GRAVIC®
Shadowbase

The Advantages of Transaction Protection

Avoid the Dangers of Non-audited Environments for Mission Critical Applications

A Gravic, Inc. White Paper

Executive Summary

Myths and hearsay still persist that the use of transaction management software (TM) and audited data comes at a considerable performance cost that outweighs the benefits. To their credit, those that believe the myths and hearsay recall results from the early iterations of TM, which did have performance issues.

However, with newer versions, TM evolved to the point that not using it brings significant issues, challenges, and danger.

Using transaction management (TM) software like the HPE Transaction Monitoring Facility (TMF) with audited data offers significant *operational*, *reliability*, and *data integrity* advantages over non-audited database applications. TM has been shown to improve overall system performance and utilization, allowing for increased application processing capacity on the system.

Table of Contents

The Advantages of Transaction Protection for Mission Critical Data

Many Applications are Written without Transaction Semantics in Mind

Application functionality and features are typically prioritized over other development items such as interoperability. A mission critical application requires different IT layers to work together for maximizing its performance and value. Problems occur when different IT groups such as application, database, and infrastructure teams make changes without coordination.

For example, when an application is written without consideration for transaction protection, an application is performing database operations (data changes) outside of the context of a transaction. These changes do not exhibit *ACID* transaction properties and protection as outlined below.

ACID Properties are the Backbone of Mission Critical Applications

ACID properties mean that transactions are:

- *Atomic* (either everything happens or nothing);
- *Consistent* (data is correctly saved);
- *Isolated* (occurs independently from the effects of other transactions);
- *Durable* (once completed, data changes are permanently stored).

Audited transactions, such as those that use HPE TMF, automatically inherit these properties.

Forgoing ACID Properties

Without transaction protection, an application and database live in a "non-audited" environment; data changes are not written to a transaction log or audit trail for the purposes of enforcing transaction semantics and recovery.[1] Non-audited applications may use other mechanisms such as application I/O intercepts for logging or journaling to avoid some of the issues inherent with non-audited databases.

Modern mission critical applications benefit from transaction protection, especially for high-volume applications or those with extremely valuable content. A technology comparison of both approaches is shown below.

Technology Comparison

	Without Transaction Protection	With Transaction Protection
Database Consistency	Not guaranteed	Guaranteed
Database Corruption	Possible	Does not occur
Database & Application Performance	Significantly worse	Dramatically better
Database & Application Capacity	Significantly worse	Dramatically better
ACID Properties	Not enforced	Enforced
Best Practice?	1. No 2. Each I/O is treated as an independent operation, regardless of its logical relationship with related database changes.	1. Yes 2. Each I/O is part of a single "unit-of-work", and either all of the I/O's will be applied, or none will. 3. It is not possible to partially execute a business transaction; the database will

[1] As opposed to *audited* data, which has ACID properties and is protected by transaction semantics and a transaction manager that writes before and after images of changed data to a transaction log (or *audit trail*) before committing changes to the actual database.

	3. It is possible to partially execute a business transaction and leave it in an inconsistent state.	always be left in a consistent state.
Risk Factors	1. In addition to data inconsistencies, without transaction protection it is possible to 'forget' to enable capturing database operations, leading to lost changes, which can also corrupt the target database. 2. These environments are delicate and easy to alter, posing great danger	1. The risk of data inconsistencies is avoided. All database changes are always captured in the DBMS's audit trail and are available for subsequent inspection if needed. 2. Helps guarantee data replication between source and target database formats to keep both databases *complete*, *accurate*, and *consistent*.
Longevity	Often viewed by new management as unsophisticated, legacy, and ancient, leading to rapid replacement with other best practice solutions that include transaction protection.	Helps future-proof applications. Newer applications and technologies can leverage transactions for their inherent benefits.

How to Enable Transaction Management Protection

To switch a non-audited application to an audited one that uses TM, HPE NonStop users can simply use an automatic transaction facility (ATF). The HPE Transaction Monitoring Facility (TMF), which is automatically included with every HPE NonStop and provides unequaled capabilities in the marketplace, does not require an application rewrite, nor typically require any application modifications at all.

