Main Memory Database Instant Recovery

Arlino Magalhães[1], José Maria Monteiro[2], Ângelo Brayner[2]

[1]Curso de Gestão de Dados
Universidade Federal do Piauí (UFPI)
Campus Ministro Petrônio Portella – Teresina – PI

[2]Departamento de Computação
Universidade Federal do Ceará (UFC)
Campus do Pici – Fortalza – CE

arlino@ufpi.edu.br, {brayner,monteiro}@dc.ufc.br

Abstract. *Main Memory Databases (MMDBs) technology handles the primary database in Random Access Memory (RAM) to provide high throughput and low latency. However, volatile memory makes MMDBs much more sensitive to system failures. The contents of the database are lost in these failures. As a result, systems may be unavailable for a long time until the database recovery process has been finished. Therefore, novel recovery techniques are needed to repair crashed MMDBs as quickly as possible. This thesis presents MM-DIRECT, a recovery technique that enables MMDBs to schedule transactions immediately after the system startup. The approach also implements a tuple-level consistent checkpoint to reduce the recovery time. To validate the proposed approach, experiments were performed in a prototype implemented on the Redis database. The results show that the proposed instant recovery technique effectively provides high transaction throughput rates even during the recovery process and normal database processing.*

1. Introduction

Several current application scenarios, such as trading, real-time bidding, advertising, weather forecasting, social gaming, etc., require massive real-time data processing. MMDBs have proved to be an efficient alternative to such applications. MMDBs keep the database in RAM to achieve very high IOPS (Input/Output Operations Per Second) rates. Such a feature makes MMDBs much more sensitive to system failures since it causes loss of main memory content [Magalhães et al. 2021a, Wu et al. 2017, Faerber et al. 2017].

In most MMDBs, the recovery process is performed offline. Thus, the database system becomes available to service new transactions only after the full recovery process is completed. One may claim that MMDBs may keep database replicas to ensure high availability. Nevertheless, the replication mechanism is not immune to errors and unpredictable defects in software and firmware. Besides, adding high availability infrastructure to a system can become expensive to deploy and maintain [Magalhães et al. 2021b, Magalhães 2021, Wu et al. 2017, Faerber et al. 2017].

This thesis presents an instant recovery mechanism for MMDBs, denoted ***MM-DIRECT*** **(Main Memory Database Instant RECovery with Tuple consistent checkpoint)**. *MM-DIRECT* implements a generic recovery approach, which can be embedded

in any main memory database system. Such an approach is able to schedule new transactions immediately after the system startup.

Figure 1 emphasizes the benefits of instant recovery (described in this thesis) concerning MMDB standard recovery (implemented by most existing MMDBs). Figure 1 represents the average transaction throughput over small time intervals. Looking more closely at Figure 1, one may observe that the standard recovery approach (represented by the orange line) has downtime after a failure while the database is recovering. On the other hand, the instant recovery approach (blue line) schedules transactions immediately after the system startup during the recovery process. Thus, applications and users do not notice the recovery process, giving the impression that the system was instantly restored. Since the proposed instant recovery approach schedules transactions as quickly as possible, it delivers higher IOPS rates, i.e., it executes workloads faster [Magalhães 2022].

Figure 1. Instant Recovery vs. Default Recovery [Magalhães 2022].

1.1. Thesis contributions

The main contributions of this thesis are the following:

- An instant recovery mechanism for MMDBs that can restore database tuples incrementally and on-demand.
- A very lightweight logging technique that can efficiently read/write log records in order to restore tuples individually without degrading transaction processing, or as little as possible.
- Two checkpoint techniques to reduce recovery time whose actions persist even if the checkpoint process does not complete (for a failure, for example).
- An analysis of the behavior of the proposed instant recovery mechanism through experiments in an OLTP workload.

2. Background

Most MMDBs implement a logical logging which records higher-level operations on secondary memory. Logical logging tends to be faster than physical logging (commonly used by disk-resident databases) during transaction processing since fewer items are recorded on the log file. MMDBs produce only Redo log records of modified data to reduce the amount of data written to secondary storage. The commit processing uses group commit, i.e., it tries to group multiple log records into one large I/O [Magalhães et al. 2021a, Stonebraker and Weisberg 2013, Faerber et al. 2017].

Most MMDBs produces a consistent checkpoint file equivalent to a materialized database state in an instant of time, commonly called snapshot. Whenever a system failure occurs in an MMDB, the primary copy of the database is lost. Thus, the recovery manager should load the last snapshot into memory and redo log records. The system can process new transactions only after complete recovery [Magalhães et al. 2021a, Stonebraker and Weisberg 2013, Faerber et al. 2017].

3. Related Work

Hekaton [Diaconu et al. 2013], VoltDB [Stonebraker and Weisberg 2013], HyPer [Funke et al. 2014], SAP HANA [Färber et al. 2012], and SiloR [Zheng et al. 2014] are examples of modern MMDBs that handle the recovery techniques described in Section 2.

PACMAN [Wu et al. 2017] and Adaptive Logging [Yao et al. 2016] utilize a dependency graph between transactions performed to identify opportunities for database recovery in parallel. Those systems must wait for the full database recovery to service new transactions.

