Come and crash our database! – Instant recovery in action

Caetano Sauer
TU Kaiserslautern, Germany
csauer@cs.uni-kl.de

Gilson Souza
TU Kaiserslautern, Germany
gsantos@rhrk.uni-kl.de

Goetz Graefe
Google, Madison, WI, USA
goetzg@google.com

Theo Härder
TU Kaiserslautern, Germany
haerder@cs.uni-kl.de

ABSTRACT

We present a demonstration of instant recovery, a family of techniques to enable incremental and on-demand recovery from different classes of failures in transactional database systems. In contrast to traditional ARIES-based algorithms, instant recovery allows transactions to run concurrently to recovery actions—not only permitting earlier access to data that requires recovery but also using the post-failure access pattern to actually guide the recovery process. This mechanism prioritizes data needed most urgently after a failure, thus dramatically reducing the mean time to repair perceived by any individual transaction.

We have implemented instant recovery in an open-source storage manager and developed a Web-based interface to showcase its recovery capabilities. Users of this demo application are able to control the execution of various benchmarks and inject different types of failures arbitrarily, observing the system behavior and recovery progress live in a dashboard utility. Furthermore, since traditional ARIES recovery is also implemented, users can select the type of recovery and obtain a live graphical comparison of the different techniques.

1. INTRODUCTION

Database availability is a key challenge of scalable and reliable information systems. Improvements in availability can be achieved on two main fronts: increasing mean time to failure (MTTF) and decreasing mean time to repair (MTTR). Large businesses and Internet-scale services have invested heavily on the first front with highly redundant hardware configurations. The latter front, however, has not seen substantial improvements in the last decades, especially when considering the algorithms for logging and recovery in transactional database systems.

The vast majority of commercial and open-source database systems rely on techniques similar in essence to the ARIES family of recovery algorithms [5]. While ARIES works very well on traditional disk-based architectures with moderate transaction throughput and limited main-memory capacity, modern hardware and the systems designed to fully exploit its potential reveal severe limitations of traditional logging and recovery. These limitations include: long time to repair due to inefficient access patterns during recovery, inability to incrementally recover and enable access to most important data first, and high overhead on normal transaction processing.

Instant recovery was proposed recently as an alternative to ARIES that addresses its limitations on both traditional and modern hardware scenarios. Instead of being restricted to system failures and restart, it is designed to recover from all classes of failures known in the database literature. Furthermore, like ARIES, it relies on write-ahead logging with physiological log records, which means it can be incrementally implemented on an ARIES system, retaining all its capabilities while eliminating its limitations. For further elaboration on the limitations of previous techniques, the contribution of instant recovery, and empirical evaluation of the techniques, we refer to previous publications [1, 2, 7, 8].

This paper describes an interactive demo application designed to showcase the benefits of instant recovery in comparison with ARIES. After a brief overview of instant recovery in Section 2 below, the architecture and functionality of the demo application is described in Section 3. We provide a high-level description of the application and its interaction with the underlying transactional system, focusing on the user interaction and what attendees of the demonstration can expect to see. Finally, Section 4 provides a summary and some concluding remarks.

2. INSTANT RECOVERY IN A NUTSHELL

Instant recovery [1] is a family of algorithms designed to address different classes of failures in transactional systems. Table 1 summarizes such failure classes and their typical causes and effects. Details of the specific recovery mechanisms employed for each class are beyond the scope of our demonstration and have been explored extensively in previous research [1, 2, 7, 8]. This section discusses the fundamental characteristic common to all instant recovery techniques: the support for recovery actions that are executed concurrently to normal processing, provide incremental access to already-recovered data items, and exploit workload access patterns to guide recovery and prioritize data needed most

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

Table 1: Failure classes, their causes, and effects (from [8])

urgently. In order to summarize these techniques and provide an overview of how instant recovery works, we discuss the case of restart after a system failure.

