From: Gerry Houlder
SendTo: X3T10 Committee
Date: 08/10/94 12:51:42 PM
Subject: System recovery on 3rd party commands   X3T10/94-184r0

This issue needs to be addressed at the XOR commands Special Interest Group meeting planned for Monday morning Sept. 12 (in Houston). The reflector can also be used to provide comments and suggestions.

The XDWRITE command (XOR Data Write) is being architected so the RAID controller can send write data to drive A. Drive A will generate XOR data by reading data from its own media and XORing it with the data received with the XDWRITE command. Next the drive will send an XPWRITE (XOR Parity Write) command to drive B (the parity drive) that includes the XOR data. When drive B completes its XOR and write operations, it returns status to drive A. Now drive A can return good status to the RAID controller if both drive operations succeed.

If the XDWRITE operation on drive A fails, it can be reported and recovered in a manner similar to existing WRITE commands. If drive A is reporting CHECK status because of a failing XPWRITE operation on drive B, however, there are extra complications.

The normal SCSI philosophy is that the initiator of any command has complete responsibility to recovery if the command fails. This assumption breaks down for RAID configurations that use 3rd party drives to initiate part of the operation. This is because there are serious drive failures that can only be "recovered" by removing the failing drive from the array and replacing it with a good spare. This can only be done by the RAID controller (which maintains the redundancy group tables) and not by a 3rd party drive (which has no knowledge of array configuration or spare availability). Such serious failures will happen and a procedure to transfer the recovery responsibility to the RAID controller must be defined.

Several methods to do this suggest themselves:
(A) Drive B's contingent allegiance to drive A must be ended before the RAID controller can issue any recovery commands. Drive A could simply clear the contingent allegiance, report CHECK status on the associated XDWRITE command, and let the RAID controller do the cleanup. The disadvantage here is that drive B has no safeguard in place to prevent other commands (possibly from other 3rd party drives, drive A again, or other RAID controllers) from being executed. These other commands could make recovery more difficult than if they were prevented from executing until the recovery procedure has completed.

(B) Drive A can transfer its "auto contingent allegiance" directly to the RAID controller's address instead of its own. There currently is no method to do this, but this is the most desirable path. The contingent allegiance mechanism is ideally suited to preventing other commands from running on the failing drive until the recovery procedure is done. This method would even lock out drive A until the recovery procedure completes. The RAID controller can clear the contingent allegiance normally when recovery is complete.
    I suggest defining a new command to "Transfer Contingent Allegiance". The command needs only to include the address of the device to transfer allegiance to (the RAID controller in this case). Now drive A can let the RAID controller know that it has the allegiance. It can return CHECK status for the XDWRITE command and define sense data that indicates that allegiance has been transferred. Alternatively, a new status response could be defined that says "3rd party CHECK condition, and the 3rd party

device has allegiance to you". This would make sure the RAID controller gets the sense bytes to read the device address. Or should the controller be required to figure out the 3rd party address from the XDWRITE command that ended with CHECK status?

(C) Drive A can issue a 3rd party RESERVE command to drive B, reserving it to the RAID controller. This would lock out other devices until the RAID controller completes its recovery action. The RAID controller would still have to learn the failing drive address from the failing XDWRITE command bytes or resulting sense bytes, just like in case (B). The problem with this method occurs when the RAID controller is done with recovery and tries to release the 3rd party reservation. The existing SCSI rules prevent the RAID controller from releasing its own reservation, only the device that sent the 3rd party RESERVE command (drive A) can release the reservation.

There are several possible ways to fix this problem:

(1) Change the 3rd party reservation rules to allow the "3rd party device" to release its own reservation. This looks like a minor rule change with few complications for backwards compatibility.

(2) Create an "indirect 3rd party release" option on the RELEASE command. This would be a command that the RAID controller would send to drive A that tells it to send a regular 3rd party RELEASE command to drive B. Drive A would return Good status after the 3rd party RELEASE command gets Good status. This seems unduly complicated, but it could work.

(3) Any better fixit suggestion?

The disadvantage of using reservations is that the system might choose to use reservations for its own needs. These needs could conflict with error recovery needs. However, I think it is unlikely that a system that makes extensive use of 3rd party devices to help with certain operations could use reservations for anything but error recovery without causing problems anyway.

(D) Any other suggestions?