Fineline [Sauer et al. 2018] implements a partitioned B-tree as a log file structure to provide persistence and instant recovery. During transaction processing, Fineline stores physical log records at commitment, i.e., transactions must wait writes in the tree. Physical records are not appropriated to MMDBs. After a crash, the system may search for multiple partitions to retrieve a page. Checkpoints are not implemented.

4. *MM-DIRECT*: An Instant Recovery Mechanism

Although modern hardware has offered promising alternatives for MMDBs to reach their full potential, this work does not require many technology improvements. In fact, the *MM-DIRECT* recovery requires a simple system with a memory hierarchy composed of two levels: main memory (for the database) and a persistent-memory level (for log files). Figure 2 shows the main components of *MM-DIRECT*. The next subsections discuss the main components of the architecture and their interactions.

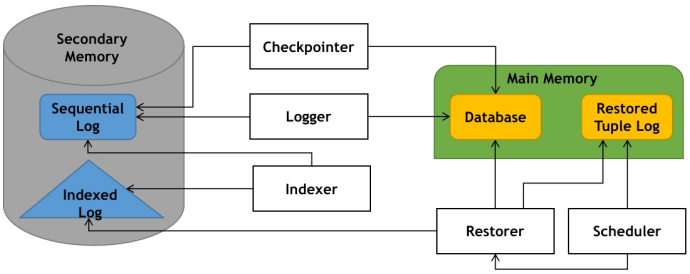

Figure 2. Architecture of the instant recovery mechanism.

4.1. The Logging Mechanism

MM-DIRECT works with a sequential log file and an indexed log file. Figure 3 depicts the structure of both files. During the execution of a given transaction Tx, log records are generated for each update action belonging to Tx and kept in a local thread, i.e., Tx performs updates locally in main memory, other transactions can see its updates only when Tx commits. During a transaction commit, the log records are flushed to the sequential

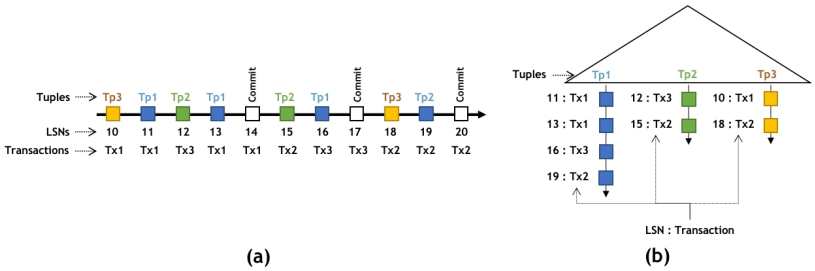

Figure 3. Sequential log (a), and indexed log (b).

log file. The Log Sequence Number (LSN) represents the order in which a record was stored. This scheme ensures log consistency to recover the database.

A sequential log does not provide the necessary support for on-demand recovery. Accordingly, after a failure, new transactions should wait for full database recovery to be executed. In order to overcome such a limitation, *MM-DIRECT* utilizes an indexed log file, which is structured as a B$^+$-tree. The database's tuple IDs are used as search keys. This way, log records for a given tuple Tp are stored in a list pointed by a single entry of a leaf node. Thus, a leaf node search key value *K* (a tuple ID) points to a list of log records for a tuple Tp with $ID = K$. The records are copied to the indexed log in the same format that they were stored in the sequential log. A single index seek operation in the B$^+$-tree can locate all the records required to restore Tp. LSN is responsible for representing the temporal order of log records belonging to a tuple Tp. The indexed log loses the global LSN order, but it maintains a local LSN order in each B$^+$-tree leaf node. Log records for a tuple in a leaf node are still sorted by LSN. This log organization ensures the consistent recovery of each tuple. In this way, the B$^+$-tree allows to recovery each tuple individually. In addition, *MM-DIRECT* supports other index structures, such as a hash table.

During normal transaction processing, transaction update records are appended to the sequential log file at the commitment. The Indexer component monitors entries in the sequential log file and inserts them into the indexed log. This process is asynchronous to the transaction commit. Such an asynchronous indexing process avoids a negative impact on transaction throughput.

To illustrate how both log files are built, consider Figure 3. In Figure 3 (a), log records generated by transactions $Tx1$, $Tx2$, and $Tx3$ are stored in the sequential log file. Figure 3 (b) illustrates the assemblage of the indexed log file. Log records for transactions $Tx1$, $Tx2$, and $Tx3$ are grouped in the B$^+$-tree leaf nodes by tuple IDs. For instance, log records with LSN 12 and 15 belonging to $Tx3$ and $Tx2$, respectively, contain write operations executed on tuple $Tp2$.

Writing records to a sequential log file is potentially faster than doing so to an indexed log file. This is because the sequential access pattern in a sequential file can write records faster than the random pattern in a B$^+$-tree. Thus, the indexed log implements an in-memory buffering mechanism to deliver write bandwidth similar to the sequential log. These buffering mechanism is responsible for mitigating the log record writing rate difference between the indexed log file and the sequential log file. As a consequence, the log tail does not increase smoothly. In this work, we refer to the log tail as the number of records in the sequential log that have not yet been stored in the indexed log.

