Instant Restore After a Media Failure

Caetano Sauer[1(✉)], Goetz Graefe[2], and Theo Härder[1]

[1] TU Kaiserslautern, Kaiserslautern, Germany
{csauer,haerder}@cs.uni-kl.de
[2] Google, Madison, WI, USA
goetzg@google.com

Abstract. Media failures usually leave database systems unavailable for several hours until recovery is complete, especially in applications with large devices and high transaction volume. Previous work introduced a technique called single-pass restore, which increases restore bandwidth and thus substantially decreases time to repair. Instant restore goes further as it permits read/write access to any data on a device undergoing restore—even data not yet restored—by restoring individual data segments on demand. Thus, the restore process is guided primarily by the needs of applications, and the observed mean time to repair is effectively reduced from several hours to a few seconds.

This paper presents an implementation and evaluation of instant restore. The technique is incrementally implemented on a system starting with the traditional ARIES design for logging and recovery. Experiments show that the transaction latency perceived after a media failure can be cut down to less than a second. The net effect is that a few "nines" of availability are added to the system using simple and low-overhead software techniques.

1 Introduction

Advancements in hardware technology have significantly improved the performance of database systems over the last decade, allowing for throughput in the order of thousands of transactions per second and data volumes in the order of petabytes. Availability, on the other hand, has not seen drastic improvements, and the research goal postulated by Jim Gray in his ACM Turing Award Lecture of a system "unavailable for less than one second per hundred years" [12] remains an open challenge. Improvements in reliable hardware and data center technology have contributed significantly to the availability goal, but proper software techniques are required to not only avoid failures but also repair failed systems as quickly as possible. This is especially relevant given that a significant share of failures is caused by human errors and unpredictable defects in software and firmware, which are immune to hardware improvements [11]. In the context of database logging and recovery, the state of the art has unfortunately not changed much since the early 90's, and no significant advancements were achieved in the software front towards the availability goal.

M. Kirikova et al. (Eds.): ADBIS 2017, LNCS 10509, pp. 311–325, 2017.
DOI: 10.1007/978-3-319-66917-5_21

Instant restore is a technique for media recovery that drastically reduces mean time to repair by means of simple software techniques. It works by extending the write-ahead logging mechanism of ARIES [19] and, as such, can be incrementally implemented on the vast majority of existing database systems. The key idea is to introduce a different organization of the log archive to enable efficient on-demand, incremental recovery of individual data pages. This allows transactions to access recovered data from a failed device orders of magnitude faster than state-of-the-art techniques, all of which require complete restoration of the entire device before access to the application's working set is allowed.

The problem of inefficient media recovery in state-of-the-art techniques, including ARIES and its optimizations, can be attributed to two major deficiencies. First, the media recovery process has a very inefficient random access pattern, which in practice encourages excessive redundancy and frequent incremental backups—solutions that only alleviate the problem instead of eliminating it. The second deficiency is that the recovery process is not incremental and requires full recovery before any data can be accessed—on-demand schedules are not possible and there is no prioritization scheme to make most needed data available earlier. Previous work addressed the first problem with a technique called single-pass restore [24], while this paper focuses on the second one.

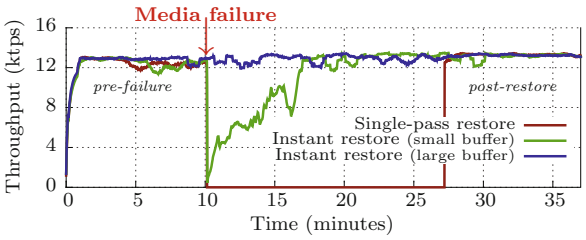

Fig. 1. Effect of instant restore (Color figure online)

The effect of instant restore is illustrated in Fig. 1, where transaction throughput is plotted over time and a media failure occurs after 10 min. In single-pass restore, as in ARIES, transaction processing halts until the device is fully restored (the red line in the chart), while instant restore continues processing transactions, using them to guide the restore process (blue and green lines). In a scenario where the application working set fits in the buffer pool (blue line), there is actually no visible effect on transaction throughput.

In the remainder of this paper, Sect. 2 describes related work, both previous work leading to the current design as well as competing approaches. Then, Sect. 3 describes the instant restore technique. Finally, Sect. 4 presents an empirical evaluation, while Sect. 5 concludes this paper.

A high-level description of instant restore was previously published in a book chapter [6] among other instant recovery techniques. The additional contribution here is a more detailed discussion of the design and implementation aspects as well as an empirical evaluation of the technique with an open-source prototype.

