Los Alamos
NATIONAL LABORATORY

Large Scale File/Storage System Indexing with GUFI

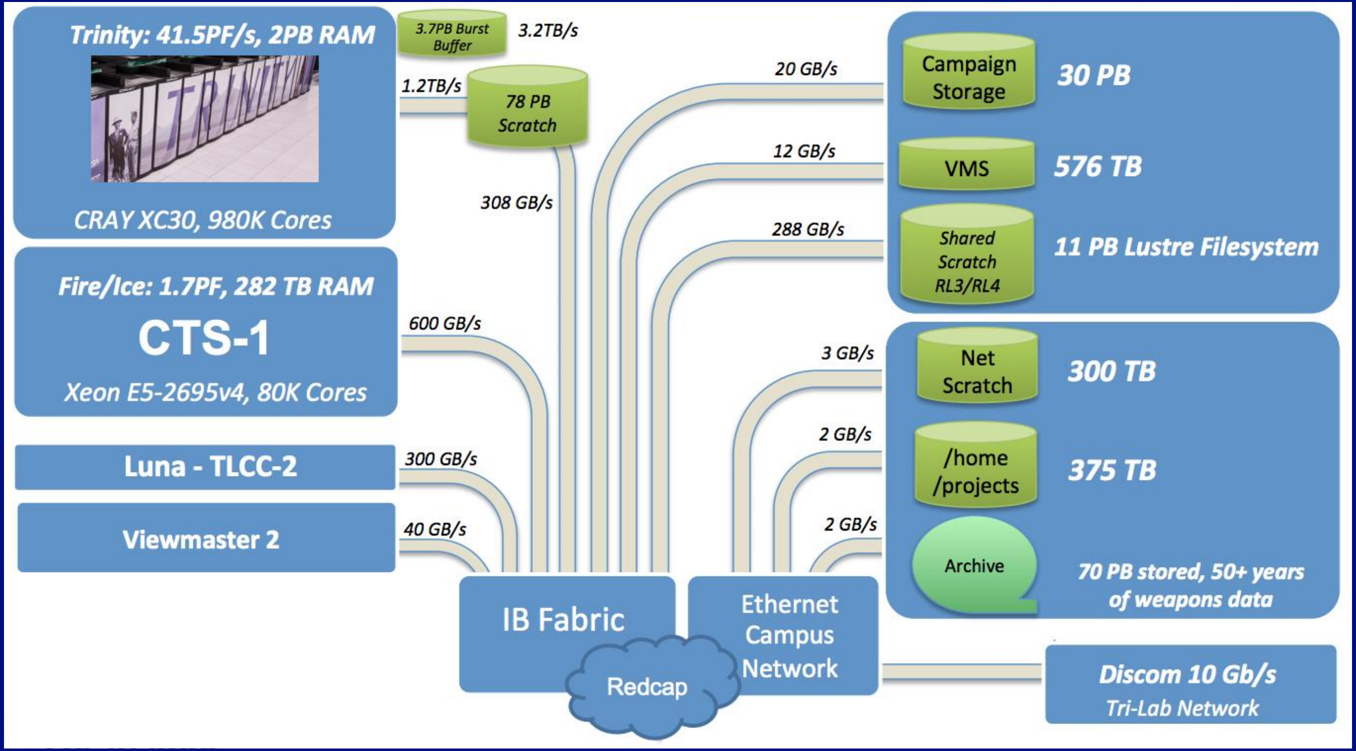
Jason Lee, Dominic Manno, Gary Grider

May 22, 2023

LA-UR-23-24629

Managed by Triad National Security, LLC, for the U.S. Department of Energy's NNSA.

LANL Compute/Storage Environment (Secure) Circa early 2017



Sampling of LANL Filesystems Circa 2020/2021 (Turquoise)

Filesystem	Directory Count (Millions)	File Count (Millions)
Home	3.8	36.6
Projects	10.7	114.2
Scratch 1	1.1	237.2
Scratch 2	5.1	857.7
Campaign	0.4	13.5
Archive 1	1.1	49.9

Filesystem Usage

- Users searching for data in files
  - Do not always know where files are
  - Lots of filesystems with lots of files
  - Files within a directory might be organized poorly
  - Want fast results (or will terminate search)

- Admins need to manage filesystem
  - Find users taking up the most space
  - Find stale files that can be deleted
  - Want reasonably fast results

No Unified Set of Performant Tools

- Different admin tools for different filesystems
  - Admins only

- Standard command line tools
  - Slow
    - Single threaded
  - Unwieldy
    - Must chain multiple commands together to get results
  - Uses resources of mission critical jobs
    - Parallel walking tool (MPI) would use more host resources

Grand Unified File Index

- Highly parallel for fast index traversal
- Stores metadata in databases
  - Complex queries with SQL
- Enforces permissions so users and admins can use the same index
  - Support for extended attributes
- Single index for all filesystems
- Leverages well developed technologies
  - POSIX filesystem hierarchy, permissions, attributes
  - SQLite 3, PCRE, jemalloc, CMake 3
  - Flash Storage

2018 R&D 100 WINNER

Fastest open-source software for supercomputer user-queried metadata

- Provides metadata queries of ultrascale file-system trees in seconds
- Maintains security structure while facilitating custom user metadata queries
- Economizes supercomputing resources — enhancing the role of the user
- Offers open-source software of a mere few thousand lines that is concise and extensible

