Transaction Management Support in the VMS Operating System Kernel

By William A. Laing, James E. Johnson, and Robert V. Landau

Abstract

Distributed transaction management support is an enhancement to the VMS operating system. This support provides services in the VMS operating system for atomic transactions that may span multiple resource managers, such as those for flat files, network databases, and relational databases. These transactions may also be distributed across multiple nodes in a network, independent of the communications mechanisms used by either the application programs or the resource managers. The Digital distributed transaction manager (DECdtm) services implement an optimized variant of the two-phase commit protocol to ensure transaction atomicity. Additionally, these services take advantage of the unique VAXcluster capabilities to greatly reduce the potential for blocking that occurs with the traditional outside the traditional transaction processing monitor environment.

Introduction

Businesses are becoming critically dependent on the availability and integrity of data stored on computer systems. As these businesses expand and merge, they acquire ever greater amounts of on-line data, often on disparate computer systems and often in disparate databases. The Digital distributed transaction manager (DECdtm) services described in this paper address the problem of integrating data from multiple computer systems and multiple databases while maintaining data integrity under transaction control.

The DECdtm services are a set of transaction processing features embedded in the VMS operating system. These services support distributed atomic two-phase commit protocol. These features, now part of the VMS operating system, are readily available to multiple resource managers and to many applications transactions and implement an optimized variant of the well-known, two-phase commit protocol.

Design Goals

Our overall design goal was to provide base services on which higher layers of software could be built. This software would support reliable and robust applications, while maintaining data integrity.

Many researchers report that an atomic transaction is a very powerful abstraction for building robust applications that consistently update data. [1,2] Supporting such an abstraction makes it possible both to respond to partial failures and to maintain data consistency. Moreover, a simplifying abstraction is crucial when one is faced with the complexity of a distributed system.

With increasingly reliable hardware and the influx of more general-purpose, fault-tolerant systems, the focus on reliability has shifted from hardware to software. [3] Recent discussions indicate that the key requirements for building systems with a 100-year mean time between failures may be (1) software-fault containment, using processes, and (2) software-fault masking, using process checkpointing and transactions. [4]

It was clear that we that supports transaction processing, as well as timesharing, office automation, and technical computing.

The design of DECdtm services also reflects several other Digital and VMS design strategies:

o Pervasive availability and reliability. As organizations become increasingly dependent on their information systems, the need for all applications to be universally available and highly reliable increases. Features that ensure application availability and data integrity, such as journaling and two-phase commit, must be available to all applications, and not limited to those traditionally thought of as "transaction processing."

o Operating environment consistency. Embedding features in the operating system that are required by a broad range of utilities ensures consistency in two areas: first, in the functionality across all layered software products, and, second, in the interface for developers. For could use transactions as a pervasive technique to increase application availability and data consistency. Further, we saw that this technique had merit in a general-purpose operating system

instance, if several distributed database products require the two-phase commit protocol, incorporating the protocol into the underlying system allows programmers to focus on providing "value-added" features for their products instead of re-creating a common routine or protocol.

- Flexibility and interoperability. Our vision includes making DECdtm interfaces available to any developer or customer, allowing a broad range of software products to take advantage of the VMS environment. Future DECdtm services are also being designed to conform to de facto and international standards for transaction processing, thereby ensuring that VMS

  applications can interoperate with applications on other vendors' systems.

Transaction Manager- Some Definitions

To grasp the concept of transaction manager, some basic terms must first be understood:

- Resource manager. A software entity that controls both the access and recovery of a resource. For example, a database manager serves as the resource manager

- Atomicity. Either all the operations of a transaction complete, or the transaction has no effect at all.

- Serializability. All operations that executed for the transaction must appear to execute serially, with respect to every other transaction.
- Durability. The effects of operations that executed on behalf of the transaction are resilient to failures.

A transaction manager supports the transaction abstraction by providing the following services:

- Demarcation operations to start, commit, and abort a transaction
- Execution operations

  for resource managers to declare themselves part of a transaction and

  for transaction branch managers to declare the distribution of a transaction

- Two-phase commit operations for resource managers and other transaction managers to change the transaction state (to either "preparing" or for a database.