2 Related Work

2.1 Failure Classes and Assumptions

Database literature traditionally considers three classes of database failures [14], which are summarized in Table 1 (along with single-page failures, a fourth class to be discussed in Sect. 2.5). In the scope of this paper, it is important to distinguish between system and media failures, which are conceptually quite different in their causes, effects, and recovery measures. System failures are usually caused by a software fault or power loss, and what is lost—hence what must be recovered—is the state of the server process in main memory; this typically entails recovering page images in the buffer pool (i.e., "repeating history" [19]) as well as lists of active transactions and their acquired locks, so that they can be properly aborted. The process of recovery from system failures is called *restart*.

Table 1. Failure classes, their causes, and effects

Failure class	Loss	Typical cause	Response
Transaction	Single-transaction progress	Deadlock	Rollback
System	Server process (in-memory state)	Software fault, power loss	Restart
Media	Stored data	Hardware fault	Restore
Single page	Local integrity	Partial writes, wear-out	Repair

Instant restart [6] is an orthogonal technique that provides on-demand, incremental data access following a system failure. While the goals are similar, the design and implementation of instant restore require quite different techniques.

In a media failure, which is the focus here, a persistent storage device fails but the system might continue running, serving transactions that only touch data in the buffer pool or on other healthy devices. If a system and media failures happen simultaneously, or perhaps one as a cause of the other, their recovery processes are executed independently, and, by recovering pages in the buffer pool, the processes coordinate transparently.

The present work makes the same assumptions as most prior research on database recovery. The log and its archive copy reside on "stable storage", i.e., they are assumed to never fail. We consider failures on the database device only, i.e., the permanent storage location of data pages. Recovery from such failures requires a backup copy (possibly days or weeks old) of the lost device and all log records since the backup was taken; these may reside either in the active transaction log or in the log archive. The process of recovery from media failures is called *restore*. The following sections briefly describe previous restore methods.

2.2 ARIES Restore

Techniques to recover databases from media failures were initially presented in the seminal work of Gray [10] and later incorporated into the ARIES family

of recovery algorithms [19]. In ARIES, restore after a media failure first loads a backup image and then applies a redo log scan, similar to the redo scan of restart after a system failure. Figure 2 illustrates the process, which we now briefly describe. After loading full and incremental backups into the replacement device, a sequential scan is performed on the log archive and each update is replayed on its corresponding page in the buffer pool. A global *minLSN* value (called "media recovery redo point" by Mohan et al. [19]) is maintained on backup devices to determine the begin point of the log scan.

Because log records are ordered strictly by LSN, pages are read into the buffer pool in random order, as illustrated in the restoration of pages A and B in Fig. 2. Furthermore, as the buffer pool fills up, they are also written in random order into the replacement device, except perhaps for some minor degree of clustering. As the log scan progresses, evicted pages might be read again multiple times, also randomly. This mechanism is quite inefficient, especially for magnetic drives with high access latencies. Thus, it is no surprise that multiple hours of downtime are required in systems with high-capacity drives and high transaction rates [24].

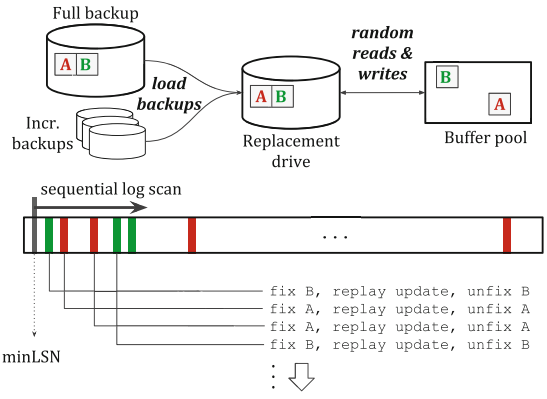

Fig. 2. Random access pattern of ARIES restore

Another fundamental limitation of the ARIES restore algorithm is that it is not incremental, i.e., pages cannot be restored to their most up-to-date version one-by-one and made available to running transactions incrementally. As shown in the example of Fig. 2, the last update to page A may be at the very end of the log; thus, page A remains out-of-date until almost the end of the log scan. Some optimizations may alleviate this situation (e.g., reusing checkpoint information), but there is no general mechanism for incremental restoration. Furthermore, even if pages could somehow be released incrementally when their last update is replayed, the hottest pages of the application working set are most likely to be released only at the very end of the log scan, and probably not even then, because they might contain updates of uncommitted transactions and thus require subsequent undo. This leads to yet another limitation of this approach: even if pages could be restored incrementally, there is no effective way to provide on-demand restoration, i.e., to restore most important pages first.