Mission Critical Data Requires the Benefits of Transaction Protection and Audited Data

There are few reasons to forgo these benefits, and to do so puts such data and the business at risk.

Myths of the Past vs. Facts of the Present

In the early days of TMF (many decades ago), certain environments experienced overhead and latency with its introduction. Those applications could not afford this overhead and latency. Over time, TMF evolved with numerous performance and enhancement fixes, to the point that using TMF makes your system run faster.

Data's Increasing Value

Data is increasingly more valuable, which makes using full ACID transaction protection necessary. Software solutions such as "Automatic Transaction Facilities" (ATF), like HPE AutoTMF, automatically provide transaction semantics to an existing non-audited application, without requiring *any* application changes, including coding and recompiling.

The Dangers of Non-audited Environments

This paper elaborates on the inherent issues and dangers of non-audited applications, database operations, and inferior methods of data protection. It effectively dispels the myth that transaction protection is unnecessary or comes at an acceptable cost. It illustrates how to easily employ ATF solutions to upgrade existing non-audited applications to achieve transactional protection along with ACID properties, and illustrates throughput of non-audited vs. audited environments.

The Non-audited Database Problem

When a non-audited I/O operation is performed against database files or tables, the only evidence of this change is the post-operation values reflected in the files or tables. That is it; there are no before images of the I/O operation that can be referenced, and the after image is the current value in the database (which means

the data is gone if the operation was a delete). Without auditing or logging database changes, it is extremely difficult, if not impossible, to know if the changes were applied correctly. This performance works fine for database operations that successfully complete, but what about when errors or failures occur partially through the application processing?

Remediating Errors and Failures in Non-audited Environments is Slow and Complicated

In some circumstances, a process, processor, disk, or system failure can occur without any notification or alert. This limitation delays recovery actions and complicates remediating these issues. This problem is common for I/Os that require multiple database operations and/or that are split across several applications or sub-routines. Without transaction protection, recovering from these failures is complex and error-prone, and likely to result in wasted time, frustrated IT teams, upset users, and database corruption.

Did All, Some, or None of the I/O Operations Complete?

Consider an application that prints bank checks. It makes changes to accounts in the database debiting the accounts, prints checks and writes a confirmation after each check prints. What if a check is printed but the application fails before the account information is updated, or when the processing is reversed, the account information is updated and the check fails to print?

Without transaction protection or some other form of logging, there maybe is no record of what did or did not happen, resulting in an inconsistent result and database corruption. There is no persistent knowledge of exactly what was done and what wasn't, and hence what should occur to recover the situation. To make matters worse, while the team is trying to figure out how to fix these issues, the application continues to run, further transmitting any incorrect or inconsistent data in the database.

Transaction Protection Greatly Simplifies Application Recovery

By contrast, with transaction protection, all steps in the business transaction are encapsulated within a single transaction context (called a "unit-of-work", or UOW). As the transaction occurs the steps are applied to the database; however, the results remain locked for other users. If an error occurred during the transaction processing, since the transaction is *atomic*, it would automatically abort, and all of the executed operations would be rolled back, undone, and unlocked. Therefore, either both the account is debited and the check is printed, or neither occurs – no other outcome is possible – including situations involving a total system failure, where the transaction manager would resolve any outstanding transactions upon recovery.

The number of possible outcomes is reduced for non-audited applications with numerous combinations of unknown states (what I/O was done vs. what was not) to audited applications with two known states (everything in the transaction happens or nothing at all). This protection ensures databases are always in a consistent state – the initial state or the final state.

Methods Used to Handle Non-audited Data

Some approaches exist to add some layer of transaction protection to an application. These alternate approaches are inferior to the full gamut of transaction protection and ACID properties and may introduce other problems.

