Backup Strategies for Molecular Dynamics

An Interview with Doug Hughes

RIK FARROW

Doug Hughes is the manager for the infrastructure team at D. E. Shaw Research, LLC in Manhattan. He is a past LOPSA Board member and an upcoming LISA '11 conference Co-Chair. Doug fell into system administration accidentally after acquiring a BE in Computer Engineering, and decided that it suited him.
doug@will.to

Rik: Backup design is important for any organization that values its data. Most sysadmins design backup systems with a focus on efficiency of backup operations as well as swift recovery of lost data. These criteria often mean full backups on the weekends and incremental backups in the evenings, after most work is done, so backups do not cut into operational performance. But what you are doing at D. E. Shaw seems to require different strategies.

Doug: I think a lot of organizations with lots of files and lots of data are struggling with the same issues. When you have 100s of TBs or PBs of data and 100s of millions of files, the traditional strategies break down. No longer can you afford to use dump to scan the file system every time you want to take a backup. Using a traditional find on a traditional file system could take many hours or even days, while the data is changing underneath you. So we've had to investigate, evaluate, and rely on different filesystem features to make this a tractable problem.

In general, I think that because of the geometric explosions of data as the cost of storage has decreased and computing power has advanced with Moore's Law, the task of doing backups has become much more complex. We started out with Linux file servers a few years ago using rdiff-backup. That was good enough for a time. From there we migrated to ZFS for its integrity guarantees and other features. ZFS "knows" what has changed between any two snapshots, and that's a very useful property. Unfortunately, ZFS still has fairly significant bugs and is missing some important features, plus there are various vendor support issues. One only need look at the monthly ZFS patch list to get a little bit nervous. We've experienced many of the bugs firsthand. Luckily, none of them has resulted in any data loss.

The big win about ZFS is that the checksum on every block protects you from mendacious disks and other components (memory, CPU, controllers, etc.). We've actually had ZFS correct data for us. The disk was certifying the data as good and returning it back through the controller, CPU, etc., all the way back to user space where ZFS noticed that the block from that disk didn't match the checksum. We've replaced a couple of disks (out of hundreds over the last couple of years) that have "silently" corrupted data. When you need to know that your data is exactly as you wrote it, this is comforting.

ZFS isn't without its problems, though. First, there are a lot of bugs, though usually with regard to zfs send, zfs recv, and intermittent, vexing performance issues. In the realm of design deficiencies, there's its inability to sanely expand a storage pool once created. Once you define a storage pool, it's very hard to expand that pool. You

can add additional RAID groups to it (RAID5, 6, even 7), but those just get stripped into the rest of the existing pool, so you end up with a bunch of full RAID chunks and an empty one getting written to equally. There are ways you can probably mitigate this by reading and rewriting every block in the file system, but that's heavyweight with 10s of TBs of data. Veritas VxFS and VxVM were awesome in this regard. They still exist but, in my opinion, the product line has not gotten enough emphasis since the purchase of Veritas by Symantec.

So we started looking for a new file system. Safety is one important property, size is another. There are few things as safe as ZFS, but some will at least check that the parity matches on read. That's not as good as block checksums, but better than nothing. Speed is also obviously important, but we're here to talk about backups. The ability to do tiering is yet another important consideration, since our data is continuously rolling, making the fresh data old and the old data ancient in a matter of months.

It's not critical to us that backups follow exactly the same strategies through generational improvements, though consistency is a goal. It was obvious that the ZFS way of doing backups was vastly superior to rsnapshot/rdiff-backup. We could keep ZFS snapshots of user data to near infinity and take differentials between them to send to a backup server. Unfortunately, no other file system is compatible with ZFS snapshots, since it's an intrinsic property. The ability to get a differential file list, which is a minor extension to ZFS snapshots and in the OpenSolaris sources, is really catching on in the filesystem community, and it seems to be making its way into a lot of enterprise file systems.

What we ended up with, for a variety of reasons, was GPFS. It provides a lot of useful features such as failover, data tiering, snapshots, and separation of metadata. It doesn't have differential snapshots yet, but hopefully that will come. We're adaptive, though, and can make use of other features to get to the same ends—not having to run the equivalent of find on 400,000,000 files. The key is separation of metadata coupled with novel arrangements of data sets.

Rik: What's special about your environment?

Doug: As has been publicized in various scientific journals and the mainstream press, we have a custom supercomputer that does molecular dynamics, as well as having several clusters of commodity machines. Actually, we have several of these supercomputers, and each outputs data at a little less than 1 MB/sec. That's not a lot, right? Well, in aggregate, it's about 10 TBs of data every 2–3 weeks, depending upon the chemistry parameters, frame storage rate, and various other things.