Despite a variety of optimizations proposed to the basic ARIES algorithm [19–21], none of them solves these problems in a general and effective manner. In summary, all proposed techniques that enable earlier access to recovered data items suffer from the same problem: early access is only provided for data for

which early access is not really needed—hot data in the application working set is not prioritized and most accesses must wait for complete recovery.

Finally, industrial database systems that implement ARIES recovery suffer from the same problems. IBM's DB2 speeds up log replay by sorting log records after restoring the backup and before applying the log records to the replacement database [13]. While a sorted log enables a more efficient access pattern, incremental and on-demand restoration is not provided. Furthermore, the delay imposed by the offline sort may be as high as the total downtime incurred by the traditional method. As another example, Oracle attempts to eliminate the overhead of reading incremental backups by incrementally maintaining a full backup image [22]. While this makes recovery slightly more efficient, it does not address the deficiencies discussed earlier.

2.3 Replication

Given the extremely high cost of media recovery in existing systems, replication solutions such as disk mirroring or RAID [2,3] are usually employed in practice to increase mean time to failure. However, it is important to emphasize that, from the database system's perspective, a failed disk in a redundant array does not constitute a media failure as long as it can be repaired automatically. Restore techniques aim to improve mean time to repair whenever a failure occurs that cannot be masked by lower levels of the system. Therefore, replication techniques can be seen largely as orthogonal to media restore techniques as implemented in database recovery mechanisms.

Nevertheless, a substantial reduction in mean time to repair, especially if done solely with simple software techniques, opens many opportunities to manage the trade-off between operational costs and availability. One option can be to maintain a highly-available infrastructure (with whatever costs it already requires) while availability is increased by deploying software with more efficient recovery. Alternatively, replication costs can be reduced (e.g., downgrading RAID-10 into RAID-5) while maintaining the same availability. Such level of flexibility, with solutions tackling both mean time to failure and mean time to repair, are essential in the pursuit of Gray's availability goal [12].

2.4 In-Memory Databases

Early work on in-memory databases focused mainly on restart after a system failure, employing traditional backup and log-replay techniques for media recovery [4,16]. The work of Levi and Silberschatz [17] was among the first to consider the challenge of incremental restart after a system failure. While an extension of their work for media recovery is conceivable, it would not address the efficiency problem discussed in Sect. 1. Thus, it would, in the best case and with a more complex algorithm, perform no better than the algorithm discussed later in Sect. 2.5.

Recent proposals for recovery on both volatile and non-volatile in-memory systems usually ignore the problem of media failures, employing the unspecific

term "recovery" to describe system restart only [1,18,23]. Therefore, recovery from media failures in modern systems either relies on the traditional techniques or is simply not supported, employing replication as the only means to maintain service upon storage hardware faults. As discussed above, while relying on replication is a valid solution to increase mean time to failure, a highly available system must also provide efficient repair facilities. In this aspect, traditional database system designs—using ARIES physiological logging and buffer management—provide more reliable behavior. Therefore, we believe that improving traditional techniques for more efficient recovery with low overhead on memory-optimized workloads is an important open research challenge.

2.5 Single-Page Repair

Single-page failures are considered a fourth class of database failures [8], along with the other classes summarized in Table 1. It covers failures restricted to a small set of individual pages of a storage device and applies online localized recovery to that individual page instead of invoking media recovery on the whole device. The single-page repair algorithm, illustrated in Fig. 3 (with backup and replacement devices omitted for simplification), has two basic requirements: first, the LSN of the most recent update of each page is known (i.e., the current PageLSN value) without having to access the page; second, starting from the most recent log record, the complete history of updates to a page can be retrieved. The former requirement can be provided with a page recovery index—a data structure mapping page identifiers to their most recent PageLSN value. Alternatively, the current PageLSN can be stored together with the parent-to-child node pointer in a B-tree data structure [9]. The latter requirement is provided by per-page log record chains, which are straight-forward to maintain using the PageLSN fields in the buffer pool.

In principle, single-page repair could be used to recover from a media failure, by simply repairing each page of the failed device individually. One advantage of this technique is that it yields incremental and on-demand restore, addressing the second deficiency of traditional media recovery algorithms mentioned in Sect. 1. To illustrate how this would work in practice, consider the example of Fig. 3. If the first page to be accessed after the failure is A, it would be the first to be restored. Using information from the page recovery index (which can be maintained in main memory or fetched directly from backups), the last red log record on the right side of the diagram would be fetched first. Then, following the per-page chain, all red log records until *minLSN* would be retrieved and replayed in the backup image of page A, thus yielding its most recent version to running transactions.