Application Logging

The *application logging* approach involves an application creating its own audit trail of operations, also known as an "application log file" or journaling to a "journal log". Every application I/O operation is written to the log file. The log file typically includes details about the actual data changes, as well as application state commentary. One example is "about to execute operation X," and subsequently "operation X completed", etc. gets written.

After a failure, there is a persistent source of information about the state of the application and its progress with database operations at the time of the failure. However, this information creates a significant performance impact. *Two or more* database I/Os must be performed for every *one* operation. How many I/Os depends on if additional application state information is written. There is one I/O to update the actual database and another to log the event.

While it is true that using transaction protection also requires additional I/Os to be written to the log file, these I/Os are highly optimized with boxcarring[2] and caching with no data loss, which is far more efficient than an application logging event.

Application logging can also suffer from similar database inconsistency issues to the approach where no journaling is performed at all. Journal I/Os may fail, leaving the journal file inconsistent with what was actually done to the file or table. Journal entry operations that are not flushed to disk or otherwise made persistent can be lost if a process, CPU, or other failure occurs. Trying to remediate this issue by persistently flushing journal entries to disk will significantly impact application and system performance.

Since application logging and database events are independent and uncoordinated, when a message such as "operation X completed" is absent from the log, it does not guarantee that operation X was, in fact, completed. Therefore, journal entries are ultimately unreliable and cause difficulty recovering from any failures, since the journal may be incomplete and the actual state of an application and database is unknown. With transaction protection, had "operation X" not completed, the transaction would have been aborted, the database changes undone, and the database would remain in a correct and consistent state.

Another downside to the application logging approach is that if this feature is not already included from the outset, an application will have to be rewritten to include the journaling functionality. This coding and testing effort is difficult at best or even impossible for teams that do not have their application's source code.

Database Triggers

The database trigger approach is a variant of *application logging*; the major difference is that an application itself does not have to be rewritten to perform the journaling. Database triggers are a mechanism provided by most enterprise database management subsystems (DBMS), in which user-written functions are automatically executed (triggered) upon the event of a database update to certain database tables.

Database triggers are customizable. Different database files and tables can have different trigger functions assigned or, optionally, none at all. Database triggers automatically execute when an I/O occurs to an assigned database file or table. The trigger function can write details about data changes, application state, etc., to the log file, which is similar to the *application logging* approach. Writing these details to persistent storage saves pertinent information for investigation after an application failure or similar problem occurs.

The database trigger approach avoids rewriting an application to perform the logging feature, but in all other respects, it suffers from the same issues as application logging such as poor performance, and the potential for database inconsistency.

Finally, it also suffers an additional problem. Database triggers must be enabled each time the DBMS is started. If this step is omitted, none of the trigger functions will be executed, and no logging will occur. There are no warnings or error messages alerting to this situation; application processing continues, compounding problems such as database inconsistencies. The database triggers approach is therefore also unreliable and error prone.

Intercept Libraries

This method is a variant of the database trigger approach, meaning that no application changes are required. Rather than effectively "inserting" logic into an application flow via a database trigger function, the intercept approach achieves the same end by intercepting (or "hijacking") the existing procedure calls used by an application to perform database operations, for example, database-provided SQL functions or operating system-provided I/O functions such as Read, Write, etc.

Rather than the default functions, customized function versions are developed and linked to or bound with the application. The customized versions maintain exactly the same syntax and semantics as the default versions, so no application changes are required. When an application calls these default functions, instead of being immediately executed, the customized function is executed first and journals the application activity (as previously discussed), before calling the default function to actually perform the required database operation.

[2] "Boxcarring" refers to a transaction manager feature that groups multiple audit records on behalf of one or more transactions into a single database (audit trail) write operation, reducing the number of writes per audit record and reducing overhead and improving performance.

This approach avoids the need to rewrite an application to perform the logging, but the intercept library approach suffers from all of the same issues as previously discussed with application logging such as poor performance, possibility of errors resulting in database inconsistency, etc. In addition, it is necessary to intercept every single application's database I/O function call. If function calls are missed, database inconsistencies are compounded while application processing continues. Hence, the intercept library approach is unreliable and error prone.