4.2. The Recovery Process

Whenever a system failure occurs, the recovery manager is initialized at system restart. Initially, the recovery manager checks if all records in the sequential log have been inserted into the indexed log file. This check is necessary because some records might not have been inserted into the B^+-tree before a failure. This is because the indexed log file insertion process runs asynchronously to the sequential log file insertion process. Thus, before starting the recovery process, the Indexer component must insert those records into the indexed log file to ensure recovery consistency. Due to the buffering mechanism implemented in the indexed log, the log tail does not grow very large. Thus, the number of records in the sequential log to be inserted into the indexed log before starting the recovery should not be too large.

Immediately after the aforementioned recovery action, the Restorer component triggers incremental recovery by traversing the B^+-tree. Each visit to an entry in the B^+-tree leaf node retrieves the log records to redo a tuple completely. As soon as a tuple is restored into memory, it is set as restored in the Restored Tuple Log component. Restored Tuple Log is an in-memory data structure (see Figure 2).

Recovery by an indexed log supports availability naturally since tuples can be accessed by transactions immediately after being restored to memory. However, transactions can request tuples that have not yet been restored to memory. Thus, in parallel to the recovery process, the Scheduler component schedules read and write operations. Such an ability is supported by the following process. As soon as the Scheduler receives an operation on a tuple Tp, it reads the Restored Tuple Log to identify if Tp has been already restored. If the answer is affirmative, the database operation on Tp is scheduled. Otherwise, the Scheduler requests the Restorer to redo Tp on-demand. This can be done by an index seek operation to look up for a search key value in the indexed log file equal to the Tp's ID.

4.3. The Tuple Consistent Checkpoint

The indexed log is a growing file. The more records stored in the log the more records to redo during database recovery, consequently increasing database recovery time. *MM-DIRECT* implements a novel checkpoint strategy - the Tuple Consistent Checkpoint (TuCC) - to overcome such a negative side-effect. The TuCC generation process scans the entire database at system runtime to obtain all the database tuples. For each tuple, the TuCC generates a new log record using the tuple contents and then flushes the record into the sequential log. Such a record is similar to the one generated by an update operation. Nonetheless, a log record generated by the checkpoint is not identified as a normal update record. It is marked with a different log record type attribute. This record is called tuple checkpoint record.

Observe that records inserted into the log before a tuple checkpoint record are no longer needed. Therefore, whenever the Indexer finds a tuple checkpoint record r for a tuple Tp in the sequential file, all records in B^+-tree node N related to Tp are removed before the Indexer inserts r in N since r reflects the state of Tp at the time it was entered into the sequential log.

In contrast to other checkpoint techniques (such as Fuzzy and Snapshot), TuCC does not lose any changes made if the checkpoint process is interrupted. This is possible

since this technique handles each leaf node individually, i.e., the checkpoint of one tuple does not interfere with the others. After all tuple checkpoint records are generated, the indexed log file reflects a database snapshot.

In TuCC algorithm, the entire database should be read to generate a checkpoint, even non-updated tuples. Such an technique may cause an unnecessary effort to perform a checkpoint in several non-updated or low-frequency updated tuples. Thus, *MM-DIRECT* implements another checkpoint technique, which is only applied for the most frequently used (MFU) tuples, denoted Tuple Consistent Checkpoint for MFU tuples (TuCC-MFU). During normal transaction processing, the system stores tuple access information. TuCC-MFU uses tuple access information to know the MFU tuples and performs an algorithm similar to TuCC, but only for MFU tuples instead of the entire database.

5. Experimental Evaluation

This section presents the main experiments conducted in order to assess the potential presented by *MM-DIRECT*. All experiments have been executed in 4-worker threads on an Intel Core i7-9700k CPU 3.60GHz x 8 machine, with 64GB RAM and 400GB SSD. The operating system was Ubuntu Linux 18.04.2 LTS.

The key goal was to compare the *MM-DIRECT*recovery to the traditional MMDB recovery. However, we also tested our instant recovery scheme in different scenarios to confirm the following expectations: (1) an indexed log must be employed to incrementally and on-demand recover the database, and (2) the asynchronous indexing of log records must be adopted to avoid transaction processing overhead. Thus, the experiments have been conducted in the three following scenarios: *(i)* Standard recovery using sequential log (SRSL); *(ii)* Instant recovery using asynchronous indexed log record insertion (IRAIL); and *(iii)* Instant recovery using synchronous indexed log record insertion (IRSIL).

In SRSL (traditional recovery), log records are written to a sequential log file during transaction processing. After a failure, the system must scan the entire log file. In IRAIL (our approach), log records are written to a sequential log during transaction processing, and stored in an indexed log asynchronously to the transaction commit. In IRSIL (scenario derived from IRAIL), log records are written directly to an indexed log during transaction processing. After a failure, for both scenarios *ii* (IRAIL) and *iii* (IRSIL), the system must traverse the B^+-tree. The IRSIL scenario was created to measure the log indexing overhead during transaction processing and instant recovery processing.

The workloads was simulated using the Memtier Benchmark [Memtier Benchmark 2020] which operated 4 worker threads, with each thread driving 50 clients. Each client submits $500,000$ requests. The requests randomly accessed 20% of the database in the 5:5 ratio between read and write operations. The database has been yielded through 20 runs of that workload. The resulting database contains $5x10^5$ tuples, with a 62.7GB sequential log file, which corresponds to $1.8x10^9$ log records. The indexed log file has been constructed from the sequential log by the Indexer component of *MM-DIRECT*. Memtier is a high-throughput benchmarking tool for Redis.

