GUFI Administrator's Guide

GUFI Developers

September 11, 2023

Contents

1 License

```
This file is part of GUFI, which is part of MarFS, which is released
under the BSD license.


Copyright (c) 2017, Los Alamos National Security (LANS), LLC
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


From Los Alamos National Security, LLC:
LA-CC-15-039

Copyright (c) 2017, Los Alamos National Security, LLC All rights reserved.
Copyright 2017. Los Alamos National Security, LLC. This software was produced
under U.S. Government contract DE-AC52-06NA25396 for Los Alamos National
Laboratory (LANL), which is operated by Los Alamos National Security, LLC for
the U.S. Department of Energy. The U.S. Government has rights to use,
reproduce, and distribute this software.  NEITHER THE GOVERNMENT NOR LOS
ALAMOS NATIONAL SECURITY, LLC MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR
ASSUMES ANY LIABILITY FOR THE USE OF THIS SOFTWARE.  If software is
```

modified to produce derivative works, such modified software should be clearly marked, so as not to confuse it with the version available from LANL.

THIS SOFTWARE IS PROVIDED BY LOS ALAMOS NATIONAL SECURITY, LLC AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL LOS ALAMOS NATIONAL SECURITY, LLC OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2 Introduction

Over the years, the amount of data we store and use has grown exponentially to the point that petabytes of storage is not uncommon. What used to be a simple task of accessing and sorting through information has been compounded into an arduous task with the size and scale of super-computing data centers. Being able to query data effectively, while also taking into account user permissions becomes paramount into accomplishing daily tasks. This is what the Grand Unified File Index (GUFI) tool aims to accomplish.

This process of efficiently accessing data is accomplished by recreating the tree structure via indexing. Each directory contains an SQL database file that stores the metadata of the files as well as summary information for that directory and optionally summary information for the entire tree below that directory.

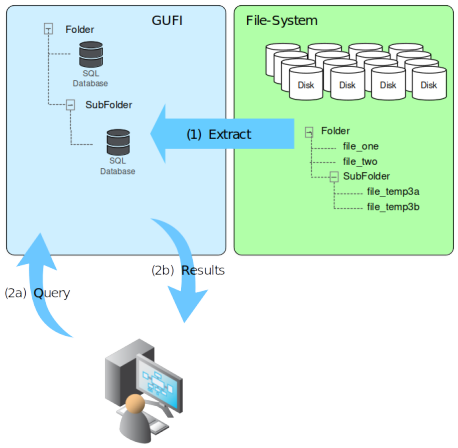

Figure 1: Layout and user interaction with GUFI

3 Dependencies

A number of dependencies that should be installed before attempting to build GUFI. There are also many optional dependencies that can enable optional features.

3.1 System Tools

Package/Executable	Required?	Comments
autotools	Yes	Some bundled packages use autotools
C Compiler	Yes	C99 support
C++ Compiler	No	C++11 support - g++ 4.9.3, clang++-3.9, or newer
CMake	Yes	Version 3.1 or higher
Make	Yes	
Python	Yes	Python 2.7 or Python 3
pip	No	Used for installing and client dependencies
git	No	1.8.5 or higher if using the performance history framework
LaTeX	No	Only needed if building LaTeX Documentation
pkg-config	Yes	
realpath	Yes	
rpmbuild	No	Only needed if building RPMs
setfattr	Yes	attr package `xattr -w` on OSX
truncate	Yes	The -s option must be available
uname	No	Only needed if building RPMs

3.2 Libraries

Name	Required?	Comments
xattr	Yes	
pcre	Yes	Version 1
fuse	No	Called osxfuse on OSX
db2	No	
gpfs	No	
matplotlib	No	Python library used for plotting numbers collected by the performance history framework
zlib	No	

3.3 Packages provided by GUFI

Several packages are provided/downloaded, built, and installed automatically:

Name	Comments
GoogleTest	https://github.com/google/googletest
jemalloc	https://github.com/jemalloc/jemalloc
paramiko	https://github.com/paramiko/paramiko patched ae3d0febef17a8ece5268bbf6c210a30573ce800 Use pip to download python-bcrypt, python-cryptography, and python-pynacl
sqlite3	https://www.sqlite.org/index.html patched sqlite-autoconf-3270200.tar.gz
sqlite3-pcre	https://github.com/mar-file-system/sqlite3-pcre

4 Download

The source code for GUFI can be found at https://github.com/mar-file-system/GUFI.

5 Build

GUFI uses CMake to generate Makefiles. CMake 3.1 or higher is required. From the root directory of the GUFI source, run:

```
mkdir build
cd build
cmake .. [options]
make
```

The recommended options for deployment are `-DCMAKE_BUILD_TYPE=Release -DCLIENT=On`.

GUFI is known to build on Ubuntu 16.04 (Xenial), Ubuntu 18.04 (Bionic), Ubuntu 20.04 (Focal), Ubuntu 22.04 (Jammy), CentOS 7, CentOS 8, RockyLinux 8, OSX 10.13 (High Sierra), OSX 11 (Big Sur), OSX 12 (Monterey), and OSX 13 (Ventura). Building on Windows with cygwin, GCC (not MinGW), and with jemalloc turned off also works. OpenSUSE 12.3 was supported in the past, and may still work.

5.1 Environment Variables

Setting	Description
`CXX=false`	Disable building of C++ code

5.2 CMake Flags

5.2.1 General

`-D<VAR>=<VALUE>`	Description
`CMAKE_INSTALL_PREFIX=<PATH>`	Install to a custom directory when running `make install`
`CMAKE_BUILD_TYPE=Debug`	Build with warnings and debugging symbols turned on

5.2.2 Dependencies

-D<VAR>=<VALUE>	Description
DEP_DOWNLOAD_PREFIX=<PATH>	Location of downloaded dependencies. If the expected files are found, they will not be downloaded. The default path points to the bundled dependencies.
DEP_BUILD_DIR_PREFIX=<PATH>	Location to build dependencies. Defaults to ${CMAKE_BINARY_DIR}/builds
DEP_INSTALL_PREFIX=<PATH>	Location to install the dependencies. Defaults to ${CMAKE_BINARY_DIR}/deps. If the dependencies are not installed in ${CMAKE_BINARY_DIR}, they will not need to be redownloaded, rebuilt, or reinstalled everytime ${CMAKE_BINARY_DIR} is deleted.
DEP_PATCH_SQLITE3_OPEN=<On|Off>	Whether or not to patch SQLite3 open
DEP_USE_JEMALLOC=<On|Off>	Whether or not to build and link with jemalloc

5.2.3 Debug

CMAKE_BUILD_TYPE must be set to Debug for these to have effect.

-D<VAR>=<VALUE>	Description
PRINT_CUMULATIVE_TIMES=<On|Off>	Print cumulative statistics at the end of some executables.
PRINT_PER_THREAD_STATS=<On|Off>	Print gufi_query event timestamps.
PRINT_QPTPOOL_QUEUE_SIZE=<On|Off>	Print size of work queues every time a work queue receives a new item or pops items off.
GPROF=<On|Off>	Compile with the -pg flag.