When a system failure occurs, the in-memory state of the database server process is lost and must therefore be recovered when the system comes back up. As in ARIES recovery, this boils down to determining which transactions must be rolled back in the UNDO phase (i.e., *active transactions*) and which pages must have their updates replayed in the REDO phase (i.e., *dirty pages*). Such information is collected with a sequential log scan in the *log analysis* phase, which covers the interval from the most recent checkpoint up to the last persisted log record.

In essence, the key characteristic that distinguishes instant restart from ARIES restart is that, once log analysis information is collected, the REDO and UNDO phases can be performed on a per-page and a per-transaction basis, respectively. In other words, recovery actions can be scheduled according to a variety of policies, and not just the schedule dictated by a sequential REDO or UNDO log scan. This means that as soon as log analysis is complete (which usually only takes a few seconds), transactions can immediately be admitted to the system and, as soon as they touch a page in need of recovery (i.e., a page marked dirty during log analysis) or incur a lock conflict with a transaction in need of rollback (i.e., a transaction marked active during log analysis), that single page or that single transaction can be recovered on demand. The basic design features that enable such on-demand recovery are (i) a per-page chain of log records that is also the basis of single-page recovery [2]; and (ii) tracking acquired locks during checkpoints and log analysis.

Fig. 1 illustrates the process of on-demand recovery during instant restart. In this scenario, log analysis has detected two dirty pages, A and B, whose expected page LSN is x and y, respectively. Furthermore, two pre-failure active transactions, T_1, holding locks with identifiers b and d, and T_2, with lock on f, were also detected. After log analysis, a new transaction is initiated—shown here as the gray box on the top left corner. This transaction attempts to fix page B in the buffer pool; because this page is marked as needing recovery, on-demand REDO processing kicks in and log records pertaining to this page are replayed by following a backward chain of log records (which is exactly the same process employed for single-page recovery [2]). Log replay of this single page happens concurrently with replay of any other page; in fact, it may even happen with asynchronous, ARIES-style REDO based on a forward log scan.

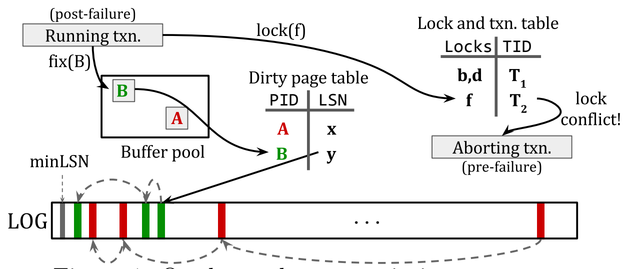


Figure 1: On-demand recovery in instant restart

After page B is restored, the new transaction attempts to acquire a lock on f. In this case, the lock table yields a conflict with transaction T_2; since this transaction is a pre-failure one, its rollback is triggered by this lock conflict. Because rollback of a pre-failure transaction works exactly in the same way as a transaction abort during normal processing (e.g., due to deadlock), the same logic is applied. Once such rollback is completed, the locks held by T_2 are released and the lock is finally granted to the waiting post-failure transaction.

The on-demand and incremental recovery schedules essentially reduce the MTTR as perceived by a single transaction by multiple orders of magnitude. ARIES restart recovery, in contrast, usually requires at least a full REDO log scan—which is typically the longest phase of recovery by far [6]—before the first post-failure transaction can complete[1].

In the next section, we describe the demo application proposed by this paper. It is the first interactive user interface ever developed for instant recovery, which can not only reproduce some of our empirical measurements, but also allow the user to interact with the workload, observe the system behavior graphically, and, most importantly, inject failures arbitrarily.

3. DEMONSTRATION

Instant recovery techniques such as single-page recovery, instant restart, and instant restore have been implemented over the past three years in the Zero [2] storage manager prototype, which is a fork of the well-known Shore-MT [3][3]. Zero has also been incorporated into the MariaDB database system (a modern fork of MySQL) as a storage engine module that can be used as an alternative to the popular InnoDB engine.