The evaluation prototype has been implemented in Redis 5.0.7 and can be down-

loaded[1]. Redis ensures database persistence similar to the one implemented by modern MMDBs. The logging technique is logical and implements a sequential log file (AOF). Periodically, database snapshots are flushed to the secondary storage. However, snapshot generation have been disabled to capture the scenario of the recovery process on very large log files. The implemented prototype utilizes Redis' AOF to represent the sequential log file required by *MM-DIRECT*. To build the indexed log file, *MM-DIRECT* utilizes the Berkeley DB B$^+$-tree library [Olson et al. 1999].

5.1. Recovery Experiments

The experiments were focused on measuring the transaction throughput, database recovery time, and logging overhead. In this sense, for each scenario, the same workload is submitted to the prototype using the same initial database during normal system processing. After 10 minutes (600 seconds) the database system has been shut down to simulate a system failure. At database restart, the workload is submitted again.

Figure 4 depicts the results of recovery experiments. The vertical dashed red line indicates the crash time. The other vertical dashed lines indicate the final recovery time of the three scenarios: SRSL (orange line), IRAIL (blue line), and IRSIL (green line).

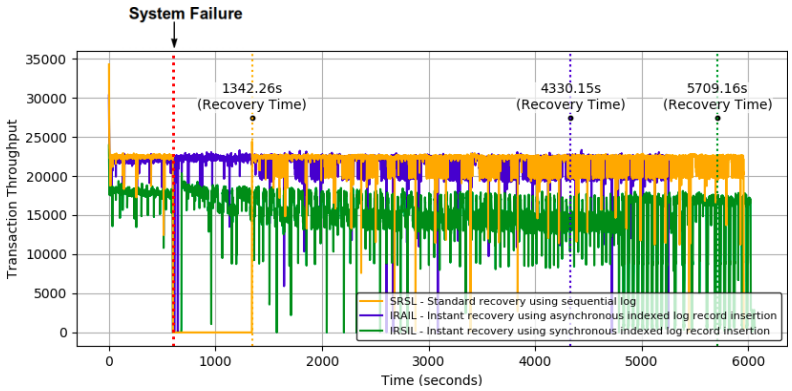

Figure 4. Transaction throughput during the recovery process.

In Figure 4, one may observe that *MM-DIRECT* (IRAIL scenario) has executed the submitted workload faster than the conventional main memory recovery approach (SRSL scenario). Such a result shows that *MM-DIRECT* provides high data availability during the database recovery. On the other hand, although in SRSL the database was restored before the IRAIL, SRSL presented long downtime after failures. This result was already expected because recovery in SRSL deals with a sequential log, which is potentially faster than manipulating an indexed log (in IRAIL).

The experiments also reveal that the *MM-DIRECT* approach does not overload transaction processing. Notice that in Figure 4 transaction throughput in IRAIL is similar to the one provided by the MMDB standard recovery (SRSL). This result was expected as well because the *MM-DIRECT* and standard recovery strategy flush log records to secondary memory in a similar manner. One may claim that *MM-DIRECT* needs to insert additional records into the indexed log, which could negatively impact MMDB throughput. However, record insertion into the indexed log file occurs asynchronously to transaction commit. Observe that in IRSIL, insertion into the indexed log file is executed

[1]https://drive.google.com/drive/folders/1LTbtY36O0kWIpxZBM-hc1BPvIjICuy2F?usp=sharing

synchronously to transaction commit, which overloaded transaction processing. In such a case, a transaction must wait for indexed log insertion to commit.

There are additional data on the experiments, which are not depicted in Figure 4. *MM-DIRECT* recovered $411,750$ tuples incrementally and $88,250$ tuples on demand. Tuples restored in an on-demand way represent 88% of data accessed by the workload during the database recovery. Before starting the recovery process, *MM-DIRECT* presented a very short downtime of 0.0079 seconds to insert 3,022 records into the indexed log file. These records were not inserted before the crash since inserts to the indexed log are performed asynchronously to transaction commit. In fact, there were few records to be inserted into the indexed log file. This is due to the fact that the log tail does not grow much due to the indexed log buffer mechanism implemented by *MM-DIRECT* (see Sections 4.1 and 4.2).

Transaction latency is the time delay between the request of a read/write operation and its effect on the database. *MM-DIRECT* implements on-demand recovery. For that reason, the latency of a transaction that requests on-demand recovery can be increased by an additional time Δ, denoted *restore latency*. Recall that during on-demand recovery, the Scheduler requests the Restorer to redo a given tuple Tp, which demands an index seek operation for looking up a search key value equal to Tp's ID in the indexed log file. Thus, restore latency only affects transactions with on-demand recovery requests. The next experiments investigate the impact of restore latency during database recovery.We have measured recovery latency in *MM-DIRECT* (IRAIL scenario) and default recovery (SRSL).