5.2.4 Client

-D<VAR>=<VALUE>	Description
CLIENT=<On|Off>	Whether or not to install paramiko and gufi_client when make install is called

5.2.5 Testing

-D<VAR>=<VALUE>	Description
TEST_WORKING_DIRECTORY=<PATH>	Directory to run tests in. Defaults to ${CMAKE_BINARY_DIR}/test

Do not turn these off unless you know what you are doing:

-D<VAR>=<VALUE>	Description
RUN_ADDQUERYFUNCS=<On|Off>	Compile gufi_query without calls to addqueryfuncs.
RUN_OPENDB=<On|Off>	Compile gufi_query without calls to opendb.
RUN_SQL_EXEC=<On|Off>	Compile gufi_query without calls to sqlite3_exec.

5.2.6 Docs

-D<VAR>=<VALUE>	Description
LATEX_BUILD=<On|Off>	Whether or not to build PDF documentation when LaTeXis found

6 Install

6.1 make

To install GUFI from the source build, run `[sudo] make install` from the build directory.

6.2 RPMs

`make package` will be available for generating RPMs if `rpmbuild` was found at configuration time. By default, only the server RPM will be generated. In order to generate the client RPM (see Section 11), the CMake flag `-DCLIENT` should be set to `On`.

7 Database Schema

GUFI stores extracted metadata in SQLite3 database files. Each directory contains a database file named db.db that contains a set of tables and views designed to facilitate efficient querying of metadata.

7.1 `entries`

The `entries` table contains file and link metadata extracted from the source filesystem using `stat(1)`, `readlink(2)`, and `llistxattr(2)`. There are also a few OS specific columns (`oss*`) that are unused in base GUFI.

7.2 Directory `summary`

The directory `summary` table contains the metadata of the current directory. Additionally, it contains columns that summarizes the `entries` table located in the same database file, such as minimum and maximum file sizes, uids, and gids. These summary columns can be used to determine whether or not the `entries` table needs to be queried at all.

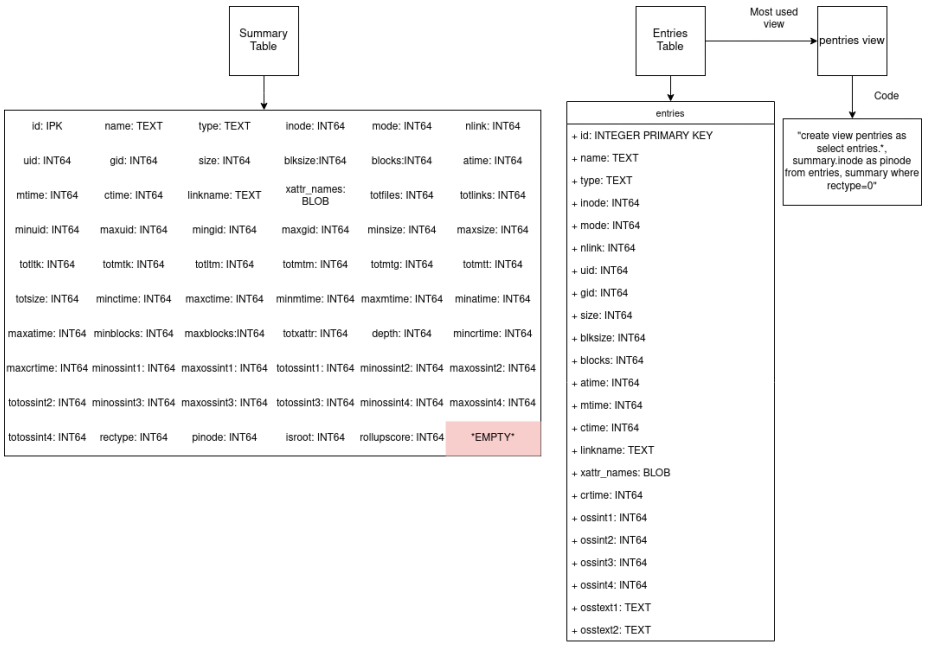

Figure 2: Database schema

7.3 pentries

The original `pentries` view was defined as:

```
SELECT entries.*, summary.inode FROM entries, summary;
```

This was done in order to not store the parent inode of each entry, which would be the same for every entry.

In general, users should query the `pentries` view over the `entries` table in order to obtain complete sets of data to query on.

7.3.1 pentries_rollup

In order to simplify rolling up indexes (see Section 9.1), the `pentries` view was modified to also union with the `pentries_rollup` table, which contains all child `pentries` views copied into the current directory's database file.

7.4 vrsummary

The `vrsummary` view is the `summary` table with a handful of columns repeated as aliases. This was done so that the `rpath` SQL function can be called to generate the path of a given directory with the same view and query whether or not the index has been rolled up, and thus should be preferred over querying the `summary` table. Using the `summary` table directly with a rolled up index is possible, but will complicate queries.

7.5 vrpentries

The `vrpentries` view is the `pentries` view with a few `summary` table columns aliased. This allows for the `rpath` SQL function can be called to generate the path of a given directory with the same view and query whether or not the index has been rolled up, and thus should be preferred over querying `entries` and `pentries`. `rpath` with the `vrpentries` view is called with the same arguments that are used with `vrsummary`. Using the `pentries` view directly with a rolled up index is possible, but will complicate queries.

7.6 treesummary

Similar to the directory `summary` table, GUFI also provides functionality to generate `treesummary` tables. Instead of summarizing only the data found in the entries table of the current directory, the `treesummary` table summarizes the contents of the entire subtree, allowing for queries to completely skip processing entire subtrees. See Section 9.2 for details.

7.7 vsummarydir

The `vsummarydir` view provides access to the entire directory summary and not a partial directory summary (say by user or group).

7.8 vsummaryuser

The `vsummaryuser` view provides access to the directory summary for each user (if this summary has been populated (not by default but easily populatable via a query)).

7.9 vsummarygroup

The `vsummarygroup` view provides access to the directory summary for each group (if this summary row has been created (not by default but easily created via a query)).

7.10 vtsummarydir

The vtsummarydir view provides access to the entire tree directory summary and not a partial directory summary (say by user or group).

7.11 vtsummaryuser

The `vtsummaryuser` view provides access to the tree directory summary for each user (if this summary row has been created (not by default but easily created via a query)).

7.12 vtsummarygroup

The `vtsummarygroup` view provides access to the tree directory summary for each group (if this summary has been populated (not by default but easily populatable via a query)).

7.13 xattrs

With the addition of extended attributes support, several more tables and views were added into db.db. Additional database files with different schemas were also added. See Section 8.3.2 for details.

8 Indexing

The first step to using GUFI is indexing a source filesystem.

An index created by GUFI retains the shape of the source filesystem: directories that exist in the source filesystem also exist in the index. If an administrator created the index, the directories will also have the same access permissions, uid, and gid.

