TSM cluster design evolution.

Half a dozen is better than four. Dozen.

Allen Rout

1. Introduction

Herein, find documented the progress of design work on the cluster of TSM server instances initially implemented by the Author beginning in 2004.

2. Background

2.1. Overview

The University of Florida's central TSM server is used by a variety of on- and off- campus units. It currently hosts some 600 clients, ranging from small, irregularly touched workstations, up to terabyte mail systems. Until 2004 we've operated as a single monolithic server; this has simplified some aspects of mainteinance, but we've grown large now, and need to do something different.

We serve a diverse university community. Administrative applications, academic support, departmental resources, and commercial interests all make use of our facilities. To support the administrative distinctions these customers need, we currently maintain 48 administrative domains, each of which has different local personnel able to administer them.

Some of these domains denote actual inter-organization barriers, but others of them describe storage use distinctions. For instance, there is a domain to support our Division of Continuing Education, but there is also one to support our archives of expired user accounts.

2.2. Client population

We have many different audiences making use of our TSM infrastructure. In addition to the usual rogue's gallery of servers and workstations, we've got several special-purpose arrangements.

One of the more important clients is our Content Manager installation. This document-management system is the center of much of our Registrar's business process. Its presence places an unusual constraint on the backup system: most TSM servers have daytime more or less free to perform mainteinance, and even momentary downtime might be acceptable. Not when you're serving as archive repository for documents.

We're serving as the log-retention archive for many central systems. That means, if someone wants to (say) analyze web logs, it might generate a large flurry of mount requests as the last six month's data is reeled off of tape.

2.3. Some Statistics

- **Occupation.** Our server is currently storing about 163TB of data. I offhand expect to be doubling this in each of the next two years. I further expect this expectation to be incorrect.

  2005-02. Our server is currently storing about 130TB of data. I offhand expect to be doubling this in each of the next two years.

  2004-08. Our server is currently storing about 78TB of data. I offhand expect to be doubling this in each of the next two years.

- **Traffic.** Our daily traffic tends to wander between 1 and 1.3 TB a night, with spikes up to 2 TB.

  2005-02. Our daily traffic tends to wander between .8 and 1.TB a night, with spikes over a 1.5 TB.

  2004-08. Our daily traffic tends to wander between 400 and 600GB a night, with spikes over a Terabyte.

- **Offsites.** We've been unsatisfied with the performance of our offsite courier and storage service, and have become sensitive to the increasing frequency of newsworthy losses of data in courier hands. We are looking into other options.

  2004-08. We have some offsite storage accessed by moving physical tapes to a location a few miles away from campus.

  2004-05. We clearly need some variety of offsites, but there's been very little support for the implementation of such until very recently. We anticipate doing this by taking physical tapes offsite; we've experimented with electronic transport of some data, but for most of our applications the transport can't keep up with the nightly load of data.

- **Database size.** The databases for our server infrastructure sum to about 230GB.

  2004-08. The databases for our server infrastructure sum to about 210GB. They are currently housed on a selection of 9-18GB SSA drives. The largest single database is 41G, not obviously too large.

  Since the servers are split, not only is each individual growth problem more tractable, but mainteinance need only affect a subset of our customers. This has made everyone happier.

2004-08. The database for this server, about 98GB, was housed on 24 9GB SSA drives. This database is dramatically ungainly to maintain; it's possible that we could reduce its' size by as much as 20% through an unload/reload, but the last time we did that, when the DB was less than 60GB, it took 40+ hours to complete. This is completely unacceptable performance for a service on which day-to-day operations depend.

2.4. Transitions

Heading into the fall of 2003, we were positioned to upgrade the infrastructure on which the TSM server was based. We were fairly confident we could purchase a single new server. it seemed unlikely that sufficient resources to separate our important workloads on different hardware would be available.

This led us to contemplate a 'virtual server' split, in a manner similar to the service virtualization we were already accustomed to deploying in other domains (such as web services).

With the decision in place to generate more than one server instance on the same box, the next issue was to determine how many servers were necessary, and what functions they should each embody.