Business Continuity and Non-audited Data

An additional consideration applies to non-audited applications that use these previously mentioned logging methods. For any mission critical application, business continuity is a prime concern, which includes application services that survive any IT situation that would normally cause an outage such as:

- power outages,
- system or data center failures,
- floods,
- earthquakes, and
- terrorist attacks.

Business continuity requires *geographic fault tolerance*; a primary or active system must have an identical backup or standby system available in a geographically separate location. While there is some debate around *how far apart* both systems need to be, best practices include considerations for earthquake fault lines, independent power grids, backup networks, and using different teams for each location, among others.

A backup system must be identical to the primary or active system in order to be reliable. Sooner or later, IT teams will need to perform a failover to the backup system. Data consistency between systems is critical and typically accomplished by using an online data replication solution, especially for mission critical environments.

Data replication solutions monitor the active system's databases for changes and replicate them to the backup database, keeping the two synchronized. When audited data is used, this process usually involves data replication software reading the active system's audit trail, a reliable source of committed database changes, and sending these changes to the backup system for replay against the target database. This process often uses the same transactional semantics that were used on the source such as BEGIN WORK to COMMIT WORK statements for SQL environments, and ensures both databases are consistent and stay synchronized over time.

When application logging or a related approach is used instead of an audit trail that leverages transaction protection, the source system's change data must be carefully and correctly configured to the data replication software so that changes can be read. However, it is very easy to miss individual files, rows, or even entire tables when manually defining this information. When these files, rows, or tables are missed in the configuration, the source application continues processing, making data changes that are not collected and replicated to the backup database, causing inconsistencies, also known as a *database divergence* between the primary and backup databases. This database divergence happens without sending any error messages or alerts to the IT team. Additionally, this database divergence *increases over time* since records are being added and data changes are being made on the source system without being properly replicated to the backup system.

Even worse, after a failover from the primary to backup system is required, processing on the backup system (now the primary) will occur based on an inconsistent database, exacerbating the database divergence. This divergence is extremely difficult to detect, and may never be detected if the primary system or database is destroyed or otherwise inaccessible after a failover. Ultimately, if a primary database is incorrect due to errors of any kind, including errors from following dangerous non-audited practices, then the backup database will be incorrect, no matter how well the backup database is being kept synchronized.

Transaction Management Facilities

Since forgoing transaction protection or using alternate methods of data change logging causes so many problems, why not always use audited data and transaction semantics[3] to avoid them? Unfortunately, the main reason for this issue is based on myths and hearsay. These myths and hearsay claim the use of transaction management software (TM) and audited data comes at a considerable performance cost that outweighs the benefits. To their credit, those that believe the myths and hearsay recall the early iterations of TM, which did have performance issues.

However, with newer iterations, TM evolved to the point to dramatically improve performance such as:

- Database disk processes can eliminate physical I/O operations by caching, without possibility of data loss – when a system fails, updates in the audit trail are reapplied instead of being lost.
- Audit trail writes are "boxcarred" – audit for many transactions from many disks is collected and written to the end of the audit trail with a single I/O operation.
    - Depending upon the TM/DBMS architecture, audit records sent from the database disk process to the audit trail disk process may also be blocked together using the "boxcarring" technique – fewer messages can efficiently represent a large number of transactions.
- Transaction management software can be massively scaled using many parallel audit trails (on HPE NonStop, using master "MAT" and one or more auxiliary "AUX" audit trails).
- Current hardware technology with multi-cored processors is orders of magnitude faster at lower cost than older platforms, and modern TM/DBMS software is architected to be parallelized, enabling the software to scale with the hardware platform.

These enhancements have effectively eliminated concerns over performance implications when using transaction semantics. A performance analysis was undertaken,[4] comparing transaction rates and response times for a sample application using audited and non-audited files. In this analysis, the absolute best transaction rate that could be achieved using non-audited files was 960 transactions per second (TPS), whereas using audited files reached 2,450 TPS (see Figure 1).