Indexes will not contain any of the files that the source filesystem contained. Instead, all metadata extracted from a single directory of the source filesystem will be placed into a single database file, called db.db, in the cooresponding directory in the index. Each database file will be created with a fixed schema that includes the tables listed in Sections 7 and 8.3.2. Additional database files may be created if extended attributes are extracted (see Section 8.3).

Index processing (creation) occurs on a per-directory basis, and thus is highly parallelizable.

8.1 Directly Indexing a Filesystem

8.1.1 `gufi_dir2index`

`gufi_dir2index` is used to directly create an index based off of the contents of a provided directory.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-x	pull xattrs from source file-sys into GUFI
-z <max_level>	maximum level to go down to
-k <filename>	file containing directory names to skip
-M <bytes>	target memory footprint
-C <count>	Number of subdirectories allowed to be enqueued for parallel processing. Any remainders will be processed in-situ
-e	compress work items

Table 1: `gufi_dir2index` Flags and Arguments

8.1.1.1 Usage

```
gufi_dir2index [flags] input_dir...  output_dir
```

The index of $\texttt{input\_dir}_i$ will be placed in `output_dir/$(basename input_dir`$_i$`)`.

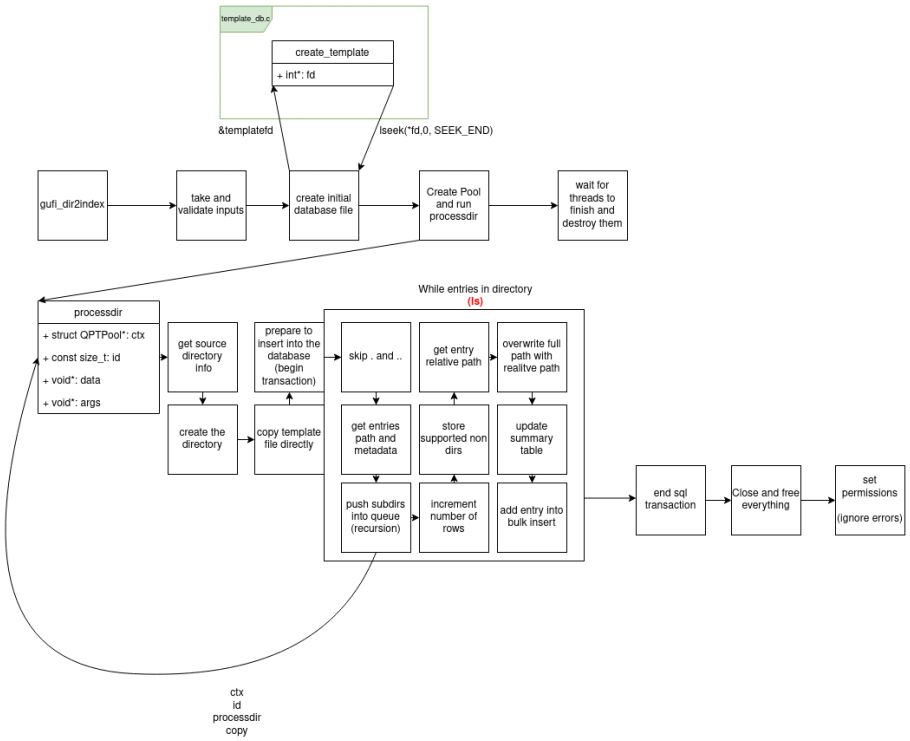

Figure 3: `gufi_dir2index` workflow

8.2 Indirectly Indexing a Filesystem

8.2.1 Trace Files

Traces files, or flat files, are text files containing `lstat(2)` information pulled from source filesystems separated by delimiters. Trace files are generated by `gufi_dir2trace` and are processed by `gufi_trace2index`.

The default delimiter is the ASCII Record Separator character `\x1E`, but can be changed to any 8-bit character. Characters that can appear in filesystems should not be used as the delimiter in order to not confuse `gufi_trace2index`.

The metadata pulled from each directory is represented by a "stanza" in a trace file. A stanza starts with a line of directory metadata followed by zero or more lines of regular file and symbolic link metadata. Note that the summary of each directory is not stored in trace files and are instead generated when the index is built.

Due to the parallelism of indexing, stanzas can appear in any order within a trace file and can be placed in any of the per thread trace files that are generated. Traces files can be concatenated together if doing so is necessary or more convenient.

8.2.2 gufi_dir2trace

gufi_dir2trace generates trace files to allow for indexes to be easily transfered to different locations rather than requiring entire trees to be copied around.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-x	pull xattrs from source file-sys into GUFI
-d <delim>	delimiter (one char) [use 'x' for 0x1E]
-k <filename>	file containing directory names to skip
-M <bytes>	target memory footprint
-C <count>	Number of subdirectories allowed to be enqueued for parallel processing. Any remainders will be processed in-situ
-e	compress work items

Table 2: gufi_dir2trace Flags and Arguments

8.2.2.1 Usage

```
gufi_dir2trace [flags] input_dir...  output_prefix
```

Trace files with the name output_prefix.${i}, where i $\in$ [0, number of threads), will be created.

8.2.3 gufi_trace2index

gufi_trace2index is used to convert trace files into indexes. It is essentially the same as gufi_dir2index except it obtains data from trace files instead of the filesystem being indexed.

Per thread trace files may be passed into gufi_trace2index directly. Note that passing in too many trace files at once might result in running out of file descriptors. Alternatively, the per thread trace files may be concatenated in any order into a smaller number of larger files for processing.

Extended attributes will be processed if they are found in the traces. There is no need to tell gufi_trace2index to process them with -x.

8.2.3.1 Usage

```
gufi_trace2index [flags] trace_file...  index_root
```

Each source filesystem found in the trace files will be converted to an index placed underneath output_dir.

Flag	Functionality
-h	help manual
-H	Show assigned input values
-n <num_threads>	define number of threads to use
-d <delim>	delimiter (one char) [use 'x' for 0x1E]
-M <bytes>	target memory footprint

Table 3: `gufi_dir2trace` Flags and Arguments

8.3 Extended Attributes

GUFI supports the indexing and querying of extended attributes (xattrs).

Reading standard filesystem permissions of files only requires read (and execute) access to the directory. Extended attribute names are visible this way. However, because xattr values are user defined data, their permissions are checked at the file level, requiring changes to how GUFI stores data. For more information on xattrs, see `xattr(7)`, `llistxattr(2)`, and `lgetxattr(2)`.

Directories containing files from multiple users might have xattrs that are not readable by all who can view the `lstat(2)` data of the directory. As GUFI was originally designed to only use directory-level permission checks, a number of modifications were made to process xattrs without violating their permissions.

8.3.1 Roll In

Extended attributes that are readable by all who have access to the directory are stored in the `xattrs_pwd` table in the main database. These xattrs are referred to as "rolled in".

