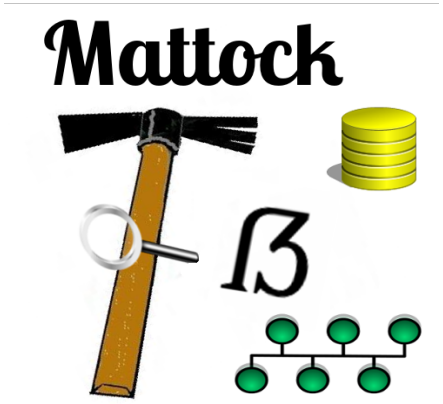
Todays schedule

- Asynchronous processing & tool-chain approach
- Integrity, privilege separation and capabilities.
- CarvFS & MinorFS
- **MattockFS core design**
- MattockFS as distributed-framework building block
- Installation (hands on)
- File-system as API (hands on)
- Python API (hands on)



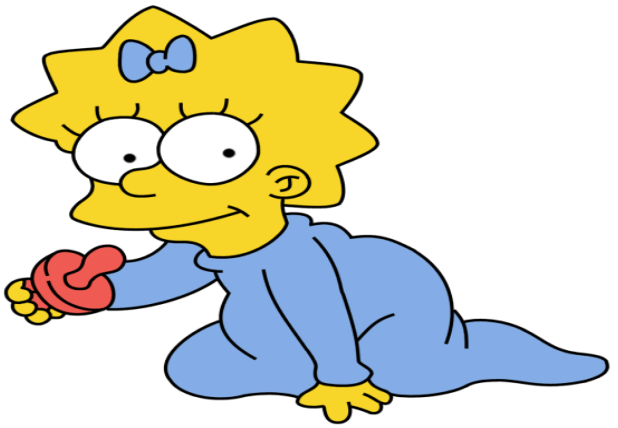
MattockFS

Computer-Forensics File-System

MattockFS core design

MattockFS

- CarvPath compatible.
- Integrated **local** message-bus.
- Capability based API.
- Data freeze & secure provenance logging.
- Designed to minimize spurious reads.



MattockFS

- User space file-system for forensic frameworks
  - Data archive
  - Integrity measures
  - Spurious read measures

- Integrated local message-bus
  - Hooks for throttling

Integrity measures

- Trusted **immutable** data and meta-data
  - Lab side sealed digital evidence bag equivalent.
- **Trusted** provenance logging
- Capability based access control
  - Sparse capabilities
  - Mandatory Access Controls

Spurious reads measures

- **Opportunistic hashing**
- Linux **fadvise** API
- **Integration** of message-bus and data-archive
- Provide **pressure** data for throttling purposes
- Job picking geared to pressure reduction and/or opportunistic hashing.

Opportunistic hashing

- Hashing done within file-system.
- Low level reads result in hashing progression high-level entities.
- Leads to delayed first-read of data-entities.
- Reduces first-read last-read time-window.

Primary OH

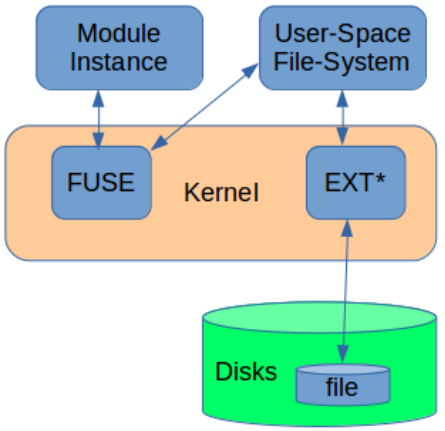
- Prioritize jobs by hashing offset.
- Hash when data is read anyway by processing module.
- Only read trailing data when hash is needed.

Secondary OH

- Data-derivation module subset only.
- Just-written data still in core (unzippers).
- Just-read data still in core (carvers)
- Hash when ‘incore’ on derive.

