Digital Document Preservation Simulation

Richard Landau

(late of Digital Equipment, Compaq, Dell)

- Research Affiliate, MIT Libraries Program on Information Science

- http://informatics.mit.edu/people/richard-landau

2014-07-22

The Problem

- How do you preserve digital documents long-term?

- Surely not on CDs, tapes, etc., with short lifetimes
- LOCKSS: **L**ots **O**f **C**opies **K**eep **S**tuff **S**afe
- What's the threat model?
  - Failures: media, format obsolescence, fires, floods, earthquakes, institutional failures, mergers, funding cuts, malicious insiders,….
- Some data exists on disk media reliability, RAID, etc.
- Little data on reliability of storage strategies

The Project

- Digital Document Preservation Simulation
- Part of Program on Information Science, MIT Libraries: Dr. Micah Altman, Director of Research
- Develop empirical data that real libraries can use to make decisions about storage strategies and costs
  - Simulate a range of situations, let the clients decide what level of risk they are willing to accept
- I do this for fun: volunteer intern, 1-2 days/week

Questions to Be Answered

- Start small: 10,000 documents for 10 years, 1-20 copies
- (Short term questions, lots more in the long term)
- Question 0: If I place a number of copies out into the network, how many will I lose over time?
  - For various error rates and copies, what's the risk?
- Question 1: If I audit the remote copies and repair them if they're broken, how many will I lose over time?
  - For various auditing strategies and frequencies, what's the risk?

Tools

- Windows 7 on largish PCs
- Cygwin for Windows (like Linux on Windows)
- Python 2.7
  - SimPy library for discrete event simulations
  - argparse module for CLI
  - csv module for reading parameter files
  - logging module for recording events
  - itertools for generating serial numbers
  - random to generate exponentials, uniforms, Gaussians
- Random seed values from random.org

