][\texttt{src}(\tau)] \mid s \in \text{dom}(\text{ts\_map})\})$ then
24         return true
25     else
26         return false
```

Algorithm 2: Pure AW-set pseudocode, defined in terms of usual set operations. This pseudocode sacrifices efficiency for clarity. Ideas are taken from [4, 5, 6].

```
POLog ← []
function add($(e, t)$):
    remove_redundant($(e, t,$ add$)$)
    for $(e', t', o') \in$ POLog do
        if redundant($((e, t,$ add$), (e', t', o'))$) then
            return
    POLog ← append(POLog, $(e, t,$ add$)$)
function delete $(e, t)$:
    remove_redundant($(e, t,$ delete$)$)
    for $(e', t', o') \in$ POLog do
        if redundant($((e, t,$ delete$), (e', t', o'))$) then
            return
    POLog ← append(POLog, $(e, t,$ delete$)$)
function read($(k)$):
    for $(e, t, o) \in$ POLog do
        if $e$.key $= k$ then
            return $e$
    return undefined
function remove_redundant $(e, t, o)$:
    for $(e', t', o') \in$ POLog do
        if redundant($((e', t', o'), (e, t, o))$) then
            POLog ← remove(POLog, $(e', t', o')$)
function redundant ((e,t,o), (e',t',o')):
    /* check whether $(e, t, o)$ is made redundant by $(e', t', o')$ */
    if $o =$ delete then
        return true
    else if $e = e' \wedge t < t'$ then
        return true
    else
        return false
periodically
    for $(e, t, o) \in$ POLog do
        if stable $(t)$ then
            POLog ← remove(POLog, $(e, t, o)$)
            POLog ← append(POLog, $(e', \perp,$ no-op$)$)
end
```

much compatibility of the existing system and therefore I take a simpler approach by resolving them based on the Erlang term order [21], an ordering of data types in Erlang. This could easily be extended with other resolution strategies as well.

More operations on Mnesia tables Apart from the basic addition and deletion of elements, Mnesia supports a range of other operations as well, including matching based on a pattern specification, iterating over a table, finding all the keys of a table, etc. As an example, the original `select/2` function in Mnesia takes a table and a pattern specification as its argument. This is modified by appending wild cards for the timestamp and operation name (since they are stored along with each tuple in the PO-Log) to the input pattern before matching. Other functions like `all_keys/1` are implemented with similar ideas.

4.5 Fault tolerance

This section continues the discussion from §2.2.5 on Mnesia's response to network partitions and talks about how Hypermnesia addresses them. We continue to use the term transient failure and communication failure to distinguish them.

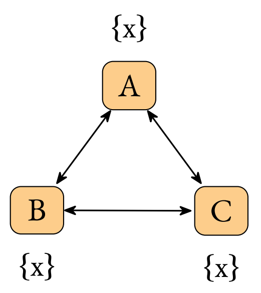
4.5.1 Communication failure

Table 4.2 shows the sequence of operations and the corresponding states of each replica, and Figure 4.2 shows the graphical representation where three Mnesia nodes are initially connected to each other and each has an element `a` in them. Then a partition occurs between node A and node B, as well as node A and node C. During this partition, an addition of element `c` occurs at A while addition of element `b` happens at B and C. When the partition recovers, replicas are now in an inconsistent state which will be reported to the developer for manual resolution.

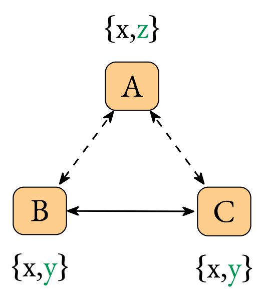
Operation	Node A	Node B	Node C
A::add(a)	{a}	{a}	{a}
B::add(b)	{a}	{a,b}	{a,b}
A::add(c)	{a,c}	{a,b}	{a,b}
`inconsistent_database`	{a,c}	{a,b}	{a,b}

Table 4.2: Operations performed on each replica. Operations that happen during the network partition sit between two horizontal lines.

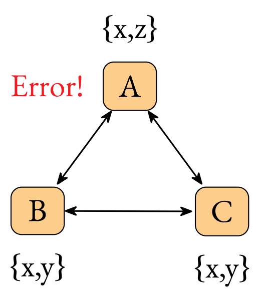
Hypermnesia resolves this issue by buffering messages during the partition, as shown in Figure 4.3c. During the communication failure, operations performed on replicas A and B are buffered until the partition heals. By the property of op-based CRDTs (Proposition 2.3.2), replicas will converge after the buffered messages are delivered. In this simple case, it is sufficient to achieve consistency by a simple



(a) Initially every replica has an `a` in its set.

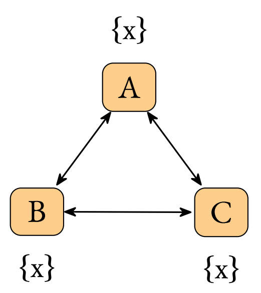


(b) A partition (dashed lines) making A temporarily unreachable. Meanwhile, y and element `z` are added to the set.

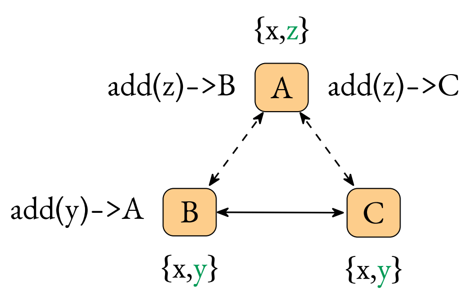


(c) When the partition heals, Mnesia would log an error message indicating `inconsistent_database` to the user for manual resolution.

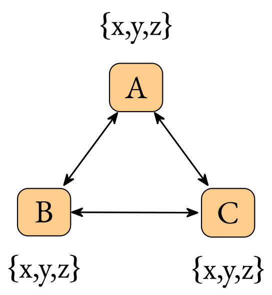
Figure 4.2: Illustration of a communication failure in Mnesia.



(a) Initially every replica has an `a` in its set.



(b) Hypermnesia buffers messages during the partition.



(c) When the communication failure recovers, buffered messages are delivered and conflicts are resolved.

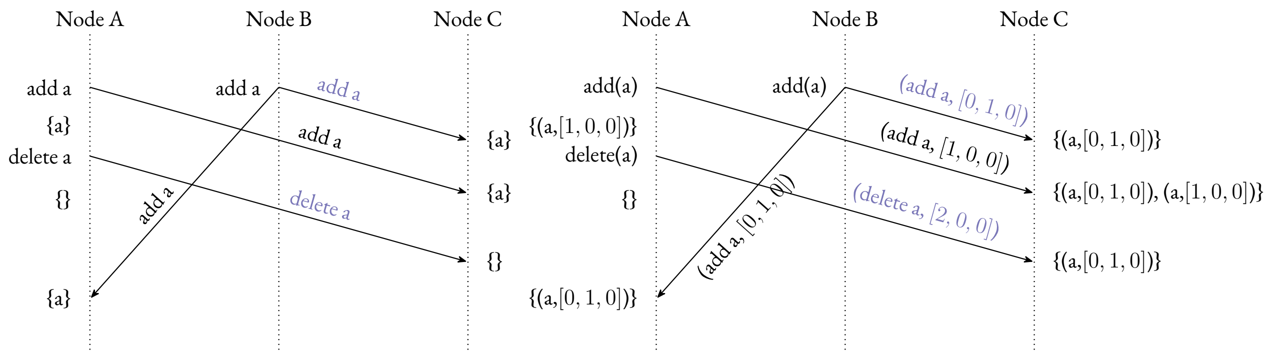
Figure 4.3: Hypermnesia buffers messages during communication failure and resolves conflicts afterwards.

set union. In a more complex case such as concurrent addition and deletion, the underlying op-based CRDT (§4.4) will handle the conflict resolution logic to ensure convergence.

The challenge here is that Mnesia, by default, considers dead nodes not to be part of the cluster and carries on operations as if they did not exist. This might be a desirable behaviour in transactions, and developers can use the `majority` option to protect mission-critical data. But in an eventually consistent system, this is less ideal since we want our system to repair itself automatically. Therefore Hypermnesia considers dead nodes as part of the cluster and buffers operations for them.

4.5.2 Transient failure

When a transient network failure happens, communication between nodes temporarily stops but is not long enough for the failure detector to act. Transactions will completely stall during this period. Although dirty operations can carry on, replicas might end up in different states due to out-of-order message delivery. For example, in Figure 4.4a, there might be a network failure between node B to A, resulting in B's `add a` being delayed. If messages are delivered as they arrive, then node A and node C will end up in an inconsistent state. This is because addition and deletion in a set do not commute, and the two purple operations (add and delete) are concurrent, and they are applied in a different order on node A and node C, resulting in different final states.



(a) Mnesia replica states can diverge depending on the ordering of delivery of operations. The purple operations are causally concurrent.

(b) Hypermnesia attaches vector timestamps to each message and stores them along with actual elements in the set for conflict resolution.

Figure 4.4: Mnesia and Hypermnesia's response to transient network failure.

In order to achieve convergence, and hence eventual consistency, concurrent operations need to commute. The exact semantics of whether addition or deletion wins depends on the actual application, and to achieve convergence, it is sufficient to define consistent semantics across replicas. Add-wins semantics is presented here but the remove-wins semantics is similar (Appendix D). To achieve add-wins semantics with a pure op-based Set CRDT, we require deletions to only remove elements that causally precede it, as captured by Line 20 in Algorithm 2. In Figure 4.4b, the deletion with timestamp $[2, 0, 0]$ removes the element (a,$[1, 0, 0]$) which is causally lower, but not (a,$[0, 1, 0]$) which is causally concurrent.

4.6 Summary

This chapter covers the design of Hypermnesia (§4.1) in terms of its APIs to minimise the amount of code refactoring needed (§4.1.1). We also examined the suitability of different CRDTs and decided to use pure op-based CRDTs primarily because of its similarity to Mnesia's communication pattern (§4.1.2). A prototype implementation is presented (§4.2), including a Tagged Causal Stable Broadcast layer by extending the basic causal broadcast with timestamp and causal stability information (§4.3.2). Practical optimisations such as the use of `bag` data structures are also considered (§4.4). We also briefly talked about two patterns of network faults and how the AW-set and buffering of operations can help us (§4.5).

Chapter 5

Evaluation

This chapter evaluates Hypermnesia against the proposed research questions (§1.1). We start by introducing new tests to ensure the correctness of the system (§5.2), and then outline the benchmarking approach (§5.3), before performing measurements on a cluster of distributed physical machines in terms of time (§5.3) and space (§5.4). This is followed by an examination of the fault tolerance properties of Hypermnesia (§5.5). We conclude the evaluation with a discussion on the API usability of Hypermnesia (§5.6).

5.1 Setup

The experiments are performed on: (a) a single physical machine with multiple Erlang VMs connected via the loopback interface, mainly used for correctness testing and some simple experiments; (b) multiple physical machines inside a data centre connected via ethernet switches, used for more extensive benchmarking.

The single physical machine has an AMD 12-Core Processor and 16GiB DDR4 memory. It runs the Ubuntu 22.04.2 LTS Operating System. The cluster consists of 10 machines, each has an Intel(R) Xeon(R) CPU with 6 cores and 64GiB memory. They all run the Ubuntu 20.04.4 LTS Operating System. More details on the test-bed setup can be found in Appendix E.

5.2 Correctness

The first question of whether eventual consistency is possible in Mnesia (Item RQ1 in §1.1) is concerned with the design and correctness of the implementation, therefore correctness testing is performed. The Mnesia source code has a regression test suite with about 5000 tests, covering various aspects of the correctness such as transaction, dirty operations, storage engine, etc. They are run against the additional code introduced by Hypermnesia. These mostly serve as tests for backwards compatibility, since there is currently no test that simulates network partitions, nor tests targeting the new eventually consistent API.

To cover the new EC API, the suite is extended by adding the following tests:

1. Unit tests on the behaviour of the AW-set and causal delivery, such as testing, among others, whether the addition of an element is reflected in a subsequent read.

2. Unit Tests on database features such as index reads.
3. Integration tests for simulating network partitions, including multiple cases of concurrent addition and deletion.

There are about 20 new tests written to cover the new API. Both unit tests and integration tests generally follow the pattern of the Erlang EUnit and Common Test framework [15, 17], respecting the existing test suite structure. The network partition is simulated with a custom Erlang distribution protocol [16], developed by RabbitMQ for their integration testing [46].

Hypermnesia passes all of the tests mentioned above, and the output log of the tests is included in Appendix F.

5.3 Benchmarking

5.3.1 Benchmark overview

The second question (Item RQ2 in §1.1) is concerned with the efficiency of the eventual consistency API in Hypermnesia. To answer this question, benchmarking is performed on the Hypermnesia codebase. The original Mnesia repository has a built-in benchmarking suite for transactions. Briefly, this benchmark is a simulation of workloads where Mnesia tables are used to store session data of, for example, users logging into a website. There are four stages in this benchmark:

population initialises tables with randomly generated data.

warmup performs operations to bring data from memory to the cache.

actual benchmarking performs the actual benchmarking.

cooldown allows the system to clean up resources and ensures isolation between consecutive runs.

Typical operations in this benchmark include reading data of a particular session, creating a new session for a subscriber of the service, deleting the details of a particular session, etc.

To compare different access contexts, the benchmark is extended from transactions to dirty and EC operations. The extension is a substitution of transaction access context for dirty/EC access contexts. One thing to note is that, unlike transactions, ACID properties are not guaranteed for dirty and EC operations. Therefore it is common to have failures while reading data as messages might still be in transit. In such cases, we retry several times before considering the operation as failed and aborting it.

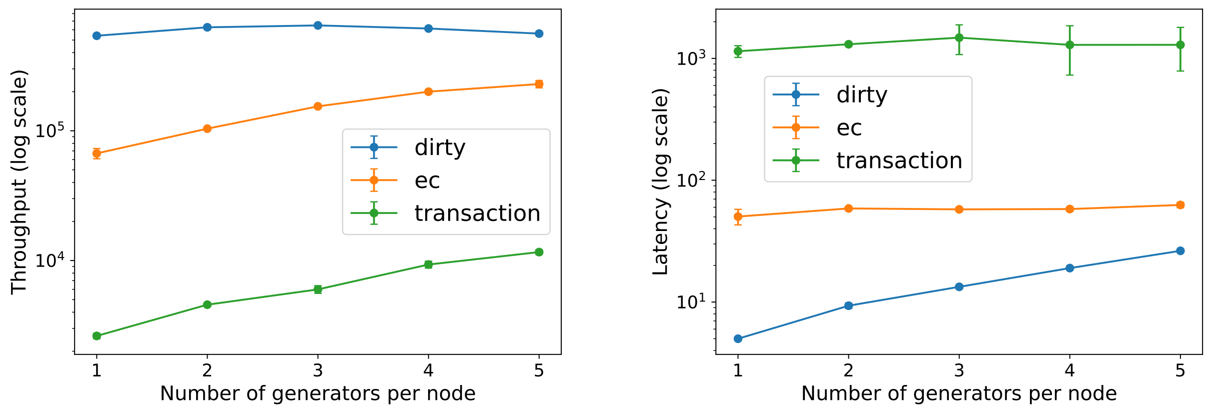
The general benchmarking strategy in the following sections is to measure the throughput and latency across three access contexts: dirty, transaction and EC, and make comparisons between them. Several metrics are considered in this benchmarking process: number of generators, number of nodes, table size, and workload types (§§5.3.2 to 5.3.5). We expect dirty operations to be the most performant in terms of throughput and latency since it has the lowest operational complexity, while transactions to be the slowest due to its synchronisation overhead. EC operations should stay between those two, and ideally as close to dirty operations as possible.