We arrange this data into chunks we call lockers. A locker is just an arbitrary chunk/division of the file system into a directory space. When a chemistry job starts, it asks for a locker and then proceeds to dump data frames grouped into trajectories. A frame is a file containing a lot of floating point numbers indicating the current position and velocity vector in three dimensions for every atom in the simulation. We get about 3 million frame and miscellaneous files per day (logs, allocation records, etc.).

Rik: So how do you set up lockers?

Doug: Lockers are managed by a database API which will hand out a directory in a probabilistic manner according to weighting. This has some flexibility, although we tend to only have one active locker at a time, because it means that there's just

a particular directory to scan for files that have changed. In actuality this is not a hard cutover since it would make life difficult for the chemists, so we only scan the most recent set of lockers for changes. This is where GPFS comes in.

There are several key features that make GPFS interesting for this sort of environment, some mentioned above.

- Quotas can be applied to arbitrary groups of files called file sets, in our case 10 TB chunks.
- Snapshots provide capability to recover from mistakes for user data, although I wish there were more. IBM is improving this.
- Storage tiering means that we can keep fast disks for the newly written data and migrate data to slower bulk-storage media for satisfying reads from analysis machines.
- A sophisticated policy engine allows for writing complex SQL-like queries to scan metadata far more efficiently than find, in fact about 20x faster.
- Metadata separation means that we can put all of our metadata on extremely fast storage, like a Texas Memory Systems RamSan620. This takes a full filesystem scan on 20 15K RPM disks from 4.5 hours down to our fastest recorded time of 12 minutes on the flash/memory hybrid device.

To scan 10 TB of a particular locker involves a reduced set of metadata. We can tell the policy engine to just start in this directory and scan those several million objects for any file that has been modified in the last hour, or the last day. This takes about 30 seconds, which is pretty impressive.

Rik: Okay, so by using a cluster file system like GPFS and special hardware for storing metadata, you can now complete your "find new/modified files" in a reasonable time. I am guessing that this allows you to create incremental backups?

Doug: We divide things into hourly sets and daily sets. The hourly sets are files that don't match .log (those may be continuing to update, and we don't need hourly backups of them). The daily sets are every file that has been modified in thepast 24 hours. The frames are the most important bits, and if we did have a disaster, we'd not want to lose too many of them. We get the frames onto the backup media expeditiously, and the rest once a day. Meanwhile, the policy engine is also taking care of other automations to move the data from tier to tier as the data rolls in, including a tier once per week of older files on media that spins down.

Rik: So that raises two questions, although I think I can answer the first. I saw a talk on Anton at a USENIX conference, and a frame represents the position of the atoms in the reaction under study during some time slice, similar to a frame in an animation. But you mentioned a "policy engine." What is this and how does it fit into your backup scheme?

Doug: The policy engine is the part of GPFS that allows you to write queries against metadata that match nearly arbitrary combinations of characteristics. Each one of these little files of quasi-SQL is a policy that is either stored in the file system (i.e., the default write policy) or activated by an administrator applying the policy on demand.

There are several base sets of policy actions one can take. There is the migrate action, which moves data from one pool (tier) to another; there is an external pool action which you can use to migrate data to an external storage pool (such as TSM [Tivoli Storage Manager]); and there are "list" actions which you can use to gener-

ate file lists. We are using this last one to quickly scan the metadata and generate lists of files. These lists are collections of inode, owner, pool, and other metadata along with filename that are passed to a script that you define. This script can do anything you want; you apply an argument to the policy to tell it how many metadata "files" are passed to the script in chunks of lines. We use the script to select only the filename from the metadata, which generates a file of filenames on which to run rsync.

So the backup process works in two parts. We have the policy engine scanning the filesystem metadata directly for things that need to be backed up (via the metadata SQL, which is very fast), which generates a metadata list that is passed to a shell script in batches of 5000 records. The shell script takes the filename and uses the first three components of the pathname to determine where it should go in backups and makes discrete files per destination. In part 2, a backup daemon (written in Perl) scans for these resulting files in a particular directory and uses rsync or Star or whatever (selectable per destination) to send these to offsite backups.

Why two parts? First, because of the way that the policy engine generates its list of metadata with extraneous data, that needs to be filtered, and it's easiest to put this output into a file so that you can later call rsync on it, for example. Second, because we actually in some cases use disk-to-cheaper-disk backup, we have backup servers that eventually get full, so it becomes necessary to have some granularity of control over the destination, which is where the path component comes in. The metadata may arrive in any order; it's just looking for stuff that has changed. This results in often arbitrarily mixed locations for things. The script that processes the metadata is responsible for categorizing these per destination so that rsync doesn't have to worry about it and can just send (using `--files-from=`) to select its source and send it to a single destination as needed.