- Transaction. The execution of a set of operations with the properties of atomicity, serializability, and durability on recoverable resources.

"committing") or to acknowledge receipt of a request to change state

Benefits of Embedding Transaction Semantics in the Kernel

Several benefits are achieved by embedding transaction semantics in the kernel of the VMS operating system. Briefly, these benefits include consistency, interoperability, and flexibility. Embedding transaction semantics in the kernel makes a set of services available to different environments and products in a consistent manner. As a consequence, interoperability between products is encouraged, as well as investment in the development of "value-added" features. The inherent flexibility allows a programmer to choose a transaction processing monitor, such as VAX ACMS, and to access multiple databases anywhere in the network. The programmer may also write an application that reads a VAX DBMS CODASYL database, updates an Rdb/VMS relational database, and writes report records to a sequential VAX RMS file-all in a single transaction. Because all database and transaction processing products use DECdtm services, a failure at any point in the transaction causes all updates to be backed out and the files to be

Two-phase Commit Protocol

DECdtm services use an optimized variant of the technique referred to as two-phase commit. The technique is a member of the class of protocols known as Atomic Commit Protocols. This class guarantees two outcomes: first, a single yes or no decision is reached among a distributed set of participants; and, second, this decision is consistently propagated to all participants, regardless of subsequent machine or communications failures. This guarantee is used in transaction processing to help achieve the atomicity property of a transaction.

The basic two-phase commit protocol is straightforward and well known. It has been the subject of considerable research and technical literature for several years. [5, 6, 7, 8, 9] The following section describes in detail this general two-phase commit protocol for those who wish to have more information on the subject.

The Basic Two-phase Commit Protocol

The two-phase commit protocol occurs between two types of participants: one coordinator and one or more subordinates. The restored to their original state.

coordinator must arrive at a yes or no decision

(typically called the "commit decision") and propagate that decision to all subordinates, regardless of any ensuing failures. Conversely, the subordinates must maintain certain guarantees (as described below) and must defer to the coordinator for the result of the commit decision. As the name suggests, two-phase commit occurs in two distinct phases, which the coordinator drives.

In the first phase, called the prepare phase, the coordinator issues "requests to prepare" to all subordinates. The subordinates then vote, either a "yes vote" or a "veto." Implicit in a "yes vote" is the guarantee that the subordinate will neither commit nor abort the transaction (decide yes or no) without an explicit order from the coordinator. This guarantee must be maintained despite any subsequent failures and usually requires the subordinate to place sufficient data on disk (prior to the "yes vote") to ensure that the operations can be either completed or backed out.

The second phase, called the commit phase, begins after the coordinator receives all expected votes. Based on the subordinate votes, the coordinator decides to commit if there are no "veto" votes; otherwise, it decides to abort. The coordinator propagates the decision to all subordinates as either an "order to commit" or an "order to abort." Because the coordinator's decision must survive failures, a record of the decision is usually stored on disk before the orders are sent to the subordinates. When the subordinates complete processing, they send an acknowledgment back to the coordinator that they are "done." This allows the coordinator to reclaim disk storage from completed transactions. Figure 1 shows a time line of the two-phase commit sequence.

A subordinate node may also function as a superior (intermediate) node to follow-on subordinates. In such cases, there is a tree-structured relationship between the coordinator and the full set of subordinates. Intermediate nodes must propagate the messages down the tree and collect responses back up the tree. Figure 2 shows a time line for a two-phase commit sequence with an intermediate node.

Most of us have had direct contact with the two-phase commit protocol. It occurs in many activities. Consider the typical wedding ceremony as presented below, which is actually a very precise two-phase commit.

Official:	Will you, Mary, take John...?
Bride:	I will.
Official:	Will you, John, take Mary...?
Groomm:	I will.
Official:	I now pronounce you man and wife.

The above dialog can be viewed as a two-phase commit:

Coordinator:	Request to Prepare?
Participant 1:	Yes Vote.
Coordinator:	Request to Prepare?
Participant 2:	Yes Vote.
Coordinator:	Commit Decision.
	Order to Commit.