Figure 5 depicts the time series of average transaction latency before, during, and after the recovery process. Analyzing Figure 5, one may notice that before the failure, the average transaction latency is quite similar (close to zero) for both scenarios. After ten minutes of normal transaction processing, the database system has been shut down to simulate a system failure. The immediate effect is that the average transaction latency spikes up in the scenario using *MM-DIRECT* (blue lines). The average transaction latency before failure and after database recovery were 0.0898 and 0.0867 microseconds, respectively. The latency during the recovery was 0.9908 microseconds. The highest average latency was 2.4151 microseconds.

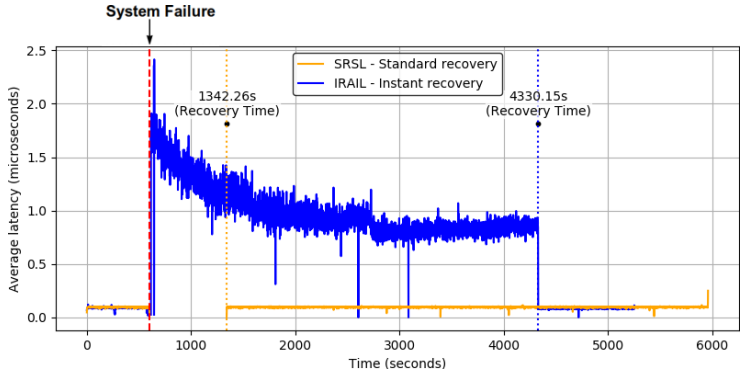

Figure 5. Average transaction latency.

The highest average transaction latencies are concentrated at the beginning of the recovery process. This delay occurs because data used by those transactions had to be recovered on-demand, i.e., it was necessary to access the indexed log in secondary memory.

Once those data have been restored, subsequent accesses to them occur directly on the main memory. Therefore, average latency gradually decreases until the database recovery process finishes when average latency becomes similar to before the failure.

There is no transaction latency measurement during recovery in the standard recovery scenario (orange lines) because there is no transaction processing. Recovery time could be considered as part of the latency time of transactions executed after the failure, as these transactions must wait for the database downtime to start executing. In this experiment, the database recovery time (1342.26 seconds) corresponded to a very long wait time for transactions to start executing. The average transaction latency throughout the default recovery scenario was 0.0957 microseconds. Besides, transaction latency before failure and after database recovery were 0.0952 and 0.0958 microseconds, respectively.

5.2. Checkpoint Experiments

The experiments presented in this section measured checkpoint efficiency and overhead (provoked by checkpoint generation). Thus, the following experiments have been executed: TuCC, TuCC-MFU, and no checkpoint. For each of these experiments, a workload has been submitted to the database system. After 4,500 seconds, the system is shut down. This timeframe was chosen to assure that at least one checkpoint had been produced before the failure. At the database restart, the workload is submitted again, as soon as the recovery process is triggered.

Figure 6 compares the TuCC checkpoint, TuCC-MFU checkpoint, and no-checkpoint. The transaction throughput rates are similar in the three experiments due to the checkpoint generation process does not interfere with transaction throughput rates. As expected, the database recovery process is faster with the proposed checkpoint techniques. TuCC-MFU was the most efficient checkpoint (faster database recovery) since it only checkpoints the most frequently updated tuples, whereas TuCC checkpoints all tuples in the database, including tuples that were not updated.

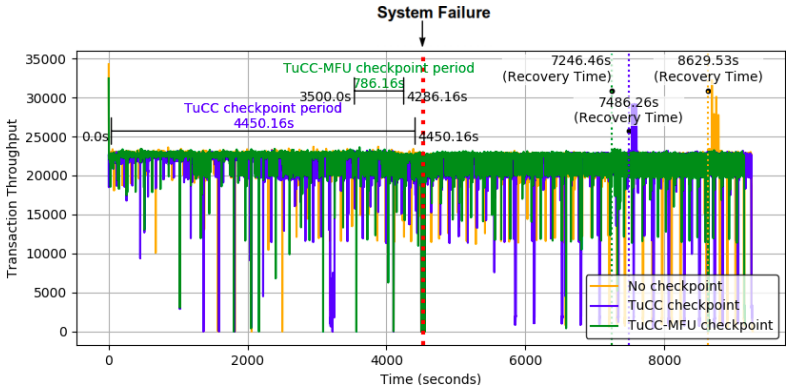

Figure 6. Checkpoint experiments.

5.3. Log Files' Write Bandwidth Experiments

Asynchronous insertion of records in the indexed log is the key to *MM-DIRECT* achieving high transaction throughput rates even during database recovery, as it is shown in Section 5.1. The sequential log write operation is potentially faster than indexed log write, as discussed in Section 4.1. Thus, the number of write operations executed per second (write

bandwidth) in a sequential log file is much higher than in the indexed log file (with a B^+-tree structure). In this sense, the experiments presented in this section measure and compare write bandwidth in the sequential log file and the indexed log file.

In Figure 7, the orange and blue lines represent the write bandwidth in the sequential log and the indexed log, respectively. Observe that both bandwidths are quite similar. Such a result is a consequence of the buffer mechanism implemented by *MM-DIRECT* (see Section 4.1). Thus, the proposed buffer mechanism mitigates the problem of a potential low write bandwidth in the indexed log file. The two lines tend downwards, i.e., the number of writes decreases with time. This is due to the behavior of the benchmark that performed more writes at the beginning of the experiments.