3. Initial Naivete

The first design pass was a simple cluster of functionally independent servers: Each server connected to our 3494 with its' own set of category numbers for tape allocation. Each server was to be capable, in principle, of being pried out of the library and re-installed in completely new hardware. The major disadvantage of this design is the complete fragmentation of drives and scratch pools. Each server's scratch would be completely separate, and furthermore moving tapes from server to server would require actually checking volumes in and out of the library. Furthermore, drives would have to be passed around from server to server. This would have been bad enough with a single set of drives, but with 3590s and 3592s, it was simply unworkable. This idea was quickly pitched.

4. Advanced Naivete

So it was clear that the drives and tapes had to be managed by a single server, and the other server instances should view the tapes as remotely managed. So, I turned my sights on the count of server instances which would be exposed to clients. We have several applications which make use of TSM as a utility, a large number of administrative domains, and several groups which use our TSM server as a sink for their remote volumes. The problem seemed simple: One each.

At this point it became clear that the number was foolishly large. The reasons may be perfectly clear, but I'll enumerate some of the more interesting ones:

- The number of different instances would make near-perfect automation of cross-server monitoring and administration a critical precondition for production service. Day one would require perfection. Not likely.
- Each server requires its' own cache and buffers: adding these up over 40 instances adds up QUICKLY
- Each server requires distinct storage resources: DB volumes, log volumes, storage volumes, etc. This means many, many small volumes to keep track of, get wrong.
- Each server needs a separate volume for a DB backup. Every day. Without fail. 40 volumes times seven days retention equals a LOT of wasted space; my 90GB database is fitting onto single volumes now; split that 40 ways?

5. The dawn of wisdom?

The complexities generated by simplistic splits having sunk in, I set about dividing my client base into divisions which logically hung together. They fell into two broad categories: Instances that client machines actually log into ("client-contact" instances) and those which served the other instances. ("administrative" instances).

In this design, a server to which clients back up data (say,'EXT') makes use of several other servers for its' resources. It requests tape mounts of a library manager, or 'control' server (say 'CTRL'), and it makes its storagepool copies to a copy server (say 'COPIES') which in turn makes reference to the library manager for its' own tapes.

- Library manager instance (administrative). If all client-contact instances view their tapes as managed remotely, then there's no special relationship to work around. TSM DB for this instance is tiny, with thousands of entries total.
- TSM DB backup instance (administrative). If we do DB backups 'remotely' to another server , then we have several optimizations available to us: We can store a DB backup on some increment of media smaller than a full volume. We can make copy pools from the primary pools that hold DB backups, and thus store identical DB backups in many places, which seems better than doing many snapshots. We can run many more DB backups simultaneously, since we don't need to allocate a physical device per backup.
- Content Manager: (client-contact) CM is a large critical application. It's desirable to insulate this server from oddities in other servers, and vice versa.
- GL Mail: (client-contact) the GatorLink Mail complex is our single largest-displacement application. It uses more than 30 Million files , and more than a terabyte of live storage. In many ways it exists in a performance envelope all its own. We'd like to mew it up in its' own problem- and failure- domain.

- ERP (client-contact) We have an administratively distinct development operation accreting around the deployment of Enterprise Resource Planning applications. This is a customer base which we would like to insulate from complexities fomented by other clients.
- Internal clients (client-contact) We have many machines, server and workstation, which are managed by folks who work for our data center. The administrative interactions we can have with these people are different than with our paying customers. From week to week it's a toss-up which set of clients might threaten the others.
- External clients (client-contact) We have a variety of paying customers whose behavior might change without warning; reinforcement or moderation of this behavior comes with the admission of next month's bill. Separation seems indicated.
- Copy pool clients (client-contact) We have several agreements with other TSM installations in which we serve as their offsite host. Some of the worst database behavior we've seen out of TSM comes as an outgrowth of these arrangements (undetected deadlocks leading to server halts, etc.) Insulation is earnestly to be desired.

The database backup problem created enough tension about wasted space that I began to consider means of performing TSM DB backups without using a full volume each time. The solution seemed to be to perform DB backups to one of the "remote" servers hosted on the same hardware. This would provide the following features:

- DB backups become disk-to-disk copies, possibly faster and definitely parallelizable.
- DB backups can be copied to remote locations with all the copypool primitives available to normal application data.
- Multiple DB backups can be stored on one volume.