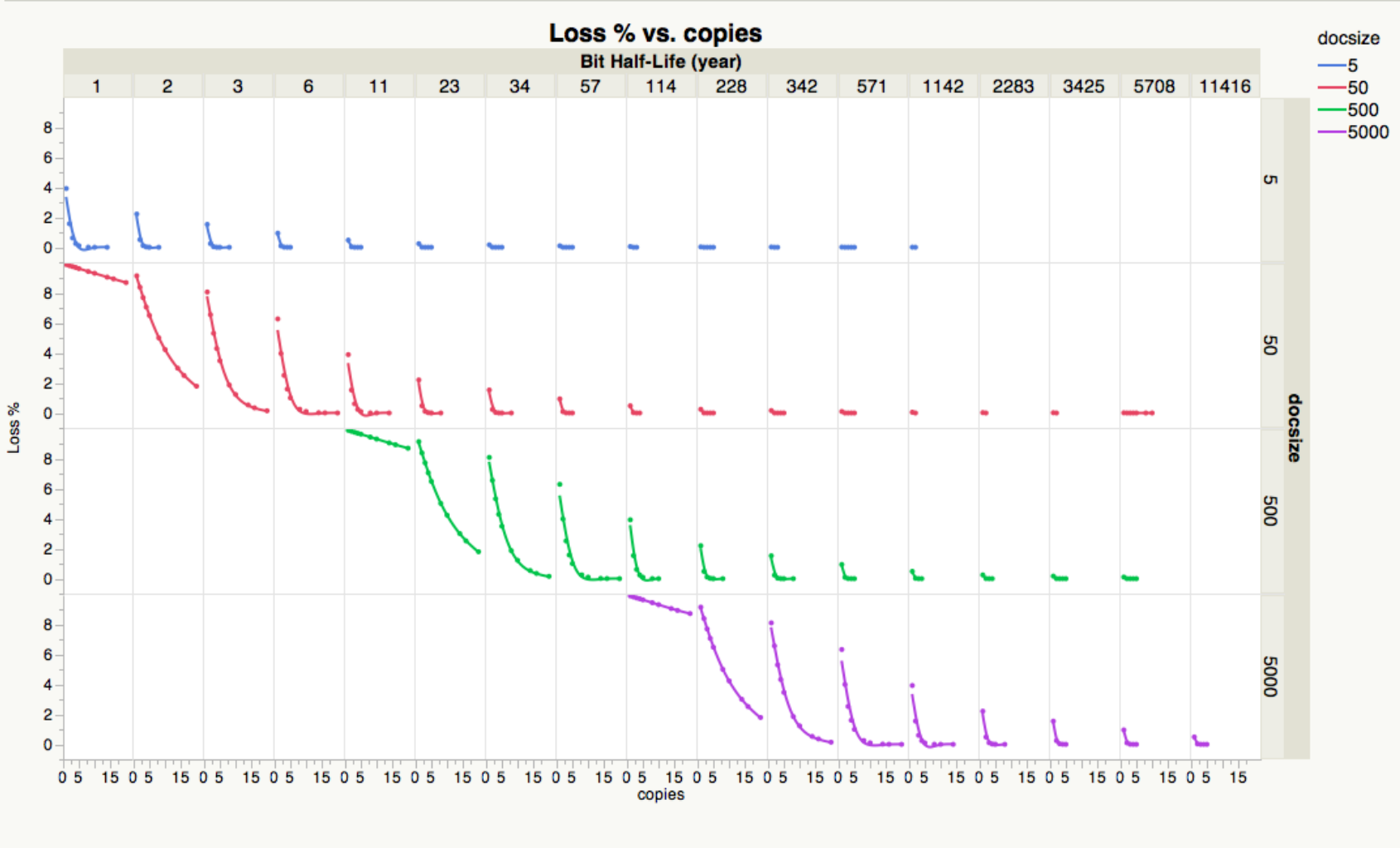
Approach

- Very traditional object and service oriented design
  - Client, Document, Collection of documents
  - Server, Copy of document, Storage Shelf
  - Auditor
- Asynchronous processes managed by SimPy
  - Sector failure, shelf failure, at exponential intervals
  - Audit a collection on a server, at regular intervals
- SimPy resource, network mutex for auditors
- Slightly perverse code (e.g., Hungarian naming)

How SimPy Works

- Library for discrete event simulation, V3
- Discrete event queue sorted by time
- Asynchronous processes, resources, events, timers
  - Process is a Python generator, **`yield`** to wait for event(s)
    - Timeouts, discrete events, resource requests
- Very simple to use, very efficient, good tutorial examples
- Time is floating point, choose your units