The basic two-phase commit protocol is straightforward, survives failures, and produces a single, consistent yes or no decision. However, this protocol is rarely used in commercial products. Optimizations are often applied to minimize message exchanges and physical disk writes. These optimizations are important particularly to the transaction processing market because the market is very performance sensitive, and two-phase commit occurs after the application is complete. Thus, two-phase commit is reasonably considered an added overhead cost. We have endeavored to reduce the cost in a number of ways, resulting in low overhead and a scalable protocol embodie`d in the DECdtm services. Some of the optimizations are described later in another section.

Components of the DECdtm Services

The DECdtm services were developed as three separate components: a transaction manager, a log manager, and a communication manager. Together, these components provide support for distributed transaction management.

The transaction manager
is the central component.
The log manager services
enable the transaction
manager to store data on
nonvolatile storage. The
communication manager

provides a location-independent interprocess communication service used by the transaction and log managers. Figure 3 shows the relationships among these components.

The Digital Distributed Transaction Manager

As the central component of the DECdtm services, the transaction manager is responsible for the application interface to the DECdtm services. This section presents the system services the transaction manager comprises.

The transaction coordinator is the core of the transaction manager. It implements the transaction state machine and knows which resource managers and subordinate transaction managers are involved in a transaction. The coordinator also controls what is written to nonvolatile storage and manages the volatile list of active transactions.

The user services are routines that implement the START_TRANSACTION, END_TRANSACTION, and ABORT_TRANSACTION transaction system services. They validate user parameters, dispense a transaction identifier, pass state transition requests to the transaction coordinator, and return information about the transaction outcome.

The branch management services support the creation and demarcation program to the transaction, to demarcate the work done in that application as part of the transaction, and finally to return information about the transaction outcome.

The resource manager services are routines that provide the interface between the DECdtm services and the cooperating resource managers. This interface allows resource managers to declare themselves to the transaction manager and to register their involvement in the "voting" stage of the two-phase commit process of a specific transaction.

Finally, the information services routines are the interface that allows resource managers to query and update transaction information stored by DECdtm services. This information is stored in either the volatile-active transaction list or the nonvolatile transaction log. Resource managers may resolve and possibly modify the state of "in-doubt" transactions through these services.

The Log Manager

The log manager provides the transaction manager with an interface for of branches in the distributed transaction tree. New branches are constructed when subordinate application programs are invoked in a distributed environment. The services are called on to attach an application

storing sufficient information in nonvolatile storage to ensure that the outcome of a transaction can be consistently resolved. This interface is available to operating system components. The log manager also supports the creation, deletion, and general management of the transaction logs used by the transaction manager. An additional utility enables operators to examine transaction logs and, in extreme cases, makes it possible to change the state of any transaction.

The Communication Manager

The communication manager provides a command/response message-passing facility to the transaction manager and the log manager. The interface is specifically designed to offer high-performance, low-latency services to operating system components. The command/response, connection-oriented, message-passing system allows clients to exchange messages. The clients may reside on the same node, within the same cluster, or within a homogeneous VMS wide area network. The communication manager also provides highly optimized local (that is, intranode) and intracluster transports. In addition, this service component multiplexes communication links across a single, cached DECnet virtual circuit to improve the performance of creating and destroying wide area links.

Transaction Processing Model

Digital's transaction processing model entails the cooperation of several distinct elements for correct execution of a distributed transaction. These elements are (1) the application programmer, (2) the resource managers, (3) the integration of the DECdtm services into the VMS operating system, (4) transaction trees, and (5) vote-gathering and the final outcome.

Application Programmer

The application programmer must bracket a series of operations with START_ TRANSACTION and END_ TRANSACTION calls. This bracketing demarcates the unit of work that the system is to treat as a single atomic unit. The application programmer may call the DECdtm services to create the branches of the distributed transaction tree.

Resource Managers

Resource managers, such as VAX RMS, VAX Rdb/VMS, and VAX DBMS, that access recoverable resources during a transaction inform the DECdtm services of their involvement in the transaction. The resource managers can then participate in the voting phase and react appropriately to the decision on the final outcome of the transaction. Resource managers must also provide recovery mechanisms to restore resources they

manage to a transaction-
consistent state in the
event of a failure.