The rules that determine whether or not an xattr pair can roll in are as follows:

- File is 0+R
- File is UG+R doesnt matter on other, with file and parent same usr and grp and parent has only UG+R with no other read
- File is U+R doesnt matter on grp and other, with file and parent same usr and parent dir has only U+R, no grp and other read
- 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

Extended attributes that cannot be read by all who can read the directory are stored in external per-uid and per-gid databases set with `uid:nobody` and `nobody:gid` owners respectively. This makes it so that non-admin users cannot access the xattrs stored in external databases that they do not have permissions to access.

8.3.2 Schema

The main database and all external databases contain the following tables and views with the 3 columns `inode`, `name`, and `value`:

- The `xattrs_pwd` table contains all extended attributes of the current directory that were placed into this database file.
- The `xattrs_rollup` table contains all extended attributes that were placed into the children directories that were subsequently rolled up into the current directory.
- The `xattrs_avail` view is the union of all extended attributes in `xattrs_pwd` and `xattrs_rollup` in this database file.

Additionally, each main database file has the following tables and views in order to keep track of which files were created by GUFI for the purposes of storing xattrs:

- The `xattrs_files_pwd` table contains a listing of external database filenames that contain xattrs that were not rolled in.
- The `xattrs_files_rollup` table contains a listing of external database filenames that contain xattrs that were not rolled in, but were brought in by rolling up.
- The `xattrs_files` view combines the listings of database filenames found in `xattrs_files_pwd` and `xattrs_files_rollup`.

8.3.3 Usage

Extended attributes are not pulled from the filesystem by default. In order to pull them, pass `-x` to `gufi_dir2index` or `gufi_dir2trace`.

Note that only xattr pairs in the user namespace (`user.*`) are extracted.

8.4 Location

GUFI is expected to be used to query the indexes of many filesystems all at once. However, the source filesystems are not expected to be accessible from each other. In order for the indexes from many disconnected systems to be queried at once, they should all be built under a common directory on a single machine.

8.4.1 Permissions

If a GUFI tree is located on a different machine than the source tree, the users and groups are probably not available on the machine with the index on it. `/etc/passwd` should be populated with the entries from the source machine and modified so that they do not have a home directory, and are not allowed to log in (`/sbin/nologin`). Similarly, `/etc/group` should be updated with the source machine's groups and modified to remove any unnecessary information.

9 Post-Indexing

After indexing, additional operations can be performed on the index in order to increase query performance.

9.1 Rollup

Rolling up is a major optimization that can reduce the number of directories that need to be opened when querying by significant amounts while still obtaining the same results as an unmodified index.

9.1.1 Bottlenecks

During querying, every single directory runs a fixed set of operations:

- `opendir(3)`
- `readdir(3)` (Looped)
- `sqlite3_open_v2`
- `sqlite3_exec_v2`
- `sqlite3_close(3)`
- `closedir(3)`

Each of these operations have costs associated with them. Some are fixed and some depend on the shape and contents of the index. Reducing the number of directories processed allows for fixed costs to be amortized.

9.1.2 Rules

9.1.2.1 Whether or not a directory *CAN* roll up

In order for a child directory to be allowed to roll up into its parent directory, it must follow two rules. First, all of its children must be rolled up. Second, all of its children must have permissions that satisfy any one of the following conditions with the parent.

- World readable and executable (o+rx)
- Matching user, group, and others permissions, with the same user and group
- Matching user and group permissions, readable and executable (ug+rx) with the same user and group, and not world readable and excutable (o-rx)
- Matching user permissions, readable and executable (u+rx) with the same user and not group or world readable and executable (go-rx)

Note that because leaf directories have no children with which to have conflicting permission with, they are considered rolled up.

9.1.2.2 Whether or not a directory *SHOULD* roll up

In addition to the permission-based rules that determine whether or not a directory can roll up, we also have a small check to say whether or not a directory should roll up.

As data is rolled upwards in the index, databases towards the top will accumulate more and more data from its subtrees. If directories are allowed to roll up without limit, some databases will become significantly larger than others, causing large amounts of tail latency.

The `gufi_rollup` executable has the `-L` flag that limits the number of entries that may be found within a single directory.

9.1.3 Steps

The processing of rolling up involves a number of steps that update the tables of the parent directory.

First, the target directory is checked to ensure it can and should roll up its children into itself using the rules listed in 9.1.2. If rollup will proceed, the parent's `summary` table is updated with a rollup score of 1.

Then, the contents each child is copied into the parent:

1. The child's `pentries` view is copied into the parent's `pentries_rollup` table. The parent's `pentries` view is updated automatically.
2. The child's `summary` table is copied into the parent's `summary` table with the names of each child directory prefixed with the parent's name.

Rolling up can be viewed as flattening the contents of the index while taking into consideration the permissions of each directory.

Rolling up will cause the size of the index to grow significantly due to the amount of data being replicated. One obvious optimization would be to only roll up to the top-most level where a directory can roll up to (one extra copy of the subtree instead of repeated copies). This however, will only allow for queries to take advantage of rollups if they start above the top-most rollup directory. Starting a query below the top-most rollup level would result in the original subtree's query time whereas the implemented method of rolling up allows for queries starting at any point in the index to take advantage of rollups.

The rollup operation can take some time, and so indexes are not rolled up automatically. The `gufi_rollup` executable must be called manually.

9.1.4 Extended Attributes

Additional steps are needed to rollup xattrs:

- The child's `xattrs_avail` view (without external database data) is copied into the parent's `xattrs_rollup` table.
- The child's `xattrs_files` view (without external databases data) is copied into the parent's `xattrs_files_rollup` table.

- The child's external database files are copied into the parent. If the parent already has an external database file with the same uid or gid, the contents of the external database are copied into the parent's external database's `xattrs_rollup` table instead.

9.1.5 `gufi_rollup` executable

In order to apply rollup to an index, run the `gufi_rollup` executable:

```
gufi_rollup index_root
```

Figure 4 shows the overall structure of the `gufi_rollup` executable. `gufi_rollup` recursively descends the index and performs the rollup operation on a directory once all of the directory's children have been processed as shown in Figure 5.

9.1.6 Undoing a rollup

To remove rollup data from an index, run the `gufi_unrollup` executable:

```
gufi_unrollup index_root
```