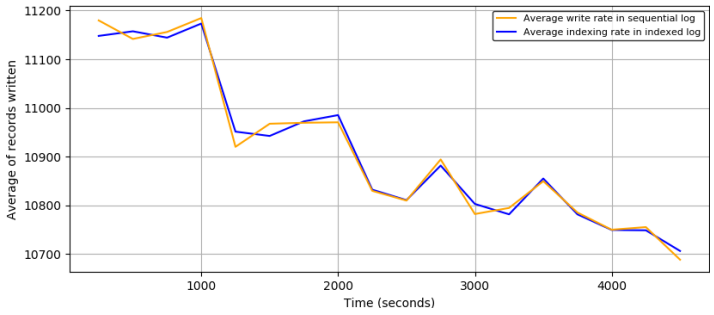

Figure 7. Write bandwidth in sequential and indexed log files.

It is important to note that the indexed log behavior shown in this section avoids a long log tail and, consequently, a downtime after a system failure. A short log tail means that a few records must be inserted into the indexed log before the recovery process begins, as discussed in Section 4.2. Thus, a short time is spent before the instant recovery begins, i.e., downtime is insignificant, as shown in Section 5.1.

5.4. Scalability Experiments

We ran further experiments to observe the behavior and performance of *MM-DIRECT* by increasing the number of threads (4, 5, 6, and 8). As the number of worker threads increases, the number of clients increases and consequently the number of requests also increases. In this way, the workload increases as the number of threads grow. In these experiments, we measured the transaction throughput, recovery time, and CPU and memory usage to analyze the scalability of *MM-DIRECT*.

Figure 8 (a) presents the average transaction throughput obtained during the full test in each experiment. The results show that the transaction throughput grows with the number of threads used, i.e., the system does not overload even if the workload increases. Figure 8 (b) presents the average latency obtained during the recovery process in each experiment. The results show that the average latency values are close in all experiments. These experiments show that *MM-DIRECT* can effectively provide low latency and high transaction throughput rates.

Figures 8 (c) and 8 (d) results show that the recovery time and full workload execution time in the experiments are close. These results were already expected because the system transaction throughput increased in each experiment, as shown in Figure 8 (a)

experiments. Besides, the increase in worker threads did not influence the latency of the system, as shown in Figure 8 (b) experiments.

Figures 8 (e) and 8 (f) show the average CPU and memory usage of the database system, respectively, obtained during the full test in each experiment. One may observe that the proposed recovery approach does not overload the CPU and memory of the system in any of the experiments.

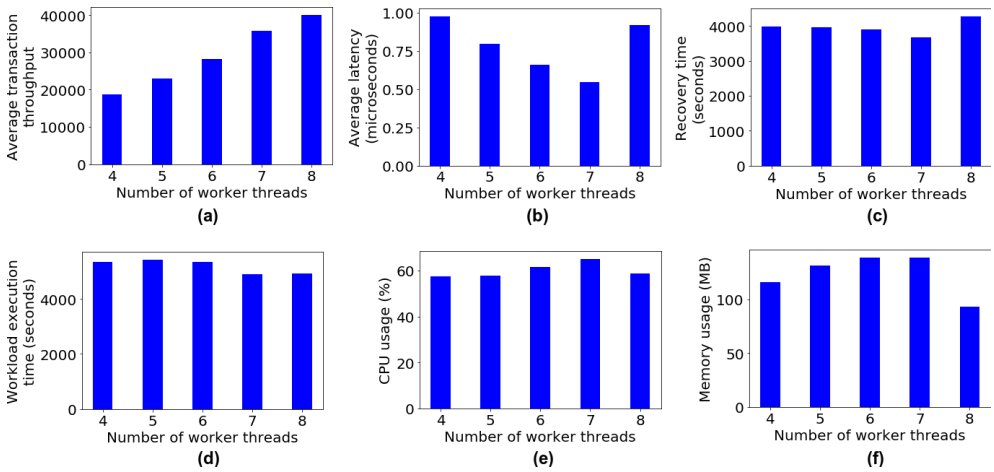

Figure 8. Scalability experiments: transaction throughput (a), latency (b), recovery time (c), workload execution time (d), CPU (e) and memory (f) usage.

6. Discussion

This section does a qualitative comparative analysis between *MM-DIRECT* and the related works discussed in Section 3. The related works need to recover the database completely so that new transactions can be executed, except Fineline which uses an instant recovery mechanism. For this reason, we will only compare Fineline with *MM-DIRECT*.

MM-DIRECT implements a logical logging technique in which logical records are flushed to a sequential log file at commitment. Besides, it implements an indexed log, through a B^+-tree, in which record writes are asynchronous to commit time, i.e., updates to the indexed log does not interfere in the transaction processing. In contrast, Fineline only implements a log file through a partitioned B-tree (indexed log). In Fineline, physical log records are flushed to the indexed log at commit time, i.e., a transaction must wait for updates to the indexed log to commit its writes. As discussed in Section 2, logical logging tends to be faster than physical logging during transaction processing. Moreover, as discussed in Section 4.1, record writes to a sequential log file is potentially faster than doing so to an indexed log file. Thus, the *MM-DIRECT* logging mechanism is much more lightweight than the Fineline logging technique. In addition, Fineline is limited to a partitioned B-tree as indexed log structure, while *MM-DIRECT* can use different index structures in the log, such as B^+-tree and Hash table.

