Implementation of Tokutek Hot Backup

Bradley C. Kuszmaul

August 22, 2020

The Tokutek Hot Backup System can back up the file-resident data of an application, such as MySQL, that runs as a single process. Applications such as PostgreSQL that operate as multiple processes could be backed up using a similar scheme with some adaptation. The application makes calls to operating system functions such as `open()`, `close()`, `dup()`, `unlink()`, `link()`, `rename()`, `read()`, `write()`, and `pwrite()`. The backup system makes a copy of the data of the application, and as it makes the copy, the backup system interposes itself in the system functions so that it observes those calls, and updates the backup copy accordingly.

For example, if there were a 100GB file that the backup system was backing up, it might take something like 15 minutes to copy the file to the backup. If while that copy operates, the application writes into the file, then the backup system observes the write and arranges for the newly written data to appear in the backup copy.

The backup system maintains a representation of the underlying file system, by keeping track of, for example,

- which open file descriptors (integers) refer to which file description,
- the read/write offset for each file description,
- which file name refer to which file in the directory, and
- which blocks in which files are being written.

An example of the state of the hotbackup system is shown in Figure **??**, which shows a mapping , called fmap, mapping from file descriptors to DESCRIPTIONs. In Figure **??**, file descriptor number 3 points to a DESCRIPTION.

A DESCRIPTION comprises an offset , which in this case is 7. The offset indicates where the next read or write will occur in the file. A DESCRIPTION further comprises a pointer to an sfile . A DESCRIPTION further contains a reference count (refcount) (which equals 1 in this case) indicating that there is one reference to the description in the fmap.

An SFILE comprises rangelocks , a set of ranges that are currently locked. In this case there are two ranges. The first range covers bytes 3–5. (In our notation, all ranges include the first value and don't include the last value. So range 3–5 includes bytes 3 and 4, but not byte 5, for example.) The second range covers bytes 18–21.

An SFILE further comprises a set of file names (one name is shown in this example); a reference count which indicates how many descriptions refer to the sfile (one in this case); and a file identifier (fileid) , which comprises a device number (13 in this case) and inode number (1431 in this case).

The system further comprises a directory which maps from names to SFILES, and an mapping structure called the imap that maps from fileids to SFILES. The directory provides a way to find an sfile given a name, and the imap provides a way to find an sfile given a file identifier.

In general the hotbackup system is employs the following structures.

- A file identifier, or *FILEID*, comprises a pair of numbers, a device identifier and an inode number. On many Unix file systems, this pair of numbers uniquely identifies a file. On Unix, the device identifer and inode number of a file can be obtained using the `fstat()` system call. Other file systems, such as windows, provide different kinds of FILEIDs, which the system can use to identify a file. In many unix systems, the device number might change when the machine restarts.

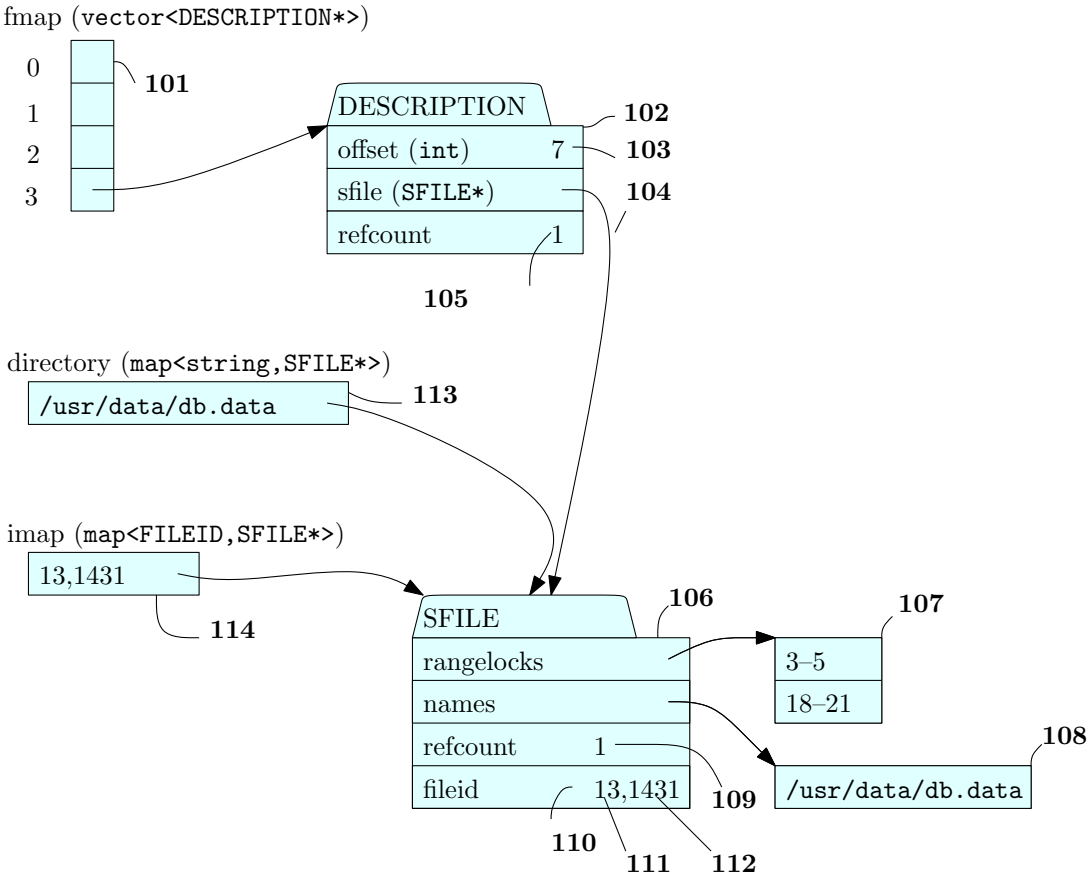

Figure 1: A typical state of the hot backup system.