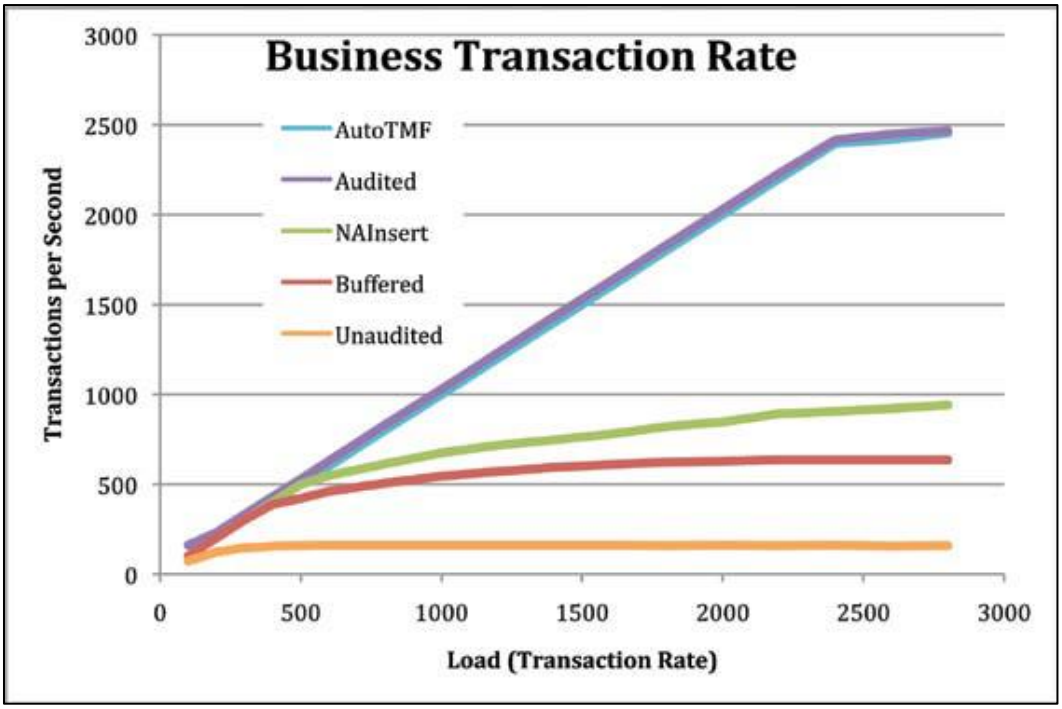

Figure 1 – Audited vs Non-audited Transaction Rate

[3] Transaction services are typically provided either by specialized transaction management (TM) subsystem software, such as the HPE Transaction Management Facility (TMF), or as part of a relational database management system (RDBMS), such as Oracle.

[4] See, "Best Practices: Using TMF to Implement Business Continuity/Disaster Recovery," by Richard Carr, The Connection Magazine, Sept – Oct 2013, Volume 34, No. 5.

Note that the best non-audited transaction rate was achieved when using a technique called “database buffering,” which comes at the cost of possible data loss in the event of a failure. This concern does not apply to audited files as the technique already includes a more efficient form of buffering.

The same result is true for transaction response times. For the same load on the system, when using unaudited files, the transaction response time was as much as 10x slower (> 500 milliseconds) than when using audited files (< 50 milliseconds) (see Figure 2).