Indexing

- `gufi_dir2index`
  - Create index from filesystem

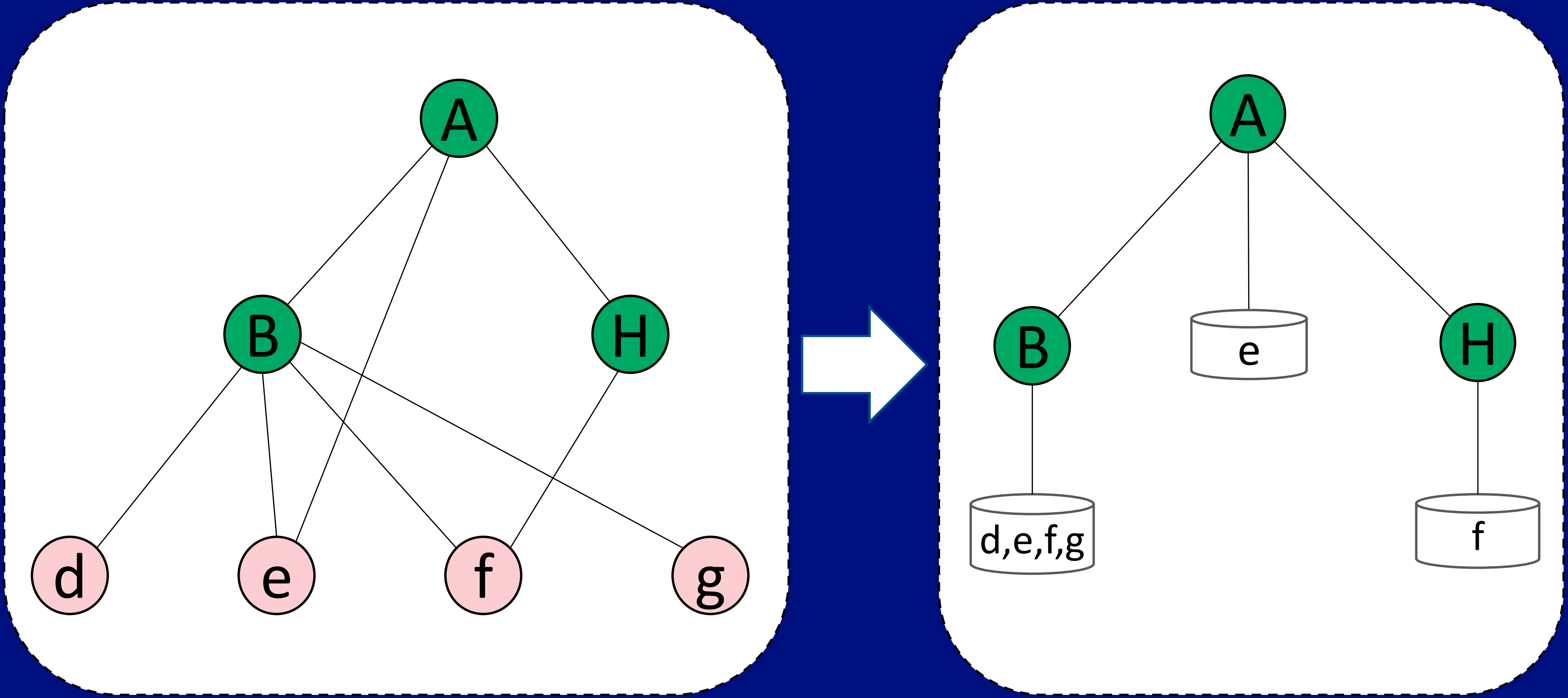
- `gufi_dir2trace` + `gufi_trace2index`
  - Create trace file(s) from filesystem
  - Create index from trace file(s)
  - Allows for indices to be moved off original system without copying directories

- Indices are not up to date
  - Scans take time to complete
  - Live filesystems are always churning (unless indexing snapshots)
  - Scan filesystems every so often
    - LANL runs full scans every 4 hours
    - Incremental updates are possible but untested

