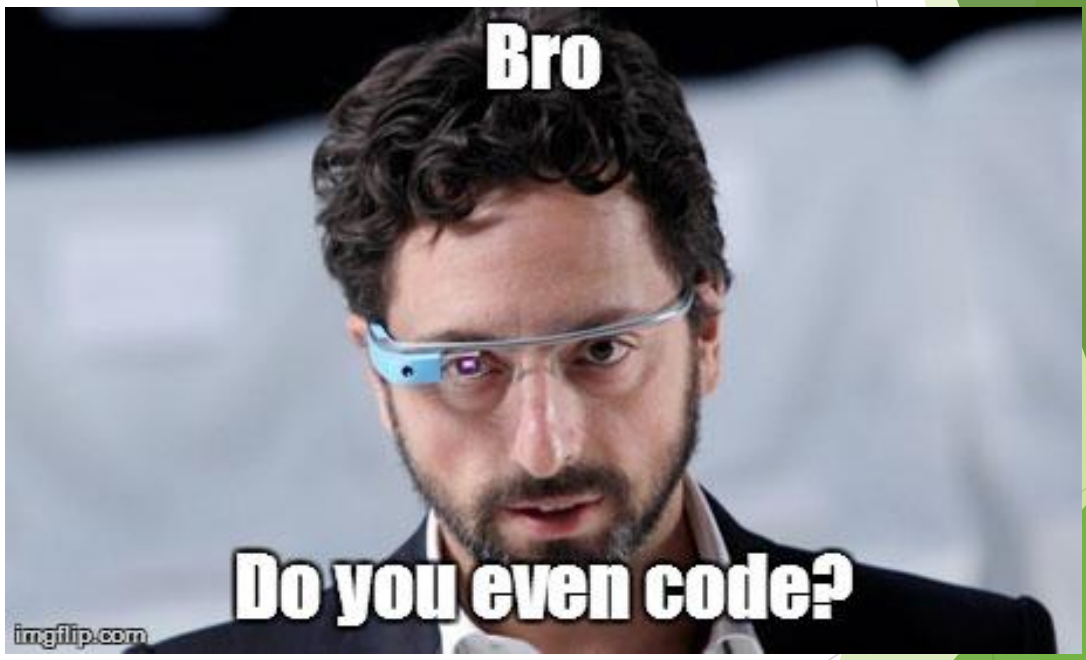
DMF Python Binding + Extras

For those of us with control “issues”

Why bother?

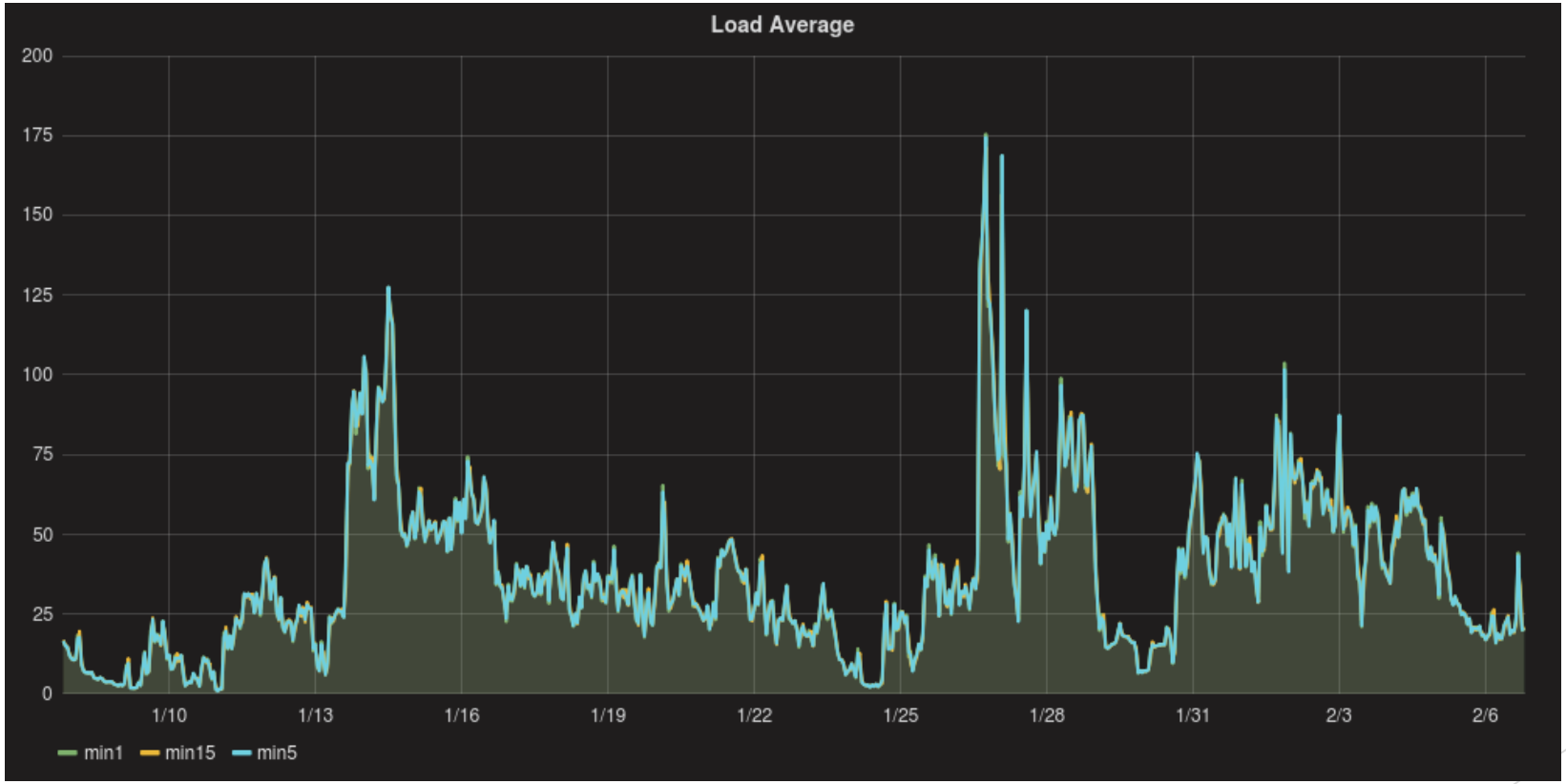
- We're not using DMF as an HSM, we're using it for archive so the in built in tooling doesn't quite fit
- We've got a queueing system written in Perl that runs the command line tools. We'd prefer that didn't happen because code is more difficult to maintain and performance isn't good when spawning subprocesses
- Python is a well understood and supported programming language in our company and integration with other tools in future will be easier
- With more control comes more flexibility and performance… maybe



