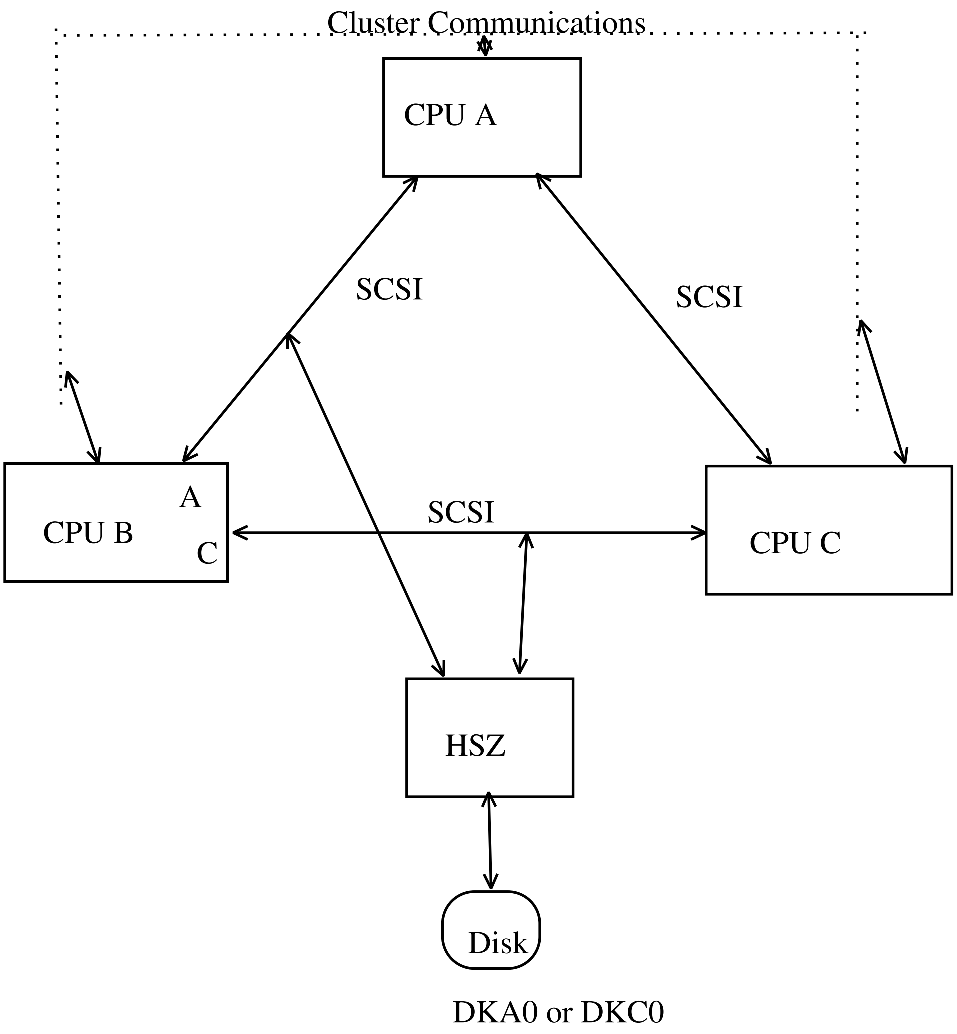
From: Glenn C. Everhart
Date: 27-Jan-1996

This note is about the design of the switching system. I would like to clarify if possible some issues.

My original design spec design speaks about a switch intercept that grabs control, crudely, just "below" the start - IO so that I/O done pointing to a UCB still uses that UCB, but the I/O
in fact gets rerouted to one or more underlying paths. That way, any existing pointers to the underlying UCB remain valid, and the device name associated with this UCB now denotes the entire tuple of paths.

Consider the following configuration:



Now CPU A can see the disk, let's say as DKA0; B can see DKA0 and DKC0, and CPU C can see DKA0: served or DKC0:.

Now suppose the system manager has specified the preferred path to be the one used for DKC0:.

If now CPU A boots from the DKA0: path that is local to it, it can tell that this is not the preferred path, but has no clue how to find the name of the other path if it is first up, unless it has a configuration file or just picks a name. If it leaves the name DKA0: in place, it will conflict with other cluster members later, so needs to know what to call the disk.