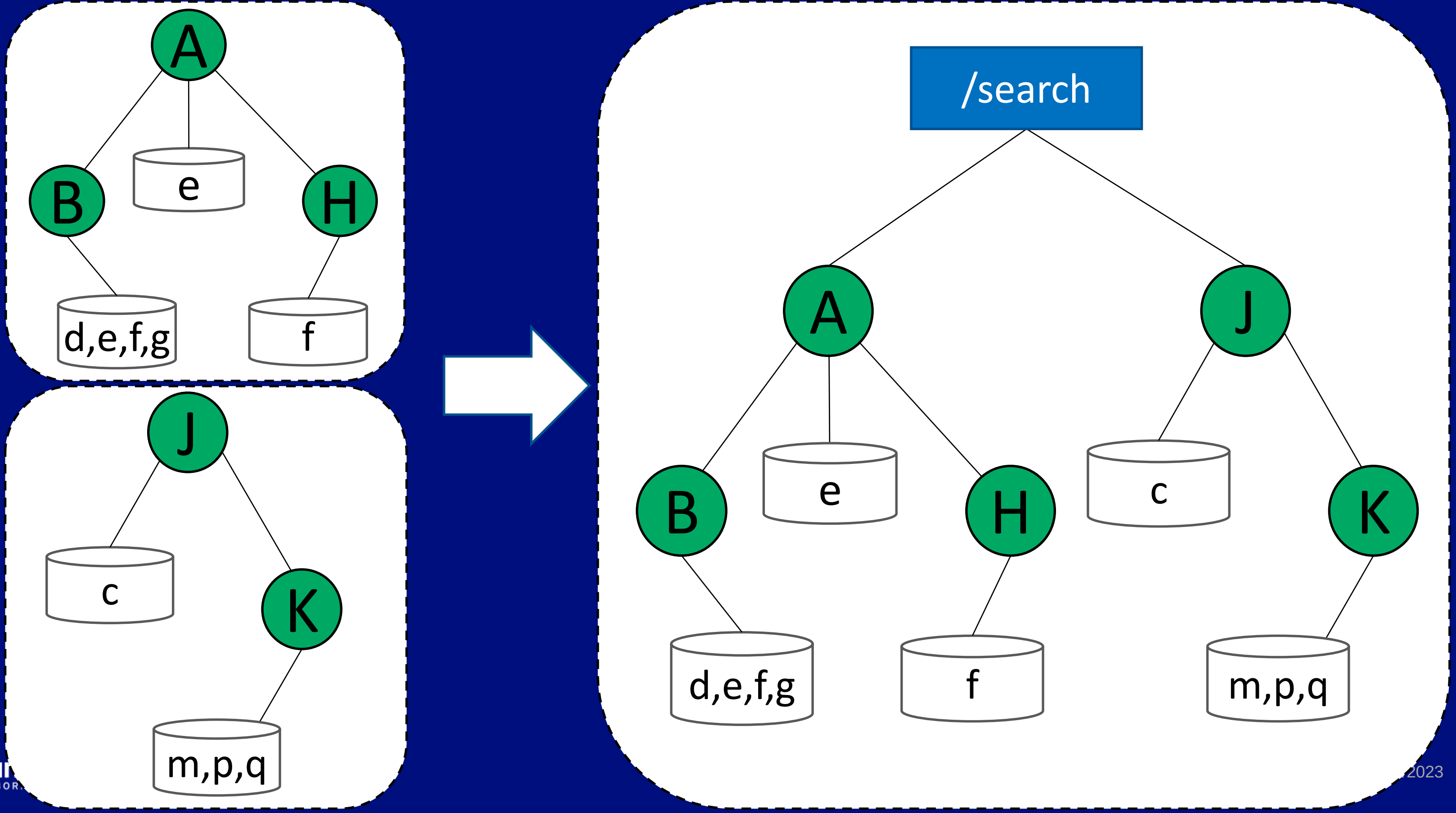
Filesystem specific optimizations

- HPSS
- GPFS
- Lustre

Source Filesystem to Index



Combining Indices



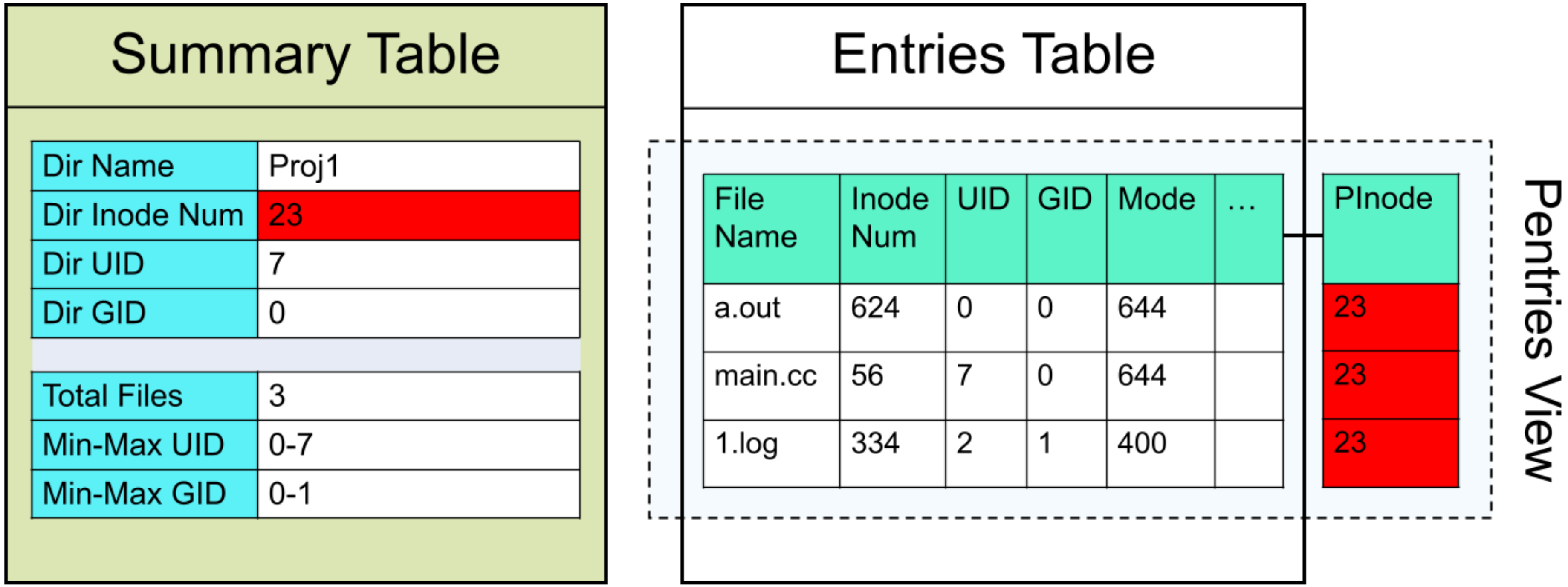
Why not a flat index?

- Very performant for simple queries
  - No tree traversal
  - One/few database(s) to open

- Multiple uids/gids in one database
  - Custom per row permission checking or admin only
    - For each path segment

- Must scan all entries when querying
  - Constant time queries
    - Queries do not scale based on caller
  - Scan multiple times when joining
  - Lots of I/O

Database Table Schema



Summary Table

Dir Name	Proj1
Dir Inode Num	23
Dir UID	7
Dir GID	0
Total Files	3
Min-Max UID	0-7
Min-Max GID	0-1

Entries Table

File Name	Inode Num	UID	GID	Mode	…	PInode
a.out	624	0	0	644		23
main.cc	56	7	0	644		23
1.log	334	2	1	400		23

