Spectrum Scale Policy
"Best Practices"

Marc A Kaplan

makaplan@us.ibm.com

2018.05.17

Policy Rules

SQL-like statements to control:

- Storage Pool selection at creation (SET POOL)
- Pool to Pool MIGRATE by mmapplypolicy command
- To/From EXTERNAL POOLs, aka TSM/HSM, HPSS, TCT
- General LIST and EXEC rules for rapid, parallel processing of files selected by WHERE (sql-expr-over-file-attributes: ACCESS_TIME, NAME, USER_ID, MODE, eXtendedATTR, ...)

Spectrum Scale Policy Rules

```
Rule 's1' SET POOL 'images' LIMIT(94)
          WHERE NAME LIKE '%.jpg' OR NAME LIKE '%.mpg'
Rule 's2' SET POOL 'data'  LIMIT(92)
Rule 's3' SET POOL 'oops'  /* default */

define([access_days],
      [(CURRENT_TIMESTAMP-ACCESS_TIME> INTERVAL '$1' DAYS)])

Rule 'm1' MIGRATE TO POOL 'cool'  WHERE access_days(14)

Rule 'd2' DELETE  WHERE access_days(31) AND
                        PATH_NAME LIKE '%/tmp/%'

Rule 'l1' EXTERNAL LIST 'LA'  EXEC ''
Rule 'l2' LIST 'L'  WHERE some_sql_condition

Rule 'l3' EXTERNAL LIST 'LB' EXEC ''
Rule 'l4' LIST 'LB' WHERE some_other_sql_boolean_expr
```

Policy engine == SQL interpreter

- code linked into the mmfsd "the daemon" binary and ...
- linked into tsapolicy, the mmapplypolicy binary
- C++ virtual methodology adapters for the different runtime environments
- based on Bob Rees' interpreter, Storage Tank... subset of ANSI SQL plus the m4 macro processor
- run time optimizations - 10/90 and fixes
- IBM Spectrum Scale: Administration Guide, Chapter 23 “Information Lifecycle Management ...”

mmapplypolicy: a Parallel, Robust File Scanner

- node and thread parallel
- one master process, multiple helper processes
- Posix threads + threads use popen command pipes
- DirectoryWalk - as directories are discovered, work is distributed to helpers and threads. Checkpointing with recovery from helper failure.

  ==> lists of (inode, path) - organized into inode number ranges from each node. [inode_range_j, node_k]
- piped sorts for inode ranges [j,*] drive parallel inodescans
- InodeScan exploits sequentiality of inode file
- rules/SQL evaluation of each inode and its attributes

File Scanner and Exec, cont'd...