5.3.2 Number of generators

Figure 5.1 shows the comparison of throughput and latency against the number of generators per node for different access contexts. Generators are clients sending requests to the Mnesia cluster. These requests are sent to the nearest node (with respect to the generator) for processing.

Generally speaking, throughput (Figure 5.1a) increases as the number of generators per node increases. For dirty operations, this saturates at around four generators per node, likely caused by I/O bandwidth limitation. Transactions and EC throughput continue to increase as more generators are added. This scalability property likely comes from the fact that Mnesia is, by default, a leaderless architecture (§2.2.2) where every node can process requests. Such design choices help Mnesia obtain more parallelism as we add more nodes into the cluster. Latency (Figure 5.1b), on the other hand, generally stays the same. Dirty operations do show a higher increasing rate than the other two, possibly because the overhead of more generators (and hence more messages to process) are more significant in a previously low-latency environment (the latency of dirty operations is on the order of 10 µs).

The increasing throughput and stable latency of EC operations demonstrate its scalability against the number of clients. Moreover, as the number of generators per node increases, the throughput of EC operations approaches about half the throughput of dirty operations, whereas initially it was only about one-tenth. This is a desirable property in a replicated system as it is scalable with respect to the number of clients (requests) it can handle.

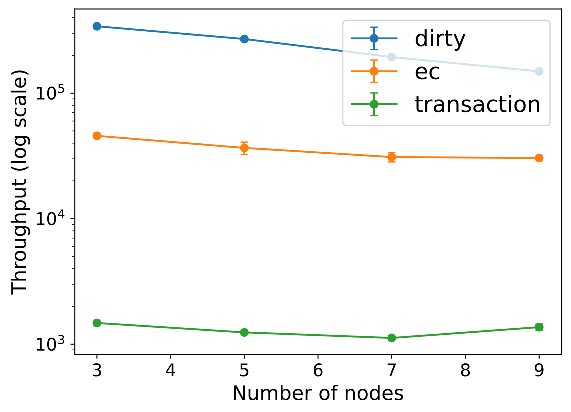


(a) Throughput **(b)** Latency

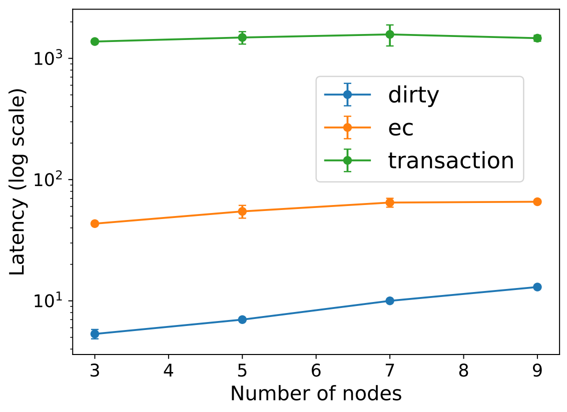
Figure 5.1: Throughput and latency against the number of generators. More generators generally bring higher throughput and latency.

5.3.3 Cluster size

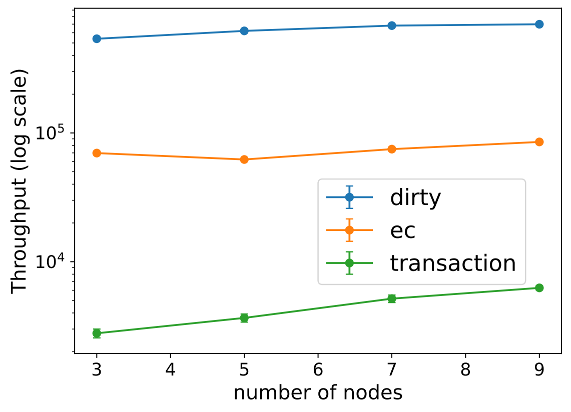
This section evaluates the throughput and latency against the number of nodes in the Mnesia cluster. Based on the original design of Mnesia (§2.2), the number of nodes used in a cluster generally does not exceed ten [28]. Figure 5.2a and Figure 5.2b show the throughput and latency change with respect to the number of nodes, with a fixed number (2) of generators. We observe that the throughput decreases and the latency increases as the number of nodes increases, suggesting that adding more nodes inevitably introduces overhead into the system, e.g. more messages to send, and more data to process. Keeping the number of generators the same means we keep the total amount of client requests the same but add duplicate work by adding more replicas, therefore the system experiences more overhead.



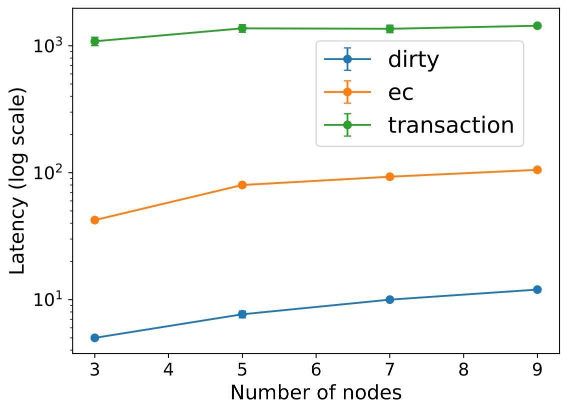
(a) Throughput with a fixed number of generators.



(b) Latency with a fixed number of generators.



(c) Throughput with varying number of generators.



(d) Latency with varying number of generators.

Figure 5.2: Throughput and latency against the number of nodes with fixed or varying number of generators. Adding replicas increases the overhead, but can be reduced by adding generators and increasing parallelism.

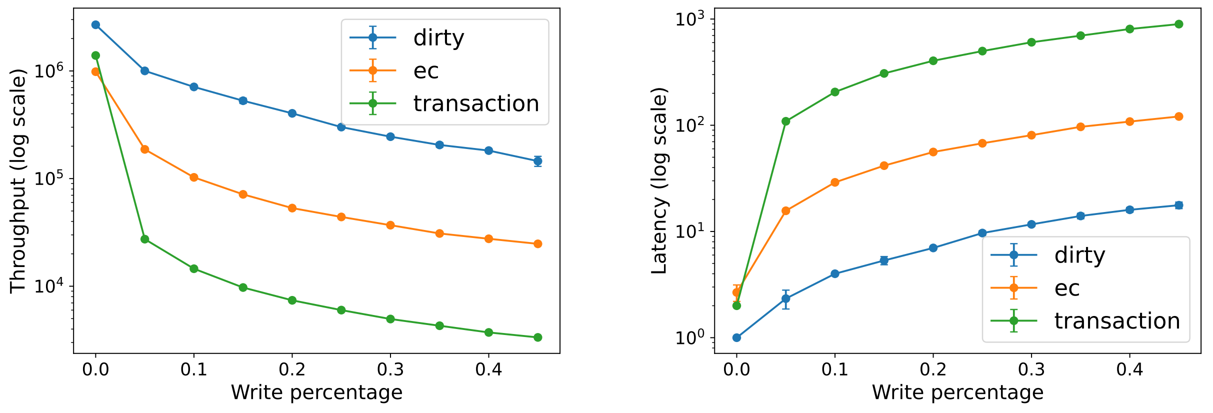
However, we could also increase the number of generators as we have more replicas, which is generally what happens in a real deployment as the number of nodes in a cluster of replicated machines increases. Figure 5.2c and Figure 5.2d show the throughput and latency as the number of generators increases linearly with the number of nodes. For all three operations, throughput increase is observed. Dirty operations' throughput saturates at about 7 nodes, when the message queue backlog starts to become the bottleneck. In terms of latency, there is an increase in all three. EC operations demonstrate around 13x higher throughput than transactions when there are nine nodes.

A higher replication factor often implies extra work for the system, thus resulting in lower throughput and higher latency. However, we could offset this with more clients and hence more parallelism. The overall effect is an increasing throughput as the number of nodes increases, albeit with an inevitable sacrifice in latency. For EC operations, this is approximately 60 µs. This property is present in all three access contexts.

5.3.4 Workload types

Figure 5.3 compares the throughput and latency of three access contexts against different workloads. Dirty operations are about 50x better than transactions, while EC operations lie between them (again), with about 10x higher throughput than transactions.

Note that if the read percentage is 100%, i.e. a read-only workload, the transaction gets close to or even surpasses the performance of EC operations. This is due to an optimisation done in the benchmark where it uses a synchronous dirty operation for reading data rather than a full two-phase commit, which removes the cost of locking, and multi-round communication time. It is even faster than EC operations as it does not need to go through the causal broadcast layer and the processing logic of an AW-set.



(a) Throughput **(b)** Latency

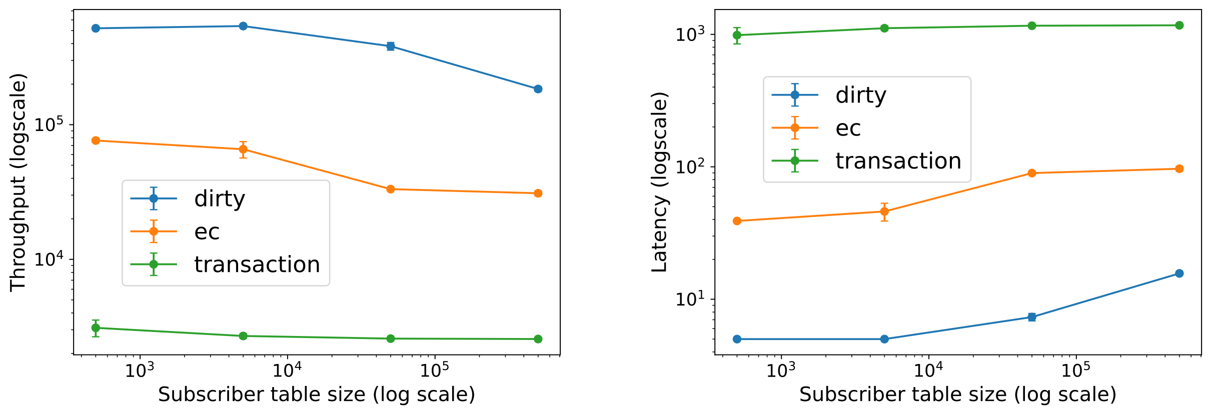
Figure 5.3: Throughput and latency against different workload types. Read-heavy workloads are faster than write-heavy ones in all three cases.

Write-intensive workloads are generally slower than read-intensive ones, since writes need to involve all replicas in the cluster, while reads can be done locally (if the data is present on the local node, which is the case in this benchmark). This trend is, again, true for all three contexts. As long as the workload is not read-only, EC operations performs better than transactions.

5.3.5 Table size

Figure 5.4 shows the change in throughput and latency as we vary the size of the subscriber table. There are five tables in total in this benchmark, and the subscriber table stores the data for users subscribing to a service, which is the most frequently changed and the largest one among all five tables. Therefore I choose to vary the subscriber table size during the initial table population.

A larger table generally makes it slower to read/write data, which affects dirty operations but has less impact on transactions and EC operations. Similar to the behaviour in §5.3.2, the low-latency dirty operations are more sensitive to even small changes in the latency of each operation, and exhibits a larger increase in latency.



(a) Throughput **(b)** Latency

Figure 5.4: Throughput and latency against table size. Dirty operations are affected the most.

Table size's impact on the performance of Hypermnesia mainly comes from the extra time in accessing elements of a larger table. For EC operations, the extra overhead of periodic cleaning of timestamps could also play a role (§4.4.2). However, this overhead is still acceptable as EC operations still have about 25x higher throughput than transactions.

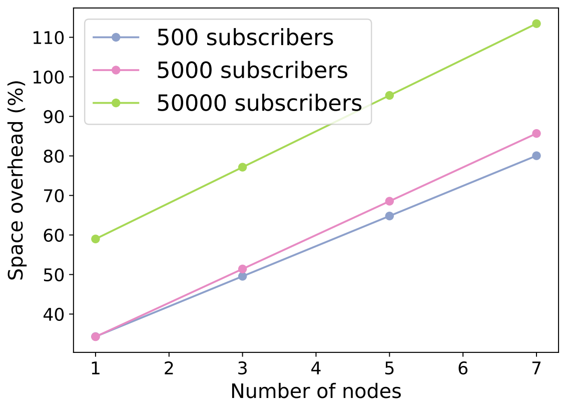
5.4 Space overhead

This section evaluates the space overhead of Hypermnesia. Generally speaking, CRDTs rely heavily on metadata to keep track of the history of operations and hence has a large overhead in its space usage [7]. In the case of implementing a pure AW-set, the primary metadata associated with each element is the vector clock timestamp, which grows linearly with the number of nodes in the cluster, although garbage collection is implemented to reduce its impact (§4.4).

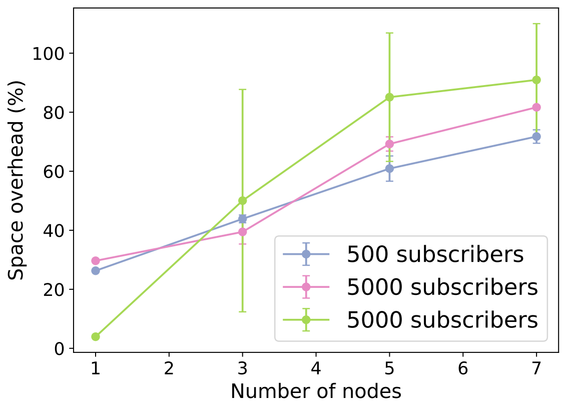
Figure 5.5a shows the pure op-based set's overhead compared to the default set in Mnesia. The overhead grows linearly with respect to the number of nodes as expected. Note that this is a static process, i.e. each time the populator will put the same number of elements into the set, therefore the lines are perfectly straight with no variance. Figure 5.5b shows the overhead after running the benchmark. This is a dynamic process during which the causal stability optimisation is applied, but not in the previous figure (we discuss the reason below). Observe that the causal stability optimisation is able to reduce the overhead by up to 30%.

Despite space optimisations, the space overhead is still relatively large, especially when the number of nodes exceeds five. There are several reasons for this:

- The underlying implementation of the PO-Log is a set-like structure rather than a partially ordered log (§4.4.4). This complicates the implementation of causal stability optimisation as it is now harder to find all elements with a timestamp smaller than the stable one.



(a) Space overhead after populating the table.



(b) Space overhead after running the benchmark.

Figure 5.5: Space overhead. The number of subscribers represents the size of the table.

- The causal stability optimisation relies on nodes continuously receiving updates from other nodes to determine timestamp stability. This is not always possible since it is common for a node to only receive messages from others (maybe no client sends requests to it). This is why the optimisation is not applied in the population phase. Bauwens and Gonzalez Boix [7] made a similar observation and proposed a solution based on an eager collection of metadata.

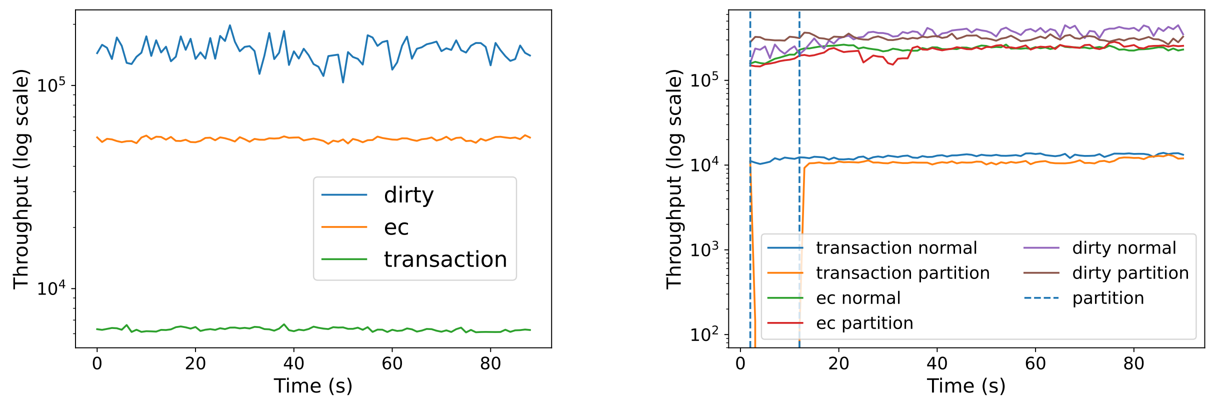
Memory management has been an important issue in CRDT research but has yet to attract much attention [7]. The current optimisation using causal stability is less effective than one would hope for. In a typical setup of three nodes, the extra space needed is around 30–40%. For this reason, Hypermnesia is limited to relatively small clusters, but this is acceptable since Mnesia is designed for small clusters in the first place [28]. We discuss more on how to optimise the space overhead of the Set CRDT in §6.2.

5.5 Fault tolerance

§4.5 discusses how Mnesia and Hypermnesia respond to network partitions. In this section, we evaluate Hypermnesia against these two situations to answer the research question Item RQ3 in §1.1.

Communication failure Mnesia does not handle communication failure by default and asks application developers to resolve conflicts. Hypermnesia buffers operations during the partition until it recovers (Figure 4.3c). It then sends the buffered message and resolves conflicts automatically.

Transient failure As discussed in §2.2.5, during a transient failure, transactions stall until the partition heals. We measure this effect by examining the throughput change in a simulated network partition, as shown in Figure 5.6. Figure 5.6a shows how the throughput changes through time, which is relatively stable when there is no partition. On the other hand, Figure 5.6b shows the throughput when there is a (simulated) network partition, lasting around 10 seconds. Dirty and EC operations remain about the same, but transaction throughput drops down to zero during the partition period. This is expected since the two-phase commit protocol requires a reply from *all* participating nodes.



(a) No partition.

(b) A partition of 10 seconds, dashed lines indicate the start and end of the partition.

Figure 5.6: Throughput changes against time. Network partition affects the transaction throughput but not dirty and EC throughputs.

In summary, Hypermnesia adds fault tolerance to Mnesia in the presence of network partitions: 1. automatic conflict resolution after a communication failure; 2. nodes remain available during transient failure, with guaranteed convergence when the partition heals.

5.6 APIs and refactoring

This section attempts to answer the research question Item RQ4 concerning refactoring. First, a summary of the code changes is given for Hypermnesia, followed by experimental modifications of real-world applications to demonstrate Hypermnesia's usability.

Listing 5.1 shows the code changes needed from transactions/dirty operations (first two columns) to EC operations (last column). Note that refactoring is often just one line of change. Another change is the `type` option while creating a Mnesia table, as shown in Listing 5.2. And that is all the change needed to use the new API.

The second step is to apply the refactoring described above in actual production codebases such as RabbitMQ [56] and ejabberd [44], and run the regression test suites against such changes. To be more cautious while refactoring, most changes are from asynchronous dirty operations to asynchronous EC operations. Hypermnesia successfully passes the test suites of both applications with little effort in changing the source code.

It is worth noting that this is not a formal usability study of the new API. A more rigorous study requires much more extensive testing and understanding of the project codebases, which requires lots of engineering effort and is beyond the scope of this project. Nevertheless, we believe this evaluation is a first step towards making Hypermnesia more usable and production-ready.