CarvPath & fadvise

- System call to communicate data-access intent.
- Mark sections of open file:
  - POSIX_FADV_WILLNEED
  - POSIX_FADV_DONTNEED
- MattockFS uses reference-counting stack.
- Allows kernel better page-cache policy.
- Reduces spurious reads.



Refcount stack : Merge up

The initial sparse-free fragments in the CarvPath to be merged is designated by C_0. Consider $F(X)$ to be the set of all byte locations designated by CarvPath X, if we look at the actual *merge up* operation, for each i, starting at zero, where : $F(C_i) \neq \emptyset$, we perform CarvPath operation so that:

- $F(S'_i) = F(C_i) \cup F(S_i)$
- $F(C_{i+1}) = F(C_i) \cap F(S_i)$

And for i=0, M being the area that we shall be marking as *hot*:

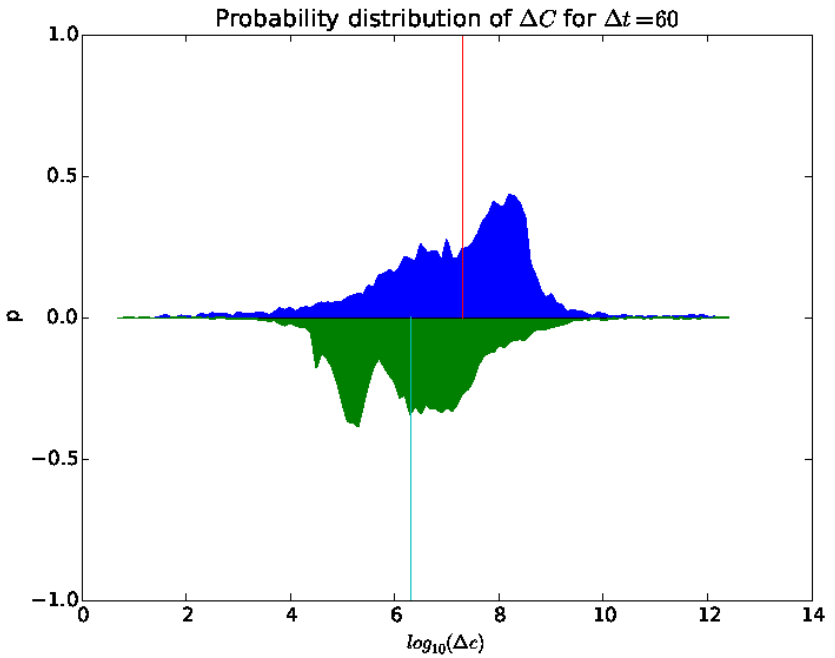
- $F(M) = F(C_0) \setminus F(S_0)$

Refcount stack: Unmerge down

- $F(S"_i) = F(S'_i) \setminus F(C_i)$
- $F(C_{i-1}) = F(C_i) \setminus F(S'_i)$
- $F(U) = F(C_0)$

Throttling

- Needed to prevent large over-commit.
- MattockFS does NOT implement throttling.
- MattockFS provides pressure-data hooks.
  - Allows framework to implement effective throttling.



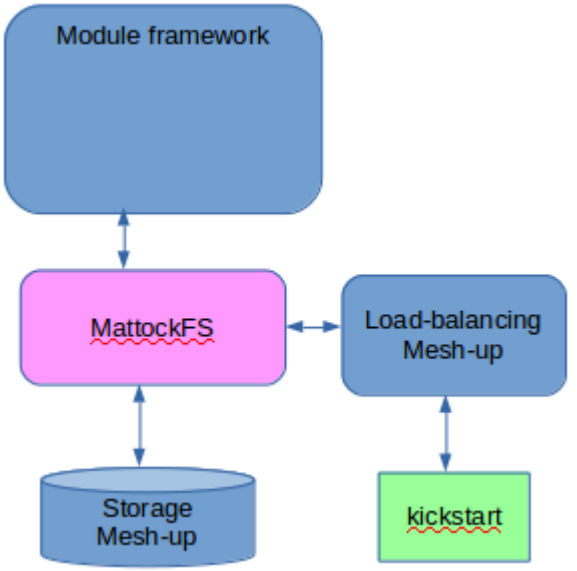
Integrated message bus

- Tool-chain level state resides in message-bus.
  - Opportunistic hash state has tool-chain lifetime.
  - Fadvise reference counting stack works with tool-chain lifetime entities.
- Interaction occurs in data-archive.
  - Fadvise invocation.
  - Opportunistic hashing on low-level reads.
  - Module picking the next job.
- Dependencies too strong to not integrate message-bus!