Integration in the Operating System

The DECdtm services are a basic component of the VMS operating system. These services are responsible for maintaining the overall state of the distributed transaction and for ensuring that sufficient information is recorded on stable storage. Such information is essential in the event of a failure so that resource managers can obtain a consistent view of the outcome of transactions.

Each VMS node in a network normally contains one transaction manager object. This object maintains a list of participants in transactions that are active on the node. This list consists of resource managers local to the node and the transaction manager objects located on other nodes.

Transaction Trees

The node on which the transaction originated (that is, the node on which the START_TRANSACTION service was called) may be viewed as the "root" of a distributed transaction tree. The transaction manager object on this node is usually responsible

The transaction identifier dispensed by the START_TRANSACTION service is an input parameter to the branch services. This parameter identifies two concerns for the local transaction manager object: (1) to which transaction tree the new branch should be added, and (2) which transaction manager object is the immediate superior in the tree.

Resource managers join specific branches in a transaction tree by calling the resource manager services of the local transaction manager object.

Vote-Gathering and the Final Outcome

When the "commit" phase of the transaction is entered (triggered by an application call to END_TRANSACTION), each transaction manager object involved in the transaction must gather the "votes" of the locally registered resource managers and the subordinate transaction manager objects. The results are forwarded to the coordinating transaction manager object.

The coordinating transaction manager object eventually informs the locally registered for coordinating the transaction commit phase of the transaction. The transaction tree grows as applications call on the branch management services of the transaction manager object.

resource managers and the subordinate transaction manager objects of the final outcome of the transaction. The subordinate transaction manager objects, in turn, propagate this information to locally registered resource managers as well as to any subordinate transaction manager objects.

Protocol Optimizations

The DECdtm services use several previously published optimizations and extend those optimizations with a number that are unique to VAXcluster systems. In this section we present these general optimizations, a discussion of VAXcluster considerations, and two VAXcluster-specific optimizations.

General Optimizations

The following sections describe some previously published optimizations.

Presumed Abort. DECdtm services use the "presumed abort" optimization. [8, 9] This optimization states that, if no information can be found for a transaction by the coordinator, the transaction aborts. This removes the need to write an abort decision to disk and to subsequently acknowledge the order to abort. In addition, subordinates that do not modify any data during the transaction (that is, they are "read only"), avoid writing information to disk or participating in the commit phase.

a "commit" record upon receipt of an order to commit. This latter record is written so that the coordinator need not be asked about the commit decision should the intermediate node fail. This refinement isolates the intermediate node's recovery from communication failures between it and the coordinator.

Performance is enhanced when the DECdtm services write the commit record on an intermediate node in a "nonurgent" or "lazy" manner. [10] The lazy write buffers the information and waits for an urgent request to trigger the group commit timer to write the data to disk. Typically, this operation avoids a disk write at the intermediate node. The increase in the length of time before the commit record is written is negligible.

One-Phase Commit. A key consideration in the design of the DECdtm services was to incur minimal impact on the performance of Digital's database products. We exploited two attributes to achieve this goal. First, all current users are limited to non-distributed transactions (those that involve only a single subordinate). Second, the two-phase

Lazy Commit Log Write. The DECdtm services can act as intermediate nodes in a distributed transaction. In this mode, they write a "prepare" record prior to responding with a "yes vote." They also write commit protocol requires that all subordinates respond with a "yes vote" to commit the transaction. This allows a highly optimized path for single subordinate transactions. Such transactions require no writes to disk by the DECdtm services and execute in one phase. The subordinate is told that it is the only voting party in the transaction and, if it is willing to respond with a "yes vote," it should proceed and perform its order to commit processing.