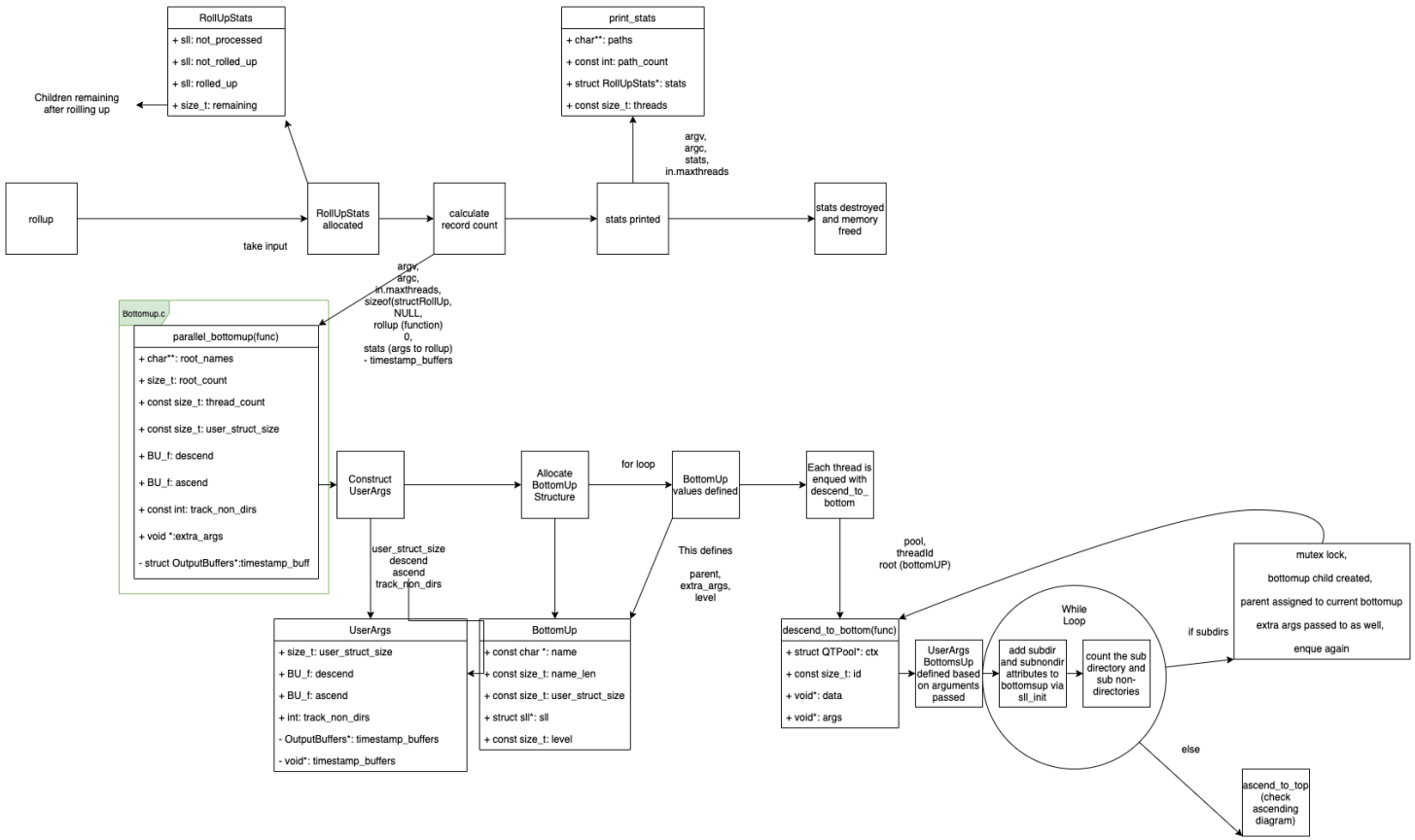

Figure 4: `gufi_rollup` Workflow

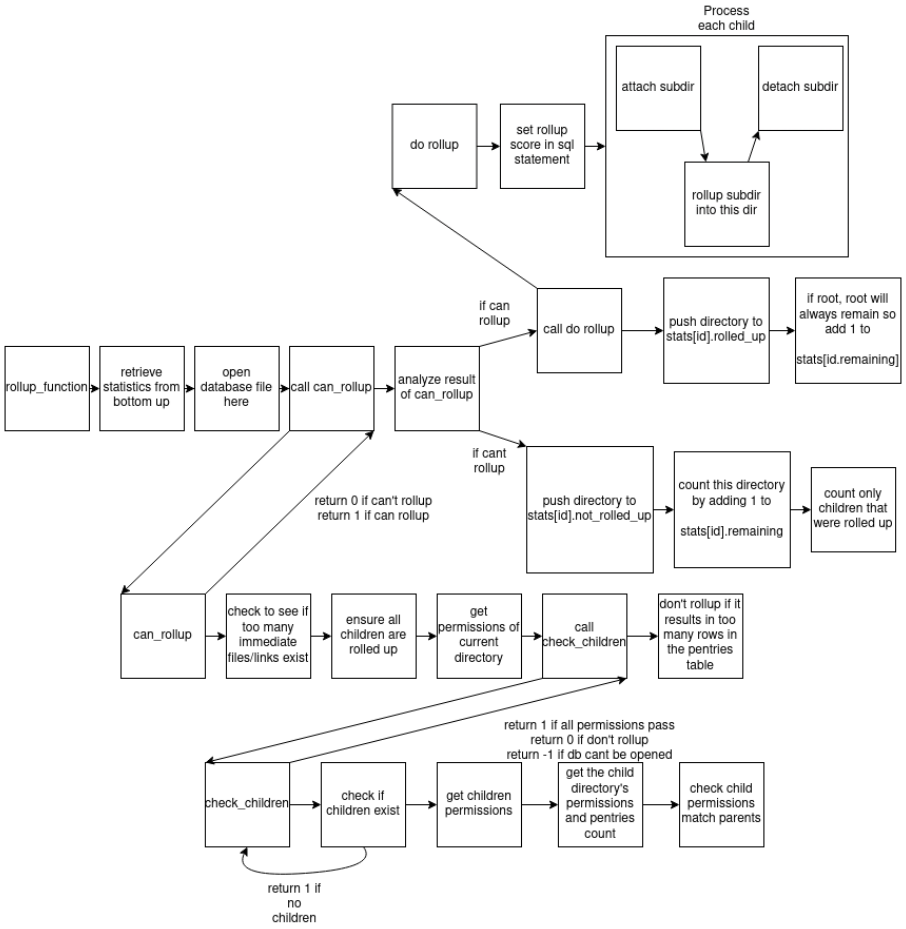

Figure 5: Rollup Function Diagram

9.2 Generate `treesummary` Tables

The `treesummary` table is an optional table that is placed into db.db. It contains a summary of the entire subtree starting at current directory using minimums, maximums, and totals of numerical values.

When a query is provided to `gufi_query -T`, the `treesummary` table is queried first. Because `treesummary` tables do not necessarily exist, `gufi_query` first checks for the existence of the `treesummary` table in the directory being processed before performing the `-T` query. If the `-T` query returns no results, the entire subtree will be skipped.

9.2.1 `gufi_treesummary`

Starting from a directory provided in the command line, `gufi_treesummary` recursively traverses to the bottom of the tree, collecting data from the `summary` table of each child directory database. If a `treesummary` table is discovered in a subdirectory, descent down the tree is stopped as the `treesummary` table contains all of the information about that directory as well as all subdirectories underneath it. Once all of the data has been collected, it is summarized and placed into the `treesummary` table of the starting directory.

Generating `treesummary` tables for all directories using this top-down approach will take a long time due to the repeated traversals across the same directories. Because of this, `gufi_treesummary` generates the `treesummary` table for the provided directory only.