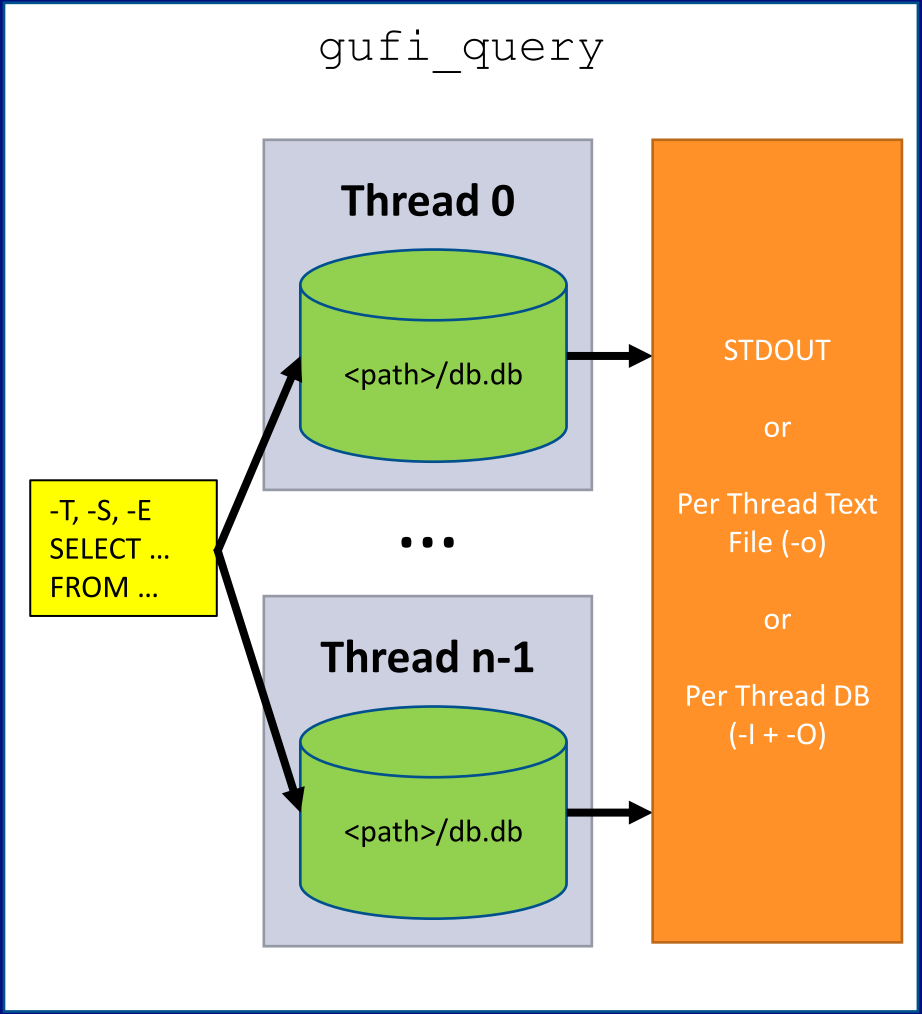
Tables and Views

- Basic Queries
  - Use summary table and pentries view

- Should actually use (will talk about later):
  - vrsummary, vrpentries
  - When xattrs are present
    - vrxsummary, vrxpentries

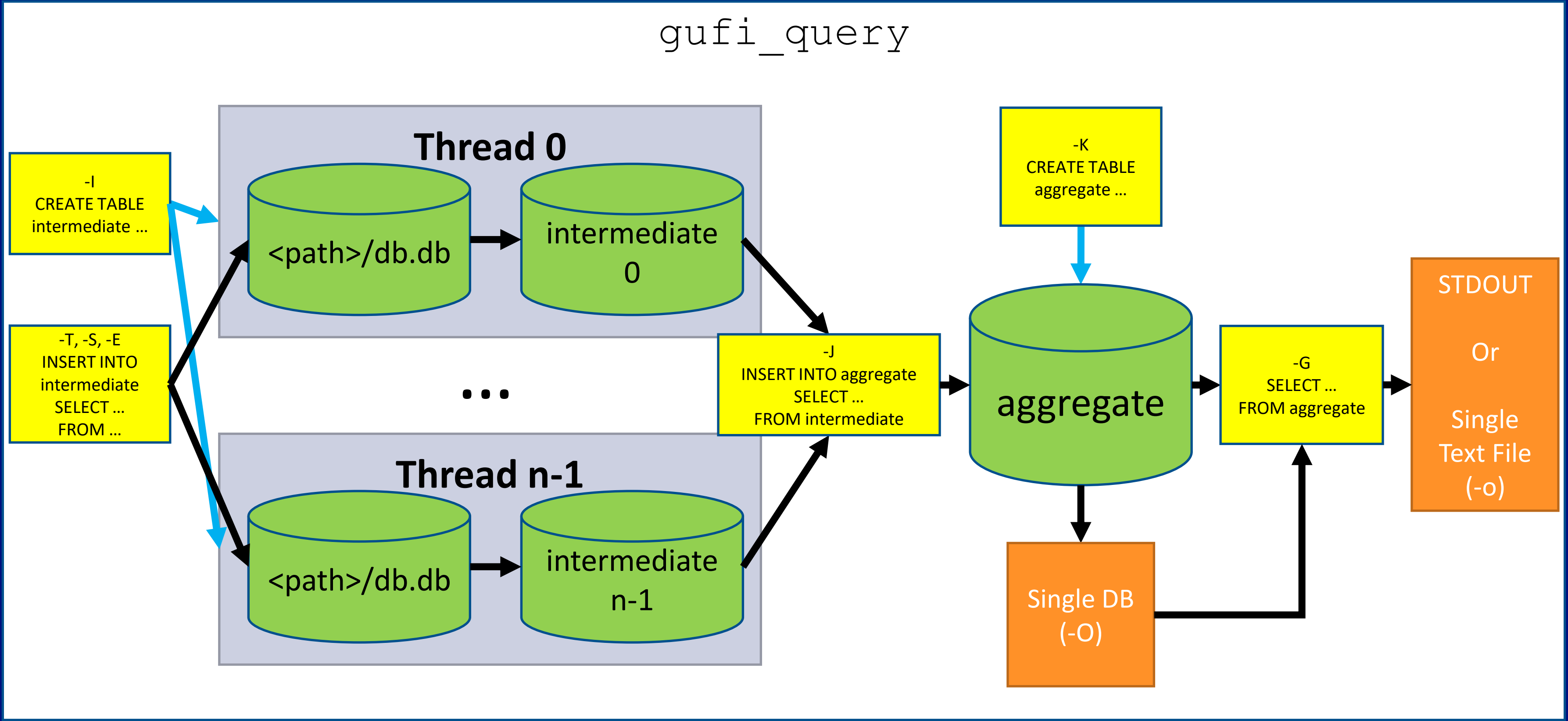
`gufi_query`

- Runs raw SQL statements
  - Need to know database and table schemas
  - Meant for admin/advanced users
    - User facing tools wrapping `gufi_query`
- Highly parallel
  - Each directory is processed by a single thread