Storing many DB backups on the same volume might cause folks to shudder, but so far it appears that doing this will actually improve recoverability. If we perform DB backups directly to a volume, then our option for making "another" copy is to either write another full or to take a snapshot. With this scheme, we can have a given DB backup replicated to N different locations as quickly as bandwidth will allow. Given the incremental nature of storagepool backups, it's not unreasonable to think that 90% of the backup might be offsite even before the primary backup completes.

There's a separate concern about how many 'eggs' (DB fulls) one might store in one 'basket' (physical tape volume). A simple answer to this is that, rather than making three DB backups, you could keep three (or four? more?) copies of the *same* backup, and occupy fewer physical volumes. This concern will become more and more relevant as we move to larger volumes. 3592 drives hold 300GB *raw*; reasonable expectations of database compression rates leaves us with an estimate that more than two terabytes of compressed database could fit on that volume. Few folks want a TSM database larger than 100GB; that's 95% wastage.

Another concern is that this generates a recursive recovery problem for recovery: in a site-wide disaster, it will be necessary to restore the database-backup (DBB) server before real work on restoring the other servers can begin. And where does the DBB server back up its own DB?

The answer to this problem is straightforward once we consider that the number of "files" (virtual volumes) resident on the DBB server is rather small compared to most "normal" TSM servers. For example, if we back up that 100GB database onto 40GB virtual volumes, then we've got three or four files a day. Even if we back up 40 servers and retain DB backups for an entire year, that's still under 60,000 files. We get this many files from the initial incremental of a moderately populated server. With more reasonable numbers (say a month retention on 20 servers) the number is even smaller: 2400 files, and a DBB server database probably measured in tens of megabytes.

A DB this small is quite reasonable to back up to FILE devclasses, which could be distributed to many locations via conventional means; RSYNC on the devclass directory would probably be entirely adequate, and could be run hourly, change being in fact minimal.

On further reflection, it seemed reasonable to make this same instance into the host of the library managers; this is another relatively small database load, and is similarly in the critical path of recovery of other servers, so it seems reasonable to combine the functions.

6. The mid-morning of wisdom

Several months of production with the before-mentioned design outlined a some important weaknesses of the pervasive use of remote volumes.

First and most obviously, when the number of processes wanting tape drives exceeded the number of tape drives, the remote server would hand off new mounts in round-robin fashion. In the worst case (and indeed a common one) this resulted in a tape dismount-remount for each virtual volume written. With reasonable virtual volume sizes (10G) this still resulted in a large number of hand-offs. This round-robin treatment is not without positive impact, though: Short processes would acquire a mount with reasonably short delay, which reduced the number of processes waiting for long periods, and all but eradicated the database locks which tend to happen when many processes hang about for a long time.

More subtly, a single process can clog all available mount points of a particular devtype. If mount retention on a virtual volume is set to (say) 60 minutes, then it's possible for a given storagepool copy to write several volumes within that retention time: This results in lots of full mount points, and many mostly-empty physical volumes. This problem, at least, was trivial to resolve: Set retention for all virtual volumes to zero.

Most importantly, there are some oddities in the way a remote volume mount is granted that make total tape deadlock not only possible, but very likely. The key lies in that remote volume mounts are granted before the target server knows where it's going to put the data.

I've defined my target servers to write data directly to tape; this results in a 90% majority of tape-to-tape copies for my workload, which is drastically more efficient than tape-to-disk and then disk-to-tape later. However, when I start a virtual-volume storagepool backup(say) the remote volume mount is granted -before- the target server has acquired a mount point. In fact, the target server doesn't even attempt to acquire the mount until the source server has mounted its' volume and sent its' first chunk of data across.

The deadlock is probably obvious at this point: If you've got four drives, it would be very easy to have four processes mount source tapes, and block forever waiting for target tapes. Since I've got two device classes for 3590 and 3592 devices, my own deadlock situations have to involve some feedback, but it's not that much more complicated.

The deadlock problem was by itself sufficient to push me out of such heavy use of remote volumes; In August 2004, I was engaged in generating copy storage pools of "local" tape served by the library manager, and in decomissioning the remote copies. Results seem good: mounts are less frequent and throughput seems better.

7. The Teatime of Tranquility

Once the copy pools were running smoothly, we entered a period in which the most prominent server challenges were again client behavior. We have several outlier client populations, each of which has some small server-configuration impact.