Why don’t the tools fit?

- Starting a subprocess for every dmput or dmget makes it difficult to track progress. Presently we periodically scan the area of the filesystem we’re recalling or backing up to to provide feedback to the user about progress. This obviously doesn’t scale well
- Building tools around command lines creates a lot of boiler plate code for managing subproccesses that is difficult to understand. Command line behavior also more likely to change and not be documented than an API
- We need access to the database for reads (more about this later) but when you have < 2billion bfids it can get locked for a long time

Graph interlude



Other changes we made….

- We backup our databases once a week (It's difficult to do it more often, *queue gasps of horror)*
- We use the backup to create a text dump of the catalogue database then put the useful information into an lmdb database – BFID – VSN, Chunknumber
- When querying lmdb for bfid information. If the request doesn't find the bfid pexpect is used to call dmcatadm and populate the lmdb database.
- We used the Spectra API to gather details about what tapes are where and put them into a mysql database

Our first surprise use case

- On our operating system SLES11sp4 we have automounts which create bind mounts. When trying to dmput via the bind mount path rather than the local path DMF is unable to migrate the selected file (We only figured this out when migrating to a new server, our queue had quite a backlog by that time)
- Converting a path which is bind mounted by the automounter to its local mount point programmatically isn't as easy as it sounds
- Getting the DMF file handle for a path is easy using the API, and the file handle can be used instead of the path to put the file, so we have our own dmput.py script that does this to avoid the bind mount problem. dmput.py is 250 lines of Python (including command line parsing, logging, error handling) with the actual work done by 40 lines of Python.

Advantages

- Libdmfuser.so which we've built our binding against allows asynchronous and synchronous requests for get, put , copy, stat etc which gives you the power to manage things more efficiently in a single process
- It's much easier to manage in flight requests of each type (backup, copy, archive, recall etc), we find when there are too many requests a lot of memory is used and performance suffers
- Scanning the filesystem or logs to provide a progress report isn't required
- Managing each type of in flight request allows you to throttle based on performance by keeping the request count low you can respond more dynamically
- Converting the catalogue db to lmdb makes it possible to quickly order requests based on the order they are on tape

Basic examples using dmf binding

```
import dmf
context = dmf.Context(futures=False)
for path in ...:
    fullstat = context.fullstat_by_path_sync(path)
    if fullstat.attr.state in (dmf.STATE_REGULAR, dmf.STATE_DUAL):
        context.put_by_path_async(path, flags=dmf.MIGRATE_FLAG_FREE)
context.await_replies(dmf.REPLY_TYPE_FINAL, 0)
```

```
"dmput all files that are online"
```

```
import dmf
context = dmf.Context(futures=False)
for path in ...:
    fullstat = context.fullstat_by_path_sync(path)
    if fullstat.attr.state in (dmf.STATE_REGULAR, dmf.STATE_DUAL, dmf.STATE_PARTIAL):
        context.put_by_path_async(path, flags=dmf.MIGRATE_FLAG_FREE)
context.await_replies(dmf.REPLY_TYPE_FINAL, 0)
```

Concurrent futures example

```
import concurrent.futures
import dmf

def process_futures(context, path_futures, handle_futures, timeout=None):
    while True:
        futures = path_futures.values() + handle_futures.values()
        if not futures:
            break
        for future in concurrent.futures.as_completed(futures, timeout=timeout):
            if isinstance(future.request_details, dmf.RequestDetailsStatPath):
                del path_futures[future.request_details.path]
                fullstat, fhandle = future.result()

                # business logic using fullstat.attr.state, fullstat.stat.ftype, fullstat.stat.size etc
                if ...:
                    handle_futures[fhandle] = context.put_by_handle_async(fhandle, flags=dmf.MIGRATE_FLAG_FREE)
            elif isinstance(future.request_details, dmf.RequestDetailsPutHandle):
                del handle_futures[future.request_details.handle]
                future.result()

path_futures = {}
handle_futures = {}
context = dmf.Context()
for path in ...:
    path_futures[path] = context.fullstat_by_path_async(path)
    try:
        process_futures(context, path_futures, handle_futures, timeout=0)
    except concurrent.futures.TimeoutError:
        pass
process_futures(context, path_futures, handle_futures)
```

Questions?


NOT SURE IF PRESENTATION WAS SO GOOD
NO ONE HAD ANY QUESTIONS
OR NO ONE WAS PAYING ATTENTION
mememaker.net