Lightweight Recoverable Virtual Memory

Satyanarayanan, Mashburn, Kumar, Steere, Kistler

Summary by : Shafeeq Sinnamohideen

Transactions, by guaranteeing the ACID properties, greatly assist in building fault-tolerant systems. While databases have long offered transactional properties, a simpler and smaller scale mechanism is desirable for use by simpler applications. These applications may include file or object servers, long computations, or any process that stores state that would be inconvenient to rebuild after a failure. The authors propose a system that offers transactional guarantees on regions of a process's address space while preserving other desirable properties of performance, functionality, and ease of use.

Virtual memory makes a good medium to offer ACID properties on, for several reasons. First, it is the most easily manipulated storage available, and almost every application stores some (or all) of its state of in memory. Memory is, however, volatile, so its contents do not survive process termination or a system reboot. If it were recoverable, so that its contents could be restored later, an application could simply continue from its last known state, without the application implementing any failure-recovery on its known. This is particularly useful for the metadata of filesystems, since updates to the metadata can fail at any time, but the filesystem won't be usable unless the metadata is consistent. The solution is to store the metadata in RVM, and perform all updates within transactions. This insures that the metadata will remain consistent, and is what the authors originally implemented for the Coda fileserver, using the Camelot transactional system.

Their experience led them to conclude that while RVM through Camelot made failure recovery easy, it had several drawbacks, and created Lightweight RVM to eliminate these drawbacks. The first major drawback was poor performance, largely due to Camelot's reliance on Mach and intensive IPC. Camelot also attempted to provide many other, in this case unused, features. The authors' solution was to trim away many of the features of RVM, and instead require the application or an intermediate library to support them. Of the ACID properties, no attempt is made to preserve Independence (serializablilty), and Durability is guaranteed only as far as the OS guarantees the durability of a file on disk. Additionally, nested and distributed transactions are not supported. Structuring RVM as a library avoids IPC and avoiding a close alliance with the VM system aids portability.

The architecture of RVM was designed to support the minimum of features required to make use of it. Persistent storage is accomplished by a backing store, usually a file on disk, that corresponds to a *segment* of memory. *Regions* of memory are mapped into VM space. A region is a page granularity chunk of a segment that usually contain related objects. The primitives offered are to begin, end, and abort transactions, manage memory mappings, and to indicate the range of addresses that a transaction will modify. A typical transaction would consist of a begin_transaction, a setrange(&X) to indicate that X will be modified, a modification of X, and an end_transaction. A transaction that commits will be logged, and the data queued for writing to the persistent backing store. To further improve performance, different levels of commit guarantees are offered. *No-flush* transactions are logged, but the log is not flushed disk. This means that some committed transactions will not be durable if the system crashes before the log is flushed again. There are cases where this is acceptable, namely the case where a set of related transactions is not meaningful unless all of them are. In this case, all but the last transaction will be *no-flush* and the last will flush the log, making all of the transactions durable. The *no-restore* transaction is similarly useful since it allows the application to specify that it does not require the transaction to be undoable in case of a voluntary abort.

After adapting Coda to Lightweight RVM, the authors conclude that the simpler design and transaction optimizations do improve performance as expected, and that it does offer enough functionality to be effective as a tool to improve fault-tolerance. The weaknesses of application level fault tolerance schemes also made apparent because they find that the most frequent cause of inconsistency is the application programmer failing to specify all the addresses involved in a transaction. They conclude that language primitives for persistent memory, such as offered by Avalon, would eliminate this problem.