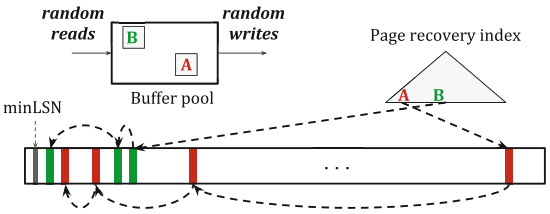

Fig. 3. Single-page repair

While the benefit of on-demand and incremental restore is a major advantage over traditional ARIES recovery, this algorithm still suffers from the first deficiency discussed in Sect. 1—namely the inefficient access pattern. The authors of the original publication even foresee the application to media failures [8], arguing that while a page is the unit of recovery, multiple pages can be repaired in bulk in a coordinated fashion. However, the access pattern with larger restoration granules would approach that of traditional ARIES restore—i.e., random access during log replay. Thus, while the technique introduces a useful degree of flexibility, it does not provide a unified solution for the two deficiencies discussed.

2.6 Single-Pass Restore

Our previous work introduced a technique called single-pass restore, which aims to perform media recovery in a single sequential pass over both backup and log archive devices [24]. Eliminating random access effectively addresses the first deficiency discussed in Sect. 1. This is achieved by partially sorting the log on page identifiers, using a stable sort to maintain LSN order within log records of the same page. The access pattern is essentially the same as that of a sort-merge join: external sort with run generation and merge followed by another merge between the two inputs—log and backup in the media recovery case.

The idea itself is as old as the first recovery algorithms (see Sect. 5.8.5.1 of Gray's paper [10]) and is even employed in DB2's "fast log apply" [13]. However, the key advantage of single-pass restore is that the two phases of the sorting process—run generation and merge—are performed independently: runs are generated during the log archiving process (i.e., moving log records from the latency-optimized transaction log device into high-capacity, bandwidth-optimized secondary storage) with negligible overhead; the merge phase, on the other hand, happens both asynchronously as a maintenance service and also during media recovery, in order to obtain a single sorted log stream for recovery. Importantly, merging runs of the log archive and applying the log records to backed-up pages can be done in a sequential pass, similar to a merge join. The process is illustrated in Fig. 4. We refer to the original publication for further details [24].

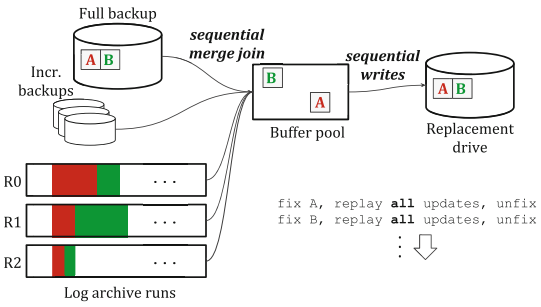

Fig. 4. Single-pass restore

Having addressed the access pattern deficiency of media recovery algorithms, single-pass restore still leaves open the problem of incremental and on-demand restoration. Nevertheless, given its superiority over traditional ARIES restore (see [6,24] for an in-depth discussion), it is a promising approach to use as starting

point in addressing the two deficiencies in a unified way. Therefore, as mentioned in Sect. 1, single-pass restore is taken as the baseline for the present work.

3 Instant Restore

The main goal of instant restore is to preserve the efficiency of single-pass restore while allowing more fine-granular restoration units (i.e., smaller than the whole device) that can be recovered incrementally and on demand. We propose a generalized approach based on segments, which consist of contiguous sets of data pages. If a segment is chosen to be as large as a whole device, our algorithm behaves exactly like single-pass restore; on the other extreme, if a segment is chosen to be a single page, the algorithm behaves like single-page repair.

This section starts by introducing the log data structure employed to provide efficient access to log records belonging to a given segment or page; after that, we present the restore algorithm based on this data structure.

3.1 Indexed Log Archive

In order to restore a given segment incrementally, instant restore requires efficient access to log records pertaining to pages in that segment. In single-page repair, such access is provided for individual pages, using the per-page chain among log records [8]. As already discussed, this is not efficient for restoration units much larger than a single page. Therefore, we build upon the partially sorted log archive organization introduced in single-pass restore [24].

In instant restore, the partially sorted log archive is extended with an index. The log archiving process sorts log records in an in-memory workspace and saves them into runs on persistent storage. These runs must then be indexed, so that log records of a given page or segment identifier can be fetched directly. Sorting and indexing of log records is done online and without any interference on transaction processing, in addition to standard archiving tasks such as compression.