If generating `treesummary` tables using `gufi_treesummary`, the tables should be generated at optimal points within the index. For example, if the index is on a home directory, it may be useful to generate `treesummary` tables at each user's home directory.

Example Call:

```
gufi_treesummary index_root
```

9.2.2 `gufi_treesummary_all`

`gufi_treesummary_all` generates `treesummary` tables for all directories. This is done by walking to the bottom of the tree and generating `treesummary` tables while walking back up, which only occurs after all subdirectories have had their `treesummary` tables generated. Leaf directories, by definition, do not have subdirectories, and further traversal down is unnecessary and impossible. Their `treesummary` tables are thus duplicates of their `summary` tables, providing the base case for the walk back up the tree. Directories above the leaves can then use the `treesummary` data found in their immediate subdirectories, which are 1) guaranteed to exist and 2) guaranteed to summarize the entire subdirectory's subtree, to generate their own `treesummary` tables.

Example Call:

```
gufi_treesummary_all index_root
```

9.2.3 `gufi_rollup`

Just as with `gufi_treesummary_all`, rolling up a tree involves walking to the bottom of the tree and working upwards. This allows for `treesummary` generation to be performed automatically during the roll up operation, resulting in `treesummary` tables being generated for all directories whether or not they were rolled up.

10 Querying

Once a GUFI tree with one or more indexes has been created, it should be queried using GUFI tools and SQL statements.

10.1 `gufi_query`

`gufi_query` is the main tool used for accessing indexes.

`gufi_query` processes each directory in parallel, passing user provided SQL statements to the current directory's database. Because of this, callers will need to know about GUFI's database and table schemas. We expect only "advanced" users such as administrators and developers to use `gufi_query` directly. Users who do not know how GUFI functions may call `gufi_query`, but might not be able to use it properly.

10.1.1 Flags

Flag	Functionality
-h	help
-H	show assigned input values (debugging)
-E <SQL ent>	SQL for entries table
-S <SQL sum>	SQL for summary table
-T <SQL tsum>	SQL for tree-summary table
-a	AND/OR (SQL query combination)
-n <threads>	number of threads (default: 1)
-j	print the information in terse form
-o <out_fname>	output file (one-per thread, with thread-id suffix)
-d <delim>	one char delimiter (default: \x1E)
-O <out_DB>	output DB
-I <SQL_init>	SQL init
-F <SQL_fin>	SQL cleanup
-y <min-level>	minimum level to descend to
-z <max-level>	maximum level to descend to
-J <SQL_interm>	SQL for intermediate results (no default. ex.: `INSERT INTO <aggregate table name> SELECT * FROM <intermediate table name>`)
-K <create aggregate>	SQL to create the final aggregation table
-G <SQL_aggregate>	SQL for aggregated results (no default. ex.: `SELECT * FROM <aggregate table name>`)
-m	Keep mtime and atime same on the database files
-B <buffer size>	size of each thread's output buffer in bytes
-w	open the database files in read-write mode instead of read only mode
-x	enable xattr processing
-k	file containing directory names to skip
-M <bytes>	target memory footprint
-e	compress work items

Table 4: `gufi_query` Flags and Arguments

10.1.2 Flag/Table Associations

A `gufi_query` call should use at least one of the `-T`, `-S`, and `-E` flags to pass in SQL statements[1].

- `-T` should be used to query the `treesummary` table.
- `-S` should be used to query the `summary` table and its variants.
- `-E` should be used to query the `entries` table and its variants.

Note that user provided SQL statements are passed directly into SQLite[2] and thus associations between flags and tables are not enforced[3].

Prior to rolling up, the `pentries` view can be treated as an optional improvement to the `entries` table. After rolling up, the `pentries` view will have been updated extensively and will contain both the original `entries` data as well as the rolled up data. Querying the `entries` table of a rolled up index will return a subset of the total results that exist in the index since it only contains the current directory's information and subdirectories are not traversed. The `summary` table was updated directly, so querying it will return all directory information.

However, instead of querying the `summary` table and switching between querying `entries` and `pentries`, the `vrpentries` and `vrsummary` views should always be used. These views contain data from the `summary` table that allow for the path of each row relative to the starting path to be generated using the `rpath` SQL function, whether or not the index has been rolled up, with consistent input arguments (see Table 8 for details). These views and function were set up to help simpliy the queries that users provide.

10.1.3 Short Circuiting with Compound Queries

At each directory, the set of user provided SQL statements run on the db.db file in the following order: if `-T` was provided, it will run if the optional `treesummary` table exists in the database. If `-T` was not run or returned at least one row of results, `-S` will run if it was provided. If `-S` was not provided or returned at least one row of results, `-E` will run if it was provided.

Conversely, if `-T` runs and does not return any results, processing is stopped before running the `-S` query. If `-S` does not return any results, the `-E` query is not run. This allows for threads to short circuit the processing of a directory if it is determined early on that no rows would be obtained from a later query. For example, if a query is searching for files larger than 1MB, but the `-S` query found that the range of file sizes is 1KB to 5KB, there is no need to run the `-E` query to get rows, since no rows will match in any case.

[1] Not passing in any SQL statements results in `gufi_query` simply walking the tree and filling up the inode and dentries caches.

[2] Anything that can be done with SQL can also be done on the databases in an index. To prevent accidental modifications, databases default to opening in read-only mode.

[3] Querying tables/views with the wrong flags may result in unexpected output.

To turn off short circuiting and always run all queries for each directory, pass the `-a` flag to `gufi_query`.

10.1.4 Extended Attributes

When querying for xattrs, pass `-x` to `gufi_query` to build the `xattrs` view for querying. This view is a SQL union of rolled in xattrs and any external databases that successfully attaches. Attaching the database files checks the permissions of the xattrs. `UNION` removes duplicate entries that showed up in multiple external databases, leaving a unique set of xattrs accessible by the caller.

The `xentries`, `xpentries`, `xsummary`, `vrxpentries`, and `vrxsummary` views are convenience views generated so that users do not have to perform joins themselves. They are `entries`, `pentries`, `summary`, `vrpentries`, and `vrsummary` enhanced with the `xattr_name` and `xattr_value` columns. Like the `xattrs` view, these views are also dropped at the end of directory processing.

Note that entries with multiple extended attributes will return multiple times in this view.

10.1.5 Aggregation

There are cases where independent per-thread results are not desirable, such as when sorting or summing, where the results from querying the index must be aggregated for final processing.

In order to handle these situations, the `-I` flag should be used to create per-thread intermediate tables that are written to by `-T`, `-S`, and `-E`. The intermediate table results will then be aggregated using `-J` into the final table created by `-K`. The rows stored in the final table are processed one last time as a whole, rather than as results from independent threads, using `-G`.

10.1.6 Per-Thread Output Files

The `-o` flag causes results to be outputted to text files. When outputting in parallel, per-thread output files are created with the thread id appended to the filename provided with the flag. When aggregating, the aggregate results are written to the filename specified by `-o` with no filename modifications.

