



Jay Almarode

# Fault Tolerance

**P**roduction applications need to be protected against the possibility of catastrophic failure. Disks fill up...hardware fails...operating systems crash... networks go down...but with proper foresight these situations do not have to lead to a loss of objects. This column describes mechanisms to achieve fault tolerance and how to recover when the bad things happen.

There are at least two ways in which systems achieve fault tolerance. One is to prevent the system from going down in the first place; the other is to bring the system back to a consistent state if it does go down. The typical way to avoid a system from going down is to duplicate, or mirror, the state of the object repository on different hardware, so that if the primary piece fails, the system will automatically switch over to the duplicate. To bring the system back up when it goes down, most transaction-based systems employ backup files and transaction logs to help the system recover to a consistent state. These same approaches apply to Smalltalk applications.

In multi-user Smalltalk, the object repository is manifested by one or more files (or possibly raw disk partitions) called extents. These are where the state of objects ultimately reside. For fault tolerance, as well as performance reasons, information about objects may first be written to other files, called transaction logs. Transaction logs contain information to re-do transactions that have been committed to the repository.

When a transaction is committed, all that's necessary is to completely write the transaction log records to consider the transaction complete. The extent files do not have to be updated with new or changed objects immediately, which can improve overall system performance and transaction throughput.

To avoid a multi-user Smalltalk system from going down, the system administrator can specify that the extent files are to be replicated. In addition to allocating extent files across multiple disk drives on different machines for performance and clustering reasons, the system administrator can allocate the replicated extents

on multiple disk drives as well. While the system is running, if a client or server process should encounter a read error on a primary extent file, the corresponding replicated extent file is automatically used instead.

In GemStone, the system administrator creates replicated extents in two ways. One way is to specify them in the configuration file used by the server process at startup time. Another way is to dynamically create new replicates at runtime by sending the message `SystemRepository createReplicateOf: extentFilename named: replicateFilename`. In both cases, you are mapping a primary extent file to a corresponding replicated extent. The replicated extent should be located on a different disk spindle to reduce IO contention, as well as to provide fault tolerance.

Even if the object repository is replicated for automatic switchover, it is still good practice to plan for recovery if the system goes down entirely. This planning involves deciding how often to back up the system, and how quickly the system must be back online. For 7 x 24 production applications, it is imperative that backups be performed while the system is online and other users are logged in. Since backups may require considerable resources for large object repositories, it is desirable to limit the IO rate of the process performing the backup to reduce its interference with other sessions.

To plan for backups and recovery, it is necessary to understand how transaction logging works. As mentioned earlier, transaction logs contain the information to re-do transactions that have been committed. Transaction logs are used to recover from an unexpected shutdown or to roll forward from a backup file. When configuring a system, an administrator supplies multiple locations where transaction logs are to be written. Therefore, if one disk becomes full, the system can automatically switch over to the next location. It is also possible to configure the maximum size of each transaction log file to balance the utilization of the disk resources. Transaction logs can be replicated to provide the same benefits as replicated extents.

Recall that objects may not be written immediately to extent files. To force the information in transaction logs to be written to the extent files, an administrator performs a checkpoint. Performing a checkpoint reduces the number of transaction logs that have to be applied when the sys-

---

Using Smalltalk since 1986, Jay Almarode has built CASE tools, interfaces to relational databases, multi-user classes, and query subsystems. He is currently a senior software engineer at GemStone Systems Inc., and can be reached at [almarode@gemstone.com](mailto:almarode@gemstone.com).

tem recovers from a crash, where the extent files are not damaged.

Transaction logging can be set up to handle two kinds of recovery situations. In the first situation, the system has unexpectedly shut down, but the extent files are not corrupt. To recover the object repository to the last committed state, only transaction log records that were written since the last checkpoint are applied. This mode of transaction logging is called partial logging, since not all transaction logs are needed to recover. To free up space, an administrator can remove any log files written prior to the most recent checkpoint, usually leaving the current log and the one immediately before.

In partial logging mode, the frequency of performing a checkpoint helps control how long it takes to recover the system. In GemStone, the system can be set up to automatically perform checkpoints at specific intervals by setting a configuration parameter; or, a checkpoint can be performed explicitly by sending `System checkpoint`. When the system is in partial logging mode, a checkpoint is also triggered when any transaction writes a log record whose size is greater than some configurable threshold.

The second kind of recovery situation occurs when the system crashes and the extent files are corrupt. In this case, the object repository must be recovered from backup files. To recover from this situation, all transaction logs that were written since the backup are needed. This type of recovery is supported by configuring the system to be in full logging mode. Full transaction logging should be used for production applications, to guarantee recoverability in the face of media failure.

One factor determining the time to recover from a backup is the frequency of backups performed. To perform a backup of the object repository in GemStone, a user performs the message: `SystemRepository fullBackupTo: aFileOrDevice Mbytes: aByteLimit`. The first argument specifies the file, raw partition, or device where the backup is to be created. The second argument specifies a byte limit so that you can create multiple backup files by limiting the size of each part.

When the first backup file is finally written, you continue writing the next part of the backup with the message `SystemRepository continueFullBackupTo: aFileOrDevice Mbytes: aByteLimit`. Since the backup procedure may consume system resources, a user can control the IO rate of the current backup session by sending `System configurationAt: #GemIOLimit put: 10`. This example allows a maximum of 10 IOs per second.

To restore the object repository, a system administrator first starts a server process on a new object repository. Then the restore operation is performed by sending `SystemRepository restoreFromBackup: backupFilename`. At this

point, the state of the repository is the same as when the backup file was created. Now the administrator can apply transaction logs to roll forward from the state of the backup to the state of the last committed transaction.

To find out the first transaction log file needed, the administrator sends `SystemRepository restoreStatus` to get the file id of the log file. When transaction log files are created, they are given a filename that includes an increasing numerical file id so that the sequence of file creation is evident. This helps in determining which transaction log files to archive (i.e. move somewhere else), and which are needed for restoration. If the needed transaction log files have been archived, the administrator sends `SystemRepository restoreFromLog: aTranLogFilename` to explicitly specify their location. If the remaining log files are located in their original location, then the administrator performs `SystemRepository restoreFromCurrentLogs`. The administrator sends the message `SystemRepository`


`commitRestore` to finish the restoration and allow other users to log in. It is also possible to restore to a specific point in time, by sending `SystemRepository timeToRestoreTo: aDateTime`, before restoring from transaction logs.

Using transaction logs, a 'warm' backup system can be built with the mechanisms described above. A 'warm' backup system is a duplicated object repository not kept in sync with the primary repository in

real time by the underlying system. Instead, the duplicated object repository is explicitly synchronized with the primary repository at specific time intervals. The advantage of a warm backup is that it places no burden on the primary system to perform IO to multiple locations; the disadvantage is that the warm backup is only up-to-date based on the last time it was explicitly synchronized with the primary system.

To build a warm backup system, a server process is started up on a copy of the primary object repository (or it could be started up on a new repository, then restored from a backup file of the primary repository). This is the warm backup server. Next, a process is spawned that continually looks for new transaction logs being created by the primary server.

When a new transaction log file is created, this process can copy the previous log file to the backup site and perform `SystemRepository restoreFromLog: aTranLogFilename`. If the primary repository goes down, the warm backup site performs `SystemRepository commitRestore`, and it is ready for duty.

Fault tolerance is a necessary consideration for production applications. System administrators need to plan for disaster and have the mechanisms in place to recover. Duplicated object repositories and transaction logging are two mechanisms that provide the functionality needed for 7 x 24 applications. 

*“One factor determining the time to recover from a backup is the frequency of backups performed..”*