In an index lookup for instant restore, the set of runs to consider would be restricted by the given *minLSN* (see Sect. 2.2) of the backup image, since runs older than that LSN are not needed. Furthermore, Bloom filters can be appended to each run to restrict this set even further. The result of the lookup in each indexed run is then fed into a merge process that delivers a single stream of log records sorted primarily by page identifier and secondarily by LSN. This stream is then used by the restore algorithm to replay updates on backup segments.

Multiple choices exist for the physical data structure of the indexed log archive. Ideally, the B-tree component of the indexing subsystem can be reused, but there is an important caveat in terms of providing atomicity and durability to this structure. A typical index relies on write-ahead logging, but that is not an option for the indexed log archive because it would introduce a kind of self-reference loop—updates to the log data structure itself would have to be logged and used later on for recovery. This self-reference loop could be dealt with by introducing special logging and recovery modes (e.g., a separate “meta”-log for

the indexed log archive), but the resulting algorithm would be too cumbersome. In our prototype, we chose a simpler solution: each partition of the log archive is maintained in its own read-only file; temporary shadow files are then used for merges and appends. In this scheme, atomicity is provided by the file rename operation, which is atomic in standard filesystems [5].

3.2 Restore Algorithm

When a media failure is detected, a restore manager component is initialized and all page read and write requests from the buffer pool are intercepted by this component. The diagram in Fig. 5 illustrates the interaction of the restore manager with the buffer pool and all persistent devices involved in the restore process: failed and replacement devices, log archive, and backup. For reasons discussed in previous work [24], incremental backups are made obsolete by the partially sorted log archive; thus, the algorithm performs just as well with full backups only. Nevertheless, incremental backups can be easily incorporated, and the description below considers a single full backup without loss of generality.

In the following discussion, the numbers in parentheses refer to the numbered steps in Fig. 5. The restore manager keeps track of which segments were already restored using a segment recovery bitmap, which is initialized with zeros. When a page access occurs, the restore manager first looks up its segment in the bitmap (1). If set to one, it indicates that the segment was already restored and can be accessed directly on the replacement device (2a). If set to zero, a segment restore request is placed into a restore scheduler (2b), which coordinates the restoration of individual segments (3).

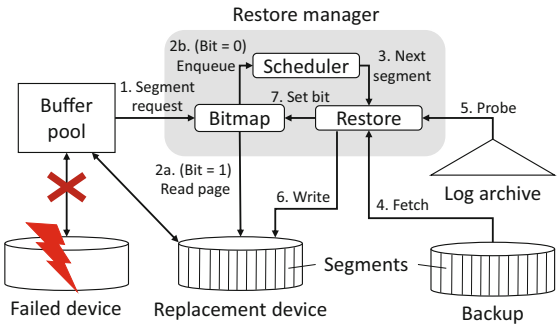

Fig. 5. Instant restore flow chart

To restore a given segment, an older version is first fetched from the backup directly (4). This is in contrast to ARIES restore, which first loads entire backups into the replacement device and then reads pages from there [19]. This has the implication that backups must reside on random-access devices (i.e., not on tape) and allow direct access to individual segments, which might require an index if backup images are compressed. These requirements, which are also present in single-page repair [8], seem quite reasonable given the very low cost per byte of current high-capacity hard disks. For moderately-sized databases, it is even advisable to maintain log archive and backups on flash storage.

While the backed-up image of a segment is loaded, the indexed log archive data structure is probed for the log records pertaining to that segment (5); the results of each probe are merged to form a single sorted log stream.

Then, log replay is performed to bring the segment to its most recent state, after which it can be written back into a replacement device (6).

Finally, once a segment is restored, the bitmap is updated (7) and all pending read and write requests can proceed. Typically, a requested page will remain in the buffer pool after its containing segment is restored, so that no additional I/O access is required on the replacement device.

All read and write operations described above—log archive index probe, segment fetch, and segment write after restoration—happen asynchronously with minimal coordination. The read operations are essentially merged index scans—a very common pattern in query processing. The write of a restored segment is also easily made asynchronous, whereby the only requirement is that marking a segment as restored on the bitmap, and consequently enabling access by waiting threads, be done by a callback function after completion of the write.