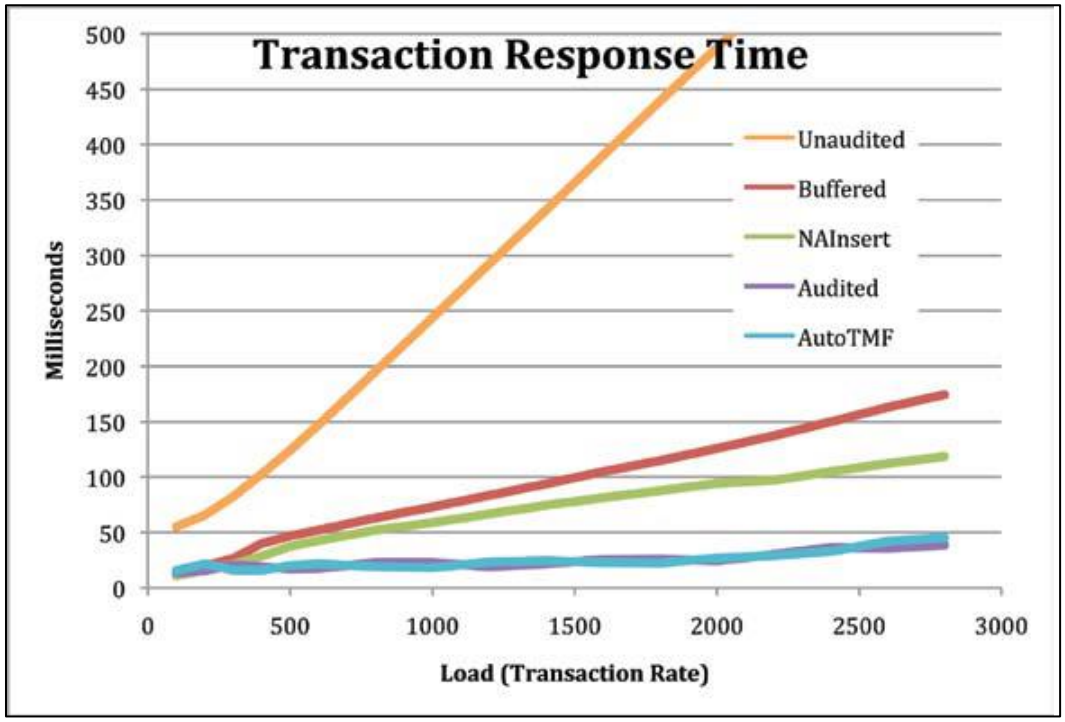

Figure 2 – Transaction Response Time

Also, there was no form of logging being performed for the non-audited case. If logging was introduced into the environment, for example to support a replication engine, then additional overhead would have made the performance *even worse* when using non-audited data.

Automatic Transaction Facilities

For many applications, however, these TM enhancements came too late, and the applications were designed with non-audited data, which posed a problem. It used to be unfeasible, or even impossible, to update existing non-audited applications to use transaction protection, despite all of the benefits offered by their use. Then, Automatic Transaction Facilities (ATF) software was introduced. ATF can enable transaction protection for non-audited applications without requiring application changes.

Automatic Transaction Facility Software Characteristics

General ATF software characteristics include:

- External configuration parameters define which automatic transactions of database files and tables are to be executed, and which transaction characteristics (such as automatic transactions) should span multiple files and tables or be exclusive to a particular file or table (isolation properties).

- ATF software uses intercept library technology, whereby the ATF software is linked with the application program and intercepts all procedure calls used by the application to perform database operations. Note that ATF intercept libraries cannot miss any relevant database I/O function calls since ATFs are written for specific database and operating system environments. This design narrows calls needed to be intercepted to a known subset, which is the same for all applications being addressed by the ATF software.

- The intercept library interacts with the underlying operating system (O/S), DBMS, and transaction manager subsystems to call the appropriate transaction functions as provided by the transaction manager, coordinating the database operations provided by the O/S and/or DBMS. This coordination enables the ATF software to execute the intercepted database I/O operations and wrap them into a transaction (as defined via the ATF configuration parameters).

An example of an ATF is the software product, "AutoTMF,"[5] sold by Hewlett Packard Enterprise (HPE). HPE NonStop systems provide world-class mission critical transaction processing platforms in the industry. TMF is a long-standing and highly evolved component of these systems. AutoTMF leverages the services provided by TMF to enable automatic transaction wrapping and protection. On HPE NonStop systems, AutoTMF supports the Enscribe file system along with SQL/MP and SQL/MX relational databases for persistent data storage.