```
mnesia:activity(transaction,
fun () ->
  mnesia:write(tab, tup),
  mnesia:read(tab, key)
end).
```

(a) transactions with `activity/2`.

```
mnesia:activity(async_dirty,
fun () ->
  mnesia:write(tab,tup),
  mnesia:read(tab,key)
end).
```

(b) dirty with `activity/2`.

```
mnesia:activity(async_ec,
fun () ->
  mnesia:write(tab,tup),
  mnesia:read(tab,key)
end).
```

(c) EC operations with `activity/2`

```
mnesia:transaction(
fun () ->
  mnesia:write(tab,tup),
  mnesia:read(tab,key)
end).
```

(d) transactions with `transaction/1`

```
mnesia:async_dirty(
fun () ->
  mnesia:write(tab,tup),
  mnesia:read(tab,key)
end).
```

(e) dirty with `async_dirty/1`

```
mnesia:async_ec(
fun () ->
  mnesia:write(tab,tup),
  mnesia:read(tab,key)
end).
```

(f) EC operations with `async_ec/1`

Listing 5.1: Changing from transaction or dirty operations to EC operations.

```
ejabberd_mnesia:create(?MODULE,
↪  oauth_client, [{disc_copies,
↪  [node()]},
 {attributes, record_info(fields,
↪  oauth_client)}, {type, set}]).
```

(a) Original code creating a Mnesia table, using a set data structure.

```
ejabberd_mnesia:create(?MODULE,
↪  oauth_client, [{disc_copies,
↪  [node()]},
 {attributes, record_info(fields,
↪  oauth_client)}, {type, pawset}]).
```

(b) New code using the pure AW-set (pawset).

Listing 5.2: Adding a type declaration when creating a Mnesia table. Code excerpt modified from ejabberd [44].

5.7 Summary

In this chapter, we evaluated Hypermnesia against the research questions of this project (§1.1). The results show that Hypermnesia can produce correct results in spite of network delays (§5.2), and partitions (§5.5), thus demonstrating the possibility of eventual consistency in Mnesia and its role in automatic conflict resolution (Items RQ1 and RQ3). Furthermore, benchmarking results show that EC operations can achieve approximately 10–20x better throughput and lower latency than transactions, and its performance can get close to dirty operations (recall that dirty operations are much faster than transactions in Mnesia from §2.2.4) when increasing the scale of the experiment (§5.3). This makes EC operations competitive for real-world applications (Item RQ2). Finally, §5.6 evaluates the usability of the new API (Item RQ4) and shows that Hypermnesia's API enables minimum code refactoring for adoption in real-world projects.

Chapter 6

Conclusions

In this project, we looked at Hypermnesia, an extension to the Mnesia DBMS incorporating eventual consistency. We conclude this report by summarising the accomplishments of Hypermnesia, highlighting key contributions (§6.1) and pointing out ways Hypermnesia can be improved in the future (§6.2).

6.1 Accomplishments

Qualitatively speaking, Hypermnesia's API is designed to minimise the amount of code refactoring (§4.1) and fits well into the existing Mnesia access contexts, making it easier to be adopted in existing open source codebases (§5.6) such as RabbitMQ and ejabberd [44, 56]. Moreover, the new eventual consistency API passes the extended Mnesia regression test suite (consisting of ≈ 5000 unit tests), including (≈ 20) additional tests that cover network partition, AW-set and RW-set behaviours and causal broadcast (§5.2). It also allows the database to continue operating while there is a partition and automatically resolves potential conflicts after the partition recovers (§5.5).

Quantitatively speaking, Hypermnesia accomplishes the above functionalities while providing about 10–20 times higher throughput and lower latency than Mnesia's transactions (§5.3). Thanks to Mnesia's performant dirty operations, Hypermnesia can be designed to guarantee eventual consistency without sacrificing much performance, taking advantage of the performant architecture of dirty operations.

6.2 Future work

Finally, I list some possible extensions to Hypermnesia:

- At the moment EC operations do not interact well with Mnesia's transactions and dirty operations, i.e. they cannot be used on the same table. As mentioned in §3.1.3, it is possible to add support for transactions that execute and update queries on a consistent snapshot and then resolve conflicts between different snapshots with CRDTs, and there are many protocols developed for this purpose [43, 51]. Now that Mnesia has built-in support for CRDTs and eventual consistency, it could be further enhanced to support lightweight but highly available transactions.

- Hypermnesia currently works for in-memory tables only. Although in-memory caching tends to be the way Mnesia is used [38], Hypermnesia could be extended to support disk tables in the future, or even custom backends. Different data structures might open the door for better space optimisation which is currently less feasible with `ets` and `dets`.
- Hypermnesia chooses to use pure op-based CRDTs for its simplicity and similarity to Mnesia's architecture. δ-CRDTs is another popular choice among eventually consistent databases [33], which might be worthwhile experimenting with.

Bibliography

[1] P. S. Almeida, A. Shoker, and C. Baquero. Delta State Replicated Data Types. *Journal of Parallel and Distributed Computing*, 111:162–173, Jan. 2018. ISSN 07437315. doi:10.1016/j.jpdc.2017.08.003.

[2] J. L. Andersen. Mnesia and CAP, Sept. 2014.

[3] V. Balegas. Introducing Rich-CRDTs · ElectricSQL, May 2022.

[4] C. Baquero, P. S. Almeida, and A. Shoker. Making operation-based CRDTs operation-based. In *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*, PaPEC '14, pages 1–2, New York, NY, USA, Apr. 2014. Association for Computing Machinery. ISBN 978-1-4503-2716-9. doi:10.1145/2596631.2596632.

[5] C. Baquero, P. S. Almeida, and A. Shoker. Pure Operation-Based Replicated Data Types, Oct. 2017.

[6] J. Bauwens and E. G. Boix. Improving the Reactivity of Pure Operation-Based CRDTs. In *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*, pages 1–6, Online United Kingdom, Apr. 2021. ACM. ISBN 978-1-4503-8338-7. doi:10.1145/3447865.3457968.

[7] J. Bauwens and E. Gonzalez Boix. Memory efficient CRDTs in dynamic environments. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages*, VMIL 2019, pages 48–57, New York, NY, USA, Oct. 2019. Association for Computing Machinery. ISBN 978-1-4503-6987-9. doi:10.1145/3358504.3361231.

[8] D. Bermbach and J. Kuhlenkamp. Consistency in Distributed Storage Systems. In V. Gramoli and R. Guerraoui, editors, *Networked Systems*, Lecture Notes in Computer Science, pages 175–189, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-40148-0. doi:10.1007/978-3-642-40148-0_13.

[9] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Pub. Co, Reading, Mass, 1987. ISBN 978-0-201-10715-9.

[10] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, Aug. 1991. ISSN 0734-2071. doi:10.1145/128738.128742.

[11] F. Cesarini. Companies Who Use Erlang, Sept. 2019.

[12] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2 edition, 2002. doi:10.1017/CBO9780511809088.

[13] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 1–12, New York, NY, USA, Dec. 1987. Association for Computing Machinery. ISBN 978-0-89791-239-6. doi:10.1145/41840.41841.

[14] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, June 1989. ISSN 0163-5808. doi:10.1145/66926.66963.

[15] Ericsson AB. Erlang – Common Test Basics. https://www.erlang.org/doc/apps/common_test/basics_chapter.html, Mar. 2023.

[16] Ericsson AB. Erlang – How to Implement an Alternative Carrier for the Erlang Distribution. https://www.erlang.org/doc/apps/erts/alt_dist.html, Mar. 2023.

[17] Ericsson AB. Erlang – EUnit - a Lightweight Unit Testing Framework for Erlang. https://www.erlang.org/doc/apps/eunit/chapter.html, Mar. 2023.

[18] Ericsson AB. Erlang – Mnesia User's Guide. https://www.erlang.org/doc/apps/mnesia/users_guide.html, Mar. 2023.

[19] Ericsson AB. Erlang – mnesia. https://www.erlang.org/doc/man/mnesia.html, Mar. 2023.

[20] Ericsson AB. Erlang – OTP Design Principles. https://www.erlang.org/doc/design_principles/users_guide.html, Mar. 2023.

[21] Ericsson AB. Erlang – Erlang Reference Manual. https://www.erlang.org/doc/reference_manual/users_guide.html, Mar. 2023.

[22] Ericsson AB. Erlang – STDLIB User's Guide. https://www.erlang.org/doc/apps/stdlib/users_guide.html, Mar. 2023.

[23] Erlang Solutions. MongooseIM platform. Erlang Solutions, Apr. 2023.

[24] B. Farinier, T. Gazagnaire, and A. Madhavapeddy. Mergeable persistent data structures. In *Vingt-Sixièmes Journées Francophones Des Langages Applicatifs (JFLA 2015)*, Jan. 2015.

[25] S. Gilbert and N. Lynch. Perspectives on the CAP Theorem. *Computer*, 45(2):30–36, Feb. 2012. ISSN 1558-0814. doi:10.1109/MC.2011.389.

[26] F. Guidec, Y. Mahéo, and C. Noûs. Supporting conflict-free replicated data types in opportunistic networks. *Peer-to-Peer Networking and Applications*, 16(1):395–419, Jan. 2023. ISSN 1936-6450. doi:10.1007/s12083-022-01404-6.

[27] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, Dec. 1983. ISSN 0360-0300. doi:10.1145/289.291.

[28] F. Hébert. *Learn You Some Erlang for Great Good!: A Beginner's Guide*. No Starch Press, San Francisco, 2013. ISBN 978-1-59327-504-4.

[29] R. Hipp. Most Widely Deployed SQL Database Engine. https://www.sqlite.org/mostdeployed.html, 2019.

[30] M. Kleppmann. *Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017. ISBN 978-1-4493-7332-0.

[31] M. Kleppmann and T. Harris. Distributed Systems, Cambridge CST Part IB lecture notes, 2022.

[32] M. Kleppmann, A. Wiggins, P. van Hardenberg, and M. McGranaghan. Local-first software: You own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, pages 154–178, New York, NY, USA, Oct. 2019. Association for Computing Machinery. ISBN 978-1-4503-6995-4. doi:10.1145/3359591.3359737.

[33] R. Klophaus. Riak Core: Building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming*, CUFP '10, page 1, New York, NY, USA, Oct. 2010. Association for Computing Machinery. ISBN 978-1-4503-0516-7. doi:10.1145/1900160.1900176.

[34] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978. ISSN 0001-0782. doi:10.1145/359545.359563.

[35] E. Levy. RabbitMQ vs Kafka: Use Cases, Performance & Architecture, Feb. 2022.

[36] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, Availability, and Convergence. Technical report, May 2012.

[37] R. C. Martin. *Design Principles and Design Patterns*. 2000.

[38] H. Mattsson, H. Nilsson, and C. Wikström. Mnesia — A Distributed Robust DBMS for Telecommunications Applications. In G. Gupta, editor, *Practical Aspects of Declarative Languages*, Lecture Notes in Computer Science, pages 152–163, Berlin, Heidelberg, 1998. Springer. ISBN 978-3-540-49201-6. doi:10.1007/3-540-49201-1_11.

[39] HAKAN. MATTSSON. Mnesia internals, Oct. 1999.

[40] HAKAN. MATTSSON. Mnesia implementation documentation, Nov. 2009.

[41] P. Mineiro. Dukes of Erl: Network partition ... oops, Mar. 2008.

[42] N. Preguiça. Conflict-free Replicated Data Types: An Overview, June 2018.

[43] N. Preguica, M. Zawirski, A. Bieniusa, S. Duarte, V. Balegas, C. Baquero, and M. Shapiro. SwiftCloud: Fault-tolerant geo-replication integrated all the way to the client machine. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, pages 30–33, Nara, Japan, Oct. 2014. IEEE. ISBN 978-1-4799-7361-3. doi:10.1109/SRDSW.2014.33.

[44] Processone. Processone/ejabberd. ProcessOne, Apr. 2023.

[45] M. Raasveldt and H. Mühleisen. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1981–1984, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-5643-5. doi:10.1145/3299869.3320212.

[46] RabbitMQ. Inet_tcp_proxy. RabbitMQ, Aug. 2022.

[47] Redis. Diving into Conflict-Free Replicated Data Types (CRDTs). https://redis.com/blog/diving-into-crdts/, Mar. 2022.

[48] J. Saltzer and M. F. Kaashoek. *Principles of Computer System Design: An Introduction*. Morgan Kaufmann, May 2009. ISBN 978-0-08-095942-9.

[49] F. B. Schmuck. The Use of Efficient Broadcast Protocols in Asynchronous Distributed Systems. Technical report, Cornell University, Aug. 1988.

[50] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free Replicated Data Types. 2011.

[51] M. Shapiro, A. Bieniusa, N. Preguiça, V. Balegas, and C. Meiklejohn. Just-Right Consistency: Reconciling availability and safety, Jan. 2018.

[52] D. Shukla. Azure Cosmos DB: Pushing the frontier of globally distributed databases, Sept. 2018.

[53] I. T. Tomter and W. Yu. Augmenting SQLite for Local-First Software. In L. Bellatreche, M. Dumas, P. Karras, R. Matulevičius, A. Awad, M. Weidlich, M. Ivanović, and O. Hartig, editors, *New Trends in Database and Information Systems*, volume 1450, pages 247–257. Springer International Publishing, Cham, 2021. ISBN 978-3-030-85081-4 978-3-030-85082-1. doi:10.1007/978-3-030-85082-1_22.

[54] A. van der Linde, J. Leitão, and N. Preguiça. Δ-CRDTs: Making δ-CRDTs delta-based. In *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data*, pages 1–4, London United Kingdom, Apr. 2016. ACM. ISBN 978-1-4503-4296-4. doi:10.1145/2911151.2911163.

[55] P. Viotti and M. Vukolić. Consistency in Non-Transactional Distributed Storage Systems. *ACM Computing Surveys*, 49(1):19:1–19:34, June 2016. ISSN 0360-0300. doi:10.1145/2926965.

[56] VMware. RabbitMQ Server. RabbitMQ, Apr. 2023.

[57] W. Vogels. Eventually Consistent: Building reliable distributed systems at a worldwide scale demands trade-offs?between consistency and availability. *Queue*, 6(6):14–19, Oct. 2008. ISSN 1542-7730. doi:10.1145/1466443.1466448.

[58] M. Vorontsov. Mikhail Vorontsov - ForgETS: A globally distributed database - Code Beam STO, June 2018.

[59] S. Weiss, P. Urso, and P. Molli. Logoot-Undo: Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1162–1174, Aug. 2010. ISSN 1558-2183. doi:10.1109/TPDS.2009.173.

[60] U. Wiger. Writing an Unsplit method, May 2023.

[61] U. Wiger. [erlang-questions] unsplit - resolving mnesia inconsistencies, Thu Feb 4 22:39:02 CET 2010.

[62] Wikipedia contributors. LYME (software bundle). *Wikipedia*, Dec. 2020.

[63] Wikipedia contributors. Eventual consistency. *Wikipedia*, Apr. 2023.

[64] W. Yu and C.-L. Ignat. Conflict-Free Replicated Relations for Multi-Synchronous Database Management at Edge. In *2020 IEEE International Conference on Smart Data Services (SMDS)*, pages 113–121, Oct. 2020. doi:10.1109/SMDS49396.2020.00021.

[65] W. Yu and S. Rostad. A low-cost set CRDT based on causal lengths. In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, PaPoC '20, pages 1–6, New York, NY, USA, Apr. 2020. Association for Computing Machinery. ISBN 978-1-4503-7524-5. doi:10.1145/3380787.3393678.

Appendix A

Benchmark configuration file

Appendix A shows an example benchmark configuration file for evaluating Hypermnesia. Some interesting fields include `generator_duration`, which is the amount of benchmarking time, `n_replicas`, which is the number of replicas in the cluster, `n_generators_per_node`, which can be thought of as the number of clients on each generator node.

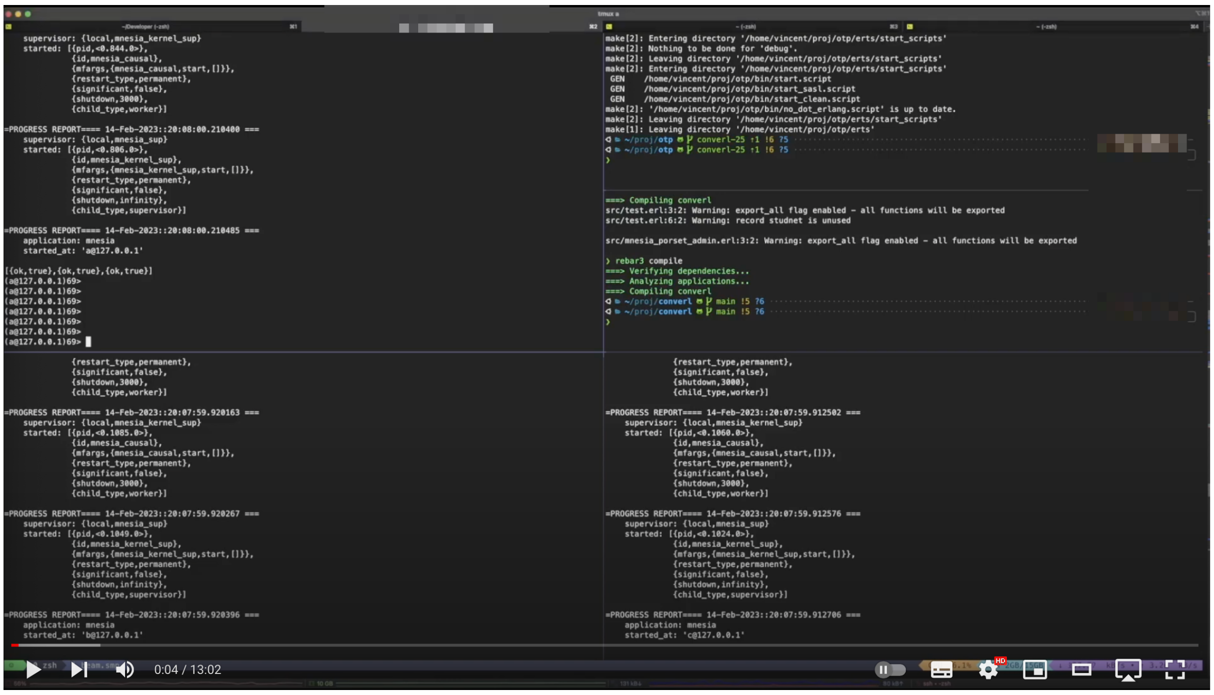
```
{start_module, slave }.
{partition_time, 0 }.
{cookie,bench_cookie}.
{activity, async_ec }.
{generator_profile, random }.
{rw_ratio, 0.5 }.
{statistics_detail, debug }.
{generator_warmup, 12000 }.
{generator_duration,                   90000 }.
{generator_cooldown,                   12000 }.
{generator_nodes,         ['bench1@my-pc', 'bench2@my-pc', 'bench3@my-pc']        }.
{use_binary_subscriber_key,         false}.
{n_generators_per_node,         1 }.
{write_lock_type,                 sticky_write}.
{table_nodes,                  ['bench1@my-pc', 'bench2@my-pc', 'bench3@my-pc'] }.
{storage_type,                       ram_copies}.
{n_replicas,                         3 }.
{n_fragments,                         1}.
{n_subscribers,                       50000 }.
{n_groups,                            100}.
{n_servers,                           20}.
```

Listing A.1: Example benchmark configuration file.

Appendix B

Repository and demo

A demo (Figure B.1) of Hypermnesia is uploaded on Youtube (link omitted for blind marking) for those who are interested in watching it. The code repository is also available on Github.