- A *file descriptor* is an nonnegative integer that the application receives when it calls the `open()` library function. Typically these integers are relatively small, since in many operating systems, the integer that's returned is the smallest unused file descriptor.
- An *fmap* maps the file descriptor to a DESCRIPTION. The system employs a growable vector of pointers to DESCRIPTIONs to implement the fmap.
- A *DESCRIPTION* comprises
  - an offset indicating where the next read or write operation will take place in the file,
  - a pointer to an SFILE, and
  - a refcount integer that keeps track of how many fmap entries refer to the description.
- A *directory* maps file name strings to SFILEs. The directory is an unordered map implmented by a hash table.
- An *imap* maps FILEIDs to SFILEs. The imap is an unordered map implemented by a hash table.
- An *sfile* comprises the following:
  - Rangelocks, a set of ranges that are locked. A range is a pair of offsets, (l, h), in a file. The locked range includes byte number l inclusive through byte number h exclusive. To lock the entire file the system may lock the range $(0, \infty)$, where ∞ is a value that is considered to be larger than any offset in a file. For example, our system uses $\infty = 2^{64} - 1$ which is the largest value that fits in a 64-bit unsigned value. The system keeps the locked ranges in an ordered map which facilities determining whether a given range intersects any range in the rangelocks. An alternative would be for the system to keep the ranges in an unordered vector, which could work well, for example, if there were typically relatively few locked ranges at any given time.
  - Names, a set of strings. A string appears in the names of an sfile if and only if, in the directory, the string maps to the sfile. The Names sets is implemented using a hash table.
  - Refcount, an integer. This is a count of the number of DESCRIPTIONs that point at the SFILE.
  - A FILEID called fileid, comprising a device number and inode number. If s is an SFILE then `imap`$[s->\text{fileid}] = s$, that is in the imap, the file identifier of an sfile maps to the sfile.

The system maintains several invariants, including the following. If S is a string, F is an sfile, integer I is an open file descriptor pointing at a regular file, and D is an description then:

1. S is in F.names if and only if directory$[S] = F$.
2. fmap$[I]$ is populated (points to a description) and the description points to an sfile.
3. The number of descriptions that refer to F equals F.refcount. (One way for the refcount of F to become greater than one is to open the same file twice.)
4. The number of fmap entries that refer to D equals D.refcount. (One way for the refcount of D to become greater than 1 is to use the `dup()` or `dup2()` unix system calls.)
5. imap$[F.\text{fileid}] = F$.
6. The directory contains exactly the names of files that are open, under the names that they were opened through. But if those names change with, e.g., `rename()` or `unlink()` we update the directory). Recall that when the directory is updated, so must the corresponding names set in the relevant sfile.

The system operates as follows:

The manipulation of F.names and directory$[S]$ is done atomically, and is affected by `open()`, `creat()`, `rename()`, and `unlink()`. The system employs a lock, called the directory lock, which is held while updating F.names$[S]$ and directory$[S]$.

The system also employs a lock, called the fmap lock, to protect the fmap from concurrent operations.

The system also employs a lock, called the imap lock, to protect the imap from concurrent operations.

During backup of a file, we go ahead and create a description just as if the user had opened() the file. Thus, the refcount of an sfile will be incremented while the underlying file is being copied during backup.

Not Doing Backup

.

When we are not doing a backup the system interposes the system calls performed by the application and operates as follows:

When the user opens a file with `open`$(S, ...)$:

1. The system finds the sfile. After opening the file, the system performs a `fstat()` on the file, yielding a file identifier.

   Then the system looks in the imap to determine if there is an existing sfile with that file identifier. If yes, then the refcount is incremented, otherwise an sfile is created with refcount equal to 1.

   The system checks to see if directory[realpath(S)] points at the sfile, if it does not, then the directory is updated as well as the names of the sfile. The system creates a description, updates the fmap, and returns the file descriptor to the user.

The hot backup library maintains a ghostly mirror of the file system. We have several things that mirror the unix file system:

- filenodes: Keeps track of the range locking for a file. A file needs to be represented by its device and inode. Represents a unix file or an inode.
- directories: The mapping of names to filenodes. Represents the directory heirarchy of the file system, but only the part that involves open files.
- description: Contains a pair $\langle \text{offset}, \text{filenode} \rangle$. Represents a unix file description.
- descriptors: Contains a pointer to a description. For every open open file we have a descriptor. (In principle several descriptors could point to the same description, but that won't happen until we implement `dup`).

What must happen:

- On `write(fd)`:

  If no backup is running, we must still maintain the descriptoin (increment the offset).

  If backup is running, we must do the write in the destination space. (It's possible that the destination file hasn't been copied yet, in which case we have a choice: We don't have to mirror the write to the backup.

  To handle parallelism: we need to lock the range being written (in the filenode) before doing any of the updates. We need to lock the description so that we can update the offset and perform the writes.

Direct-IO