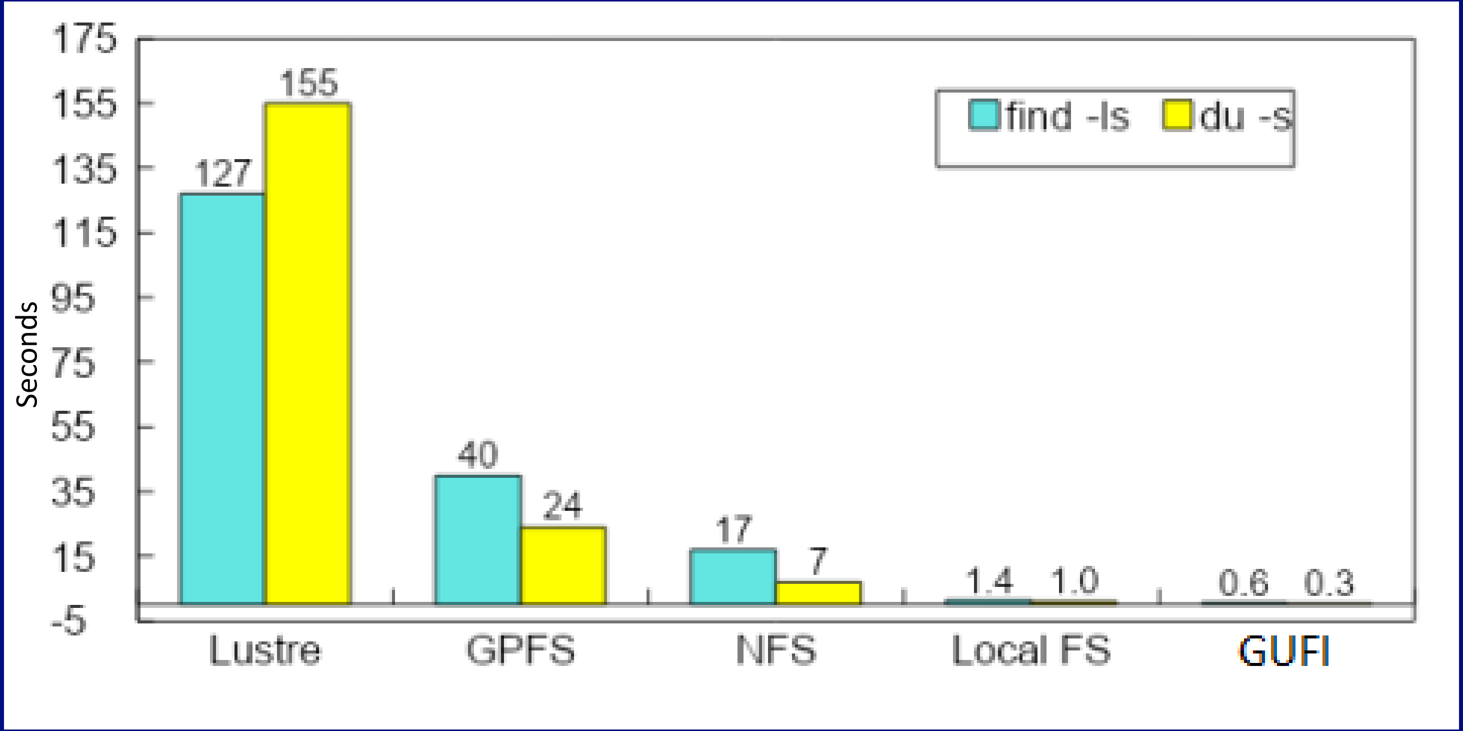


Los Alamos NATIONAL LABORATORY

Aggregation


gufi_query
Thread 0
<path>/db.db
intermediate 0
...
Thread n-1
<path>/db.db
intermediate n-1
-I CREATE TABLE intermediate ...
-T, -S, -E INSERT INTO intermediate SELECT ... FROM ...
-J INSERT INTO aggregate SELECT ... FROM intermediate
-K CREATE TABLE aggregate ...
aggregate
-G SELECT ... FROM aggregate
STDOUT Or Single Text File (-o)
Single DB (-O)

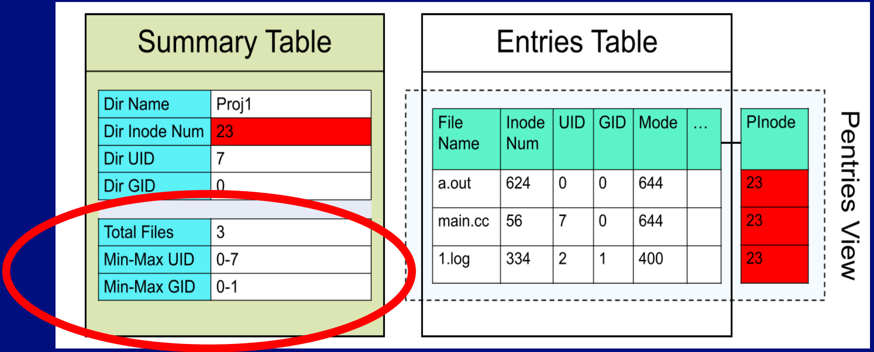
Querying Linux Kernel 5.8.9 Source (74K dirs + files)



Compound Queries

- Use the [directory] summary table to determine whether to run query on entries table
  - Quickly find out if the current directory contains an entry with value X
  - Automatically created

- Use the tree summary table
  - Summary of entire subtree starting at current directory
  - Determine whether a subtree should be traversed
    - Quickly find out if a subtree contains an entry with value X
    - Quickly get a value without walking the subtree
  - Not generated by default



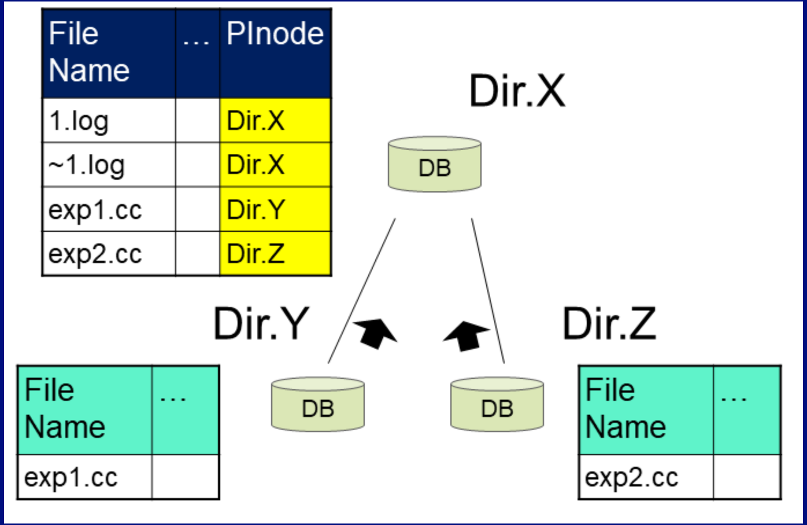
Dir Name	Proj1
Dir Inode Num	23
Dir UID	7
Dir GID	0
Total Files	3
Min-Max UID	0-7
Min-Max GID	0-1

File Name	Inode Num	UID	GID	Mode	…	PInode
a.out	624	0	0	644		23
main.cc	56	7	0	644		23
1.log	334	2	1	400		23

Where X can be
- Subdir count
- File count
- Link count
- Min/max
  - size
  - uid
  - gid
  - …
- User defined values
  - Minor schema/code changes

Rollup (Permission Based Sharding)

- Most subdirectories under a directory have compatible uid, gid, and read permissions
  - Why traverse the subtree and open multiple directories/databases if one will suffice?
- Copy data from subdirectories upwards if uid, gid, and permissions allow for it
  - Skip traversing entire subtrees and still get the same data
- Copy data up one level at a time
  - Lots of duplicate data and used space
  - Allows for querying to start at any level and still take advantage of rollup
- Don't always roll up fully
  - Large directories can cause large tail latency

