Ventisrv file formats

Mechiel Lukkien
mechiel@xs4all.nl

Google Summer of Code, for the Plan 9/Inferno project
August 2007

ABSTRACT

This document describes the file format used by *ventisrv* for its data and index file and also explains design decisions. It should provide enough insight in the file format to help with data recovery.

Introduction

At startup, *ventisrv* reads the index file sequentially from start to end and places a part of the score of each stored block in memory. The data file keeps the actual data and all relevant meta-data, enough to reconstruct an index file (though to reconstruct an index file, the entire data file has to be read). The data file is a concatenation of blocks, each consisting of a data header followed by the data itself; compression changes this slightly. The index file is a concatenation of index headers, each referencing a block in the data file (and in the same order). The file formats are described first for the simple case, without support for compression; first the data file format, followed by the index file format. Then the file format that supports compressed blocks is described.

Format of data file

An empty (zero length) data file is simply a data file without any blocks stored in it. A block is stored by writing a header, called *Dhdr*, to the data file, followed by the data itself. The header is 31 bytes long: a 4 byte *magic*, 20 byte *score*, 1 byte *data type*, 2 byte *size*, 4 bytes *connection time*. The fixed magic value is `0x2f9d81e5`. *Size* indicates the number of bytes following the header, thus size of the data. Even though 2 bytes can address up to 64 kilobytes, only values up to 56 kilobytes are valid since the venti protocol does not allow larger blocks. Note that the ‘zero score’   the score belonging to the zero-length data block   is never stored on disk. It is handled internally by *ventisrv*, though such a block is valid in the file format. The *score* in the header is the score of the data following the header. During operation *ventisrv* checks whether the score in the header matches the score it calculates from the data, to detect e.g. disk failures. The *connection time* is the time (in seconds since UNIX epoch) at which the venti session (TCP connection) was started. It can be used to group and relate blocks to an accidental or malicious batch of writes.

This is the definition of the `Dhdr` in Limbo (with functions removed), along with the *magic*:

```
Dhdrmagic:      con big 16r2f9d81e5;

Dhdr: adt {
       score:          Score;  # 20 bytes
       dtype:          int;    # 1 byte
       size:           int;    # 2 bytes
       conntime:       big;    # 4 bytes
};
```

Format of the index file

For each block (header and data) written to the data file, an *Ihdr* is written to the index file. An *Ihdr* is 15 bytes long: the first 8 bytes of the score called *halfscore*, a 1 byte *data type* and a 6 byte *offset* into the data file.

Only 8 bytes from the score are stored. Storing more is not useful: if more bytes were needed, main memory requirements would be exceed the amount of memory that fits in a computer. Also, the index file has to be read into memory at *ventisrv* startup, so it is best to keep it as small as possible. Even 8 bytes are more than needed for almost all *ventisrv* installations. Note that index headers do not contain a ‘magic’ and do not have data following them.

Only 6 bytes are used for storing the offset into the data file. More address space will never be needed because main memory will run out first when storing such large amounts of data. *Offset* is the offset in bytes into the data file to the header of the block that is referenced. Headers in the index and data file always occur in the same order. This means the offsets in the consecutive index headers are monotonically increasing.

Below the definition of an `Ihdr` in Limbo (with functions removed) is given. The field *compressed* is used only for compression and can be ignored for now.

```
Ihdr: adt {
       halfscore:      array of byte;  # 8 bytes
       dtype:          int;            # 1 byte
       offset:         big;            # 6 bytes
       compressed:     int;
};
```

File format changes to support compression

After the basic *ventisrv* functionality had been implemented, support for compressing blocks of data was added. The most straight-forward implementation is to add a bit to the *Ihdr* to indicate whether the block is compressed; and add a similar bit to the *Dhdr*, along with the size of the compressed payload (i.e. data actually on disk, which will be decompressed to the actual data). The actual implementation is a bit different. A new header can now occur in the data file, the *Fhdr* (‘F’ for flate, the compression algorithm used, implemented by Inferno’s *filter-deflate(2)* module. An *Fhdr* is of variable length, it contains information about one or more data blocks. This is necessary because the compressed payload following the header contains data for multiple blocks. The only reason for compressing multiple blocks into a single ‘compressed payload’ is that the compression ratio will be higher: the search history for the compression algorithm will be larger, and it does not have to build up such a history for each block to compress.

The fixed-length part of an *Fhdr* is 7 bytes long: a 4 byte *magic*, a 1 byte *count* for the number of blocks stored in the compressed payload, and a 2 byte *size* of the compressed payload. The fixed magic value is `0x78c66a15`. The maximum number of compressed blocks in a single *Fhdr* is 256. The size of the compressed payload is currently kept <= 56 kilobytes, though they can be up to 64 kilobytes. The maximum size cannot be much larger because the entire compressed payload up to the needed block has to be decompressed to read that block.

The variable-length part of the header immediately follows the fixed-length part. This variable part is made up of a header for each block stored in the compressed payload. Each such header looks much like a *Dhdr*, it is 27 bytes in size: a 20 byte *score*, 1 byte *data type*, 2 byte *size* and 4 byte *connection time*. *Size* is the size of the uncompressed data. To illustrate, consider an *Fhdr* that represents two blocks. On disk, it will start off with 7 bytes of fixed-size header. The *count* will be set to 2. This header is followed immediately by 27 bytes for the first block and another 27 bytes for the second block. After this a compressed payload follows with a *size* specified in the fixed-length part of the header. Note that the entire header is stored uncompressed on file. This allows for determining whether a score is present by only reading the header. Compressing the 27 bytes would not be of much use anyway, since 20 bytes of out 27 are the score, which is random data to the compression algorithm.

The index header changes only slightly: the most significant bit of the data file *offset* now indicates whether the header in the data file it points to is a *Dhdr* (when the bit is not set) or an *Fhdr* (when it is set).

This value is represented in the `Ihdr` by the field *compressed*. Headers in the index file are still in the same order of appearance as the blocks in the data file. Note that each stored score is given a header in the index file. This includes possible multiple scores in a single *Fhdr* in the data file: they each get an *Ihdr*, with the data file offset pointing to the same location in the data file. A program performing a lookup has to find the right score in the *Fhdr* itself.

Ventisrv assumes data will compress to at most 90% of the original size. When the current compression buffer has no room for another block the *Fhdr* is written to disk. If *ventisrv* tries adding a compressed block which was supposed to fit based on the 90%-size assumption but the maximum compressed buffer size is exceeded, the blocks are written to the data file without compression, as *Dhdr* blocks. In other words, *Fhdr* and *Dhdr* blocks can be mixed freely in the data file.

Conclusions

Support for compression makes the file format more complex, but not significantly so. Improvements could be made in the area of compression. For example, another compression algorithm could be used, one that depends less on building up compression history, or has some predefined histories to choose from. Also, since compression is relatively slow, a faster compression algorithm would be welcome. Early detection of whether data is compressible can alleviate the problem of slow compression somewhat. The header format would not necessarily have to change to accommodate for this.

The index and data files contain enough information to cross-check the validity of the data blocks. *Ventisrv* performs such checks on the most recently written blocks in these files at startup. The data file is always written before the index header is written, though not flushed explicitly, so the index file may be flushed in the background by a file system scheduler. In any case, missing headers in the index file are automatically added by *ventisrv* at startup, missing data file blocks are a fatal error and need to be resolved manually (by removing the index headers). The only remaining question is what to do with permanently damaged and non-recoverable (e.g. from backup) data blocks. Ideally, it should be possible to mark a data block as invalid, at least in the data file. There is currently no way to mark a block as such.