```
    supervisor: {local,mnesia_kernel_sup}
    started: [{pid,<0.844.0>},
              {id,mnesia_causal},
              {mfargs,{mnesia_causal,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,3000},
              {child_type,worker}]

=PROGRESS REPORT==== 14-Feb-2023::20:08:00.210400 ===
    supervisor: {local,mnesia_sup}
    started: [{pid,<0.806.0>},
              {id,mnesia_kernel_sup},
              {mfargs,{mnesia_kernel_sup,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,infinity},
              {child_type,supervisor}]

=PROGRESS REPORT==== 14-Feb-2023::20:08:00.210485 ===
    application: mnesia
    started_at: 'a@127.0.0.1'

[{ok,true},{ok,true},{ok,true}]
(a@127.0.0.1)69>
(a@127.0.0.1)69>
(a@127.0.0.1)69>
(a@127.0.0.1)69>
(a@127.0.0.1)69>
(a@127.0.0.1)69>
(a@127.0.0.1)69>
```

```
make[2]: Entering directory '/home/vincent/proj/otp/erts/start_scripts'
make[2]: Nothing to be done for 'debug'.
make[2]: Leaving directory '/home/vincent/proj/otp/erts/start_scripts'
make[2]: Entering directory '/home/vincent/proj/otp/erts/start_scripts'
 GEN    /home/vincent/proj/otp/bin/start.script
 GEN    /home/vincent/proj/otp/bin/start_sasl.script
 GEN    /home/vincent/proj/otp/bin/start_clean.script
make[2]: '/home/vincent/proj/otp/bin/no_dot_erlang.script' is up to date.
make[2]: Leaving directory '/home/vincent/proj/otp/erts/start_scripts'
make[1]: Leaving directory '/home/vincent/proj/otp/erts'
 ~/proj/otp   convert-25 ↑1 !6 ?5
 ~/proj/otp   convert-25 ↑1 !6 ?5
❯
```

```
===> Compiling converl
src/test.erl:3:2: Warning: export_all flag enabled - all functions will be exported
src/test.erl:6:2: Warning: record studnet is unused

src/mnesia_porset_admin.erl:3:2: Warning: export_all flag enabled - all functions will be exported

❯ rebar3 compile
===> Verifying dependencies...
===> Analyzing applications...
===> Compiling converl
 ~/proj/converl   main !5 ?6
 ~/proj/converl   main !5 ?6
❯
```

```
              {restart_type,permanent},
              {significant,false},
              {shutdown,3000},
              {child_type,worker}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.920163 ===
    supervisor: {local,mnesia_kernel_sup}
    started: [{pid,<0.1085.0>},
              {id,mnesia_causal},
              {mfargs,{mnesia_causal,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,3000},
              {child_type,worker}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.920267 ===
    supervisor: {local,mnesia_sup}
    started: [{pid,<0.1049.0>},
              {id,mnesia_kernel_sup},
              {mfargs,{mnesia_kernel_sup,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,infinity},
              {child_type,supervisor}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.920396 ===
    application: mnesia
    started_at: 'b@127.0.0.1'
```

```
              {restart_type,permanent},
              {significant,false},
              {shutdown,3000},
              {child_type,worker}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.912502 ===
    supervisor: {local,mnesia_kernel_sup}
    started: [{pid,<0.1060.0>},
              {id,mnesia_causal},
              {mfargs,{mnesia_causal,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,3000},
              {child_type,worker}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.912576 ===
    supervisor: {local,mnesia_sup}
    started: [{pid,<0.1024.0>},
              {id,mnesia_kernel_sup},
              {mfargs,{mnesia_kernel_sup,start,[]}},
              {restart_type,permanent},
              {significant,false},
              {shutdown,infinity},
              {child_type,supervisor}]

=PROGRESS REPORT==== 14-Feb-2023::20:07:59.912706 ===
    application: mnesia
    started_at: 'c@127.0.0.1'
```

Figure B.1: Demo

Appendix C

State-based CRDTs details

This chapter complements §2.3.1 with more formal details of state-based CRDTs, along with examples.

C.1 Properties

A state-based CRDT communicates by propagating the entire state of the data type with other parties, and perform the merge operation with a merge (sometimes called join) operator to converge towards the least upper bound of the two states. They are sometimes also referred to as Convergent Replicated Data Types (CvRDTs). I start by defining some of the mathematical properties that state-based CRDTs, leading towards its convergence property (Proposition C.1.1), which are essential in eventual consistency.

Definition C.1.1 (Least Upper Bound (LUB) [50]). We define the LUB of two states x and y, partially ordered by $\sqsubseteq_v$ to be $m = x \sqcup_v y$, and there is no m' such that $x \sqsubseteq_v m' \wedge y \sqsubseteq_v m'$. The operator defined above $\sqcup_v$ is also called the join operator.

We observe that the join operator is idempotent ($x \sqcup y = x \sqcup y \sqcup y = x \sqcup x \sqcup y$), commutative ($x \sqcup y = y \sqcup x$) and associative. We then define the join semilattice to be:

Definition C.1.2 (Join Semilattice [12]). An ordered set $(S, \sqsubseteq_v)$, equipped with a join operator $\sqcup_v$ is a join semilattice if and only if $\forall x, y \in S, \exists m = x \sqcup_v y$

Now a state-based CRDT consists of a triple (S, M, Q) where S is a join semilattice, Q is a set of *query functions* and M is a set of *mutators*, where each m takes in one state $X \in S$ and produce a new state $X' = m(X)$. Moreover, mutators are defined to be *inflations*, i.e., for any m and X, we have $X \sqsubseteq m(X)$.

This definition gives us some nice properties of state-based CRDTs. In particular, if we define our merge operator to be the join operator over the join semilattice, then our merge process is converging towards the LUB of the most recent values. As an example, we consider the ordered set of natural numbers as $(\mathbb{N}, <)$, ordered by the less-than relation. Then this is indeed a join semilattice since the max operator acts as the LUB of every two natural numbers x and y, i.e. $x \sqcup y = \max(x, y)$. This property of the join semilattice gives the following guarantee:

Proposition C.1.1 ([50]). *Any two object replicas of a state-based CRDTs eventually converge, assuming the system transmits payload infinitely often between pairs of replicas over eventually-reliable point-to-point channels.*

Note that we still require delivery of states to guarantee convergence, i.e. we need every message to eventually reach every replica of the cluster. However, this is the only requirement, since the `merge` operator is idempotent and commutative, we can tolerate arbitrary message reordering and repeated delivery of messages.

The downside of the state-based CRDT is that it requires the dissemination of the entire state, which makes it not suitable for collections of large states, such as a big set of elements. Almeida et al. [1] proposes a variant of state-based CRDTs to address this issue, which we discuss below.

C.2 Delta-state CRDTs

Delta-State Conflict-Free Replicated Data Types (δ-CRDTs) [1] is a new kind of state-based CRDTs that disseminates only the changing part of the operation as a δ-state, hence reducing the communication cost.

Definition C.2.1 (Delta-mutator [1])**.** A delta-mutator m^δ is a function, corresponding to an update operation, which takes a state X in a join semilattice S as its parameter and returns a delta-mutation $m^\delta(X) \in S$.

Definition C.2.2 (Delta-group [1])**.** A delta-group is inductively defined as either a delta-mutation or a join of several delta-groups.

Definition C.2.3 (δ-CRDT [1])**.** A δ-CRDT consists of a triple (S, M^δ, Q), where S and Q are defined as before, and M^δ is a set of delta-mutators. The state transition is defined to be one of the below:

Joining with a delta-state:

$$X' = X \sqcup m^\delta(X)$$

Joining with a delta-group D:

$$X' = X \sqcup D$$

This definition decouples the state transition from applying the mutation, since now we first mutate to get a delta-state, then we apply the join to change our state. The results of a δ-mutator looks almost like producing an operation in op-based CRDTs, apart from the requirement that the result must also be a state in the join semilattice.

This definition is sufficient if the CRDT that we want to build does not require causal consistency, such as a counter. But more is needed if we do care about causal order, like (the addition and deletion in a) set. We therefore make some additional constructions:

A *causal context* (sometimes also called *causal history*) is a collection of events/dots defined as follows [1]:

$$
\begin{aligned}
\text{CausalContext} &= \mathcal{P}(\mathbb{I} \times \mathbb{N}) \\
\max_i(c) &= \max(\{n \mid (i, n) \in c\} \cup \{0\}) \\
\text{next}_i(c) &= (i, \max_i(c) + 1)
\end{aligned}
$$

where $\mathbb{I}$ is the set of replica identifiers, $\mathbb{N}$ is the natural numbers. A causal context allows us to tag each event in our system with a unique identifier (or a *dot*) (i, n), signalling that the event happens at replica i at point n.

A *dot store* acts as a container for data-type specific information. It can be queried with the `dots` function which takes in a dot store and returns the set of dots in the store. There are three types of dot store, defined as [1]

$$
\begin{aligned}
\text{DotSet} : \text{DotStore} &= \mathcal{P}(\mathbb{I} \times \mathbb{N}) \\
\text{DotFun}\langle V : \text{Lattice}\rangle : \text{DotStore} &= \mathcal{P}(\mathbb{I} \times \mathbb{N}) \hookrightarrow V \\
\text{DotMap}\langle K, V : \text{DotStore}\rangle : \text{DotStore} &= K \hookrightarrow V
\end{aligned}
$$

And we can now combine the causal context and dot store to construct states for δ-CRDTs [1]:

$$
\begin{aligned}
\text{Causal}\langle T : \text{DotStore}\rangle &= T \times \text{CausalContext} \\
\textbf{when} \quad & T : \text{DotSet} \\
(s, c) \sqcup (s', c') &= ((s \cap s') \cup (s \setminus c') \cup (s' \setminus c), c \cup c') \\
\textbf{when} \quad & T : \text{DotMap}\langle\_,\_\rangle \\
(m, c) \sqcup (m', c') &= (\{k \mapsto v(k) \mid k \in \text{dom}\, m \cup \text{dom}\, m' \wedge v(k) \neq \bot\}, c \cup c') \\
\textbf{where}\ v(k) &= \text{fst}((m(k), c) \sqcup (m'(k), c))
\end{aligned}
$$

Listing C.1: The join semilattice for δ-CRDTs, adapted from [1].

C.3 Example: Delta-State set CRDT

With the causal context and dot store above, we can now define our delta-state add-wins set in Listing C.2. A δ-CRDT add-wins set consists of a pair of a dot map and a causal context. For such a set, when we add an element, the delta mutator add^{δ} generates a singleton map from the element to a dot, and the causal context is only concerned with the new tag and all the previous tags associated with the added element. When we remove an element, the dot map is replaced with an empty map and the causal context with all the previous tags associated with the element, this will cause the element e to be removed during the next join (because it is in the causal context but not in the dot store, according to the join rule in Listing C.1). Look-up is just examining the domain of the dot map, i.e. everything that is in the dot map m is considered to be in the set.

$$\begin{aligned}
\mathrm{AWSet}\langle E\rangle &= \mathrm{Causal}\langle \mathrm{DotMap}\langle E, \mathrm{DotSet}\rangle\rangle \\
\mathrm{add}_i^{\delta}(e,(m,c)) &= (\{e \mapsto d\}, d \cup m(e)) \quad \textbf{where}\ d = \{\mathrm{next}_i(c)\} \\
\mathrm{remove}_i^{\delta}(e,(m,c)) &= (\{\}, m(e)) \\
\mathrm{clear}_i^{\delta}((m,c)) &= (\{\}, \mathrm{dots}(m)) \\
\mathrm{elements}((\mathrm{m},\mathrm{c})) &= \mathrm{dom}\, m
\end{aligned}$$

Listing C.2: δ-CRDT set specification, adapted from [1].

Appendix D

Remove-wins set algorithm

An algorithm for the pure op-based RW-set is shown in Algorithm 3.

Algorithm 3: Pure RW-set pseudocode, defined in terms of usual set operations. Note this is pseudocode sacrifices efficiency for clarity. Ideas are taken from [4, 5, 6].