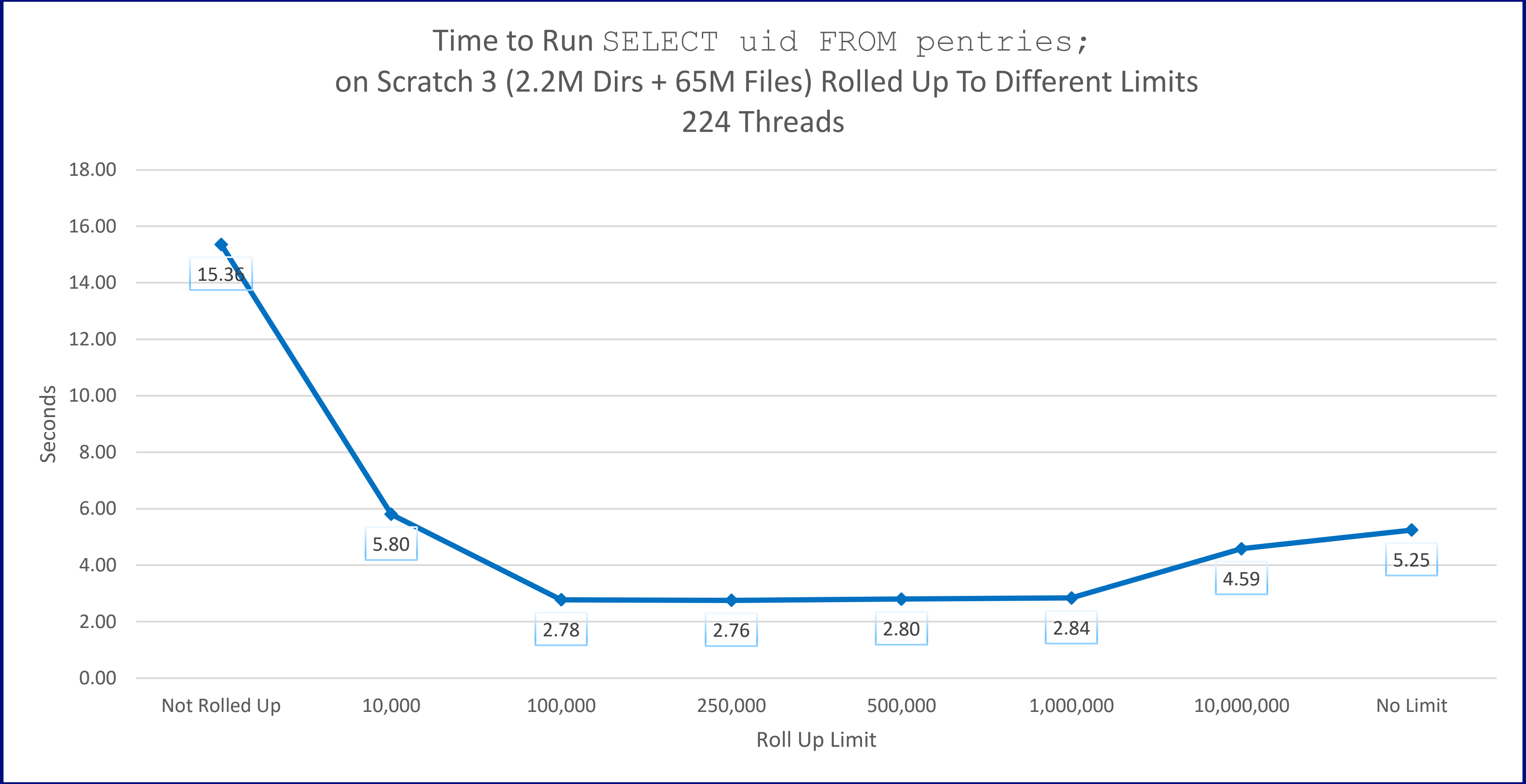


File Name	…	PInode
1.log		Dir.X
~1.log		Dir.X
exp1.cc		Dir.Y
exp2.cc		Dir.Z

File Name	…
exp1.cc	

File Name	…
exp2.cc	

Index	Original Directory Count	# of Rolled Up Directories to Traverse	% of Rolled Up Directories to Traverse
anony	7.35M	2873	0.04%
yelluser	1.62M	6406	0.39%
scratch 3	2.20M	5049	0.23%


Time to Run `SELECT uid FROM pentries;`
on Scratch 3 (2.2M Dirs + 65M Files) Rolled Up To Different Limits
224 Threads
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
Seconds
15.36
5.80
2.78
2.76
2.80
2.84
4.59
5.25
Not Rolled Up
10,000
100,000
250,000
500,000
1,000,000
10,000,000
No Limit
Roll Up Limit

Rollup Rules

1. World read and exec (i.e. o+rx)

2. Matching perms (usr, grp, and other), with same usr and grp

3. Matching usr and grp perms, read and exec (ug+rx) with same usr and grp, and not world read and exec (i.e. o-rx)

4. Matching usr perms, read and exec (u+rx) with the same usr, and not grp or world read and exec (go-rx)

Extended Attributes (xattrs)

- Small user data stored with metadata
  - Tag files

- Different permission handling than `stat(2)` data
  - Need read permission of files instead of the directory they are in
  - Compatible xattrs are stored in the main database
  - Incompatible xattrs are stored in per-uid and per-gid databases are attached during querying
    - Successful attach indicates that the user can read the xattr values

- Includes rolling up xattrs

xattr Rules

1. File is 0+R (doesn't matter what the parent dir perms or ownership is)

2. File is UG+R doesn't matter on other, with file and parent same usr and grp and parent has only UG+R with no other read

3. File is U+R doesn't matter on grp and other, with file and parent same usr and parent dir has only U+R, no grp and other read

4. Directory has write for every read: `drw*rw_*rw*` or `drw*rw*___` or `drw*______` - if you can write the dir you can chmod the files to see the xattrs

User Facing Tools

- Requires on `/etc/GUFI/config`
  - `gufi_find`
    - `find(1)`
  - `gufi_ls`
    - `ls(1)`
  - `gufi_getfattr`
    - `getfattr(1)`
  - `gufi_stats`
    - Queries that are probably useful
- `gufi_stat`
  - `stat(1)`

More Information

- Source Code, Documentation, and Previous Presentations
  - https://github.com/mar-file-system/GUFI
  - Contributions are welcome!

