the client side, we built a local, caching, write-behind proxy. When a caller asks for an image, this client image repository either returns it directly from local cache or downloads the file into local cache, and then returns it. Either way, callers remain blissfully ignorant.

Likewise, when adding images on the client, the client image repository uploads it to the server. We use a pool of threads to run background transfers so the user doesn't have to wait on uploads.

Both the client and server repositories are heavily multithreaded. We created a system of locking called "reservations." Reservations are a soft form of collaborative locking. When a client wants to add an image to the repository, it must first request and hold a "write reservation." This way, we can be sure that no other thread is reading the image file when we issue the reservation. Readers have to acquire a "read reservation," naturally.

Although we did not implement distributed transactions or two-phase commit, in practice there is only a small window between when the client image repository grants a write reservation and when the server side grants a corresponding write reservation. When that second reservation is granted, we can be confident that we will avoid file corruption.

In practice, even lock contention is rare. It requires two photographers at two different workstations to access exactly the same customer's session. Still, there are several workstations in every studio, and each workstation has many threads, so it pays to be careful.

NIO image transfer

Obviously, that leaves the problem of getting the images from the client to the server. One option we considered and rejected early was CIFS—Windows shared drives. Our main concern here was fault-tolerance, but transfer speed also worried us. These machines needed to move a lot of data back and forth, while photographers and customers were sitting around waiting.

In our matrix of off-the-shelf options, nothing had the right mix of speed, parallelism, fault-tolerance, and information hiding. Reluctantly, we decided to build our own file transfer protocol, which led us into one of the most complex areas of Creation Center. Image transfer became a severe trial, but we emerged, at last, with one of the most robust features of the whole system.

I had some prior experience with Java NIO, so I knew we could use it to build a blazing-fast image transfer mechanism. Building the NIO data transfer itself wasn't particularly difficult. We used the common leader-follower pattern to provide concurrency while still keeping NIO selector operations on a single thread.

Although the protocol wasn't difficult to implement, there were a number of nuances to deal with:

- Either end can close a socket, particularly if the client crashes. Sample code never deals with this properly.