```
POLog ← []
function add($(e, t)$):
    remove_redundant($(e, t, \text{add})$)
    for $(e', t', o') \in$ POLog do
        if redundant($((e, t, \text{add}), (e', t', o'))$) then
            return
    POLog ← append(POLog, $(e, t, \text{add})$)
function delete $(e, t)$:
    remove_redundant($(e, t, \text{delete})$)
    for $(e', t', o') \in$ POLog do
        if redundant($((e, t, \text{delete}), (e', t', o'))$) then
            return
    POLog ← append(POLog, $(e, t, \text{delete})$)
function read($(k)$):
    for $(e, t, o) \in$ POLog do
        if $e.\text{key} = k$ then
            nodel ← true
            for $e, t, o \in$ POLog do
                if $e.\text{key} = k \wedge o = \text{delete}$ then
                    nodel ← false
            if nodel then
                return $e$
    return undefined
function remove_redundant $(e, t, o)$:
    for $(e', t', o') \in$ POLog do
        if redundant($((e', t', o'), (e, t, o))$) then
            POLog ← remove(POLog, $(e', t', o')$)
function redundant ((e,t,o), (e’,t’,o’)):
    /* check whether $(e, t, o)$ is made redundant by $(e', t', o')$ */
    if $e = e' \wedge t < t'$ then
        return true
    else
        return false
function reify($(e, t, o)$):
    remove_redundant($(e, t, o)$)
```

Appendix E

Evaluation set-up

E.1 Single machine specification

The single phyiscal machine has a AMD Ryzen 9 3900X 12-Core Processor and 16GiB DIMM DDR4 2667 MHz memory. It runs Ubuntu 22.04.2 LTS Operating System.

E.2 Cluster configuration

The cluster consists of 10 nodes within our own small, research cluster. These ten nodes are placed inside three racks. Each node in the cluster has a 6-core Intel Xeon CPU E5-2430L @ 2.40GHz CPU and 64GB of DDR3 memory. They all run Ubuntu 20.04 LTS operating system. There are 4 out of 9 worker nodes with hyper-threading enabled.

The average network latency between each machine connected via Ethernet is about 0.19 ms[*] and the average throughput is about 918 Mbit s^{-1}[†].

[*] measured with `ping(8)` https://linux.die.net/man/8/ping
[†] measured with `iperf(1)` https://linux.die.net/man/1/iperf

Appendix F

Hypermnesia test suite output

The output of the Mnesia test suites, with the additional tests for Hypermnesia is listed below.