- Anonymized Traces From LANL Systems
  - https://github.com/mar-file-system/GUFI-Filesystem-Traces

- Dominic Manno, Jason Lee, Prajwal Challa, Qing Zheng, David Bonnie, Gary Grider, and Bradley Settlemyer. 2022. GUFI: fast, secure file system metadata search for both privileged and unprivileged users. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22). IEEE Press, Article 57, 1-14.

Future Work

- Index Object Stores

- Index and Query Even Larger Datasets
  - University of Michigan Advanced Research Computing scans their archives
  - Recently helped Texas Advanced Computing Center index BeeGFS
    - 56.6M Directories with 16 Threads took ~35.75 Hours
    - 38.6M Directories with 16 Threads took ~15 Hours
  - 1 Billion Directories?

- Longitudinal Studies on LANL Filesystems

Use Case: Full Text Search on Hierarchical Archival Data

- Received archival data stored in directory tree
  - Scraped data from archive and inserted into GUFI
    - Image Metadata + OCR
    - MS Office Files
    - PDF Text
    - etc.

- Full Text Search in GUFI with minor changes
  - Added new table and created view joining on inode == full text inode

- Full Text Search in SQLite
  - Built-in Extensions (FTS 3/4, 5)
  - Custom Extensions

Tutorial/Demo

Source and Traces

1. `git clone https://github.com/mar-file-system/GUFI`
2. `git clone https://github.com/mar-file-system/GUFI-Filesystem-Traces`
   - ~11GB
   - Recommend using a local directory for this tutorial
3. `cat GUFI-Traces.part* > GUFI-Traces.tar.bz2`
4. `tar -xjf GUFI-Traces.tar.bz2`
   - ~134GB

GUFI Dependencies (Minimal)

- System Tools
  - attr
  - autotools
  - bash
  - C compiler
    - C++ is optional
  - CMake 3.1+
  - coreutils
  - pkg-config
  - Python 2.7 or 3

- Libraries
  - xattr
  - pcre
    - Version 1
  - zlib (optional)
- Comes with GUFI
  - jemalloc
  - Sqlite3 3.27
  - sqlite3-pcre
  - GoogleTest

GUFI Dependencies Installer Scripts

- GUFI is tested on GitHub Actions with many different setups

- See scripts in `contrib/CI`
  - Ubuntu
  - CentOS 7 and 8
  - Rocky Linux 8
  - macOS

Building GUFI

- `mkdir build`
- `cd build`
- `cmake <GUFI source dir>`
  - `-DDEP_INSTALL_PREFIX="${SWHOME}"`
  - `-DDEP_BUILD_THREADS=2`
  - `-DENABLE_JEMALLOC=Off`
- `make -j`
- `sudo make install`
  - (optional)

Environment

- Live filesystem
  - `SRC="${HOME}/GUFI/build/test"`
- Trace file prefix
  - `TRACE="${HOME}/traces/trace"`
- Index
  - `SEARCH="${HOME}/search"`
    - Make sure `"${SEARCH}"` is not a subdirectory of `"${SRC}"`
  - `INDEX="${SEARCH}/test"`
    - Interchangeable with `"${SEARCH}"` when querying
- Thread count to run with
  - `THREADS="$(nproc --all)"`

Add some xattrs

- `setfattr -n user.type -v directory "${SRC}"`
- `setfattr -n user.type -v file "${SRC}/Makefile"`
- `setfattr -n user.tag -v tagdata "${SRC}/Makefile"`

Create An Index From A Live Filesystem

- `gufi_dir2index -x -n "${THREADS}" "${SRC}" "${SEARCH}"`

- Use `-e` to compress work items

- Make sure `"${SEARCH}"` is either the parent of `"${SRC}"` (in-tree index) or a completely different directory (out-of-tree index)
  - If `"${SEARCH}"` or `"${INDEX}"` is under `"${SRC}"`, `gufi_dir2index` will index the index being generated and never stop

Create Trace Files From A Live Filesystem

- `gufi_dir2trace -x -n "${THREADS}" "${SRC}" "${TRACE}"`

- Use `-e` to compress work items

- Will create files called `"${TRACE}.0"`… `"${TRACE}.$((THREADS - 1))"`

- Expect weird filesystem metadata
  - Unexpected characters in paths
    - Can change trace field delimiter to attempt to handle (`-d`)
  - Duplicate inodes

Create An Index From A Trace

- `gufi_trace2index -n "${THREADS}" "${TRACE}".* "${SEARCH}"`
  - Might run out of file descriptors

or

- `cat "${TRACE}".* > "${TRACE}"`
- `gufi_trace2index -n "${THREADS}" "${TRACE}" "${SEARCH}"`

- Extended attributes are processed if found
- Use `-d` for non-default field delimiters

Basic Querying

`gufi_query -d " " -n "${THREADS}" <args> "${INDEX}"`

- Use `-e` to compress work items

- Get all file/link basenames
  - `-E "SELECT name FROM vrpentries;"`

- Get all file names with full paths
  - `-E "SELECT rpath(sname, sroll) || '/' || name FROM vrpentries;"`

- Get all directory basenames
  - `-S "SELECT name FROM vrsummary;"`

- Get all directory names with full paths
  - `-S "SELECT rpath(sname, sroll) FROM vrsummary;"`

Getting Full Paths from [vr]summary and [vr]pentries

- `-S "SELECT rpath(sname, sroll) FROM vrsummary;" -E "SELECT rpath(sname, sroll) || '/' || name FROM vrpentries;"`

- GNU find

Compound Queries

-
  ```
  gufi_query "${INDEX}" \
      -S "SELECT rpath(sname, sroll) FROM vrsummary WHERE totfiles < 5;" \
      -E "SELECT rpath(sname, sroll) || '/' || name FROM vrpentries;"
  ```

  - Directory names where the number of files in the directories is less than 5

  AND

  - File names from directories that returned results from -S

-
  ```
  gufi_query "${INDEX}" \
      -S "SELECT rpath(sname, sroll) FROM vrsummary WHERE totfiles < 5;" \
      -E "SELECT rpath(sname, sroll) || '/' || name FROM vrpentries;" \
      -a
  ```

  - Directory names where the number of files in the directories is less than 5

  OR

  - File names

-o was already used for outputting

Total File Size with row data from the Entries Table

-
  ```
  gufi_query -d " " -n "${THREADS}" "${INDEX}" \
      -I "CREATE TABLE intermediate(size INTEGER);" \
      -E "INSERT INTO intermediate SELECT size FROM vrpentries WHERE type ==
      \"f\";" \
      -K "CREATE TABLE aggregate(size INTEGER);" \
      -J "INSERT INTO aggregate SELECT size FROM intermediate;" \
      -G "SELECT SUM(size) FROM aggregate;"
  ```