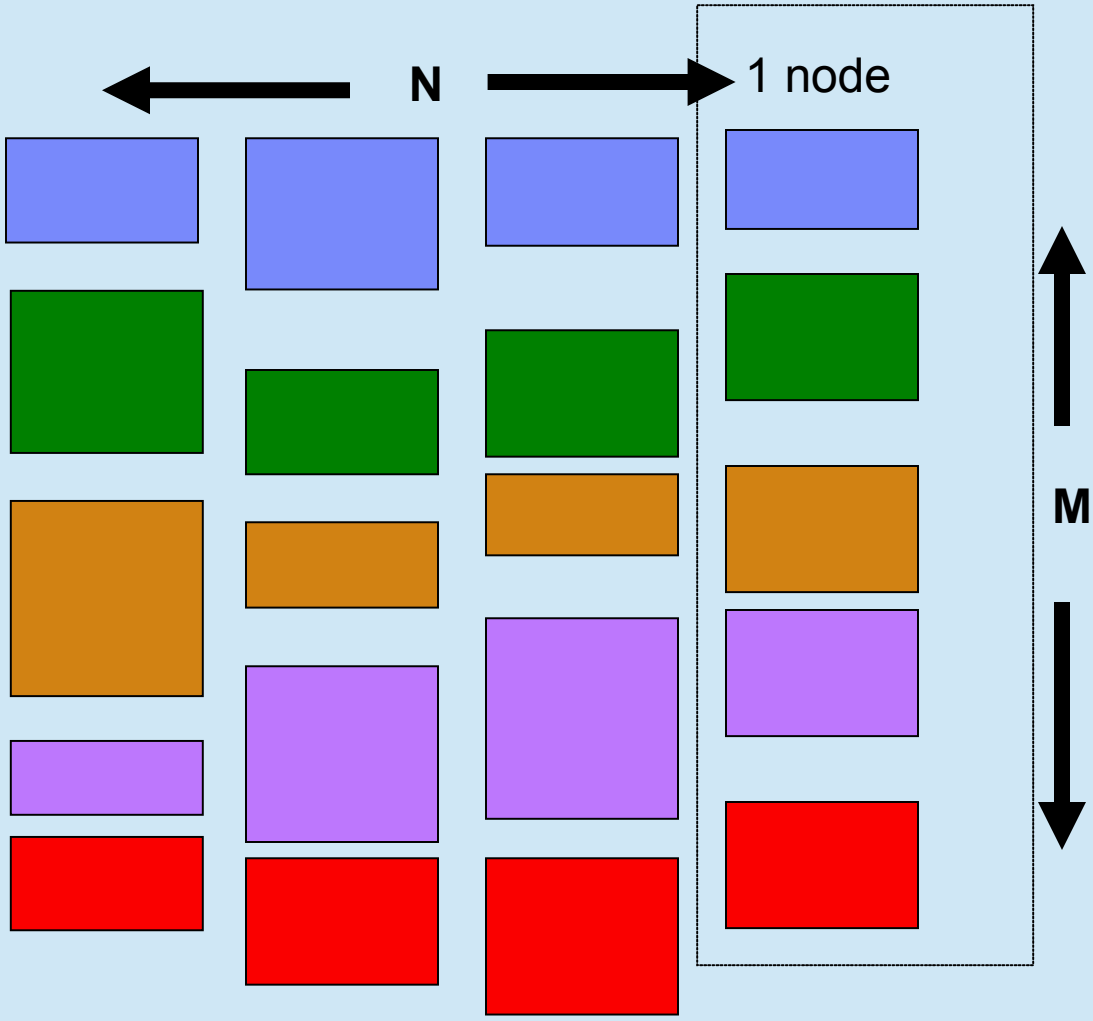
- results of inode-scan-SQL-eval is set of Policy Decision Records : (weight,inode, path, rule_index, pool, other-attributes,...) (WEIGHT(sql_numeric_expr) in rule)
- multiple PDR files [node_k, m]
- parallel sort-merge of PDR files... popen(/bin/sort ...) for WEIGHT(age) THRESHOLD(90,60) based choices of MIGRATE, DELETE, and/or LIST-EXEC - the PdrScan
- parallel execution of migrate, delete, EXEC-script,

by "bunches" of PDRs

- robust against helper failures - redo of inode range, redo of PDR bunch

Parallel Directory and Inodes Scans

1. Each of N nodes starts with a directory to walk.
2. Each directory entry is assigned to one of M (sub)buckets based on high bits of inode number.
3. The N nodes store entries into N x M sub-buckets. (typ. choose M>N)
4. In the next phase, nodes are assigned rows of work. Each row has 1/M of all inodes.
5. Each row of N sub-buckets is sorted in inode order for policy evaluation. Each node evaluates policy rules on the inodes in its assigned rows.



Also...

- mmbackup - built on mmapplypolicy
- mmimgbackup and restore - mmapplypolicy with dynamically linked (shlib) special options and hooks - HSM based disaster recovery or file system export /import.
- AFM - maintenance and recovery commands use mmapplypolicy, special options: scan all inode numbers, with or without matching directory entries, supply [(inode,path)...] input file, ...

parallel `find ... | xargs ...` ==> `mmfind ...-xargs ...`

- find files that match criteria and execute a command or any script on each file
- implemented as a perl script that translates classic find predicates (-f -o -a -newer ...) to gpfs-policy-sql and invokes mmapplypolicy
- fully parallelized, multi-node, multi-threaded directory walk, inodescan, sorting/selection, command executions
- **samples/ilm/mmfind** & friends
- FindTo Sql translator – tr_findToPol.pl - may be used as an assistant or "crutch"
- mmfind ... --help --polflags '-N all -g /gpfs/tdir' ...