```
{mnesia_SUITE,all}.
[{4067,
  {{mnesia_SUITE,all},
   [{0,{crash,{mnesia_SUITE,app},{test_case_failed,4}}},
    {0,{ok,{mnesia_SUITE,appup},[]}},
    {233,
     {{mnesia_SUITE,{group,light}},
      [{1,
        {{mnesia_SUITE,{group,install}},
         [{1,
           {{mnesia_install_test,all},
            [{0,{ok,{mnesia_install_test,silly_durability},[]}},
             {0,{ok,{mnesia_install_test,silly_move},[]}},
             {1,{ok,{mnesia_install_test,silly_upgrade},[]}}]}}]}},
       {0,
        {{mnesia_SUITE,{group,nice}},
         [{0,
           {{mnesia_nice_coverage_test,all},
            [{0,{ok,{mnesia_nice_coverage_test,nice},[]}}]}}]}},
       {66,
        {{mnesia_SUITE,{group,evil}},
         [{66,
           {{mnesia_evil_coverage_test,all},
            [{0,{ok,{mnesia_evil_coverage_test,system_info},[]}},
             {0,{ok,{mnesia_evil_coverage_test,table_info},[]}},
             {0,{ok,{mnesia_evil_coverage_test,error_description},[]}},
             {0,{ok,{mnesia_evil_coverage_test,db_node_lifecycle},[]}},
             {0,{ok,{mnesia_evil_coverage_test,evil_delete_db_node},[]}},
             {0,{ok,{mnesia_evil_coverage_test,start_and_stop},[]}},
             {0,{ok,{mnesia_evil_coverage_test,checkpoint},[]}},
             {0,{ok,{mnesia_evil_coverage_test,table_lifecycle},[]}},
             {0,{ok,{mnesia_evil_coverage_test,storage_options},[]}},
             {5,{ok,{mnesia_evil_coverage_test,add_copy_conflict},[]}},
             {4,{ok,{mnesia_evil_coverage_test,add_copy_when_going_down},[]}},
             {1,
              {ok,{mnesia_evil_coverage_test,add_copy_when_dst_going_down},
                  []}},
             {3,{ok,{mnesia_evil_coverage_test,add_copy_with_down},[]}},
             {3,{ok,{mnesia_evil_coverage_test,replica_management},[]}},
             {1,{ok,{mnesia_evil_coverage_test,clear_table_during_load},[]}},
             {0,{ok,{mnesia_evil_coverage_test,schema_availability},[]}},
             {0,{ok,{mnesia_evil_coverage_test,local_content},[]}},
             {0,
              {{mnesia_evil_coverage_test,{group,table_access_modifications}},
               [{0,
                 {ok,{mnesia_evil_coverage_test,change_table_access_mode},[]}},
                {0,
                 {ok,{mnesia_evil_coverage_test,change_table_load_order},[]}},
                {0,{ok,{mnesia_evil_coverage_test,set_master_nodes},[]}},
                {0,
                 {ok,{mnesia_evil_coverage_test,offline_set_master_nodes},
                     []}}]}},
             {1,{ok,{mnesia_evil_coverage_test,replica_location},[]}},
             {5,
              {{mnesia_evil_coverage_test,{group,table_sync}},
               [{0,{ok,{mnesia_evil_coverage_test,dump_tables},[]}},
                {0,{ok,{mnesia_evil_coverage_test,dump_log},[]}},
                {5,{ok,{mnesia_evil_coverage_test,wait_for_tables},[]}},
                {0,{ok,{mnesia_evil_coverage_test,force_load_table},[]}}]}},
             {0,{ok,{mnesia_evil_coverage_test,user_properties},[]}},
             {0,{ok,{mnesia_evil_coverage_test,unsupp_user_props},[]}},
             {1,
              {{mnesia_evil_coverage_test,{group,record_name}},
               [{1,
                 {{mnesia_evil_coverage_test,{group,record_name_dirty_access}},
                  [{0,
                    {ok,{mnesia_evil_coverage_test,
                            record_name_dirty_access_ram},
                        []}},
                   {0,
                    {ok,{mnesia_evil_coverage_test,
                            record_name_dirty_access_disc},
                        []}},
                   {0,
                    {ok,{mnesia_evil_coverage_test,
                            record_name_dirty_access_disc_only},
                        []}},
                   {0,
                    {ok,{mnesia_evil_coverage_test,
                            record_name_dirty_access_xets},
                        []}}]}}]}},
             {3,
              {{mnesia_evil_coverage_test,{group,snmp_access}},
               [{0,{ok,{mnesia_evil_coverage_test,snmp_open_table},[]}},
                {0,{ok,{mnesia_evil_coverage_test,snmp_close_table},[]}},
                {0,{ok,{mnesia_evil_coverage_test,snmp_get_next_index},[]}},
                {1,{ok,{mnesia_evil_coverage_test,snmp_get_row},[]}},
                {0,{ok,{mnesia_evil_coverage_test,snmp_get_mnesia_key},[]}},
                {0,{ok,{mnesia_evil_coverage_test,snmp_update_counter},[]}},
                {0,{ok,{mnesia_evil_coverage_test,snmp_order},[]}}]}},
             {7,
              {{mnesia_evil_coverage_test,{group,subscriptions}},
               [{7,{ok,{mnesia_evil_coverage_test,subscribe_standard},[]}},
                {0,{ok,{mnesia_evil_coverage_test,subscribe_extended},[]}}]}},
             {0,
              {{mnesia_evil_coverage_test,{group,iteration}},
               [{0,{ok,{mnesia_evil_coverage_test,foldl},[]}}]}},
             {0,
              {{mnesia_evil_coverage_test,{group,debug_support}},
               [{0,{ok,{mnesia_evil_coverage_test,info},[]}},
                {0,{ok,{mnesia_evil_coverage_test,schema_0},[]}},
                {0,{ok,{mnesia_evil_coverage_test,schema_1},[]}},
                {0,{ok,{mnesia_evil_coverage_test,view_0},[]}},
                {0,{ok,{mnesia_evil_coverage_test,view_1},[]}},
                {0,{ok,{mnesia_evil_coverage_test,view_2},[]}},
                {0,{ok,{mnesia_evil_coverage_test,lkill},[]}},
                {0,{ok,{mnesia_evil_coverage_test,kill},[]}}]}},
             {1,{ok,{mnesia_evil_coverage_test,sorted_ets},[]}},
             {0,{ok,{mnesia_evil_coverage_test,index_cleanup},[]}},
             {5,
              {{mnesia_dirty_access_test,all},
               [{0,
                 {{mnesia_dirty_access_test,{group,dirty_write}},
                  [{0,{ok,{mnesia_dirty_access_test,dirty_write_ram},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_write_disc},[]}},
                   {0,
                    {ok,{mnesia_dirty_access_test,dirty_write_disc_only},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_write_xets},[]}}]}},
                {0,
                 {{mnesia_dirty_access_test,{group,dirty_read}},
                  [{0,{ok,{mnesia_dirty_access_test,dirty_read_ram},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_read_disc},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_read_disc_only},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_read_xets},[]}}]}},
                {0,
                 {{mnesia_dirty_access_test,{group,dirty_update_counter}},
                  [{0,
                    {ok,{mnesia_dirty_access_test,dirty_update_counter_ram},
                        []}},
                   {0,
                    {ok,{mnesia_dirty_access_test,dirty_update_counter_disc},
                        []}},
                   {0,
                    {ok,{mnesia_dirty_access_test,
                            dirty_update_counter_disc_only},
                        []}},
                   {0,
                    {ok,{mnesia_dirty_access_test,dirty_update_counter_xets},
                        []}}]}},
                {0,
                 {{mnesia_dirty_access_test,{group,dirty_delete}},
                   [{0,{ok,{mnesia_dirty_access_test,dirty_delete_ram},[]}},
                    {0,{ok,{mnesia_dirty_access_test,dirty_delete_disc},[]}},
                    {0,
                     {ok,{mnesia_dirty_access_test,dirty_delete_disc_only},[]}},
                    {0,{ok,{mnesia_dirty_access_test,dirty_delete_xets},[]}}]}},
                 {0,
                  {{mnesia_dirty_access_test,{group,dirty_delete_object}},
                   [{0,
                     {ok,{mnesia_dirty_access_test,dirty_delete_object_ram},
                         []}},
                    {0,
                     {ok,{mnesia_dirty_access_test,dirty_delete_object_disc},
                         []}},
                    {0,
                     {ok,{mnesia_dirty_access_test,
                             dirty_delete_object_disc_only},
                         []}},
                    {0,
                     {ok,{mnesia_dirty_access_test,dirty_delete_object_xets},
                         []}}]}},
                 {0,
                  {{mnesia_dirty_access_test,{group,dirty_match_object}},
                   [{0,
                     {ok,{mnesia_dirty_access_test,dirty_match_object_ram},[]}},
                    {0,
                     {ok,{mnesia_dirty_access_test,dirty_match_object_disc},
                         []}},
                    {0,
                     {ok,{mnesia_dirty_access_test,
                             dirty_match_object_disc_only},
                         []}},
                    {0,
                     {ok,{mnesia_dirty_access_test,dirty_match_object_xets},
                         []}}]}},
                 {1,
                  {{mnesia_dirty_access_test,{group,dirty_index}},
                   [{0,
                     {{mnesia_dirty_access_test,
                          {group,dirty_index_match_object}},
                      [{0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_match_object_ram},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_match_object_disc},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_match_object_disc_only},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_match_object_xets},
                            []}}]}},
                    {0,
                     {{mnesia_dirty_access_test,{group,dirty_index_read}},
                      [{0,
                        {ok,{mnesia_dirty_access_test,dirty_index_read_ram},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,dirty_index_read_disc},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_read_disc_only},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,dirty_index_read_xets},
                            []}}]}},
                    {0,
                     {{mnesia_dirty_access_test,{group,dirty_index_update}},
                      [{0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_update_set_ram},
                            []}},
                       {0,
                        {ok,{mnesia_dirty_access_test,
                                dirty_index_update_set_disc},
                            []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_set_disc_only},
                           []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_set_xets},
                           []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_bag_ram},
                           []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_bag_disc},
                           []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_bag_disc_only},
                           []}},
                      {0,
                       {ok,{mnesia_dirty_access_test,
                               dirty_index_update_bag_xets},
                           []}}]}}]}},
                {0,
                 {{mnesia_dirty_access_test,{group,dirty_iter}},
                  [{0,{ok,{mnesia_dirty_access_test,dirty_iter_ram},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_iter_disc},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_iter_disc_only},[]}},
                   {0,{ok,{mnesia_dirty_access_test,dirty_iter_xets},[]}}]}},
                {3,
                 {{mnesia_dirty_access_test,{group,admin_tests}},
                  [{0,{ok,{mnesia_dirty_access_test,del_table_copy_1},[]}},
                   {0,{ok,{mnesia_dirty_access_test,del_table_copy_2},[]}},
                   {0,{ok,{mnesia_dirty_access_test,del_table_copy_3},[]}},
                   {0,{ok,{mnesia_dirty_access_test,add_table_copy_1},[]}},
                   {0,{ok,{mnesia_dirty_access_test,add_table_copy_2},[]}},
                   {0,{ok,{mnesia_dirty_access_test,add_table_copy_3},[]}},
                   {0,{ok,{mnesia_dirty_access_test,add_table_copy_4},[]}},
                   {0,{ok,{mnesia_dirty_access_test,move_table_copy_1},[]}},
                   {0,{ok,{mnesia_dirty_access_test,move_table_copy_2},[]}},
                   {0,{ok,{mnesia_dirty_access_test,move_table_copy_3},[]}},
                   {0,{ok,{mnesia_dirty_access_test,move_table_copy_4},[]}}]}},
                {0,
                 {ok,{mnesia_dirty_access_test,dirty_error_stacktrace},[]}}]}},
             {5,
              {{mnesia_trans_access_test,all},
               [{0,{ok,{mnesia_trans_access_test,write},[]}},
                {0,{ok,{mnesia_trans_access_test,read},[]}},
                {0,{ok,{mnesia_trans_access_test,wread},[]}},
                {0,{ok,{mnesia_trans_access_test,delete},[]}},
                {0,{ok,{mnesia_trans_access_test,delete_object_bag},[]}},
                {0,{ok,{mnesia_trans_access_test,delete_object_set},[]}},
                {0,{ok,{mnesia_trans_access_test,match_object},[]}},
                {0,{ok,{mnesia_trans_access_test,select},[]}},
                {0,{ok,{mnesia_trans_access_test,select14},[]}},
                {0,{ok,{mnesia_trans_access_test,all_keys},[]}},
                {0,{ok,{mnesia_trans_access_test,transaction},[]}},
                {2,
                 {{mnesia_trans_access_test,{group,nested_activities}},
                  [{0,{ok,{mnesia_trans_access_test,basic_nested},[]}},
                   {1,
                    {{mnesia_trans_access_test,{group,nested_transactions}},
                     [{0,
                       {ok,{mnesia_trans_access_test,nested_trans_both_ok},
                           []}},
                      {0,
                       {ok,{mnesia_trans_access_test,nested_trans_child_dies},
                           []}},
                      {0,
                       {ok,{mnesia_trans_access_test,nested_trans_parent_dies},
                           []}},
                      {0,
                       {ok,{mnesia_trans_access_test,nested_trans_both_dies},
                           []}}]}},
                   {1,
                    {ok,{mnesia_trans_access_test,mix_of_nested_activities},
                        []}}]}},
                {0,
                 {{mnesia_trans_access_test,{group,index_tabs}},
                        [{0,{ok,{mnesia_trans_access_test,index_match_object},[]}},
                         {0,{ok,{mnesia_trans_access_test,index_read},[]}},
                         {0,
                          {{mnesia_trans_access_test,{group,index_update}},
                           [{0,{ok,{mnesia_trans_access_test,index_update_set},[]}},
                            {0,
                             {ok,{mnesia_trans_access_test,index_update_bag},[]}}]}},
                         {0,{ok,{mnesia_trans_access_test,index_write},[]}},
                         {0,
                          {ok,{mnesia_trans_access_test,index_delete_object},[]}}]}},
                      {2,
                       {{mnesia_trans_access_test,{group,index_lifecycle}},
                        [{0,{ok,{mnesia_trans_access_test,add_table_index_ram},[]}},
                         {0,{ok,{mnesia_trans_access_test,add_table_index_disc},[]}},
                         {0,
                          {ok,{mnesia_trans_access_test,add_table_index_disc_only},
                              []}},
                         {0,
                          {ok,{mnesia_trans_access_test,create_live_table_index_ram},
                              []}},
                         {0,
                          {ok,{mnesia_trans_access_test,
                                  create_live_table_index_disc},
                              []}},
                         {0,
                          {ok,{mnesia_trans_access_test,
                                  create_live_table_index_disc_only},
                              []}},
                         {0,{ok,{mnesia_trans_access_test,del_table_index_ram},[]}},
                         {0,{ok,{mnesia_trans_access_test,del_table_index_disc},[]}},
                         {0,
                          {ok,{mnesia_trans_access_test,del_table_index_disc_only},
                              []}},
                         {0,
                          {{mnesia_trans_access_test,{group,idx_schema_changes}},
                           [{0,
                             {ok,{mnesia_trans_access_test,idx_schema_changes_ram},
                                 []}},
                            {0,
                             {ok,{mnesia_trans_access_test,idx_schema_changes_disc},
                                 []}},
                            {0,
                             {ok,{mnesia_trans_access_test,
                                     idx_schema_changes_disc_only},
                                 []}}]}}]}}]}},
                   {10,
                    {{mnesia_evil_backup,all},
                     [{0,{ok,{mnesia_evil_backup,backup},[]}},
                      {0,{ok,{mnesia_evil_backup,bad_backup},[]}},
                      {0,{ok,{mnesia_evil_backup,global_backup_checkpoint},[]}},
                      {3,
                       {{mnesia_evil_backup,{group,restore_tables}},
                        [{0,{ok,{mnesia_evil_backup,restore_errors},[]}},
                         {0,{ok,{mnesia_evil_backup,restore_clear},[]}},
                         {0,{ok,{mnesia_evil_backup,restore_keep},[]}},
                         {0,{ok,{mnesia_evil_backup,restore_recreate},[]}},
                         {0,{ok,{mnesia_evil_backup,restore_clear_ram},[]}}]}},
                      {0,{ok,{mnesia_evil_backup,traverse_backup},[]}},
                      {0,{ok,{mnesia_evil_backup,selective_backup_checkpoint},[]}},
                      {0,{ok,{mnesia_evil_backup,incremental_backup_checkpoint},[]}},
                      {2,{ok,{mnesia_evil_backup,install_fallback},[]}},
                      {1,{ok,{mnesia_evil_backup,uninstall_fallback},[]}},
                      {1,{ok,{mnesia_evil_backup,local_fallback},[]}},
                      {0,{ok,{mnesia_evil_backup,sops_with_checkpoint},[]}}]}}]}}]}},
             {2,
              {{mnesia_frag_test,{group,light}},
               [{1,
                 {{mnesia_frag_test,{group,nice}},
                  [{0,{ok,{mnesia_frag_test,nice_single},[]}},
                   {0,{ok,{mnesia_frag_test,nice_multi},[]}},
                   {0,{ok,{mnesia_frag_test,nice_access},[]}},
                   {0,{ok,{mnesia_frag_test,iter_access},[]}}]}},
                {0,
                 {{mnesia_frag_test,{group,evil}},
                  [{0,{ok,{mnesia_frag_test,evil_create},[]}},
                   {0,
                    {skip,
                        {mnesia_frag_test,evil_delete},
                        "Not yet implemented (NYI).\n"}},
                   {0,{ok,{mnesia_frag_test,evil_change},[]}},
                   {0,{ok,{mnesia_frag_test,evil_combine},[]}},
                   {0,{ok,{mnesia_frag_test,evil_loop},[]}},
                   {0,{ok,{mnesia_frag_test,evil_delete_db_node},[]}}]}}]}},
             {1,
              {{mnesia_SUITE,{group,qlc}},
               [{1,
                 {{mnesia_qlc_test,all},
                  [{0,
                    {{mnesia_qlc_test,{group,dirty}},
                     [{0,{ok,{mnesia_qlc_test,dirty_nice_ram_copies},[]}},
                      {0,{ok,{mnesia_qlc_test,dirty_nice_disc_copies},[]}},
                      {0,{ok,{mnesia_qlc_test,dirty_nice_disc_only_copies},[]}}]}},
                   {0,
                    {{mnesia_qlc_test,{group,trans}},
                     [{0,{ok,{mnesia_qlc_test,trans_nice_ram_copies},[]}},
                      {0,{ok,{mnesia_qlc_test,trans_nice_disc_copies},[]}},
                      {0,{ok,{mnesia_qlc_test,trans_nice_disc_only_copies},[]}},
                      {0,
                       {{mnesia_qlc_test,{group,atomic}},
                        [{0,{ok,{mnesia_qlc_test,atomic_eval},[]}}]}}]}},
                   {0,{ok,{mnesia_qlc_test,frag},[]}},
                   {0,{ok,{mnesia_qlc_test,info},[]}},
                   {0,{ok,{mnesia_qlc_test,mnesia_down},[]}}]}}]}},
             {0,
              {{mnesia_SUITE,{group,index_plugins}},
               [{0,
                 {{mnesia_index_plugin_test,all},
                  [{0,{ok,{mnesia_index_plugin_test,add_rm_plugin},[]}},
                   {0,{ok,{mnesia_index_plugin_test,tab_with_plugin_index},[]}},
                   {0,
                    {ok,{mnesia_index_plugin_test,tab_with_multiple_plugin_indexes},
                        []}},
                   {0,{ok,{mnesia_index_plugin_test,ix_match_w_plugin},[]}},
                   {0,{ok,{mnesia_index_plugin_test,ix_match_w_plugin_ordered},[]}},
                   {0,{ok,{mnesia_index_plugin_test,ix_match_w_plugin_bag},[]}},
                   {0,{ok,{mnesia_index_plugin_test,ix_update_w_plugin},[]}}]}}]}},
             {0,
              {{mnesia_SUITE,{group,registry}},
               [{0,
                 {{mnesia_registry_test,all},
                  [{0,{ok,{mnesia_registry_test,good_dump},[]}},
                   {0,{ok,{mnesia_registry_test,bad_dump},[]}}]}}]}},
             {108,
              {{mnesia_SUITE,{group,config}},
              [{109,
              {{mnesia_config_test,all},
               [{2,{ok,{mnesia_config_test,access_module},[]}},
                {2,{ok,{mnesia_config_test,auto_repair},[]}},
                {4,{ok,{mnesia_config_test,backup_module},[]}},
                {10,{ok,{mnesia_config_test,debug},[]}},
                {2,{ok,{mnesia_config_test,dir},[]}},
                {16,{ok,{mnesia_config_test,dump_log_load_regulation},[]}},
                {11,
                 {{mnesia_config_test,{group,dump_log_thresholds}},
                  [{8,{ok,{mnesia_config_test,dump_log_time_threshold},[]}},
                   {3,{ok,{mnesia_config_test,dump_log_write_threshold},[]}}]}},
                {4,{ok,{mnesia_config_test,dump_log_update_in_place},[]}},
                {8,{ok,{mnesia_config_test,event_module},[]}},
                {2,{ok,{mnesia_config_test,backend_plugin_registration},[]}},