Our demonstration will provide a hands-on experience to interact with these systems under a variety of benchmark workloads. Focusing mainly on the logging and recovery aspects, the demo program enables users to inject different types of failures in a database workload running on Zero and observe the recovery process live in an intuitive graphical interface. Combined with a choice of parametrized workloads, this rich interface allows users to interactively explore

[1]Improvements to the ARIES algorithm aimed at enabling earlier access during recovery have been proposed [5, 4], but, in summary, none of them provides fully on-demand and incremental recovery to fine-granular objects, especially those needed most urgently by the application. The limitations of these "extended" versions of ARIES are also discussed in previous work [1, 8]

[2]https://github.com/caetanosauer/zero

[3]https://sites.google.com/site/shoremt/

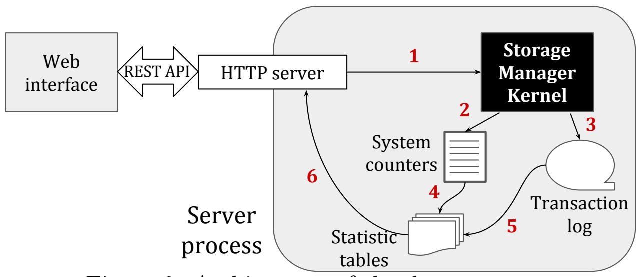


Figure 2: Architecture of the demo program

the potential of instant recovery techniques in improving database reliability.

To demonstrate the various instant recovery techniques and compare them with traditional ARIES-based recovery, we have developed a Web-based interface to control the execution of benchmarks and display relevant system statistics in a graphical way. This section describes how our demo program is organized, how it achieves these goals, and how the user can interact with it.

3.1 Demo program architecture

The diagram in Fig. 2 illustrates the main components of the demo program and their interaction. The Web interface serves two main purposes: first, it sends commands to the Zero storage manager (which may be running either independently or as a storage engine inside MariaDB) to control the execution of workloads as well as adjust system and benchmark parameters dynamically; and second, it serves as a "dashboard" to visualize real-time statistics.

The communication between Web client and server is implemented with a JSON-based REST API. Commands are available to start and stop a certain benchmark workload and—especially for the purposes of instant recovery—inject system crashes and persistent storage failures. System crashes cause the immediate destruction of all in-memory data structures of the server process—most importantly buffer pool, transaction manager, and lock table. In Fig. 2, the process of receiving a command via the REST API and forwarding it to the storage manager kernel is illustrated by arrow number 1. The following paragraphs discuss the remainder of the demo program architecture by referring to the numbered arrows.

While the storage manager kernel processes transactions, it generates two types of information which are relevant for the demo. First, a collection of system counters (arrow 2) is used to keep track of system events for which only the total number of occurrences is of interest. For example, counters of transaction commits, page reads and writes, log volume generated, number of active transactions, number of dirty pages, etc. are typical measures of interest.

The second collection mechanism is through the transaction log (arrow 3). By continuously analyzing the logs and performing various aggregations, events can be collected in a time-dependent manner, allowing more detailed statistics than those provided by simple counters. The information collected from the system counters and the transaction log is stored in a collection of statistic tables (arrows 4 and 5), which are finally serialized into the JSON format for display in the demo (arrow 6).

3.2 Visualizing the recovery process

As mentioned above, the demo application provides commands to inject failures into the running workload. We support injection of three of the four classes summarized in Table 1: system, media, and single-page failures. The remaining class—transaction failure—is not supported because there is not much to demonstrate in that case, as the abort of a single transaction does not cause any noticeable impact on system behavior.

When injecting a failure, the user may also choose which recovery method to employ: traditional ARIES or instant recovery. In addition to system failures, users can also inject a failure on persistent storage (e.g., single-page or whole-device failures) independently of system failures—i.e., different failure modes can be mixed and matched freely. Furthermore, failures can be re-injected at any time during recovery from a previous failure, demonstrating the independence and idempotency of recovery modes.