MattockFS is **NOT** a CF Framework!

- Foundational services for framework.

- Needs symbiotic relationship other components
  - Future module-framework.
  - Future load-balancing facilities.
  - Distributed storage mesh-up

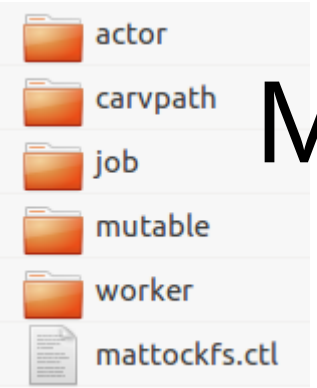


MattockFS: Hooks

- Hooks for forensic frameworks
  - Framework based routing functionality
  - Pressure data for throttling new-data input
  - Queue size for throttling
- Hooks for multi-server setup
  - Geared to locality of data
  - Migrating CPU intensive jobs

File-system interface as API

- Sparse capability as file-name and as secure textual object reference.
- Extended attributes as methods.
  - RVAL=$(attr -qg foo job/bc1286d623953.ctl)
    - val = myjob.foo()
  - attr -s bar \”$ARGUMENT\” job/bc1286d623953.ctl
    - myjob.bar(myargument)
- NOTE: MattockFS reserves “;” as argument separator!
  - attr -s bar \”$ARG1\;$ARG2\” job/bc1286d623953.ctl



MattockFS directory structure

- **carvpath/** : global mount-point level read-only carvpath access to repository
- **mattockfs.ctl**: mountpoint level methods
- **actor/**: Access to module (actor) level methods.
- **worker/**: Access to module-instance (worker) level methods.
- **job/**: Access to task (job) level methods
- **mutable/**: Creation of new data within the context of a task/job

carvpath/

- carvpath/<carvpath>.<ext>
  - Regular read-only repository data.
- carvpath/<cp1>/<cp2> → ../<cp3>
  - Self flattening symbolic link resolving.
- carvpath/<cp1>/<cp2>.<ext> → ../<cp3>.<ext>
  - Self flattening symbolic link resolving.

carvpath/<carvpath>.<ext>

- attr -g **opportunistic_hash** $CP
  - Fetch opportunistic hashing status
  - $hashresult;$hashingoffset
  - NOTE: opportunistic hashing state flushes at refcount=0
- Attr -g **fadvise_status** $CP
  - Fetch fadvise status
  - $hotsize;$coldsize
- Attr -s **force_fadvise** $val $CP
  - Overrule MattockFS use of fadvise
  - NORMAL / DONTNEED / WILLNEED / RANDOM / SEQUENTIAL / NOREUSE

mattockfs.ctl

- attr -g **fadvise_status** mattockfs.ctl
  - Information on fadvise status of the whole archive
  - $hotsize;$coldsize

- attr -g **full_archive** mattockfs.ctl
  - Carvpath representing the entire archive section that this mount-point manages.

actor/

- actor/<ACTOR>.inf
  - Provides info on the actor to workers of OTHER actor.
- actor/<ACTOR>.ctl
  - Provides registration interface for workers of THIS actor.
  - Mandatory Access Control is advised for POLA reasons!

actor/<ACTOR>.inf

- Attr -g worker_count actor/ocr.inf
  - Fetch the current number of locally registered workers for this actor.
- Attr -g anycast_status actor/ocr.inf
  - Fetch potential throttling info for data targeted at this module.
  - $setsize;setvolume

actor/<ACTOR>.ctl

- weight/overflow
  - Settable attributes for future networked load balancing purposes.
  - Weight: used for determining jobs of what module to migrate to other nodes.
  - Overflow: Never migrate jobs to an other node unless overflow is exceeded.