mmxargs – take care of “special characters” in pathnames when using LIST rules

```
#  mmapplypolicy path -P rules  -M OLDU=matt -M NEWU=makaplan

RULE 'x' EXTERNAL LIST 'x' EXEC '/usr/lpp/mmfs/bin/mmxargs'
                           OPTS 'chown NEWU'

RULE 'x1' LIST 'x' DIRECTORIES_PLUS
          WHERE USER_NAME=OLDU
```

```
# mmapplypolicy … -I defer -f /path/pre … #  to save file LISTs
```

Pathnames can contain any byte values 0x01..0xFF, not-necessarily UTF-8
Default policy LIST format only escapes \\ and \n.
Alternative:

```
 rule 'x' external list 'x' exec '…' opts '…' ESCAPE '%/+@'
  /* RFC3986  %xx encoding of non-alpha-numerics with a few exceptions */
```

Learn Policy/SQL by “Examples and Tips”

in Spectrum Scale Administration Guide

Dates and Weights:

RULE ’a’ MIGRATE TO POOL ’A’
 WEIGHT(*CURRENT_TIMESTAMP - ACCESS_TIME*)
 WHERE
  CURRENT_TIMESTAMP - MODIFICATION_TIME > INTERVAL ’10’ DAYS

Use `m4`, SHOW, -I test, -L 6: *(often -L2 or -L3 is enough)*

define*(access_age_in_days,*
 (INTEGER(((CURRENT_TIMESTAMP – ACCESS_TIME)
*SECONDS)) /(24*3600.0)))*

RULE external list 'w' exec ''
RULE list 'w' ***WEIGHT****(access_age_in_days)* ***SHOW****(access_age_in_days)*

 # mmapplypolicy /root/test_dir -P rules ***-I test -L 6***

Policy TimeStamp to Unix Seconds

```
define([toSeconds],[(($1) SECONDS(12,6))])
define([toUnixSeconds],[toSeconds($1 - '1970-1-1@0:00')])

RULE external list b ...

RULE list b
 SHOW('sinceNow=' toSeconds(current_timestamp-modification_time) )

RULE external list c ...

RULE list c
  SHOW('sinceUnixEpoch=' toUnixSeconds(modification_time) )
```

LIKE is nice but sometimes you want the power of Regex

… WHERE REGEX(name,['^[a-z]*$']) /* only accept lowercase names */

… WHERE NOT REGEX(STRING_VALUE,['^[^z]*$|^[^y]*$|^[^x]*$|[abc]'])
/* test if STRING_VALUE contains ***all*** of the characters x, y, and z, in any order,
and ***none*** of the characters a, b, or c. */

Say less, do more ...

Rule 'm' MIGRATE TO POOL 'data'
/* no FROM POOL ==> all pools
no WHERE == WHERE TRUE == WHERE NAME LIKE '%' */