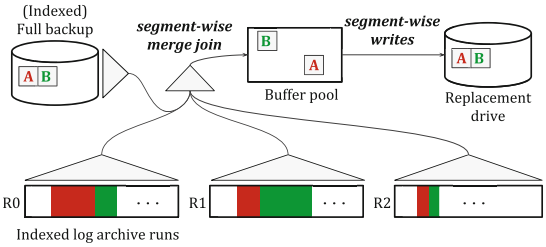

Fig. 6. Instant restore

To illustrate the access pattern of instant restore, similarly to the diagrams in Sect. 2, Fig. 6 shows an example scenario with three log archive runs and two pages, A and B, belonging to the same segment. The main difference to the previous diagrams is the segment-wise, incremental access pattern, which delivers the efficiency of pure sequential access with the responsiveness of on-demand random reads.

Using this mechanism, user transactions accessing data either in the buffer pool or on segments already restored can execute without any additional delay, whereby the media failure goes completely unnoticed. Access to segments not yet restored are used to guide the restore process, triggering the restoration of individual segments on demand. As such, the time to repair observed by transactions accessing data not yet restored is multiple orders of magnitude lower than the time to repair the whole device. Furthermore, time to repair observed by an individual transaction is independent of the total capacity of the failed device. This is in contrast to previous methods, which require longer downtime for larger devices.

4 Experiments

Our experimental evaluation covers three main measures of interest during recovery from a media failure: restore latency, restore bandwidth, and transaction throughput. Before presenting the empirical analysis, a brief summary of our experimental environment is provided.

4.1 Environment

We implemented instant restore in a fork of the Shore-MT storage manager [15] called *Zero*. The code is available as open source[1]. The workload consists of the TPC-C benchmark as implemented in Shore-MT, but adapted to use the Foster B-tree [7] data structure for both table and index data.

All experiments were performed on dual six-core CPUs with HyperThreading. The system has 100 GB of high-speed RAM and several Samsung 840 Pro 250 GB SSDs. The operating system is Ubuntu Linux 14.04 with Kernel 3.13.0-68 and all code is compiled with `gcc` 4.8 and `-O3` optimization.

The experiments all use the same workload, with media failure and recovery set up as follows. Initial database size is 100 GB, with full backup and log archive of the same size—i.e., recovery starts from a full backup of 100 GB and must replay roughly the same amount of log records. Log archive runs are a little over 1.5 GB in size, resulting in 64 inputs in the restore merge logic. All persistent data is stored on SSDs and 24 worker threads are used at all times.

4.2 Restore Latency and Bandwidth

Our first experiment evaluates restore latency by analyzing the total latency of individual transactions before and after a media failure. The hypothesis under test is that average transaction latency immediately following a media failure is in the order of a few seconds or less, after which is gradually decreases to the pre-failure latency. Furthermore, with larger memory, i.e., where a larger portion of the working set fits in the buffer pool, average latency should remain at the pre-failure level throughout the recovery process.

The results are shown in Fig. 7a. After ten minutes of normal processing, during which the average latency is 1–2 ms, a media failure occurs. The immediate effect is that average transaction latency spikes up (to about 100 ms in the buffer pool size of 30 GB) but then decreases linearly until pre-failure latency is reestablished. For the largest buffer pool size of 45 GB, there is a small perturbation in the observed latency, but the average value seems to remain between 1 and 2 ms. From this, we can conclude that for any buffer pool size above 45 GB, a media failure goes completely unnoticed.

These results successfully confirm our hypothesis: average latency of a transaction accessing failed media is reduced from several minutes to 100 ms, which corresponds to three orders of magnitude or three additional 9's of availability. Note that the average restore latency is independent of total device capacity, and thus of total recovery time. Therefore, the availability improvement could be in the order of four or five orders of magnitude in certain cases. This would be expected, for instance, for very large databases (in the order of terabytes) stored on relatively low-latency devices. In these cases, the gap between a full sequential read and a single random read—hence, between mean time to repair with single-pass restore and with instant restore—is very pronounced.

[1] http://github.com/caetanosauer/zero.

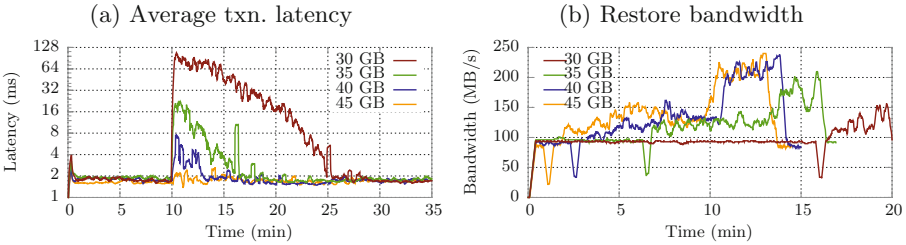

Fig. 7. Transaction latency and restore bandwidth observed with instant restore

Next, we evaluate restore bandwidth for the same experiment. The hypothesis here is that, in general, restore bandwidth gradually increases throughout the recovery process until it reaches the bandwidth of single-pass restore. From these two general behaviors, two special cases are, again, the small and large buffer pools. In the former, bandwidth may not reach single-pass speeds due to prioritization of low latency for the many incoming requests (recall that each buffer pool miss incurs a read on the replacement device, which, in turn, incurs a restore request). In the latter case, restore bandwidth should be as large as single-pass restore.