actor/<ACTOR>.ctl (2)

- Attr -g register_worker actor/mmls.ctl
  - Register current process as a worker of this actor.
  - Returns a $workercap sparse capability needed
  - worker/Cec6ab930c3833d02d60764fa8ad1e9e0e3eaab30d43ba14e63295cdd4aed65d2.ctl
  - WCPATH=$(attr -g register_worker actor/mmls.ctl)
- attr -s reset 1 actor/mmls.ctl
  - Force unregister any worker currently registered for the named actor.

worker/<WOKERCAP>.ctl

- attr -s unregister 1 $WCPATH
  - Unregister a worker that was registered to an actor.
- job_select_policy: settable policy for fetching the next job:
  - H (default): lowest opportunistic hashing index.
  - K : create a starting job, meant for kickstart module.
  - O: lowest offset.
  - R/r: Prefer entities with highest/lowest possible refcount
  - D/d: highest/lowest possible density of chunks with the maximum refcount
  - S: smallest data

worker/<WORKERCAP>.ctl (2)

- module_select_policy: settable policy meant for load balancing purposes.
- Attr -g accept_job $WCPATH
  - Get a handle to the next job we can process.
  - Returns a $jobcap sparse capability needed for job level operations.
  - job/Cd06cb930c3833d02d60764fa8ad1e9e0e3eaab30d43ba14e63295cdd4aed65d2.ctl
  - JCPATH=$(attr -g accept_job $WCPATH)

job/<JOBCAP>.ctl

- attr -g job_carvpath $JCPATH
  - Get the carvpath of the data this job relates to
- attr -g routing_info $JCPATH
  - Get info meant for the module lib routing engine.
  - $targetactor;$routerstate
- attr -s routing_info “$actor\;$rstate” $JCPATH
  - Forward job to the next actor.

job/<JOBCAP>.ctl (2)

- attr -s submit_child $CHILDINFO $JCPATH
  - Child carvpath
  - Next actor
  - Initial child router state
  - Mime-type
  - File extension

job/<JOBCAP>.ctl (3)

- Allocating storage for child entity:
  - attr -s allocate_mutable $BYTES $JCPATH
  - attr -g current_mutable $JCPATH
    - Returns a $mutable capability needed for operations on the allocated mutable data.
- Freezing the newly created data.
  - attr -g frozen_mutable $JCPATH
    - Returns the read only carvpath to the frozen data.
    - Carvpath should be used in submit_child operation!

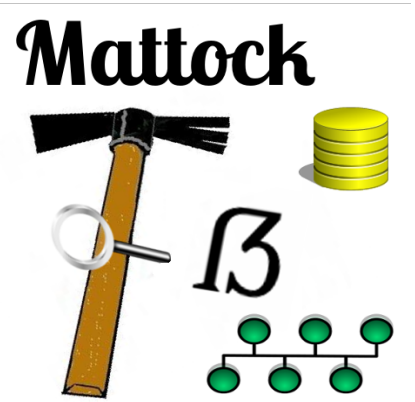
mutable/<MUTABLECAP>.dat

- File should be opened as “r+”
- Allocated space must be honored
- Non written chunks will end up sparse.

Trusted provenance log

- Fuse file-system runs as different user
- MattockFS is responsible for writing provenance log
- Completed tool-chains.
- /var/mattock/log/<MPNR>.provenance

Todo:



- Address module_select_policy safety.
- Address redis usage robustness concerns.
- Implement recovery features:
  - Restore from journal
  - Quarantine for module-crashing data.
- Implement secondary opportunistic hashing
- Write basic language bindings

Todo:

- Address *module_select_policy* safety.
- Address redis usage robustness concerns.
- Implement recovery features:
  - Restore from journal
  - Quarantine for module-crashing data.
- Implement secondary opportunistic hashing
- **Write basic language bindings**
  - Consider writing one for your favorite language!