LIKE From pools, For Filesets ... but more

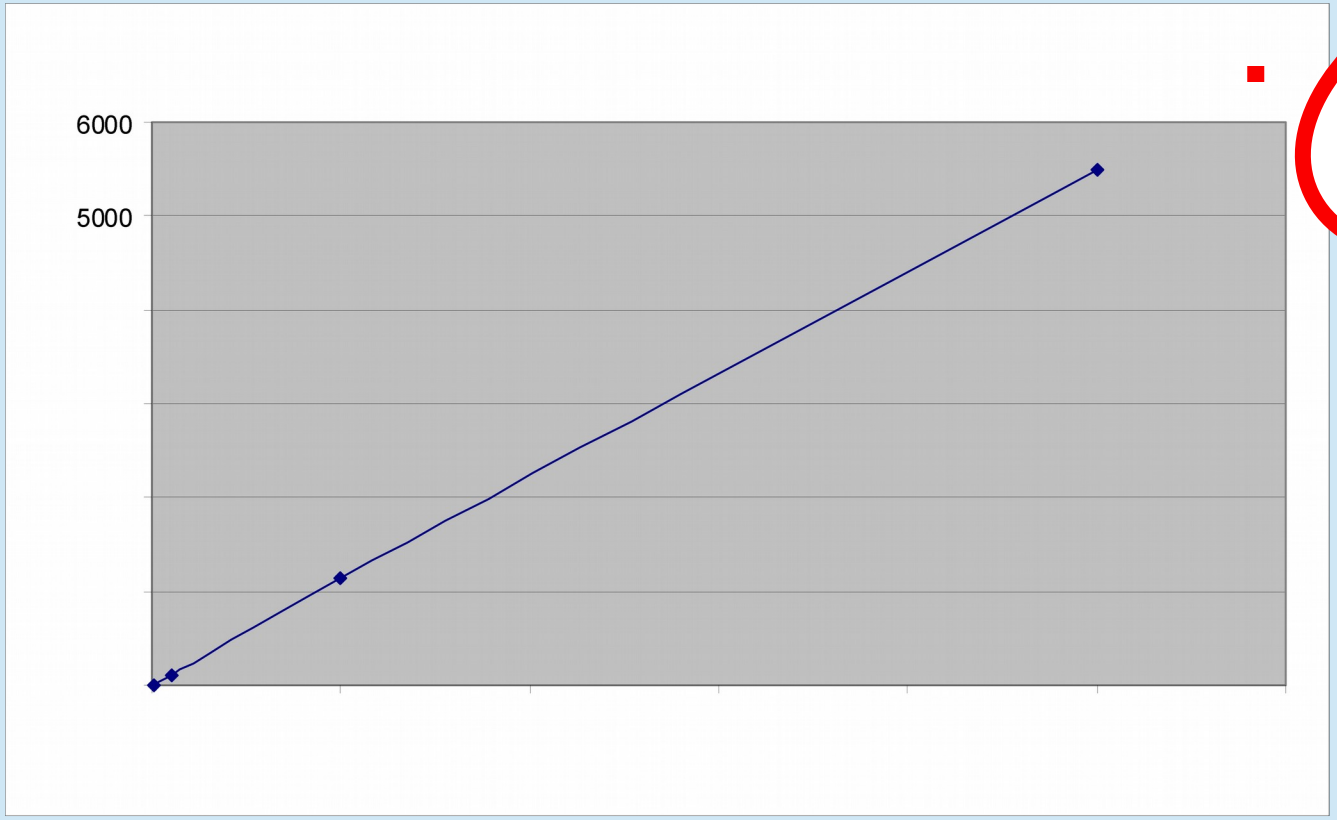
WHERE POOL_NAME LIKE 'dat%'
WHERE FILESET_NAME LIKE 'fx%'