- Reading every row → Lots of I/O
- Storing lots of rows in memory → High Memory Utilization
  - Can reduce by computing `SUM` in `-E` and `-J`

Total File Size with the Summary Table

-
  ```
  gufi_query -d " " -n "${THREADS}" "${INDEX}" \
      -I "CREATE TABLE intermediate(totsize INTEGER);" \
      -S "INSERT INTO intermediate SELECT totsize FROM vrsummary;" \
      -K "CREATE TABLE aggregate(totsize INTEGER);" \
      -J "INSERT INTO aggregate SELECT totsize FROM intermediate;" \
      -G "SELECT SUM(totsize) FROM aggregate;"
  ```

- Only reading 1 row from each database file in each directory
- Storing moderate number of rows in memory
  - Can reduce by computing `SUM` in `-J`

Total File Size with the Tree Summary Table

- `gufi_treesummary -n "${THREADS}" "${INDEX}"`

-
  ```
  gufi_query "${INDEX}" \
    -T "SELECT totsize FROM treesummary;"
  ```

- Only reading 1 row from 1 table from 1 database in 1 directory
- First time running `-T` requires traversing through tree (via `gufi_treesummary`)
  - Afterwards, queries are basically free

Aggregation
File Extension Histogram

-
  ```
  gufi_query -d " " -n "${THREADS}" "${INDEX}" \
  -I "CREATE TABLE intermediate(ext TEXT, count INTEGER);" \
  -E "INSERT INTO intermediate SELECT REPLACE(name, RTRIM(name,
  REPLACE(name, \".\", \"\")), \"\") AS ext, COUNT(inode) FROM
  vrpentries GROUP BY ext;" \
  -K "CREATE TABLE aggregate(ext TEXT, count INTEGER);" \
  -J "INSERT INTO aggregate SELECT ext, SUM(count) FROM
  intermediate GROUP BY ext;" \
  -G "SELECT ext, SUM(count) FROM aggregate GROUP BY ext;"
  ```

Aggregation
Top 10 Largest Files by UID

-
  ```
  gufi_query -d " " -n "${THREADS}" "${INDEX}" \
  -I "CREATE TABLE intermediate (uid INTEGER, size INTEGER, name TEXT);" \
  -E "INSERT INTO intermediate SELECT uid, size, fullpath FROM (SELECT uid,
  size, rpath(sname, sroll) || '/' || name AS fullpath, row_number() OVER
  (PARTITION BY uid ORDER BY size DESC) AS rownum FROM vrpentries) WHERE
  rownum <= 10;" \
  -K "CREATE TABLE aggregate (uid INTEGER, size INTEGER, name TEXT);" \
  -J "INSERT INTO aggregate SELECT uid, size, name FROM (SELECT uid, size,
  name, row_number() OVER (PARTITION BY uid ORDER BY size DESC) AS rownum
  FROM intermediate) WHERE rownum <= 10;" \
  -G "SELECT uid, size, name FROM (SELECT uid, size, name, row_number() OVER
  (PARTITION BY uid ORDER BY size DESC) AS rownum FROM aggregate) WHERE
  rownum <= 10;"
  ```

Rollup

- `gufi_rollup -n "${THREADS}" "${INDEX}"`
  - In-place
  - `-L <count>` to limit entries table row count
  - `-X` to do dry run

- `vrsummary`, `vrpentries`
  - Views of `summary` and `pentries` with aliased columns
  - Can be used for both origin index and rolled up index
  - `SELECT rpath(sname, sroll), … FROM vr* …`
    - Use `rpath` over `path`

- Can use `gufi_unrollup` executable to remove rollup data

Extended Attributes

- Create Index with `-x`
  - `gufi_dir2index -x …`
  - `gufi_dir2trace -x …`

- New views are available when running `gufi_query -x`
  - `xattrs`
    - inode, name, value
  - `xentries, xpentries, vrxpentries`
    - `entries` and `pentries` with associated xattr name and value
  - `xsummary, vrxsummary`
    - `summary` with associated xattr name and value

-
```
gufi_query -d " " -x "${INDEX}" \
-S "SELECT rpath(sname, sroll), xattr_name, xattr_value FROM vrxsummary;" \
-E "SELECT rpath(sname, sroll) || '/' || name, xattr_name, xattr_value FROM
vrxpentries;"
```
  - If original entry had multiple xattrs, queries will return multiple rows for the same entry

User Facing Tools

- Requires on `/etc/GUFI/config`
  - `gufi_find`
    - `find(1)`
  - `gufi_ls`
    - `ls(1)`
  - `gufi_getfattr`
    - `getfattr(1)`
  - `gufi_stats`
    - Queries that are probably useful
- `gufi_stat`
  - `stat(1)`

Thank you!

Sampling of LANL Filesystems Circa 2020/2021 (Yellow)

Filesystem	Directory Count (Millions)	File Count (Millions)
Home	3.3	23.4
Projects	18.5	178.9
Scratch 3	5.9	165.1
Scratch 4	16.5	225.0
Scratch 5	7.4	159.3
Archive 2	5.6	161.3

Shell vs Aggregation

Goal	Shell	GUFI

Top 10 Largest Files	`find -printf "%p %s\n" | sort -n -r -k 2 | head -n 10`	`INSERT INTO aggregate SELECT name, size FROM intermediate ORDER BY size DESC LIMIT 10;`
		
		`SELECT name, size FROM aggregate ORDER BY size DESC LIMIT 10;`

Top 10 Largest Files by UID	`find -printf "%P %s %U\n" |` **???**
	
	• awk/perl/shell associative arrays + limit?
	• `sort -n -r -k 3 -k 2`?
	  • Cannot limit by count during sort
	  • After sort, output has been flattened
	• Loop through /etc/passwd and run as user?
	  • Why run as user if you are root?
		`INSERT INTO aggregate SELECT name, size, uid FROM (SELECT name, size, uid, row_number() OVER (PARTITION BY uid ORDER BY size DESC) AS rownum FROM intermediate) WHERE rownum <= 10;`
		
		`SELECT name, size, uid FROM (SELECT name, size, uid, row_number() OVER (PARTITION BY uid ORDER BY size DESC) AS rownum FROM aggregate) WHERE rownum <= 10;`