VAXcluster Considerations

The optimizations listed above (and others not described here) provide the DECdtm services with a competitive two-phase commit protocol. VAXcluster technology, though, offers other untapped potential. VAXcluster systems offer several unique features, in particular, the guarantee against partitioning, the distributed lock manager, and the ability to share disk access between CPUs. [11]

Within a VAXcluster system, use of these unique features allows the DECdtm services to avoid a blocked condition which occurs during the short period of time when a subordinate node responds with a "yes vote" and communication with its coordinator is lost. Normally, the subordinate is unable to proceed with that transaction's commit until communications have communication is lost, a subordinate node knows, as a result of the guarantee against partitioning, that its coordinator has failed. Because a subordinate node can access the transaction log of the failed coordinator, it may immediately "host" its failed coordinator's recovery. Communications to the hosted coordinator are quickly restored, and the subordinate node is able to complete the transaction commit.

VAXcluster-specific Optimizations

Once the blocking potential was removed from intra-VAXcluster transactions, several additional protocol optimizations became practical. The optimizations described in this section are dynamically enabled if the subordinate and its coordinator are both in the same VAXcluster system.

Early Prepare Log Write. As noted earlier, an intermediate node must write a "prepare" record prior to responding with a "yes vote." The presence of this record in an intermediate node's log indicates that the node must get the outcome of the transaction from the been restored.

Outside a VAXcluster system, the DECdtm services would indeed be blocked. If, however, the subordinate and its coordinator are in the same VAXcluster system, this will not occur. If coordinator and, thus, it

is subject to blocking. Therefore, the prepare record is typically written after all the expected votes are returned, which adds to commit-time latency.

The DECdtm services are free from blocking concerns within a VAXcluster system; the vast majority of transactions do commit. This factor prompted an optimization that writes a prepare record while simultaneously collecting the subordinate votes. This reduces commit-time latency.

No Commit Log Write. The lazy commit log write optimization described above causes the intermediate node's commit record to be written and, thus, minimizes the potential for blocking should the intermediate node fail. Note that this is not a concern for the intra-VAXcluster case. Therefore, no commit record is written at the intermediate node.

Performance Evaluation

Table 1 describes the message and log write costs of the DECdtm services protocol and compares it to the basic two-phase commit protocol, as well as to the standard presumed abort variant previously described. [8,9]

Table 1

Logging and Message Cost by Two-phase Commit (2PC) Protocol Variant

	Coordinator		Intermediate	
Coordinator	Log_Write	Message	Log_Write	Message
Basic 2PC:	2, 1 forced	2N	2, 2 forced	2
Presumed Abort:	2, 1 forced	2N	2, 2 forced	2
(RO intermediate)	2, 1 forced	1N	0	1
Normal DECdtm:	2, 1 forced	2N	2, 1 forced	2
(RO intermediate)	2, 1 forced	1N	0	1
Intracluster:	2, 1 forced	2N	1, 1 forced*	2
(RO intermediate)	2, 1 forced	1N	0	1

DECdtm_1PC:_________0_____________1___________-_____________-______

Notes:
Log writes are total writes, forced. The table entry 2,1 forced means that there are two total log writes, one of which is forced. A forced write must complete before the protocol makes a transition to the next state.

RO means Read Only.

Where a message is listed as xN, N represents the number of intermediates that fit that category.
* Forced means that the log write is initiated optimistically; thus, it has lower latency.

Ease-of-use Evaluation

A primary goal in providing transaction processing primitives within the VMS kernel was to supply many disparate applications with a straightforward interface to distributed transaction management. This contrasts with most commercially available systems, where distributed transaction management functionality is available only from a transaction processing monitor. This latter form restricts the functionality to applications written to execute under the control of the transaction processing monitor, and it effectively precludes other applications from making use of the technology.

From the outset of development, we endeavored to provide an interface that was suitable for as many applications as possible. We made early versions of the DECdtm services available within Digital to decrease These products are VAX Rdb/VMS, VAX DBMS, VAX RMS Journaling, VAX ACMS, DECintact, VAX RALLY, and VAX SQL.

In general, the modifications to these products have been relatively minor, as might be inferred from the short time it took to make the required changes. Based on this experience, we expect third-party software vendors to rapidly take advantage of the DECdtm services as they become available as part of the standard VMS operating system.

To incorporate the DECdtm services into a recoverable resource manager, the existing internal transaction management module with calls to the DECdtm services must be replaced. The resource manager must also be modified to correctly respond to the prepare and commit callbacks by the DECdtm services. Further, the recovery logic of the resource manager must be modified to obtain from the DECdtm services the state of "in doubt" transactions.