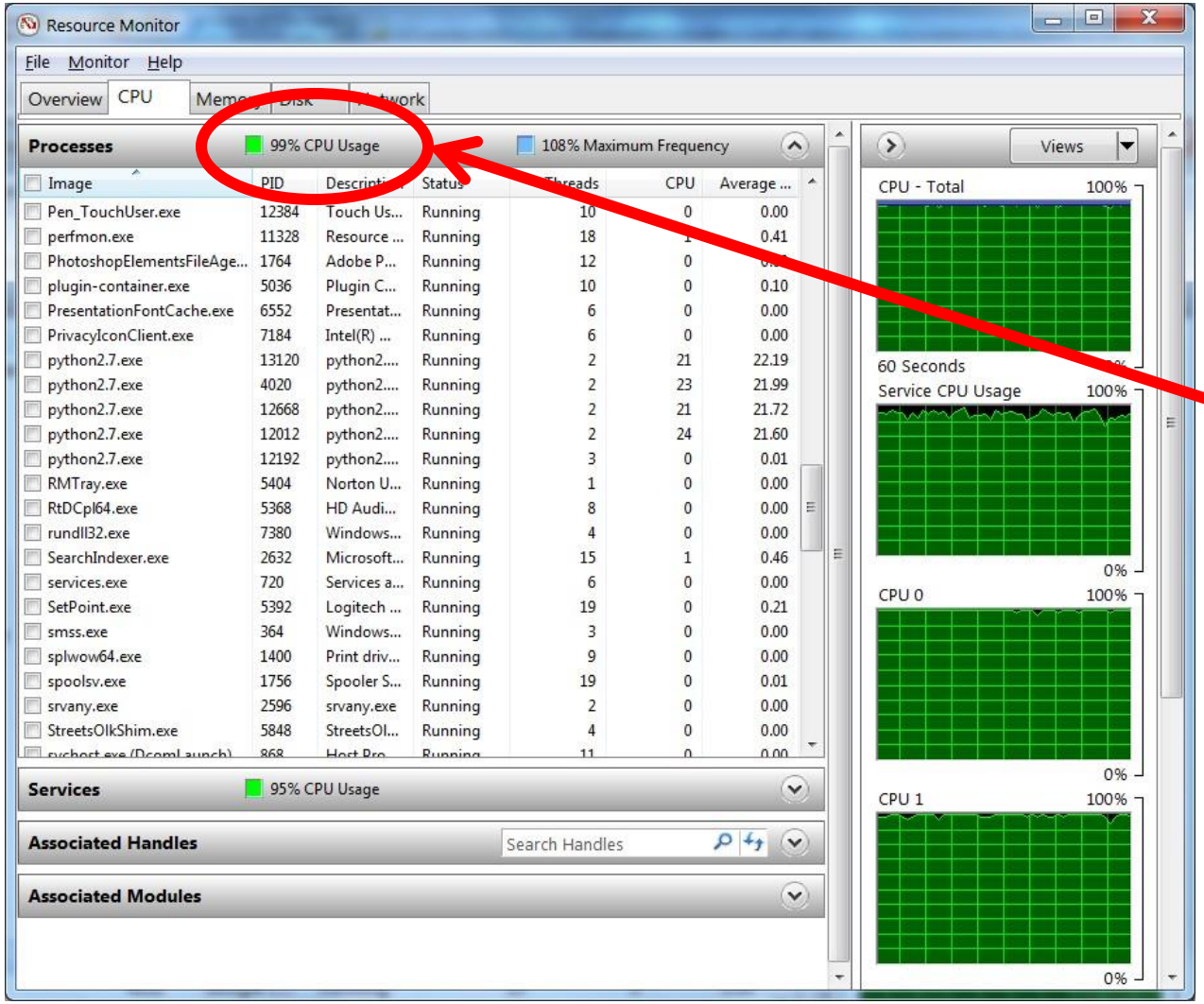
Early Results for Q0



Is Python Fast Enough?

- Yes, if one is careful
  - Code it all in C++? No. Get results faster with Python.
  - "Premature optimization is the root of all evil" -- Knuth
- Numerous small optimizations on the squeaky wheels
  - Just a few reduced CPU and I/O by 98%
- If it's still not fast enough, use a bigger computer

Burn Those CPUs!



- One test only
  2 sec to 8 min
- 2000-4000 tests
- Scheduler runs
  5-7 programs at once
- All 4 cores
- (Poor man's parallelism)
- 99% CPU use

Class Instances Should Have Names

- `<client.CDocument object at 0x6ffff9f6690>` is imperfectly informative
  - `D127` is much better for humans: name, not address
  - With zillions of instances, identifying one during debugging is hard
- Suggestion:
  - Assign a unique ID to every instance that denotes class and a serial number
  - Always always always pass IDs as arguments rather than instance pointers

How I Assign IDs

- Every class begins like this:

```
class CDocument(object):
    getID = itertools.count(1).next

(in __init__)
    self.ID = "D" + str(self.getID())
    G.dID2Document[self.ID] = self
```

- **`itertools.count(1).next`** function returns a unique integer for this class, starting at 1, when you invoke it
- Global dictionary **`dID2Xxxx`** translates from an ID string to an instance of class Xxxx
- Pass ID as argument; when a function needs the pointer, it can use a single fast dictionary lookup

Lessons Learned

- Python is fast enough
- **`SimPy`** is really dandy
- **`argparse`** is a great lib for CLIs
- **`csv.DictReader`** makes everything a string, beware
- item-in-list is really slow, O(n/2), use dict or set instead
- Lots of comments in the code!

Code is on github

- On github: **MIT-Informatics/PreservationSimulation**
- Questions: mailto:**landau@ricksoft.com**

Further Details (Not Tonight)

- Many small **optimizations** add up
  - Fast find of victim document on a shelf
  - Shorten log file by removing many or all details
  - Minimize just-in-time checks during auditing
  - Change item-in-list checks to item-in-dict
- Wide variety of **scenarios** covered
  - 1000X span on document size, 100X on storage shelf size, 10000X span on failure rates, 20X span on number of copies
  - A test run takes from 2 seconds to 8 minutes (CPU)
- Running **external programs**
- **Post-processing** of structured log files

Forming and Running External Commands

- Substitute multiple arguments into a string with **`format()`**

```
TemplateCmd = "ps | grep {Name} | wc -l"
RealCmd = TemplateCmd.format(**dictArgs)
```

- Run command and capture output into string (or list)

```
ResultString = ""
for Line in os.popen(RealCmd):
    ResultString += Line.strip()

nProcesses = int(ResultString)
if nProcesses < nProcessMax:
    . . .
```

A More Complicated Command

- Read a dict of parameters with **csv.DictReader**

```
dir,specif,len,seed,cop,ber,berm,extra1
../q1,v3d50,0,919028296,01,0050,0050000,
```

- And then substitute lots of arguments with **format()**

```
python main.py {family} {specif} {len} {seed}        \
  --ncopies={cop} --ber={berm} {extra1} >             \
  {dir}/{specif}/log{ber}/c{cop}b{ber}s{seed}.log     \
  2>&1  &
```

- Execute and capture output with **os.popen()**

Subprocess Module Instead

- `os.popen` is being replaced by more general functions
  - `subprocess.check_output`
  - `subprocess.Popen`
  - `subprocess.Pipe`
  - `subprocess.communicate`

Supersede That Old Function

- Have a complicated function that works, but you want to replace it with a spiffier version?
  - Edit in place?
    - Might break the whole thing for a while
  - Comment it out with ''' or """?
    - Not perfectly reliable, like "**`if 0`**" in C
- Supersede it: add the new version after the old
  - Last version replaces previous one in module or class dict

```
def Foo(…):
    (moldy old code)
def Foo(…):
    (shiny new code)
```