Regardless of whether recovery activities are being carried on or not, the dashboard of the demo program constantly displays statistics collected from the statistic tables described above, some of which can be selected for plotting. For instance, the user can observe the transaction throughput along with the buffer pool hit ratio. During instant restart after a system failure, these values should gradually rise as on-demand, incremental recovery progresses.

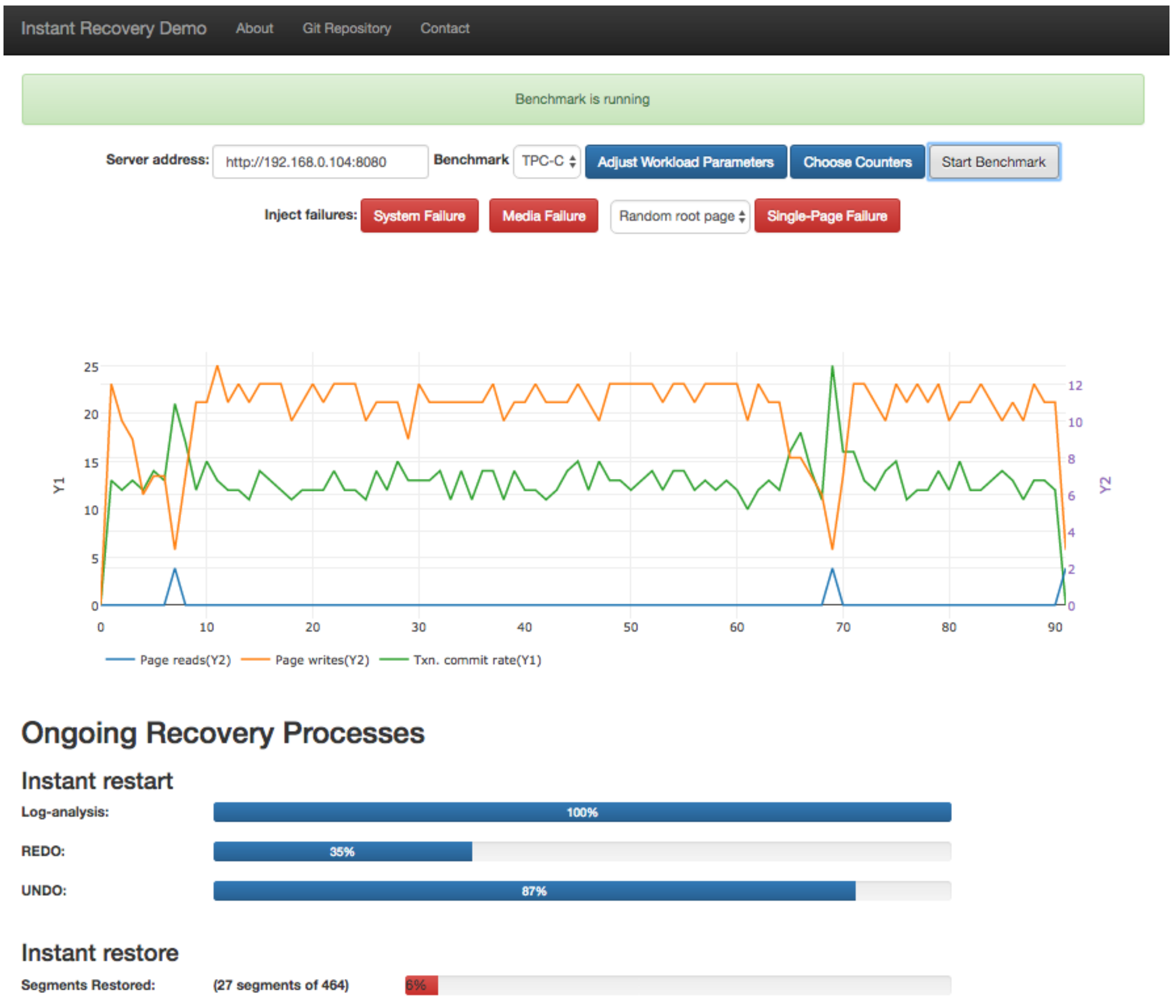
In addition to statistics available during normal processing, special statistics are collected and displayed during the recovery process—for instance, progress bars indicate the percentage of completion of each recovery phase (log analysis, REDO, and UNDO).