ERP. Our ERP (Enterprise Resource Planning) organization is founded on a series of databases, each of which has been receiving a daily full backup since the deployment of the system. None of these database fulls have been deemed obsolete, neither the transaction logs which trace the path between them. The largest single node has an occupation somewhat over 115 TB. This client population is an outlier in the 'bulk capacity' direction. Pleasantly, other than separating these large-capacity, seldom-used files onto larger tapes, little explicit configuration work has been necessary to accommodate the work flow.

Offsite behavior, however, has been significantly more expensive than one might imagine. This is because the database full backups tend to dominate the nightly incremental traffic. Since the fulls never expire, offsite tapes with full backups on them seldom become candidates for reclamation. This has resulted in a mostly write-only offsite infrastructure; tapes leave, but they don't come back. The technical complexity of this is not different from normal offsite operations, but we never get to the position in which offsite tape count becomes grossly constant.

Content Management. UF has a document-management application infrastructure, making use of IBM's Content Manager (CM) product. CM in turn makes use of TSM as a data store. Many of the day-to-day operations of the University rely on this document store. This has resulted in the CM TSM infrastructure having a more rigorous uptime requirement during daytime than is often the case. Many of my backup-usage TSM servers can be restarted in the middle of the day without any actual users noticing the change. This is not the case with CM. Separating the CM utility to its' own TSM instance has been enough to protect CM usage from the vagaries of the rest of the backup infrastructure.

GatorLink Mail. UF's central mail system currently has about 1.2TB of active mailbox space, with a current total of 74.3 million files. We run continuous targeted backups of files known to be changed, and

our incremental timing has recently exceeded the number of hours in the day.

This application isn't too small in the 'bulk size' direction, though each of the two back ends totals somewhat less than 3TB total occupation. Where it really shines, though, are in the 'file count' and 'file coverage' directions. The mail stores use a single file per message, using file names of increasing integers.

We split this workload off from the rest of our TSM work, and then split each back-end machine into its' own TSM server. This has finally resulted in our being able to complete expiration on a daily basis.

8. Hurricanes (Current projects)

2004 was a banner year for hurricanes in Florida. Among the administrative results of this has been renewed interest in disaster recovery planning. The decision was reached that moving physical tapes a few miles was probably inadequate protection.

We are therefore engaged in a plan to set up remote TSM infrastructure in a machine room several hundred miles away. Once we have a remote copy there, we will be decomissioning our physical offsites, and likely our local copies, too.

9. Conclusion, and predictions

And there we stand. At current writing (Fall of 2005) we anticipate greater than 50% annual growth in existing customers. We are contemplating several new endeavors, including a 200+TB HSM installation. That our system design will evolve further is beyond question. There's no telling, yet, which way we'll go.

10. Addenda: Obstructions and potholes

TCPADMINPORT. The first problem I encountered with this configuration is that, regardless of which port is specified as the server port, the server listens on 1580 for administrative commands. It's necessary to also set the TCPADMINPORT to the same port as the TCPPort.

Mount thrashing. Since all of the tape mounts were to virtual volumes, the default mount retention (60 minutes) created a 'thrash' of mount points when the target server is writing directly to tape. For example:

- Server SOURCE completes virtual volume 'A' on server TARGET, writing to tape volume T01, and closes it. Volume 'A' is retained in 'mounted' state on SOURCE for 60 minutes, which also retains volume T01.

- Server SOURCE requests a scratch virtual volume ’B’ from TARGET. SOURCE has authorization for more than one mount point, so TARGET has to mount tape volume T02 to write the volume. When volume ’B’ is complete and closed, it too is retained in a mounted state, thus pinning T02.
- [etc.]

This gets especially interesting when more than one SOURCE server is attempting to access the TARGET server at the same time. If N servers are working with at least N+1 mount points, then every single new virtual-volume mount will request a new physical-media mount. Of course, this plays havoc with efficiency.

The solution to this problem was to set mount retention of the virtual volume devclass to zero: In this way, the old volume is "dismounted" (and thus the underlying tape drive is, too) immediately. This permits new mount requests from the same server to be granted from the same tape. In this structure we only need sufficient mount points to service the number of simultaneous sessions.