Outline of a Digital Preservation System

October 18, 2010

Original link

I hate losing data. Don't you wish there was some way to store stuff so that it will be available forever, so that your websites could live on even after you die? But it's a tricky problem. There are a number of things you have to worry about:

- Hardware failure. The disk you had your data on died and you can't get data off it anymore. Or, even worse, your data has been silently corrupted.
- Hardware skew. Your disks are still good, but they're ancient and obscure and no one has drivers for them anymore. (I have some old IBM tapes that I can't read because the devices are so obscure.)
- Filesystem skew. You can read the raw devices but the data was stored on it using a striping technology no modern operating systems implement, with a filesystem which stopped being supported after its lead developer was arrested for murder, and then encoded with a higher-level file distribution system that used erasure coding to spread data across multiple machines.
- Format skew. Your documents may be stored in an obscure format that nobody has a reader for anymore.
- Institutional failure. The people in charge of making sure all the other problems get addressed die or move on.

There's a basic outline to the skew problems: use the simplest, most popular choice possible and then move to a replacement as it shows signs of going permanently extinct. It seems unlikely to me that ASCII text on a Unix file system in a JBOD configuration will ever completely die off, but if it does you'll be safe for a long time before you'll have to move things over.

Similarly, for the failure problems you want: a) to choose reliable material, b) to implement constant sanity checks, and c) have several layers of backups if the sanity checks don't get executed or fail.

OK, so let's get to a specific outline. You have a bunch of standard files (mostly ASCII derivatives, maybe PDFs) in a standard file system (UFS?) mounted read-only (with hardware-level write-prevention) from a bunch of standard SATA disks (no RAID or anything like that) on some machines running a modern Unix. You can serve these files up using bog-standard HTTP (or

any other file transfer protocol that can be mapped to the model of a Unix file system). The file names (less a prefix for mounting the drive) are designed to be globally unique. The best way to do this is probably to combine a date and a DNS name (ala `tag:`). So you could imagine this file to be at:

```
/hdd7/2002/www.aaronsw.com/weblog/preservation.html
```

Alongside the files, for each drive you have a manifest that lists the date the file was added to the preservation system followed by the globally-unique filename and two hashes: a fast hash (probably CRC32) and a standard secure hash (probably SHA1). As a background process, you (nightly? weekly?—depends on how long it takes, I guess) go through each file and compare its fast hash against the one listed in the manifest. If the comparison fails, you go offline and sound the alarm.

Going offline isn't so bad, because you have at least two other replicas in different geographic locations under different political regimes. Any request traffic can go to them instead while you restore your system.

For safety, you should probably toss the whole drive and get it restored from one of the replica sites.

The manifest from each system at each site is replicated on a series of index servers (at least one at each site). The index servers run a simple HTTP redirector that maps from the secure hash or the globally unique filename to a box storing the file, using a table they generate from the manifests.

If bandwidth isn't an issue, the index server should proxy the data instead of redirecting you to preserve URLs. If it is an issue, you should probably try to reset the DNS name to any server which has failed to point to another redirector. So you can imagine the stable URL being:

```
http://preservation.example.org/2002/www.aaronsw.com/weblog/preservation.html
```

and that redirecting to:

```
http://27.preservation.example.org/hdd7/2002/www.aaronsw.com/weblog/preservation.html
```

which serves the file. When 27 fails (i.e. is no longer serving all the data it was previously serving), it's given a new number and the DNS `A` record for 27.web.resource.org is changed to point to the index servers. The index servers strip off any hdd prefix (such prefixes are prohibited at the root of the global namespace) and serve the request as they do a request for the stable URL.

The index servers also have a manifest of the manifests (with similar regular verification procedures). Each data server regularly publishes a file with the date and the secure hash of the manifest that its regular check was compared

against. The index servers regularly request this file from each of the data servers and take offline any servers whose manifest hash doesn't match the one in their manifest of manifests.

Finally, a master manifest is generated by combining all of the manifests, sorting by date, and removing duplicates. (If in this process you find two different hashes for the same file, go offline and sound the alarm.) At the end of every day during which files were added to the collection, the system verifies that the secure hash of all the files up through yesterday is the same as it was yesterday and then makes sure the index servers all agree on the new hash. If there is consensus, the hash of this new master manifest is calculated and very widely published (perhaps including a classified ad in the largest newspaper in each country). You should always be able to work backward from this widely-published hash to a correct state of the entire system. And since all of the manifests are public, anyone with a recent newspaper and some time on their hands should be able to verify the accuracy of their data.

Now this protects against accidental day-to-day hardware failure, but you also want to protect against long-tail catastrophic failures. You should use hetrogenous systems to avoid simultaneous hardware failure or software vulnerability, there should be no one with physical or electronic access to all the systems, as much as possible you should not be able to delete data without physical access, and the locations should be physically secure. Does it make sense to make an alternate copy (i.e. paper with bytes written in OCR-A or something?) in case of an EMP blast or something like that? Thinking about how to protect against these obscure failures seems like the trickiest part.

This system should be fairly simple to implement and operate (I can't imagine it being more than a couple pages of Python code) but even so you need someone to implement and operate it.

Recall that we have at least three sites in three political jurisdictions. Each site should be operated by an independent organization in that political jurisdiction. Each board should be governed by respected community members with an interest in preservation. Each board should have at least five seats and move quickly to fill any vacancies. An engineer would supervise the systems, an executive director would supervise the engineer, the board would supervise the executive director, and the public would supervise the board.

There are some basic fixed costs for operating such a system. One should calculate the high-end estimate for such costs along with high-end estimates of their growth rate and low-end estimates of the riskless interest rate and set up an endowment in that amount. The endowment would be distributed evenly to each board who would invest it in riskless securities (probably in banks whose deposits are ensured by their political systems).

Whenever someone wants to add something to the collection, you use the same procedure to figure out what to charge them, calculating the high-end cost of

maintaining that much more data, and add that fee to the endowments (split evenly as before).

What would the rough cost of such a system be? Perhaps the board and other basic administrative functions would cost $100,000 a year, and the same for an executive director and an engineer. That would be $300,000 a year. Assuming a riskless real interest rate of 1%, a perpetuity for that amount would cost $30 million. Thus the cost for three such institutions would be around $100 million. Expensive, but not unmanageable. (For comparison, the Internet Archive has an annual budget of $10-15M, so this whole project could be funded until the end of time for about what 6-10 years of the Archive costs.)

Storage costs are trickier because the cost of storage and so on falls so rapidly, but a very conservative estimate would be around $2000 a gigabyte. Again, expensive but not unmanageable. For the price of a laptop, you could have a gigabyte of data preserved for perpetuity.

These are both very high-end estimates. I imagine that were someone to try operating such a system it would quickly become apparent that it could be done for much less. Indeed, I suspect a Mad Archivist could set up such a system using only hobbyist levels of money. You can recruit board members in your free time, setting up the paperwork would be a little annoying but not too expensive, and to get started you'd just need three servers. (I'll volunteer to write the Python code.) You could then build up the endowment through the interest money left over after your lower-than-expected annual costs. (If annual interest payments ever got truly excessive, the money could go to reducing the accession costs for new material.)

Any Mad Archivists around?

[Thanks to Brewster Kahle and the Internet Archive for inspiring many of the technical details of this scheme.]