Figure 7b shows the results of this experiment for four buffer pool sizes. For the smallest buffer pool of 30 GB, restore bandwidth remains roughly constant in the first 15 min. This indicates that during this initial period, most segments are restored individually in response to an on-demand request resulting from a buffer pool miss. As the buffer size increases, the rate of on-demand requests decreases as restore progresses, resulting in more opportunities for multiple segments being restored at once. In all cases, restore bandwidth gradually increases throughout the recovery process, reaching the maximum speed of 240 MB/s towards the end in the larger buffer pool sizes.

4.3 Transaction Throughput

The next experiments evaluate how media failure and recovery impact transaction throughput with instant restore. We take the same experiment performed in the previous section and look at transaction throughput for each buffer pool size individually. As instant restore progresses, transactions continue to access data in the buffer pool, triggering restore requests for each page miss. Therefore, we expect that the larger the buffer pool is (i.e., more of the working set fits into main memory), the less impact a media failure has on transaction throughput. This effect was already presented in the diagram of Fig. 1—the present section analyzes that in more detail.

Figure 8 presents the results. In the four plots shown, transaction throughput is measured with the red line on the left y-axis. At minute 10, a media failure occurs, after which a green straight line shows the pre-failure average throughput. The number of page reads per second is shown with the blue line on the right y-axis.

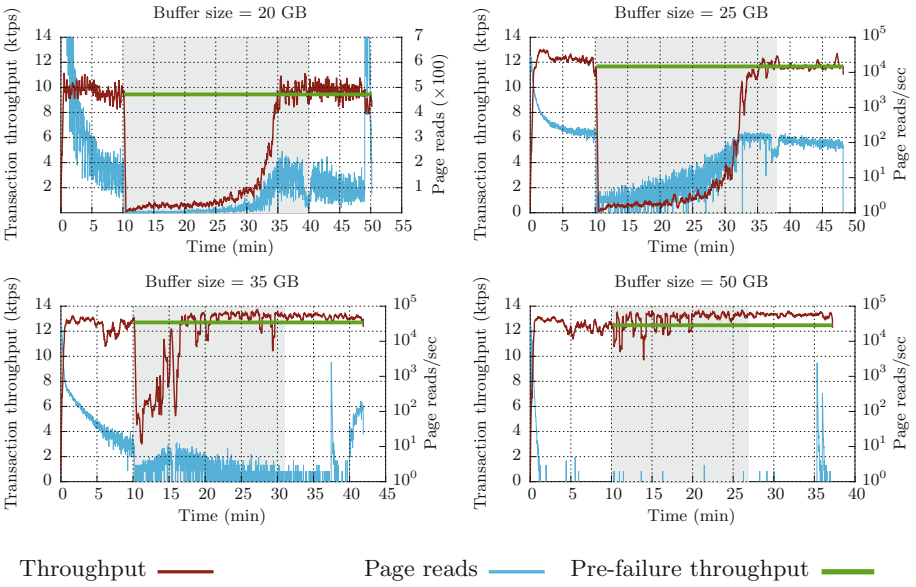

Fig. 8. Impact of instant restore on txn. Throughput at varying buffer pool sizes.

Moreover, total recovery time, which also varies depending on the buffer pool size, is also shown as the shaded interval on the x-axis.

The goal of instant restore in this experiment is to re-establish the pre-failure transaction throughput (i.e., the dotted green line) as soon as possible. Similar to the evaluation on previous experiments, our hypothesis is that this occurs sooner the larger the buffer pool is. The results show that for a small buffer pool of 20 GB, transaction throughput drops substantially, and it only regains the pre-failure level at the very end of the recovery process. As the buffer size is increased to 25 and then 35 GB, pre-failure throughput is re-established at around minute 7, i.e., 1/3 of the total recovery time. Lastly, for the largest buffer pool of 50 GB, the media failure does not produce any noticeable slowdown, as predicted in our hypothesis.