It is possible to load switching code, since we know the device is dualpathed and an HSZ, and the switching code CAN rename the device, either by setting up a new UCB of its own, or by setting up a new DDB if need be and naming the device in that way. In either case, there are only two ways to get the name:

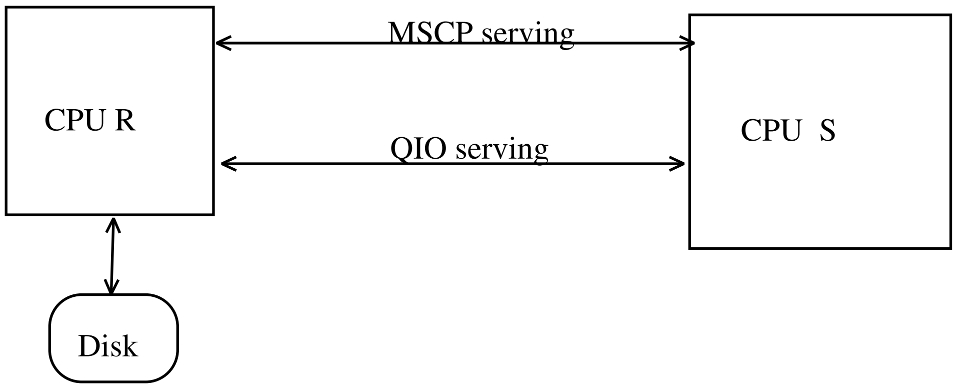
1. Algorithmically, via for example some counter maintained at config time, or
2. From a configuration file (which can be initially populated with names found algorithmically.)

The latter is what I think is needed, using the former at first to set the config file up, and checking the results as noted in my Design Spec.

For this situation, either a brand new UCB can be used, or an existing one can be used, since we can find out before any use that a dual path situation exists that a dual path exists and get switching code loaded which can fix things. Note that if CPU A boots the wrong side, it cannot rename DKA0: to DKC0: without config file, since it has no idea natively where the other side of the HSZ is, or what bus it's on or how it's named. If the other path is preferred, it isn't at CPU A locally at all, and its name is controlled by something else. CPU A has to know the right name from something that will be used for the tuple, or print a message and halt, to avoid data corruption. (And the naming must be checked early on once a cluster is up, to be sure everyone agrees on it.)

What I need to know, though, is whether an entirely new UCB is connected and to be made visible, or whether an existing UCB should be made to switch.

In the case where both QIOserver and MSCP server exist, a single node can see both paths. Consider the following configuration:



Note that CPU S will see two paths to the disk, both named the same, from the two servers. To avoid this, the MSCP server and QIO server would have to look for one another's paths and try to hide their new ones if they pre-exist, since by the time QIOserver comes up, the MSCP served path may have become active and have pointers all over.

This might mean a server must look for local paths direct connected, or paths of the other server. DKdriver and DUdriver would have to look for multiple kinds of served paths. If this were done, only one path might show up on CPU S, but getting MSCP to/from QIOserver failover would be a bear to do.

It is not feasible to use a wholly different device name here. A new UCB could create one and be renamed to the same as an old one, but if anyone had pointers to the old UCB, switching would be impossible. The name is controlled from CPU R above.

It is however feasible to use an intercept that forces a UCB to switch in this case. Existing UCB pointers remain valid and we "just" switch I/O around.

The need to support this sometime is a problem with the entirely-separate-UCB scheme so far as I can see. If dkdriver, dudriver, mscp server,and qioserver all have to cooperate to eliminate extra UCB paths, the need for each to check all others begins to look like a future combinatorial explosion as well.

The only way to use a new UCB to represent the switched paths would have to be that the switch code gets loaded first, in basically all cases, whether needed or not. That is akin to making EVERY disk on a system a one volume shadowset and seems wasteful. On the other hand, using the first UCB to show up avoids the problem with active UCBs gaining switching, and needs the switch code only where switching is to be done.

I will mention in addition that handling things like IO$_DIAGNOSE can work with something like this test:

When doing the start-MV processing, mark each IRP in the queue (if not done already) as to which driver's FDT routines it used. At start-io, if the driver number is the same path number, let the I/O by; it is using the path it was initially processed on. If not, then if the FDT routine is at the same address (or if this is a special case that's OK anyway, as write logical on a disk gets checked for shadow set member for example) let it by. Otherwise flag illegal operation, or operation derailed, or some such and junk it. That way DK to DK and DU to DU switching would basically pass everything, and cross switching would pass whatever made sense, and the number of special cases can be kept small.

The code needed to handle either strategy pretty well exists, but it looks to me that the strategy of intercepting a UCB is what is needed, to avoid having to redo the whole thing down the road not too far.

As for UCB status, I'm inclined to have some UCB areas that are just maintained from the currently active underlying UCB, and to track individual statistics elsewhere and teach some display(s) how to show them. For simple cases MSCP handles this seems ok now. For more complex I/O I think it can be managed in the future; there are only a few cells known outside the driver anyway that can report status. Status needed by the underlying driver itself will be there since underlying control blocks are kept carefully in place.

Glenn Everhart
1/27/1996