10.1.7 Per-Thread Output Database Files

The `-O` flag allows for results to be written to SQLite database files instead of text files. The resulting filenames follow the same creation rules as `-o`. However, the queries passed into `gufi_query` need modifications. When writing in parallel, `-I` is needed to create the table for each per-thread output database. `-T`, `-S`, and `-E` should be modified to write to the per-thread tables in the same way as writing to the intermediate tables when aggregating. When writing aggregate results to a database, `-G` is not needed as `-J` already wrote the results into the

aggregate table. However, `-G` may still be provided to get results during the `gufi_query` in addition to queries on the results database file later on.

Output database files may be passed to the `querydbs` executable for further processing.

10.1.8 Example Calls

Table 5: Parallel Results

Output	`gufi_query /path/to/index`
`stdout`	`-S "SELECT * FROM vrsummary;"`
Per-thread Files	`-E "SELECT * FROM vrpentries;"` `-o results`

Per-thread Database Files	`-I "CREATE TABLE results(name TEXT, size INT64);"`
	`-S "INSERT INTO results SELECT name, size FROM vrsummary;"`
	`-E "INSERT INTO results SELECT name, size FROM vrpentries;"`
	`-O results`

Table 6: Aggregate Results

Output	`gufi_query /path/to/index`

`stdout`	`-I "CREATE TABLE intermediate(name TEXT, size INT64);"`
	`-S "INSERT INTO intermediate SELECT name, size FROM vrsummary;"`
	`-K "CREATE TABLE aggregate(name TEXT, size INT64);"`
	`-J "INSERT INSERT aggregate SELECT * FROM intermediate;"`
	`-G "SELECT * FROM aggregate ORDER BY size DESC;"`

Single File	`-I "CREATE TABLE intermediate(name TEXT, size INT64);"`
	`-E "INSERT INTO intermediate SELECT name, size FROM vrpentries;"`
	`-K "CREATE TABLE aggregate(name TEXT, size INT64);"`
	`-J "INSERT INSERT aggregate SELECT * FROM intermediate;"`
	`-G "SELECT * FROM aggregate ORDER BY size DESC;"`
	`-o results`

Single Database File	`-I "CREATE TABLE intermediate(name TEXT, size INT64);"`
	`-S "INSERT INTO intermediate SELECT name, size FROM vrsummary;"`
	`-E "INSERT INTO intermediate SELECT name, size FROM vrpentries;"`
	`-K "CREATE TABLE aggregate(name TEXT, size INT64);"`
	`-J "INSERT INSERT aggregate SELECT * FROM intermediate;"`
	`-O results`

10.1.9 Behavior When Traversing Rolled Up Indexes

When `gufi_query` detects that a directory being processed has been rolled up, the thread processing that directory does not enqueue work to descend further down into the tree, as all of the data underneath the current directory is available in the current directory. While this may seem to only reduce tree traversal by

"some" amount, in practise, rolling up indexes of real filesystems reduces the number of directories (and thus the bottlenecks listed in Section 9.1.1) that need to be processed by over 99%, significantly reducing index query time.

Queries are not expected to have to change too often when switching between unmodified indexes and rolled up indexes. If they do, they should not have to change by much.

10.1.10 Visualizing the Workflow

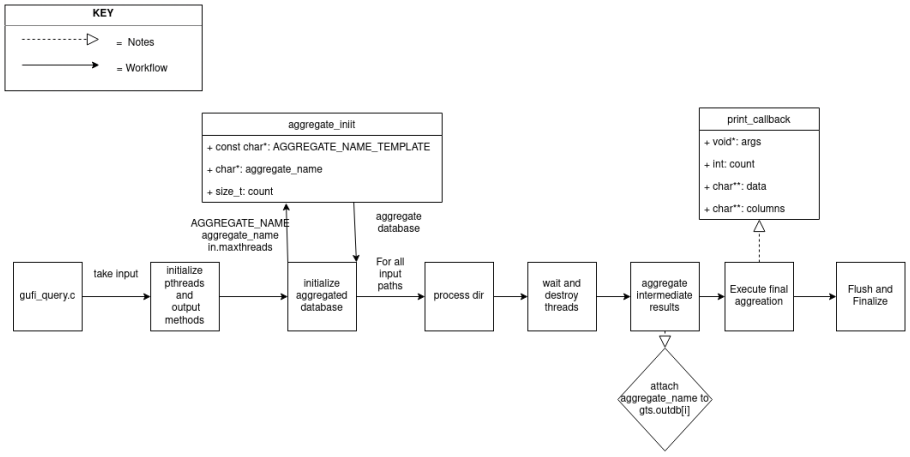

Figure 6: Workflow of `gufi_query`

10.1.10.1 `processdir` The core of `gufi_query` is the `processdir` function. This is where the -T, -S, and -E flags are processed. Multiple instances of this function are run in parallel via the thread pool in order to quickly traverse and process an index.

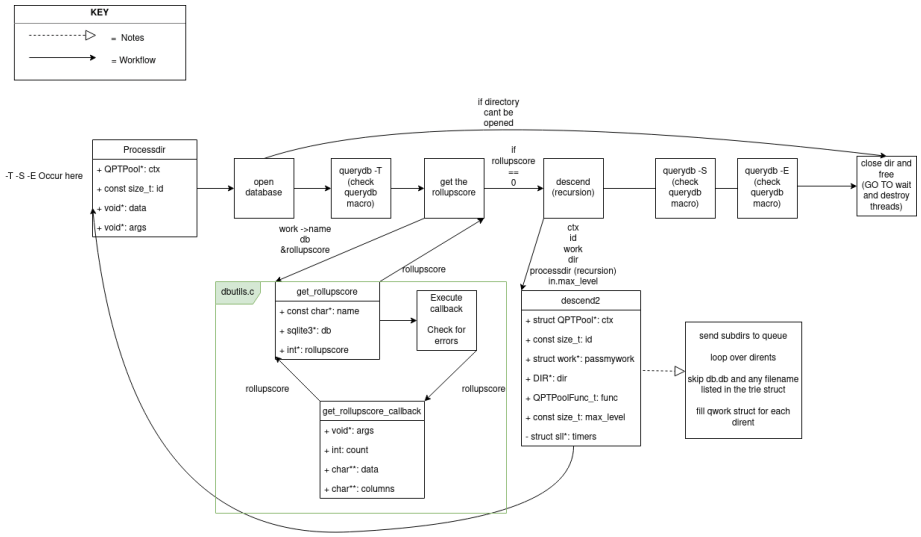

Figure 7: Workflow of `processdir`

10.1.10.2 `querydb` The `querydb` macro is used to execute SQL statements and handle errors.