                {4,{ok,{mnesia_config_test,inconsistent_database},[]}},
                {0,{skip,{mnesia_config_test,max_wait_for_decision},'NYI'}},
                {1,{ok,{mnesia_config_test,send_compressed},[]}},
                {0,{ok,{mnesia_config_test,app_test},[]}},
                {39,
                 {{mnesia_config_test,{group,schema_config}},
                  [{2,{ok,{mnesia_config_test,start_one_disc_full_then_one_disc_less},[]}},
                   {4,
                    {ok,{mnesia_config_test,start_first_one_disc_less_then_one_disc_full},
                        []}},
                   {2,
                    {ok,{mnesia_config_test,start_first_one_disc_less_then_two_more_disc_less},
                        []}},
                   {2,
                    {ok,{mnesia_config_test,schema_location_and_extra_db_nodes_combinations},
                        []}},
                   {2,{ok,{mnesia_config_test,table_load_to_disc_less_nodes},[]}},
                   {10,{ok,{mnesia_config_test,schema_merge},[]}},
                   {16,
                    {{mnesia_config_test,{group,dynamic_connect}},
                     [{4,{ok,{mnesia_config_test,dynamic_basic},[]}},
                         {1,{ok,{mnesia_config_test,dynamic_ext},[]}},
                         {10,{ok,{mnesia_config_test,dynamic_bad},[]}}]}]}]}},
                   {2,{ok,{mnesia_config_test,unknown_config},[]}}]}]}]}},
                {51,
                 {{mnesia_SUITE,{group,examples}},
                  [{51,
                    {{mnesia_examples_test,all},
                     [{0,{ok,{mnesia_examples_test,bup},[]}},
                      {0,{skip,{mnesia_examples_test,company},'NYI'}},
                      {0,{ok,{mnesia_examples_test,meter},[]}},
                      {51,
                       {{mnesia_examples_test,{group,tpcb}},
                        [{6,{ok,{mnesia_examples_test,replica_test},[]}},
                         {6,{ok,{mnesia_examples_test,sticky_replica_test},[]}},
                         {7,{ok,{mnesia_examples_test,dist_test},[]}},
                         {6,{ok,{mnesia_examples_test,conflict_test},[]}},
                         {6,{ok,{mnesia_examples_test,frag_test},[]}},
                         {6,{ok,{mnesia_examples_test,frag2_test},[]}},
                         {6,{ok,{mnesia_examples_test,remote_test},[]}},
                         {6,
                          {ok,{mnesia_examples_test,remote_frag2_test},[]}}]}}]}}]}}]}},
             {930,
              {{mnesia_SUITE,{group,medium}},
               [{1,
                 {{mnesia_SUITE,{group,install}},
                  [{1,
                    {{mnesia_install_test,all},
                     [{0,{ok,{mnesia_install_test,silly_durability},[]}},
                      {0,{ok,{mnesia_install_test,silly_move},[]}},
                      {1,{ok,{mnesia_install_test,silly_upgrade},[]}}]}}]}},
                {40,
                 {{mnesia_SUITE,{group,atomicity}},
                  [{40,
                    {{mnesia_atomicity_test,all},
                     [{0,
                       {ok,{mnesia_atomicity_test,explicit_abort_in_middle_of_trans},
                           []}},
                      {0,
                       {ok,{mnesia_atomicity_test,runtime_error_in_middle_of_trans},
                           []}},
                      {0,{ok,{mnesia_atomicity_test,kill_self_in_middle_of_trans},[]}},
                      {0,{ok,{mnesia_atomicity_test,throw_in_middle_of_trans},[]}},
                      {40,
                       {{mnesia_atomicity_test,{group,mnesia_down_in_middle_of_trans}},
                        [{8,
                          {ok,{mnesia_atomicity_test,mnesia_down_during_infinite_trans},
                              []}},
                         {21,
                          {{mnesia_atomicity_test,{group,lock_waiter}},
                           [{0,{ok,{mnesia_atomicity_test,lock_waiter_sw_r},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_rt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wr_r},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_srw_r},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_wr},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_sw_srw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wr_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_srw_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wr_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_srw_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wr_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_srw_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_r_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_r_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_r_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_rt_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_rt_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_rt_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wr_wr},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_srw_srw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_r},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_w},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_rt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_wt},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_wr},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_srw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_wt_sw},[]}},
                            {0,{ok,{mnesia_atomicity_test,lock_waiter_w_wr},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_srw},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_sw},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_r},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_w},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_rt},[]}},
                         {0,{ok,{mnesia_atomicity_test,lock_waiter_w_wt},[]}}]}},
                      {11,
                       {{mnesia_atomicity_test,{group,restart_check}},
                        [{0,{ok,{mnesia_atomicity_test,restart_r_one},[]}},
                         {0,{ok,{mnesia_atomicity_test,restart_w_one},[]}},
                         {0,{ok,{mnesia_atomicity_test,restart_rt_one},[]}},
                         {0,{ok,{mnesia_atomicity_test,restart_wt_one},[]}},
                         {0,{ok,{mnesia_atomicity_test,restart_wr_one},[]}},
                         {0,{ok,{mnesia_atomicity_test,restart_sw_one},[]}},
                         {1,{ok,{mnesia_atomicity_test,restart_r_two},[]}},
                         {1,{ok,{mnesia_atomicity_test,restart_w_two},[]}},
                         {1,{ok,{mnesia_atomicity_test,restart_rt_two},[]}},
                         {1,{ok,{mnesia_atomicity_test,restart_wt_two},[]}},
                         {1,{ok,{mnesia_atomicity_test,restart_wr_two},[]}},
                         {0,
                          {ok,{mnesia_atomicity_test,restart_sw_two},
                              []}}]}}]}}]}}]}},
             {106,
              {{mnesia_SUITE,{group,isolation}},
               [{106,
                 {{mnesia_isolation_test,all},
                  [{105,
                    {{mnesia_isolation_test,{group,locking}},
                     [{0,{ok,{mnesia_isolation_test,no_conflict},[]}},
                      {0,{ok,{mnesia_isolation_test,simple_queue_conflict},[]}},
                      {12,{ok,{mnesia_isolation_test,advanced_queue_conflict},[]}},
                      {0,{ok,{mnesia_isolation_test,simple_deadlock_conflict},[]}},
                      {0,{ok,{mnesia_isolation_test,advanced_deadlock_conflict},[]}},
                      {1,{ok,{mnesia_isolation_test,schema_deadlock},[]}},
                      {1,{ok,{mnesia_isolation_test,lock_burst},[]}},
                      {26,
                       {{mnesia_isolation_test,{group,sticky_locks}},
                        [{1,
                          {ok,{mnesia_isolation_test,basic_sticky_functionality},
                              []}},
                         {25,{ok,{mnesia_isolation_test,sticky_sync},[]}}]}},
                      {0,
                       {{mnesia_isolation_test,{group,unbound_locking}},
                        [{0,{ok,{mnesia_isolation_test,unbound1},[]}},
                         {0,{ok,{mnesia_isolation_test,unbound2},[]}}]}},
                      {57,
                       {{mnesia_isolation_test,{group,admin_conflict}},
                        [{0,{ok,{mnesia_isolation_test,create_table},[]}},
                         {4,{ok,{mnesia_isolation_test,delete_table},[]}},
                         {4,{ok,{mnesia_isolation_test,move_table_copy},[]}},
                         {4,{ok,{mnesia_isolation_test,add_table_index},[]}},
                         {4,{ok,{mnesia_isolation_test,del_table_index},[]}},
                         {4,{ok,{mnesia_isolation_test,transform_table},[]}},
                         {4,{ok,{mnesia_isolation_test,snmp_open_table},[]}},
                         {4,{ok,{mnesia_isolation_test,snmp_close_table},[]}},
                         {4,{ok,{mnesia_isolation_test,change_table_copy_type},[]}},
                         {4,{ok,{mnesia_isolation_test,change_table_access},[]}},
                         {4,{ok,{mnesia_isolation_test,add_table_copy},[]}},
                         {4,{ok,{mnesia_isolation_test,del_table_copy},[]}},
                         {4,{ok,{mnesia_isolation_test,dump_tables},[]}},
                         {5,
                          {{mnesia_isolation_test,{group,extra_admin_tests}},
                           [{0,{ok,{mnesia_isolation_test,del_table_copy_1},[]}},
                            {0,{ok,{mnesia_isolation_test,del_table_copy_2},[]}},
                            {0,{ok,{mnesia_isolation_test,del_table_copy_3},[]}},
                            {0,{ok,{mnesia_isolation_test,add_table_copy_1},[]}},
                            {0,{ok,{mnesia_isolation_test,add_table_copy_2},[]}},
                            {0,{ok,{mnesia_isolation_test,add_table_copy_3},[]}},
                            {0,{ok,{mnesia_isolation_test,add_table_copy_4},[]}},
                            {0,{ok,{mnesia_isolation_test,move_table_copy_1},[]}},
                            {0,{ok,{mnesia_isolation_test,move_table_copy_2},[]}},
                            {0,{ok,{mnesia_isolation_test,move_table_copy_3},[]}},
                            {0,
                             {ok,{mnesia_isolation_test,move_table_copy_4},
                                 []}}]}}]}},
                      {5,{ok,{mnesia_isolation_test,nasty},[]}}]}},
                   {0,
                    {{mnesia_isolation_test,{group,visibility}},
                     [{0,
                       {ok,{mnesia_isolation_test,dirty_updates_visible_direct},[]}},
                             {0,
                              {ok,{mnesia_isolation_test,dirty_reads_regardless_of_trans},
                                  []}},
                             {0,
                              {ok,{mnesia_isolation_test,
                                      trans_update_invisibible_outside_trans},
                                  []}},
                             {0,
                              {ok,{mnesia_isolation_test,trans_update_visible_inside_trans},
                                  []}},
                             {0,{ok,{mnesia_isolation_test,write_shadows},[]}},
                             {0,{ok,{mnesia_isolation_test,delete_shadows},[]}},
                             {0,{ok,{mnesia_isolation_test,write_delete_shadows_bag},[]}},
                             {0,{ok,{mnesia_isolation_test,write_delete_shadows_bag2},[]}},
                             {0,
                              {{mnesia_isolation_test,{group,iteration}},
                               [{0,{ok,{mnesia_isolation_test,foldl},[]}},
                                {0,{ok,{mnesia_isolation_test,first_next},[]}}]}},
                             {0,{ok,{mnesia_isolation_test,shadow_search},[]}},
                             {0,{ok,{mnesia_isolation_test,snmp_shadows},[]}}]}]}]}]}},
                    {63,
                     {{mnesia_SUITE,{group,durability}},
                      [{63,
                        {{mnesia_durability_test,all},
                         [{52,
                           {{mnesia_durability_test,{group,load_tables}},
                            [{4,{ok,{mnesia_durability_test,load_latest_data},[]}},
                             {0,
                              {ok,{mnesia_durability_test,load_local_contents_directly},
                                  []}},
                             {0,
                              {ok,{mnesia_durability_test,
                                      load_directly_when_all_are_ram_copiesA},
                                  []}},
                             {0,
                              {ok,{mnesia_durability_test,
                                      load_directly_when_all_are_ram_copiesB},
                                  []}},
                             {4,
                              {{mnesia_durability_test,
                                   {group,late_load_when_all_are_ram_copies_on_ram_nodes}},
                               [{2,
                                 {ok,{mnesia_durability_test,
                                         late_load_all_ram_cs_ram_nodes1},
                                     []}},
                                {2,
                                 {ok,{mnesia_durability_test,
                                         late_load_all_ram_cs_ram_nodes2},
                                     []}}]}},
                             {20,
                              {ok,{mnesia_durability_test,
                                      load_when_last_replica_becomes_available},
                                  []}},
                             {0,
                              {ok,{mnesia_durability_test,
                                      load_when_down_from_all_other_replica_nodes},
                                  []}},
                             {0,
                              {skip,
                                  {mnesia_durability_test,
                                      late_load_transforms_into_disc_load},
                                  "Mnesia is not debug compiled, test case ignored.\n"}},
                             {0,
                              {skip,
                                  {mnesia_durability_test,late_load_leads_to_hanging},
                                  "Mnesia is not debug compiled, test case ignored.\n"}},
                             {3,
                              {ok,{mnesia_durability_test,
                                      force_load_when_nobody_intents_to_load},
                                  []}},
                             {0,
                              {skip,
                                  {mnesia_durability_test,
                                      force_load_when_someone_has_decided_to_load},
                                  "Mnesia is not debug compiled, test case ignored.\n"}},
                             {0,
                              {ok,{mnesia_durability_test,
                                      force_load_when_someone_else_has_loaded},
                                  []}},
                             {0,
                       {ok,{mnesia_durability_test,force_load_when_we_has_loaded},
                           []}},
                      {0,
                       {ok,{mnesia_durability_test,force_load_on_a_non_local_table},
                           []}},
                      {0,
                       {ok,{mnesia_durability_test,
                               force_load_when_the_table_does_not_exist},
                           []}},
                      {19,
                       {{mnesia_durability_test,
                            {group,load_tables_with_master_tables}},
                        [{4,{ok,{mnesia_durability_test,master_nodes},[]}},
                         {2,{ok,{mnesia_durability_test,starting_master_nodes},[]}},
                         {4,
                          {ok,{mnesia_durability_test,master_on_non_local_tables},
                              []}},
                         {0,
                          {ok,{mnesia_durability_test,
                                  remote_force_load_with_local_master_node},
                              []}},
                         {3,
                          {ok,{mnesia_durability_test,master_node_with_ram_copy_2},
                              []}},
                         {4,
                          {ok,{mnesia_durability_test,master_node_with_ram_copy_3},
                              []}}]}}]}},
                   {7,
                    {{mnesia_durability_test,{group,durability_of_dump_tables}},
                     [{5,{ok,{mnesia_durability_test,dump_ram_copies},[]}},
                      {1,{ok,{mnesia_durability_test,dump_disc_copies},[]}},
                      {1,{ok,{mnesia_durability_test,dump_disc_only},[]}}]}},
                   {1,{ok,{mnesia_durability_test,durability_of_disc_copies},[]}},
                   {1,
                    {ok,{mnesia_durability_test,durability_of_disc_only_copies},
                        []}}]}}]}},
             {159,
              {{mnesia_SUITE,{group,recovery}},
               [{159,
                 {{mnesia_recovery_test,all},
                  [{21,
                    {{mnesia_recovery_test,{group,mnesia_down}},
                     [{2,
                       {{mnesia_recovery_test,{group,mnesia_down_during_startup}},
                        [{2,
                          {ok,{mnesia_recovery_test,
                                  mnesia_down_during_startup_disk_ram},
                              []}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_init_ram},
                              "Mnesia is not debug compiled, test case ignored.\n"}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_init_disc},
                              "Mnesia is not debug compiled, test case ignored.\n"}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_init_disc_only},
                              "Mnesia is not debug compiled, test case ignored.\n"}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_tm_ram},
                              "Mnesia is not debug compiled, test case ignored.\n"}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_tm_disc},
                              "Mnesia is not debug compiled, test case ignored.\n"}},
                         {0,
                          {skip,
                              {mnesia_recovery_test,
                                  mnesia_down_during_startup_tm_disc_only},
                              "Mnesia is not debug compiled, test case ignored.\n"}}]}},
                      {10,
                       {{mnesia_recovery_test,{group,master_node_tests}},
                   [{0,{ok,{mnesia_recovery_test,no_master_2},[]}},
                    {2,{ok,{mnesia_recovery_test,no_master_3},[]}},
                    {0,{ok,{mnesia_recovery_test,one_master_2},[]}},
                    {1,{ok,{mnesia_recovery_test,one_master_3},[]}},
                    {0,{ok,{mnesia_recovery_test,two_master_2},[]}},
                    {2,{ok,{mnesia_recovery_test,two_master_3},[]}},
                    {0,{ok,{mnesia_recovery_test,all_master_2},[]}},
                    {2,{ok,{mnesia_recovery_test,all_master_3},[]}}]}},
                 {5,
                  {{mnesia_recovery_test,{group,read_during_down}},
                   [{2,{ok,{mnesia_recovery_test,dirty_read_during_down},[]}},
                    {2,
                     {ok,{mnesia_recovery_test,trans_read_during_down},[]}}]}},
                 {1,
                  {{mnesia_recovery_test,{group,with_checkpoint}},
                   [{0,{ok,{mnesia_recovery_test,with_checkpoint_same},[]}},
                    {1,{ok,{mnesia_recovery_test,with_checkpoint_other},[]}}]}},
                 {0,{ok,{mnesia_recovery_test,delete_during_start},[]}}]}},
              {15,
               {{mnesia_recovery_test,{group,explicit_stop}},
                [{15,
                  {ok,{mnesia_recovery_test,explicit_stop_during_snmp},[]}}]}},
              {0,{ok,{mnesia_recovery_test,coord_dies},[]}},
              {2,
               {{mnesia_recovery_test,{group,schema_trans}},
                [{2,
                  {{mnesia_schema_recovery_test,all},
                   [{0,
                     {{mnesia_schema_recovery_test,
                          {group,interrupted_before_log_dump}},
                      [{0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_create_ram},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_create_disc},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_create_do},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_create_nostore},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_delete_ram},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_delete_disc},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_delete_do},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_add_ram},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_add_disc},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
                       {0,
                        {skip,
                            {mnesia_schema_recovery_test,
                                interrupted_before_add_do},