MM-DIRECT is enable to retrieve all log records to restore a given tuple by only one search in the indexed log B^+-tree. A B^+-tree leaf node points to a list that contains only log records to restore a tuple completely. In Fineline, a indexed log B^+-tree node points to all partitions that contains the log records that updated a given page. Each

partition can contains update records of several pages. A partition is implemented as a flat-file. Thus, Fineline must traverse multiple partitions to restore a page, i.e., it must access multiple files. Besides, in each partition, the log records to restore that page must be probed by a flat-file index. In this way, accessing log records in *MM-DIRECT* is much simpler and potentially much faster than in Fineline.

The *MM-DIRECT* checkpoint technique reduces the number of log records in the indexed log in order to accelerate the recovery process. On the other hand, Fineline implements only a technique to merge partitions to provide acceptable read performance during recovery. However, recovery time tends to increase as the number of log records increases. Thus, *MM-DIRECT* checkpoint technique is effectively enable to reduce the recovery time, while the Fineline does not implement checkpoints. A checkpoint in Fineline would be a very expensive process, as the system would have to update multiple partitions for each page.

7. Thesis by-products

Database recovery is a critical task in any database system since it guarantees database consistency in the presence of failures. Nonetheless, in-memory database systems are not taught sufficiently in database courses or addressed in database textbooks. Thus, we produced a survey entitled ***Main Memory Database Recovery: a Survey*** to elucidate the main issues regarding in-memory database recovery. The survey was published in *ACM Computing Surveys* (CSUR, Qualis A1, Impact Factor: 14.324) [Magalhães et al. 2021a].
Link: `https://doi.org/10.1145/3442197`

The first version of our in-memeory-database recovery mechanism has been published in the paper ***Indexed Log File: Towards Main Memory Database Instant Recovery*** in the *24th International Conference on Extending Database Technology* (EDBT 2021, Qualis A1) [Magalhães et al. 2021b].
Link: `https://doi.org/10.5441/002/edbt.2021.34`

We presented our Ph.D. work in the paper ***Main Memory Databases Instant Recovery*** at the *VLDB Ph.D. Workshop* in the *47th International Conference on Very Large Data Bases* (VLDB 2021, Qualis A1) [Magalhães 2021].
Link: `http://ceur-ws.org/Vol-2971/paper10.pdf`

We presented the work ***Main Memory Database Recovery: a Survey*** that we published in *ACM Computing Surveys* as *Distinguished Published Paper* at the *36th Brazilian Symposium on Databases* (SBBD 2021, Qualis A4).
Link: `https://sbbd.org.br/2021/full-papers-ts/`

We have the tutorial ***Main memory database recovery strategies*** accepted to be presented at the *48th ACM SIGMOD/PODS International Conference on Management of Data* (SIGMOD 2023, Qualis A1) in June 2023 [Magalhaes et al. 2023].
Link: `https://doi.org/10.1145/3555041.3589402`

We presented the tutorial ***Main memory database recovery strategies*** at the *37th Brazilian Symposium on Databases* (SBBD 2022, Qualis A4) [Magalhães et al. 2022].
Link: `https://sol.sbc.org.br/index.php/sbbd_estendido/article/view/21861`

We produced and presented the work ***Big Data Management and Processing with***

In-Memory Databases at the *1st Latin American Computer Update Journey* (Jolai 2018) that was a satellite event of the *44th Latin American Computer Conference* (CLEI 2018, Qualis B1) [Magalhães et al. 2018a].
Link: `https://sol.sbc.org.br/index.php/jolai_clei/`

We produced and presented the work ***In-Memory Database Management Systems*** at the *14th Brazilian Symposium on Information Systems* (SBSI 2018, Qualis A4) [Magalhães et al. 2018b].
Link: `https://www.ucs.br/site/midia/arquivos/topicos-sistema-informacao.pdf`

A prototype has been developed to evaluate the feasibility of the recovery mechanism proposed in this thesis. The evaluation prototype has been implemented in Redis (REmote DIctionary Server) 5.0.7 and can be downloaded. We also produce a manual in English and Portuguese versions. The manual serves as a guide for operating the prototype and redoing all the experiments in the thesis.
Link: `https://drive.google.com/drive/folders/1LTbtY36O0kWIpxZBM-hc1BPvIjICuy2F?usp=sharing`

The conferences and the journal in which we published and presented our thesis are recognized as very relevant in the database area. The best in the world in the area. Very few Brazilian researchers managed to publish in ACM Computing Surveys and presented works at the VLDB conference. Furthermore, so far as we know, our group is the first Brazilian group to have a tutorial accepted at SIGMOD conference tutorial track. Finally, it is important to highlight that we have relevant citations. We received a citation from a Huawei research group at the VLDB conference [Lee et al. 2022].

8. Future Works

We intend to evaluate *MM-DIRECT* performance by submitting different workload patterns on the system during normal database execution and database recovery using different indexed log data structures. In this way, perhaps we may be able to identify whether a particular indexed log data structure is more appropriate for a particular workload pattern.