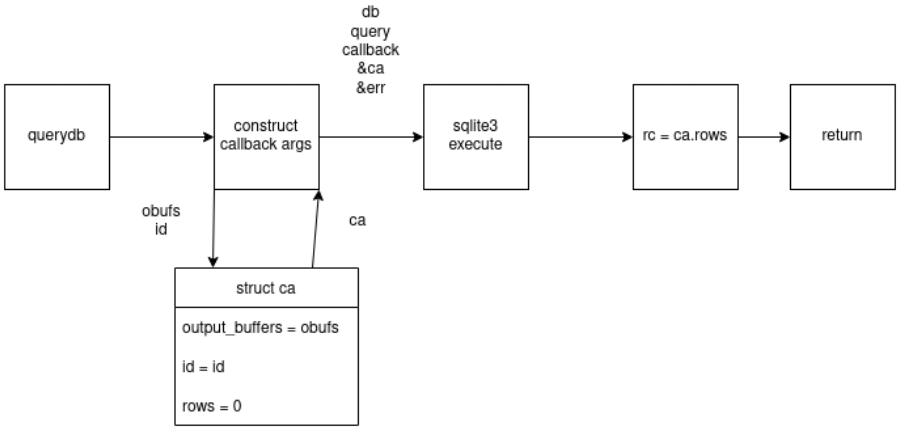

Figure 8: `querydb` macro workflow

10.2 `querydbs`

`querydbs` attaches all database files it is given into a single view for querying. All tables in the provided database files should have the same name and schema. For tables called `table`, the view combining all tables will be named `vtable`.

This was intended to allow for easy post-processing of `gufi_query -O` output database files.

`querydbs` does not have a `gufi_` prefix because it will work for any set of input databases with the same schema. For example, `querydbs` can also be used directly on individual database files from an index.

Note that there is a hard limit of 125 databases that can be queried by querydbs that is set by SQLite3.

10.3 SQLite Functions

Several convenience functions are added into each database instance opened for querying.

Table 7: SQLite functions that are available in `gufi_query` and `querydbs`.

Function	Purpose
`uidtouser(uid)`	Converts a UID to a user name
`gidtogroup(gid)`	Converts a GID to a group name
`modetotxt(mode)`	Converts numerical permission bits to text
`strftime(format, timestamp)`	Replaces SQLite's custom `strftime` with `strftime(3)`
`blocksize(bytes, unit)`	Converts a size to a the number of blocks of size `unit` needed to store the first argument. `unit` is the combination of at least one integer and/or a prefix and a suffix: Prefix: `K`, `M`, `G`, `T`, `P`, `E` Suffix (multiplier): *no suffix* (1024), `B` (1000), `iB` (1024) Return value is a string. Note that this function was implemented to replicate `ls` output and is meant for use with `gufi_ls` only, so use with caution. Examples: Call / Output blocksize(1024, '1000') / 2 blocksize(1024, '1024') / 1 blocksize(1024, 'K') / 1K blocksize(1024, '1K') / 1 blocksize(1024, 'KB') / 2KB blocksize(1024, '1KB') / 2 blocksize(1024, 'KiB') / 1KiB blocksize(1024, '1KiB') / 1
`human_readable_size(bytes)`	Converts a size to a human readable string

Table 8: SQLite functions that require the context of an index and thus are only available in `gufi_query`.

Function	Purpose
`path()`	Current directory relative to path passed into executable
`epath()`	Current directory basename
`fpath()`	Full path of current directory

`rpath` (sname, sroll)	Current directory relative to path passed into executable taking into account the rolled up name in summary table and rollup score. Should only be used with the `sname` and `sroll` columns of the `vrpentries`, `vrsummary`, `vrxpentries`, and `vrxsummary` views.
	
	Usage:
	`SELECT rpath(sname, sroll)`
	`FROM vrsummary/vrxsummary;`
	
	`SELECT rpath(sname, sroll) || "/" || name`
	`FROM vrpentries/vrxpentries;`

`level()`	Depth of the current directory from the starting directory
`starting_point()`	Path of the starting directory
`subdirs_walked()`	Number of subdirectories that were enqueued while processing the current directory. This function will return 0 if the current directory was rolled up. Only available in `-S` and `-E`. Not recommended for use with `-E`.

10.4 gufi_stat_bin

`gufi_stat` is a script that can be used to extract individual entries from an index like `stat(1)`. However, it is not a wrapper for `gufi_query`. Instead, it is a wrapper for `gufi_stat_bin`, a compiled executable. `gufi_stat_bin` does not use the configuration file and thus needs to be provided the search path.

10.4.1 Flags

Option	Description
`-f <FORMAT>`	use the specified FORMAT instead of the default. A newline is outputted after each use of FORMAT.
`-j`	print the information in terse form; Same as `-t`/`--terse` in `stat(1)`.

11 Deployment

GUFI functionality as described so far has been local. However, users are not expected to log into the machine with the indexes with an interactive shell. Instead, they are expected to call wrapper scripts to forward commands to the server (installed with the client RPM).

11.1 RPMs

When client building is enabled and the rpms built (`make package`), two rpms will be created. The server rpm will contain GUFI as described so far. The client rpm will contain executables that should be installed on the client node. The user executables on the client side simply forward commands to the server. A jailed shell should be set up on the server to prevent arbitrary command execution - see Section 11.4 for details.

11.2 SSH

Normal ssh, using passwords or user keys (`~/.ssh/id_rsa`), should be disabled on the server. Instead, ssh should be done using host based authentication (`/etc/ssh/ssh_host_rsa_key`).

11.3 Runtime Configuration

`gufi_query` provides the core GUFI functionality. However it is not expected to be used by users, as the syntax of `gufi_query` is somewhat unwieldy, and users are not expected to understand how GUFI works. Wrappers were created in order to provide users (and administrators) with predetermined queries that are used often and extract useful information (see the user guide for more details). The server has the actual implementation of the wrappers while the clients have command forwarders.

These wrappers require configuration files located at `/etc/GUFI/config` (with read permissions for all GUFI users) to function. These configuration files are simple text files containing lines of `<key>=<value>`. Duplicate configuration keys are overwritten by the bottom-most line. Empty lines and lines starting with # are ignored. The server and client require different configuration values. Example configuration files can be found in the `config` directory of the GUFI source, as well as in the build directory. The corresponding examples will also be installed with GUFI.

If a server and client exist on the same node, the server and client configuration file may be combined into one file.

11.3.1 Server

Key	Value Type	Purpose
Threads	Positive Integer	Number of threads to use when running `gufi_query`.
Query	File Path	The absolute path of `gufi_query`.
Stat	File Path	The absolute path of `gufi_stat_bin`.
IndexRoot	Directory Path	The absolute path of a GUFI tree (or parent of multiple indexes).
OutputBuffer	Non-negative Integer	The size of each per-thread buffer used to buffer writes to stdout. Setting this too low will affect performance. Setting this too high will use too much resources.

11.3.2 Client

Key Value	Type	Purpose
Server	URI	The URI of the server (IP address, URL, etc.)
Port	Non-negative integer	Port number
Paramiko	Directory Path	The absolute path to the paramiko directory on the client side

11.4 `gufi_jail`

`gufi_jail` is a simple script that limits the commands users logging in with ssh are able to run to only a subset of GUFI commands. `/etc/ssh/sshd_config` should be modified with the following lines:

```
Match Group gufi
    ForceCommand /path/to/gufi_jail
```