                            "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_add_kill_copier},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_move_ram},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_move_disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_move_do},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_move_kill_copier},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delcopy_ram},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delcopy_disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delcopy_do},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delcopy_kill_copier},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_addindex_ram},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_addindex_disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_addindex_do},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delindex_ram},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delindex_disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_delindex_do},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_before_change_type_ram2disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_ram2do},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_disc2ram},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_disc2do},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_do2ram},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_do2disc},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_type_other_node},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_before_change_schema_type},
                     "Mnesia is not debug compiled, test case ignored.\n"}}]}},
             {2,
              {{mnesia_schema_recovery_test,
                   {group,interrupted_after_log_dump}},
               [{0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_create_ram},
                     []}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_create_disc},
                     []}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_create_do},
                     []}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_create_nostore},
                     []}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_after_delete_ram},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_after_delete_disc},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {skip,
                     {mnesia_schema_recovery_test,
                         interrupted_after_delete_do},
                     "Mnesia is not debug compiled, test case ignored.\n"}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_add_ram},
                     []}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_add_disc},
                     []}},
                {0,
                 {ok,{mnesia_schema_recovery_test,
                         interrupted_after_add_do},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_add_kill_copier},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_move_ram},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_move_disc},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_move_do},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_move_kill_copier},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delcopy_ram},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delcopy_disc},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delcopy_do},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delcopy_kill_copier},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_addindex_ram},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_addindex_disc},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_addindex_do},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delindex_ram},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delindex_disc},
                  []}},
             {0,
              {ok,{mnesia_schema_recovery_test,
                      interrupted_after_delindex_do},
                  []}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_after_change_type_ram2disc},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_after_change_type_ram2do},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_after_change_type_disc2ram},
                  "Mnesia is not debug compiled, test case ignored.\n"}},
             {0,
              {skip,
                  {mnesia_schema_recovery_test,
                      interrupted_after_change_type_disc2do},
                                "Mnesia is not debug compiled, test case ignored.\n"}},
                           {0,
                            {skip,
                                {mnesia_schema_recovery_test,
                                    interrupted_after_change_type_do2ram},
                                "Mnesia is not debug compiled, test case ignored.\n"}},
                           {0,
                            {skip,
                                {mnesia_schema_recovery_test,
                                    interrupted_after_change_type_do2disc},
                                "Mnesia is not debug compiled, test case ignored.\n"}},
                           {0,
                            {skip,
                                {mnesia_schema_recovery_test,
                                    interrupted_after_change_type_other_node},
                                "Mnesia is not debug compiled, test case ignored.\n"}},
                           {0,
                            {skip,
                                {mnesia_schema_recovery_test,
                                    interrupted_after_change_schema_type},
                                "Mnesia is not debug compiled, test case ignored.\n"}}]}]}]}]}},
                  {0,
                   {{mnesia_recovery_test,{group,async_dirty}},
                    [{0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_pre_kill_part},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_pre_kill_coord_node},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_pre_kill_coord_pid},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_post_kill_part},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_post_kill_coord_node},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,async_dirty_post_kill_coord_pid},
                          "Mnesia is not debug compiled, test case ignored.\n"}}]}},
                  {0,
                   {{mnesia_recovery_test,{group,sync_dirty}},
                    [{0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_pre_kill_part},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_pre_kill_coord_node},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_pre_kill_coord_pid},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_post_kill_part},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_post_kill_coord_node},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
                     {0,
                      {skip,
                          {mnesia_recovery_test,sync_dirty_post_kill_coord_pid},
                          "Mnesia is not debug compiled, test case ignored.\n"}}]}},
                  {0,
                   {{mnesia_recovery_test,{group,sym_trans}},
                    [{0,
                      {skip,
                          {mnesia_recovery_test,
                              sym_trans_before_commit_kill_coord_node},
                          "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_before_commit_kill_coord_pid},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_before_commit_kill_part_after_ask},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_before_commit_kill_part_before_ask},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_after_commit_kill_coord_node},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_after_commit_kill_coord_pid},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_after_commit_kill_part_after_ask},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_after_commit_kill_part_do_commit_pre},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,
                        sym_trans_after_commit_kill_part_do_commit_post},
                    "Mnesia is not debug compiled, test case ignored.\n"}}]}},
             {0,
              {{mnesia_recovery_test,{group,asym_trans}},
               [{0,
                {skip,
                    {mnesia_recovery_test,asymtrans_part_ask},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_part_commit_vote},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_part_pre_commit},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_part_log_commit},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_part_do_commit},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_coord_got_votes},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_coord_pid_got_votes},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_coord_log_commit_rec},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                    {mnesia_recovery_test,asymtrans_coord_pid_log_commit_rec},
                    "Mnesia is not debug compiled, test case ignored.\n"}},
               {0,
                {skip,
                {mnesia_recovery_test,asymtrans_coord_log_commit_dec},
                "Mnesia is not debug compiled, test case ignored.\n"}},
           {0,
            {skip,
                {mnesia_recovery_test,asymtrans_coord_pid_log_commit_dec},
                "Mnesia is not debug compiled, test case ignored.\n"}},
           {0,
            {skip,
                {mnesia_recovery_test,
                    asymtrans_coord_rec_acc_pre_commit_log_commit},
                "Mnesia is not debug compiled, test case ignored.\n"}},
           {0,
            {skip,
                {mnesia_recovery_test,
                    asymtrans_coord_pid_rec_acc_pre_commit_log_commit},
                "Mnesia is not debug compiled, test case ignored.\n"}},
           {0,
            {skip,
                {mnesia_recovery_test,
                    asymtrans_coord_rec_acc_pre_commit_done_commit},
                "Mnesia is not debug compiled, test case ignored.\n"}},
           {0,
            {skip,
                {mnesia_recovery_test,
                    asymtrans_coord_pid_rec_acc_pre_commit_done_commit},
                "Mnesia is not debug compiled, test case ignored.\n"}}]}},
        {0,{skip,{mnesia_recovery_test,after_full_disc_partition},'NYI'}},
        {116,
         {{mnesia_recovery_test,{group,after_corrupt_files}},
          [{10,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_decision_log_head},
                []}},
           {10,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_decision_log_tail},
                []}},
           {10,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_latest_log_head},
                []}},
           {10,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_latest_log_tail},
                []}},
           {22,
            {ok,{mnesia_recovery_test,after_corrupt_files_table_dat_head},
                []}},
           {10,
            {ok,{mnesia_recovery_test,after_corrupt_files_table_dat_tail},
                []}},
           {22,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_schema_dat_head},
                []}},
           {22,
            {ok,{mnesia_recovery_test,
                    after_corrupt_files_schema_dat_tail},
                []}}]}},
        {1,{ok,{mnesia_recovery_test,disc_less},[]}},
        {2,{ok,{mnesia_recovery_test,garb_decision},[]}},
        {0,{skip,{mnesia_recovery_test,system_upgrade},'NYI'}}]}}]}},
  {559,
   {{mnesia_SUITE,{group,consistency}},
    [{559,
      {{mnesia_consistency_test,all},
       [{83,
         {{mnesia_consistency_test,{group,consistency_after_restart}},
          [{13,
            {ok,{mnesia_consistency_test,consistency_after_restart_1_ram},
                []}},
           {13,
            {ok,{mnesia_consistency_test,
                    consistency_after_restart_1_disc},
                []}},
           {13,
            {ok,{mnesia_consistency_test,
                    consistency_after_restart_1_disc_only},
                []}},
           {13,
    {ok,{mnesia_consistency_test,consistency_after_restart_2_ram},
        []}},
   {14,
    {ok,{mnesia_consistency_test,
            consistency_after_restart_2_disc},
        []}},
   {14,
    {ok,{mnesia_consistency_test,
            consistency_after_restart_2_disc_only},
        []}}]}},
{23,
 {{mnesia_consistency_test,{group,consistency_after_dump_tables}},
  [{11,
    {ok,{mnesia_consistency_test,
            consistency_after_dump_tables_1_ram},
        []}},
   {12,
    {ok,{mnesia_consistency_test,
            consistency_after_dump_tables_2_ram},
        []}}]}},
{63,
 {{mnesia_consistency_test,{group,consistency_after_add_replica}},
  [{10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_2_ram},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_2_disc},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_2_disc_only},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_3_ram},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_3_disc},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_add_replica_3_disc_only},
        []}}]}},
{63,
 {{mnesia_consistency_test,{group,consistency_after_del_replica}},
  [{10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_2_ram},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_2_disc},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_2_disc_only},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_3_ram},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_3_disc},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            consistency_after_del_replica_3_disc_only},
        []}}]}},
{65,
 {{mnesia_consistency_test,
      {group,consistency_after_move_replica}},
  [{10,
    {ok,{mnesia_consistency_test,
            consistency_after_move_replica_2_ram},
        []}},
   {10,
      {ok,{mnesia_consistency_test,
              consistency_after_move_replica_2_disc},
          []}},
     {11,
      {ok,{mnesia_consistency_test,
              consistency_after_move_replica_2_disc_only},
          []}},
     {10,
      {ok,{mnesia_consistency_test,
              consistency_after_move_replica_3_ram},
          []}},
     {10,
      {ok,{mnesia_consistency_test,
              consistency_after_move_replica_3_disc},
          []}},
     {11,
      {ok,{mnesia_consistency_test,
              consistency_after_move_replica_3_disc_only},
          []}}]}},
  {3,
   {{mnesia_consistency_test,
        {group,consistency_after_transform_table}},
    [{1,
      {ok,{mnesia_consistency_test,
              consistency_after_transform_table_ram},
          []}},
     {0,
      {ok,{mnesia_consistency_test,
              consistency_after_transform_table_disc},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_transform_table_disc_only},
          []}}]}},
  {0,
   {skip,
       {mnesia_consistency_test,
           consistency_after_change_table_copy_type},
       'NYI'}},
  {8,
   {{mnesia_consistency_test,{group,consistency_after_restore}},
    [{1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_clear_ram},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_clear_disc},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_clear_disc_only},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_recreate_ram},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_recreate_disc},
          []}},
     {1,
      {ok,{mnesia_consistency_test,
              consistency_after_restore_recreate_disc_only},
          []}}]}},
  {0,
   {skip,
       {mnesia_consistency_test,consistency_after_rename_of_node},
       'NYI'}},
  {248,
   {{mnesia_consistency_test,
        {group,checkpoint_retainer_consistency}},
    [{141,
      {{mnesia_consistency_test,
           {group,updates_during_checkpoint_activation}},
       [{15,
         {ok,{mnesia_consistency_test,
                 updates_during_checkpoint_activation_1_ram},
             []}},
        {15,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_1_disc},
        []}},
   {15,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_1_disc_only},
        []}},
   {15,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_2_ram},
        []}},
   {15,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_2_disc},
        []}},
   {15,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_2_disc_only},
        []}},
   {16,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_3_ram},
        []}},
   {16,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_3_disc},
        []}},
   {16,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_activation_3_disc_only},
        []}}]}},
{32,
 {{mnesia_consistency_test,
      {group,updates_during_checkpoint_iteration}},
  [{10,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_iteration_2_ram},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_iteration_2_disc},
        []}},
   {10,
    {ok,{mnesia_consistency_test,
            updates_during_checkpoint_iteration_2_disc_only},
        []}}]}},
{3,
 {{mnesia_consistency_test,
      {group,load_table_with_activated_checkpoint}},
  [{1,
    {ok,{mnesia_consistency_test,
            load_table_with_activated_checkpoint_ram},
        []}},
   {1,
    {ok,{mnesia_consistency_test,
            load_table_with_activated_checkpoint_disc},
        []}},
   {1,
    {ok,{mnesia_consistency_test,
            load_table_with_activated_checkpoint_disc_only},
        []}}]}},
{0,
 {{mnesia_consistency_test,
      {group,add_table_copy_to_table_checkpoint}},
  [{0,
    {ok,{mnesia_consistency_test,
            add_table_copy_to_table_checkpoint_ram},
        []}},
   {0,
    {ok,{mnesia_consistency_test,
            add_table_copy_to_table_checkpoint_disc},
        []}},
   {0,
    {ok,{mnesia_consistency_test,
            add_table_copy_to_table_checkpoint_disc_only},
        []}}]}},
{71,
 {{mnesia_consistency_test,{group,consistency_after_fallback}},
  [{10,
    {ok,{mnesia_consistency_test,
                     consistency_after_fallback_2_ram},
                 []}},
            {10,
             {ok,{mnesia_consistency_test,
                     consistency_after_fallback_2_disc},
                 []}},
            {13,
             {ok,{mnesia_consistency_test,
                     consistency_after_fallback_2_disc_only},
                 []}},
            {11,
             {ok,{mnesia_consistency_test,
                     consistency_after_fallback_3_ram},
                 []}},
            {11,
             {ok,{mnesia_consistency_test,
                     consistency_after_fallback_3_disc},
                 []}},
            {14,
             {ok,{mnesia_consistency_test,
                     consistency_after_fallback_3_disc_only},
                 []}}]}}]}},
      {0,
       {{mnesia_consistency_test,{group,backup_consistency}},
        [{0,
          {{mnesia_consistency_test,
               {group,interupted_install_fallback}},
           [{0,
             {skip,
                 {mnesia_consistency_test,inst_fallback_process_dies},
                 "Mnesia is not debug compiled, test case ignored.\n"}},
            {0,
             {skip,
                 {mnesia_consistency_test,fatal_when_inconsistency},
                 "Mnesia is not debug compiled, test case ignored.\n"}}]}},
         {0,
          {{mnesia_consistency_test,
               {group,interupted_uninstall_fallback}},
           [{0,
             {skip,
                 {mnesia_consistency_test,after_delete},
                 "Mnesia is not debug compiled, test case ignored.\n"}}]}},
         {0,
          {{mnesia_consistency_test,
               {group,mnesia_down_during_backup_causes_switch}},
           [{0,
             {skip,
                 {mnesia_consistency_test,cause_switch_before},
                 "Mnesia is not debug compiled, test case ignored.\n"}},
            {0,
             {skip,
                 {mnesia_consistency_test,cause_switch_after},
                 "Mnesia is not debug compiled, test case ignored.\n"}}]}},
         {0,
          {{mnesia_consistency_test,
               {group,mnesia_down_during_backup_causes_abort}},
           [{0,
             {skip,
                 {mnesia_consistency_test,cause_abort_before},
                 "Mnesia is not debug compiled, test case ignored.\n"}},
            {0,
             {skip,
                 {mnesia_consistency_test,cause_abort_after},
                 "Mnesia is not debug compiled, test case ignored.\n"}}]}},
         {0,
          {{mnesia_consistency_test,
               {group,schema_transactions_during_backup}},
           [{0,
             {skip,
                 {mnesia_consistency_test,change_schema_before},
                 "Mnesia is not debug compiled, test case ignored.\n"}},
            {0,
             {skip,
                 {mnesia_consistency_test,change_schema_after},
                 "Mnesia is not debug compiled, test case ignored.\n"}}]}}]}}]}}]}},
{0,
 {{mnesia_SUITE,{group,majority}},
  [{0,
    {{mnesia_majority_test,all},
     [{0,{ok,{mnesia_majority_test,write},[]}},
          {0,{ok,{mnesia_majority_test,wread},[]}},
          {0,{ok,{mnesia_majority_test,delete},[]}},
          {0,{ok,{mnesia_majority_test,clear_table},[]}},
          {0,{ok,{mnesia_majority_test,frag},[]}},
          {0,{ok,{mnesia_majority_test,change_majority},[]}},
          {0,{ok,{mnesia_majority_test,frag_change_majority},[]}}]}]}]}},
    {0,
     {{mnesia_frag_test,{group,medium}},
      [{0,
        {skip,
            {mnesia_frag_test,consistency},
            "Not yet implemented (NYI).\n"}}]}]}]}},
 {2903,
  {{mnesia_SUITE,{group,heavy}},
   [{2903,
     {{mnesia_SUITE,{group,measure}},
      [{2903,
        {{mnesia_measure_test,all},
         [{2903,
           {{mnesia_measure_test,{group,benchmarks}},
            [{26,
              {{mnesia_measure_test,{group,meter}},
               [{5,{ok,{mnesia_measure_test,ram_meter},[]}},
                {6,{ok,{mnesia_measure_test,disc_meter},[]}},
                {13,{ok,{mnesia_measure_test,disc_only_meter},[]}}]}},
             {1,{ok,{mnesia_measure_test,cost},[]}},
             {1,{ok,{mnesia_measure_test,dbn_meters},[]}},
             {2874,
              {{mnesia_measure_test,{group,tpcb}},
               [{937,{ok,{mnesia_measure_test,ram_tpcb},[]}},
                {945,{ok,{mnesia_measure_test,disc_tpcb},[]}},
                {991,
                 {ok,{mnesia_measure_test,disc_only_tpcb},
                     []}}]}}]}}]}}]}}]}}]}},
 {0,{ok,{mnesia_SUITE,clean_up_suite},[]}}]}]}.
```