5 Conclusions

Instant restore improves perceived mean time to repair and thus database availability in the presence of media failures. We identified two main deficiencies with traditional recovery techniques, such as the ARIES design [19]: (i) media recovery is very inefficient due to its random access pattern on database pages, which means that time to repair is unacceptably long; and (ii) data on a failed device cannot be accessed before recovery is completed. The first deficiency was addressed with single-pass restore [24], which introduces a partial sort order on the log archive, eliminating the random access pattern of log replay.

The second deficiency is addressed with the instant restore technique, which was first described in earlier work [6] and discussed in more detail, implemented, and evaluated in this paper. By generalizing single-pass restore and other recovery methods such as single-page repair, instant restore is the first media recovery method to effectively eliminate the two deficiencies discussed. In comparison with traditional ARIES media restore, instant restore delivers not only the benefits of single-pass restore (i.e., substantially higher bandwidth and therefore shorter recovery time), but also much quicker access (e.g., seconds instead of hours) to the application working set after a failure.

References

1. Arulraj, J., Pavlo, A., Dulloor, S.: Let's talk about storage & recovery methods for non-volatile memory database systems. In: Proceedings of SIGMOD, pp. 707–722 (2015)
2. Bitton, D., Gray, J.: Disk shadowing. In: Proceedings of VLDB, pp. 331–338 (1988)
3. Chen, P.M., et al.: RAID: high-performance, reliable secondary storage. ACM Comput. Surv. **26**(2), 145–185 (1994)
4. Eich, M.H.: A classification and comparison of main memory database recovery techniques. In: Proceedings of ICDE, pp. 332–339 (1987)
5. GLIBC: The GNU C Library Reference Manual (2014), http://www.gnu.org/software/libc/manual/html_node/Renaming-Files.html. Accessed 06 Oct 2014
6. Graefe, G., Guy, W., Sauer, C.: Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, 2nd edn. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2016)
7. Graefe, G., Kimura, H., Kuno, H.A.: Foster B-trees. ACM Trans. Database Syst. **37**(3), 17 (2012)
8. Graefe, G., Kuno, H.A.: Definition, detection, and recovery of single-page failures, a fourth class of database failures. PVLDB **5**(7), 646–655 (2012)
9. Graefe, G., Kuno, H.A., Seeger, B.: Self-diagnosing and self-healing indexes. In: Proceedings of DBTest, p. 8 (2012)
10. Gray, J.N.: Notes on data base operating systems. In: Bayer, R., Graham, R.M., Seegmüller, G. (eds.) Operating Systems. LNCS, vol. 60, pp. 393–481. Springer, Heidelberg (1978). doi:10.1007/3-540-08755-9_9
11. Gray, J.: Why do computers stop and what can be done about it? In: Symposium on Reliability in Distributed Software and Database Systems, pp. 3–12 (1986)
12. Gray, J.: What next?: a dozen information-technology research goals. J. ACM **50**(1), 41–57 (2003)
13. Haderle, D.J., Majithia, T.: Fast log apply, US Patent 6,289,355, 11 September 2001
14. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. **15**(4), 287–317 (1983)
15. Johnson, R., Pandis, I., Hardavellas, N., Ailamaki, A., Falsafi, B.: Shore-MT: a scalable storage manager for the multicore era. In: Proceedings of EDBT, pp. 24–35 (2009)
16. Lehman, T.J., Carey, M.J.: A recovery algorithm for a high-performance memory-resident database system. In: Proceedings of SIGMOD, pp. 104–117 (1987)
17. Levy, E., Silberschatz, A.: Incremental recovery in main memory database systems. IEEE Trans. Knowl. Data Eng. **4**(6), 529–540 (1992)

18. Malviya, N., Weisberg, A., Madden, S., Stonebraker, M.: Rethinking main memory OLTP recovery. In: Proceedings of ICDE, pp. 604–615 (2014)
19. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst. **17**(1), 94–162 (1992)
20. Mohan, C., Narang, I.: An efficient and flexible method for archiving a data base. SIGMOD Rec. **22**(2), 139–146 (1993)
21. Mohan, C., Treiber, K., Obermarck, R.: Algorithms for the management of remote backup data bases for disaster recovery. In: Proceedings of ICDE, pp. 511–518 (1993)
22. Oracle Corporation: RMAN Incremental Backups, Oracle Database Documentation 10g, Sect. 4.4 (2015)
23. Oukid, I., et al.: SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery. In: Proceedings of DaMoN, pp. 8:1–8:7 (2014)
24. Sauer, C., Graefe, G., Härder, T.: Single-pass restore after a media failure. In: Proceedings of BTW. LNI, vol. 241, pp. 217–236 (2015)