file scan benchmarks

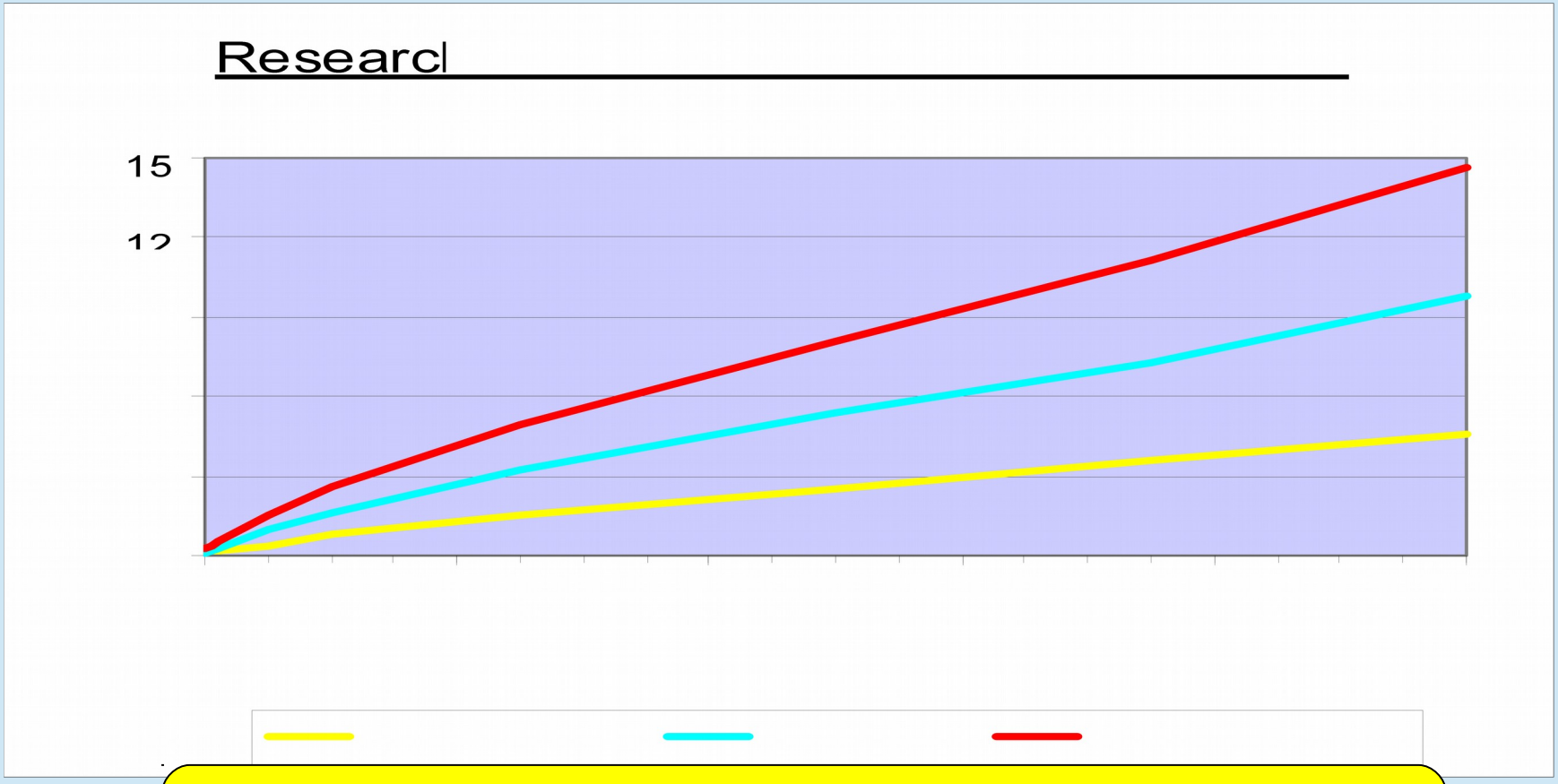
- IBM intros Elastic Storage - as used by HPC brain Watson 10 billion files, 43 mins ... *Where've we heard that before?* by Chris Mellor (The Register: 2014)
- GPFS Scans 10 Billion Files in 43 Minutes 10 Billion F iles in 43 Minutes 10 Billion Files in 43 Minutes - DS Con (2011: Freitas, Slember, Sawdon, et.al.) on two SSD boxes with 10 nodes
- 1 Billion in 20 minutes (2007) on real disks with 8 n odes

GPFS/HPSS Billion File Demo at SC'07

Show complete HSM/BA solution for 1B files



375 M fph



Less than 1 Second per Million Files!
Less than 15 Minutes per Billion Files!

Using 8x IBM eServer xSeries 336 (3.2GHz Intel Xeon)
attached at 4Gb/s to
IBM DS-4800 (2.1 TB RAID1) & IBM DS-4100 (10.2TB RAID5)

mmapplypolicy: File Scanning Performance Tips

- Use fastest storage for system pool == metadata == directories + inodes + …
  - SSD, Flash, or Fastest Disks
    ... but probably NOT Raid-5 – stripe update problem
    … independently seeking devices, non-interfering IO paths
    for thread and node parallel access
  - mmcrnsd … %nsd ... pool=system,usage=metadataOnly
  - consider metadata blocksize vs data blocksize
    mmcrfs … -B nnnn … --metadata-block-size mmmm
- more files per directory ==> faster directory scanning
  or small directories ==> directory in inode

- mmapplypolicy … **-N** nodelist … **-g** sharedTmpDir
  mmchconfig … **defaultHelperNodes** … **sharedTmpDir**
  - Release 5.0.1 defaults: -N **managerNodes** and -g **.mmSharedTmpDir**
  - Prior default: single node execution with multi-threading

- mmapplypolicy … --choice-algorithm fast && ... WEIGHT(0) …
  (avoids final sort of all selected files by weight)

- mmapplypolicy /pathToIndependentFileset --scope inodeSpace
  (scan only the files and inodes in an independent fileset)

Be aggressive...