3.3 Screenshot walk-through

Fig. 3 shows a screenshot of our demo application in which a TPC-C workload is running and two recovery processes are currently active: instant restart (from a system failure) and instant restore (from a media failure). At the top, the IP address of the server process is provided along with the chosen workload—in this case TPC-C. An additional button opens up a pop-up window in which system and workload parameters can be adjusted—some of which can also be changed while the benchmark runs. Below that, three red buttons are provided to inject a system, media, or single-page failure. In the latter case, the user can additionally specify what kind of page should be selected for failure (in this case, a root page of an index). These buttons remain available even when recovery is already being carried out, allowing users to simulate failure-on-failure scenarios.

The middle part of the screenshot shows the graphical display component, in which three statistics are currently selected—transaction commit rate, page reads, and page writes. Using the "Choose counters" button on the top, users can select different statistics to plot. A text output of all stats and counters is also available but omitted here.

Finally, the bottom part shows the progress of currently ongoing recovery processes. In this case, a system restart is being carried on, and the progress bars show how much of the REDO and UNDO phases has been completed. Below that, a single progress bar displays the progress of a concomitant restore process, in which segments of the failed device are also restored incrementally. Note that transactions are running despite the ongoing recovery—this is the crucial feature of instant recovery.



Benchmark is running

Server address: http://192.168.0.104:8080 Benchmark TPC-C

Inject failures: Random root page

Ongoing Recovery Processes

Instant restart

Log-analysis: 100%

REDO: 35%

UNDO: 87%

Instant restore

Segments Restored: (27 segments of 464) 6%

Figure 3: Screenshot of the Instant Recovery Demo

4. SUMMARY AND CONCLUSIONS

Instant recovery drastically improves database system availability in the presence of failures, allowing incremental and on-demand recovery. Using a prototype transactional storage manager, we aim to demonstrate how these new recovery techniques behave in a real system. The demo application proposed here will enable attendees at the conference to interactively inject failures into a running workload and observe the instant recovery process live. The dashboard utility constantly collects relevant statistics from the database server process and displays them graphically according to customized selections made by the user. The Zero storage manager on which the demo is based is the first—and so far only—implementation of instant recovery. Therefore, the demo application will also provide a novel experience to most attendees.

5. REFERENCES

[1] G. Graefe, W. Guy, and C. Sauer. *Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, Second Edition*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2016.

[2] G. Graefe and H. A. Kuno. Definition, Detection, and Recovery of Single-Page Failures, a Fourth Class of Database Failures. *PVLDB*, 5(7):646–655, 2012.

[3] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi. Shore-MT: a scalable storage manager for the multicore era. In *Proc. EDBT*, pages 24–35, 2009.

[4] C. Mohan. A cost-effective method for providing improved data availability during DBMS restart recovery after a failure. In *Proc. VLDB*, pages 368–379, 1993.

[5] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. 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.

[6] C. Sauer, G. Graefe, and T. Härder. An empirical analysis of database recovery costs. In *RDSS (SIGMOD Workshops)*, 2014.

[7] C. Sauer, G. Graefe, and T. Härder. Single-pass restore after a media failure. In *Proc. BTW, LNI 241*, pages 217–236, 2015.

[8] C. Sauer, G. Graefe, and T. Härder. Instant restore after a media failure. Under submission, 2016.