Transaction protection and related properties for non-audited files and tables can be configured in AutoTMF. The non-audited application is linked with the AutoTMF intercept library. When the application calls a relevant non-audited database function, the AutoTMF intercept library executes its version of that function instead. This function checks with the AutoTMF configuration information (for example, whether the operation is intended for a file or table which is configured for automatic transactions). TMF makes the appropriate function call (for example, to BEGIN, COMMIT or RESUME a transaction) dependent upon this information, the database operation being performed, and the current state of any currently in-process transaction intercepted by AutoTMF. Then, the AutoTMF intercept library issues the database function as originally called by the application, and returns control to the application with the results of that operation. This action is how AutoTMF enables non-audited applications running on HPE NonStop systems to derive transactional protection and its related ACID properties for operations performed on Enscribe, SQL/MP and SQL/MX databases, without requiring any application modifications.

Debunking the Myths about Transactions

Transaction protection provides numerous benefits such as ACID properties, simplicity when defining data replication configurations and capturing all relevant changes, etc. The myths and hearsay regarding performance issues has largely been eliminated including the fact that most if not all applications will actually see *dramatically* improved performance, and ATF software can be used for existing non-audited applications without requiring application changes. Despite how this technology has evolved, claims are continuously raised against transaction protection, for example[6]:

- *"Managing transactions introduces extra steps in the application workflow, and thus degrades performance."* **However, it is actually the opposite.**
  - If a non-audited data replication solution is being used, then every data change (insert, update, or delete) requires an extra application I/O to the data replication processes and/or log files. This step adds considerable overhead to the non-audited file I/O pathlength and latency, and significantly degrades system and application performance more so than any additional overhead introduced by a TM/ATF.
  - Using an ATF intercept method does not add any more overhead than using other intercept methods to log non-audited I/O data changes.
  - When *any* part of a non-audited data replication intercept or solution fails, it may negatively impact the source application's processing and even stop the application. This problem is avoidable by using TM/ATF: replication software is a completely independent component from the application and transaction management subsystem, and a failure of any part of the replication software does not directly impact the application's processing.

- *"Using a TM/ATF can lose data that needs to be replicated."* **However, it is actually the opposite.**
  - When a user-written I/O intercept mechanism is being used to log data changes for non-audited data, it is very easy to miss individual files, rows, or even entire tables when adding the intercept library to new applications or performing application updates.

[5] "Transaction Manager Facility" (TMF) is the transaction manager subsystem found on HPE NonStop Server platforms, hence the name, "AutoTMF".
[6] Please see the white paper, *Only the Truth: Debunking HPE NonStop TMF Data Protection Myths*.

  - The extra steps required to save non-audited I/O into a replication solution's log files are fraught with additional failure modes, possibly leading to data loss when failures occur (e.g., certain application failures, I/O cancellations, replication process failures, CPU failures, even system failures and restarts).

- *"Using a TM/ATF requires more disk space and retention for the audit trails."* **However, it is actually the opposite.**
  - Regardless of the collection method used, sufficient and persistent disk space must be allocated to log the change data that needs to be replicated to the target system, and the log files that hold the change data need to remain available until that data is replicated successfully. The audit trail requirements to satisfy this need when using a TM/ATF should not be substantially different from the disk needs required by the non-audited solution.

- *"Use of a TM/ATF impacts database consistency."* **However, it is actually the opposite.**
  - Without the use of transaction semantics, applications may make data changes that are not fully completed, especially if a data change requires multiple database operations, and/or is split across several applications or sub-routines.
    1. Recovery from failures in such circumstances is complex and error-prone, and is very likely to leave the database in an inconsistent state, which can be very costly. By using a TM/ATF and audited data, data integrity, data consistency, and error recovery is completely handled by the transaction manager; either all of the database operations complete or none of them do, always leaving the database in a consistent state.
    2. Using a TM/ATF also avoids the possibility of a base file becoming inconsistent with its alternate key file or index; this possibility can and does arise for non-audited files and tables.
  - There are several error scenarios discussed above whereby a non-audited application will continue processing even though the journaling method used is not operating correctly (for example, by forgetting to enable database triggers).
    1. A system or related failure will result in the inability to fully recover because there is no log information to recover from, resulting in an incorrect database. This scenario cannot arise when using a TM/ATF; if a database operation is executed against an audited file or table, and the TM/ATF has not been correctly configured and started, no transaction will be in effect and the operation will consequently fail, as will any such subsequent database operation. The application cannot "blindly" proceed with database changes on the assumption that all such changes are being logged, as is true in the non-audited case.