Example of DECdtm Usage

The model and pseudocode presented in Figure 4 the "time to market" for software products that wished to exploit distributed transaction processing technology. As of July 1990, at least seven Digital software products have been modified to use the DECdtm services.

illustrate the use of DECdtm services in a simple example of a distributed transaction. The transaction spans two nodes, NODE_A and NODE_B, in a VMS network. During the course of the transaction, recoverable resources managed by resource managers, RM_A and RM_B, are modified. Two "application" programs, APPL_A and APPL_B, that run on NODE_A and NODE_B, respectively, make normal procedural calls to RM_ A and RM_B. APPL_A and APPL_B use an interprocess communication mechanism to communicate information across the network. The DECdtm service calls are prefixed with a dollar sign ($).

The code for the resource managers, RM_A and RM_B, is identical with respect to calls for the DECdtm services. The resource manager routine, ROUTINE RM_A_EVENT, is invoked by the DECdtm services during transaction state transitions.

Conclusions

The addition of a distributed transaction manager to the kernel of the general-purpose VMS operating system makes distributed transactions available to a wide spectrum of applications. This design and implementation was accomplished with comparative ease and with quality performance. In addition to utilizing the most commonly described optimizations of the two-phase commit protocol, we have used optimizations that exploit some of the unique benefits of the VAXcluster system.

Acknowledgments

We wish to gratefully acknowledge the contributions of all the transaction processing architects involved, and in particular Vijay

Trehan, for delivering to us an understandable and implementable architecture. We also extend our thanks to Phil Bernstein for his encouragement and advice, and to our initial users, Bill Wright, Peter Spiro, and Lenny Szubowicz, for their persistence and good nature. Tony Hasler, Mark Howell, Dave Marsh, Julian Palmer, Kevin Playford, and Chris Whitaker.

References

1. R. Haskin, Y. Malachi, W. Sawdon, and G. Chan, "Recovery Management in Quicksilver," ACM Transactions on Computer Systems, vol. 6, no. 1 (February 1988).

2. A. Spector et al., Camelot: A Distributed Transaction Facility for Mach and the Internet-An Interim Report (Pittsburgh: Carnegie Mellon University, Department of Computer Science, June 1987).

3. W. Bruckert, C. Alonso, and J. Melvin, "Verification of the First Fault-tolerant VAX System," Digital Technical Journal, vol. 3, no. 1 (Winter 1991, this issue): 79-85.

4. J. Gray, "A Census of Tandem System Availability between 1985 and 1990," Tandem Technical Report 90.1, part no. 33579 (January 1990).
5. P. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery

Finally, and most importantly, we would like to thank all the DECdtm development engineers and the others who helped ship the product: Stuart Bayley, Cathy Foley, Mike Grossmith, Tom Harding,

in Database Systems (Reading, MA: Addison-Wesley, 1987).

6. J. Gray, "Notes on Database Operating Systems," In Operating Systems: An Advanced Course (Berlin: Springer-Verlag, 1978).

7. B. Lampson, "Atomic Transactions," In Distributed Systems-Architecture and Implementation: An Advanced Course, edited by G. Goos and J. Hartmanis (Berlin: Springer-Verlag, 1981).

8. C. Mohan, B. Lindsay, and R. Obermarck, "Transaction Management in the R* Distributed Database Management System," ACM Transactions on Computer Systems, vol. 11, no. 4 (December 1986).

9. C. Mohan and B. Lindsay, "Efficient Commit Protocol for the Tree of Processes Model of Distributed Transactions," Proceedings of the 2nd ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing (Montreal: August 1983).

10. D. Duchamp, "Analysis of Transaction Management Performance," Proceedings of the Twelfth ACM Symposium on Operating Systems Principles (Special issue), vol.23, no.5 (December 1989): 177-190.

11. N. Kronenberg, H. Levy, and W. Strecker, "VAXclusters: A Closely-Coupled Distributed System," ACM Transactions on Computer Systems, vol. 4, no. 2 (May 1986).