Another option of future works is partitioning the indexed log file in multiple files in order to reduce the impact of a log file corruption. By partitioning the log into multiple files, not all files should be corrupted if a log file corruption event happens.

Implement a parallel recovery technique to speed up the recovery process. The technique can process different log partitions through different threads, for example.

Implement a machine learning technique to chose the best checkpoint technique, for a given workload and database cardinality, in order to improve the system performance.

9. Conclusion

This document summarizes the contributions of the thesis [Magalhães 2022], which addresses the issues of providing main memory database instant recovery. This thesis presented *MM-DIRECT*, an instant recovery mechanism for MMDBs. *MM-DIRECT* allows new transactions to run concurrently with the recovery process. It implements an indexed log file to speed up fetch operation to access log records to restore data. Furthermore, the

proposed recovery mechanism restores data in on-demand and incremental ways. *MM-DIRECT* implements two novel checkpoints (TuCC and TuCC-MFU). Besides reducing recovery time, both checkpoint techniques do not interfere with transaction throughput.

The experiment results performed in *MM-DIRECT* show that the proposed instant recovery reduces the perceived database repair time, as transactions can be performed as soon as the moment the system is restarted. The experiments also analyzed the impact of using an indexed log structure on transaction throughput rates, transaction latency, and checkpoint efficiency. The experiments reveal that *MM-DIRECT* does not overload the database system.

The contributions produced during research have been featured in several premier venues in the area in terms of scientific publications, short courses, tutorials, and presentations. In addition, we produced a prototype to evaluate the feasibility of the *MM-DIRECT* recovery. The prototype has manuals to guide the reproducibility of the thesis experiments.

References

Diaconu, C., Freedman, C., Ismert, E., Larson, P.-A., Mittal, P., Stonecipher, R., Verma, N., and Zwilling, M. (2013). Hekaton: Sql server's memory-optimized oltp engine. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*.

Faerber, F., Kemper, A., Larson, P.-Å., Levandoski, J., Neumann, T., Pavlo, A., et al. (2017). Main memory database systems. *Foundations and Trends® in Databases*.

Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., and Lehner, W. (2012). Sap hana database: data management for modern business applications. *ACM Sigmod Record*.

Funke, F., Kemper, A., Mühlbauer, T., Neumann, T., and Leis, V. (2014). Hyper beyond software: Exploiting modern hardware for main-memory database systems. *Datenbank-Spektrum*.

Lee, L., Xie, S., Ma, Y., and Chen, S. (2022). Index checkpoints for instant recovery in in-memory database systems. *Proc. VLDB Endow.*, 15(8):1671–1683.

Magalhães, A. (2021). Main memory databases instant recovery. In *Proceedings of the VLDB PhD Workshop*.

Magalhães, A. (2022). *Main memory database instant recovery*. PhD thesis, Federal University of Ceara, Brazil.

Magalhães, A., Brayner, A., and Monteiro, J. M. (2022). Main memory database recovery strategies. In *Anais Estendidos do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 175–180. SBC.

Magalhaes, A., Brayner, A., and Monteiro, J. M. (2023). Main memory database recovery strategies. In *SIGMOD/PODS '23: Companion of the 2023 International Conference on Management of Data*, pages 31–35.

Magalhães, A., Monteiro, J. M., and Brayner, A. (2018a). Gerenciamento e processamento de big data com bancos de dados em memória. In *I Jornada latino-americana de atualização em informática , JOLAI 2018, São Paulo, SP, Brazil, 2018*.

Magalhães, A., Monteiro, J. M., and Brayner, A. (2018b). Sistemas de gerenciamento de banco de dados em memória. In *XIV Simpósio Brasileiro de Sistemas de Informação , SBSI 2018, Caxias do Sul, RS, Brazil, 2018*.

Magalhães, A., Monteiro, J. M., and Brayner, A. (2021a). Main memory database recovery: A survey. *ACM Computing Surveys (CSUR)*.

Magalhães, A., Monteiro, J. M., Brayner, A., and Moraes, G. (2021b). Indexed log file: Towards main memory database instant recovery. In *EDBT*.

Memtier Benchmark (2020). Github - redislabs. https://github.com/RedisLabs/memtier_benchmark. Accessed: August 26, 2020.

Olson, M. A., Bostic, K., and Seltzer, M. I. (1999). Berkeley db. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191.

Sauer, C., Graefe, G., and Härder, T. (2018). Fineline: log-structured transactional storage and recovery. *Proceedings of the VLDB Endowment*.

Stonebraker, M. and Weisberg, A. (2013). The voltdb main memory dbms. *IEEE Data Eng. Bull.*

Wu, Y., Guo, W., Chan, C.-Y., and Tan, K.-L. (2017). Fast failure recovery for main-memory dbmss on multicores. In *Proceedings of the 2017 ACM International Conference on Management of Data*.

Yao, C., Agrawal, D., Chen, G., Ooi, B. C., and Wu, S. (2016). Adaptive logging: Optimizing logging and recovery costs in distributed in-memory databases. In *Proceedings of the 2016 International Conference on Management of Data*.

Zheng, W., Tu, S., Kohler, E., and Liskov, B. (2014). Fast databases with fast durability and recovery through multicore parallelism. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*.