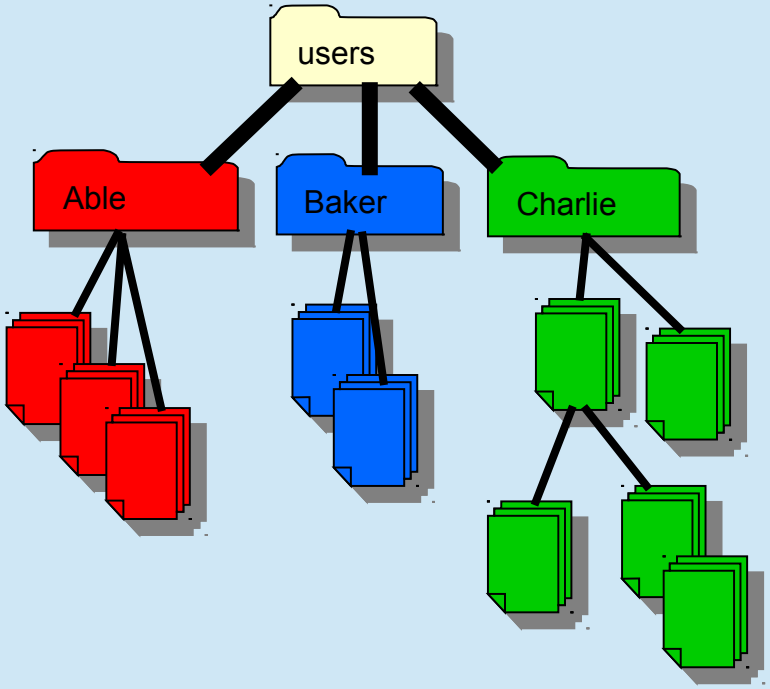
mmapplypolicy ... -a IscanThreads
= number of inode scanning threads per node
with one sort process feeding each inodescan (Use sparingly!)

Or pace yourself with QOS…

mmchqos FS --enable pool=*,maintenance=100iops
mmapplypolicy …

```
[I] Qos 'maintenance' configured as 100.0IOPS
```

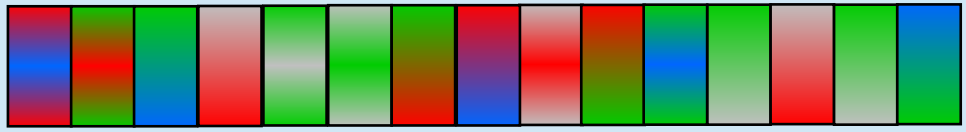
“independent” Filesets



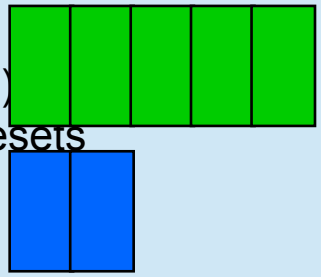
- Filesets divide name space
  - Named subtree may be unlinked & moved
  - Share underlying storage
- Original Filesets
  - Shared inode space (within blocks)
    - Cost proportional to file system size
- New Filesets
  - Logically have private inode space
    - Cost proportional to fileset size

  How to make Inode numbers unique?
  (without fixed partitions)

Inode Space 0 → 2^{60}



- *Dynamic* Inode Space Partitioning
  - Shared inode space (by block ranges)
  - No fixed limit on number of files or filesets
    - 64 bit limit on total



⇒ Per fileset Snapshots, Backup, Restore, Data Management, …

LWE-beyond file creation and mmapplypolicy...

- Light Weight Events - "hook" OPEN, READ, WRITE, CLOSE, RMDIR, ..., posix and GPFS api.
- EVENT 'OPEN' ACTION(any) WHERE (sql-expr)
- ACTION(any) can be any sql expr: eval TRUE|FALSE, including functions with effects:
- SetXattr, System(any-program-and-args-as-sql-string-expr), SetSpecial(caching-controls), other callbacks to internal file system methods
- Basis for TCT, mmaudit, other audit-like APIs coming