And, there are other less immediately tangible benefits of using a TM/ATF with audited data:

- **Future-proofing IT architectures**
  Newer and younger management teams tend to view a non-audited application and database that cannot maintain or guarantee data integrity and consistency as "antiquated" or "legacy" – out of touch with current best-practices for relational database management systems (RDBMS).
- **Advanced capabilities**
  Future technical improvements in data protection often require auditing in order to leverage these advanced capabilities. New zero data loss (ZDL) data replication technology ensures that customers' data changes are safe-stored on a target system before the source data changes are allowed to commit. Any subsequent failure of the source system will not lose any critical data, regardless of the type of failure that occurs at the source system, datacenter, or communications network. This additional capability is simply not available without the use of transactional data change auditing.

Summary

Myths persist that the use of transaction management (TM) software and audited data comes at considerable cost, which outweighs the benefits. However, far from being an impediment, use of a TM and audited data not only offers significant operational, reliability, and data integrity advantages, it also improves overall system performance and utilization, allowing for increased application processing capacity on the system.

Alternate methods of protecting data may be employed instead of using transactions, but have significant issues of their own. Additionally, migrating from an existing non-audited application to an audited application

is very simply made, without changing the application and using the facilities provided by an automatic transaction facility (ATF). Key points:

- Using TM/ATF does not impact performance; in fact, *in most cases, performance dramatically improves*.
    - Alternate methods of providing transaction-like semantics are manual, error prone, inefficient, and difficult to scale.
- Using an ATF *does not require an application rewrite*, nor typically any application modifications at all.
- Using a TM/ATF *ensures database consistency* and brings an unaudited database up-to-date, utilizing ACID transaction properties.
- Using TM/ATF *increases application sustainability and longevity* since newer applications and technologies leverage transactions for their inherent benefits.
    - Newer management may view non-audited applications that avoid transaction protection as unsophisticated, legacy, and ancient. Some are alarmed by learning this point, and demand rapid application *and hardware* replacement in favor of newer applications that use transaction protection.
- Using TM/ATF is *one of the top best practices available* for DBMS data management.

Consequently, for mission critical data, there is no valid reason to forgo the benefits of transaction protection and audited data. Doing so is disadvantageous and puts an application, data, and the business at risk. Especially since existing non-audited applications can easily gain transaction protection by configuring ATF such as AutoTMF for HPE NonStop. The HPE Transaction Monitoring Facility (TMF) provides unequaled capabilities in the industry.

International Partner Information

Global

Hewlett Packard Enterprise
6280 America Center Drive
San Jose, CA 95002
USA
Tel: +1.800.607.3567
www.hpe.com

Japan

High Availability Systems Co. Ltd
MS Shibaura Bldg.
4-13-23 Shibaura
Minato-ku, Tokyo 108-0023
Japan
Tel: +81 3 5730 8870
Fax: +81 3 5730 8629
www.ha-sys.co.jp

Gravic, Inc. Contact Information

17 General Warren Blvd.
Malvern, PA 19355-1245
USA
Tel: +1.610.647.6250
Fax: +1.610.647.7958
www.shadowbasesoftware.com
Email Sales: shadowbase@gravic.com
Email Support: sbsupport@gravic.com

Hewlett Packard Enterprise Business Partner Information

Hewlett Packard Enterprise directly sells and supports Shadowbase Solutions under the name ***HPE Shadowbase***. For more information, please contact your local HPE account team or visit our website.