UNIVERSITY OF COPENHAGEN

Hindsight: Flexible, secure and efficient backup

Johan Brinch and Morten Brøns-Pedersen

http://github.com/mortenbp/hindsight

Abstract
Hindsight is a snapshot-based backup system providing guaranteed safe backup to untrusted sources such as Amazon S3 or DropBox. Our design features global de-duplicity through content-based indexing [1]; a means for exploiting data redundancy across all files in all snapshots.

Introduction

Hindsight is a backup system for filesystems. It backs up files and directories — and their metadata — to an external location. Multiple named snapshots are supported. A snapshot reflects the state of a directory at a particular time. Though usually only requiring a fraction of the space, each snapshot contains all files and directories that were in the file system when the snapshot was taken.

Hindsight is **flexible**. It can search through directory structures in snapshots and download just the directories or files you need; no more. This is especially useful when a range of snapshots must be searched for a file.

Hindsight is **efficient**. It uses compression to save space and eliminates redundant data across files; if more copies of the same file — or even just parts of it — exist, only the first one will travel to the back-end and be stored there. This is true across all snapshot ever taken. It processes several files concurrently to better utilise the disk and network, and it supports multi-core systems.

Hindsight is **secure**. It uses the NaCl [2] encryption library by Bernstein et al. to encrypt and authenticate the stored data. This prevents people who eavesdrop or has access to the back-end from reading or modifying the user's private data.

Hindsight is **crash-safe**. If the system shuts down (e.g. due to power failure) while taking a backup, it still works. As soon as the system is running again, Hindsight can efficiently continue from where it left of; nothing is wasted.

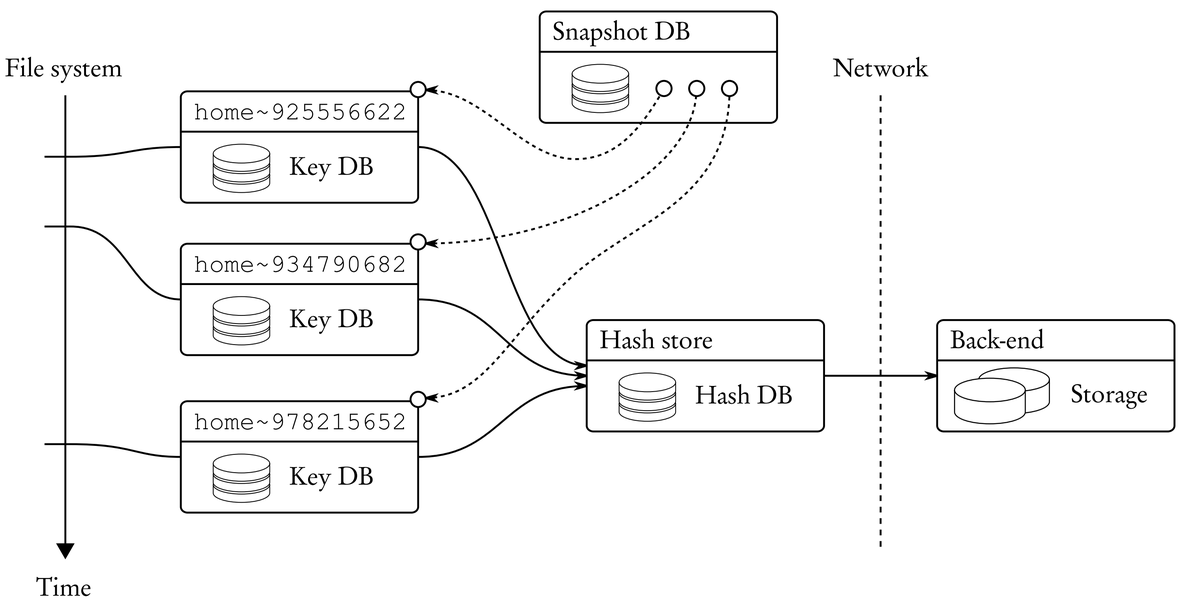
Overview

Each snapshot is represented as a database which contains all file system paths associated with that snapshot (figure: key DB). All the snapshots share a single global file-block database (figure: hash DB). Each file-block goes through this database which prevents multiple instances of the same block from being stored.

A database keeps track of all the snapshots (figure: snapshot DB), and is used when restoring or searching old snapshots. This is the entry point into the saved data.

Everything that is stored on the back-end is stored under a random ID, except the snapshot database. To restore the snapshot database in the event of a disk crash, it must be saved under a fixed name.

We require a minimal API of the back-end: `PUT`, `GET`, and `DELETE`. This makes it trivial to support new back-ends. So far, Hindsight supports remote storage in form of SSH, CouchDB, Amazon S3 [4] and local path (mount point).

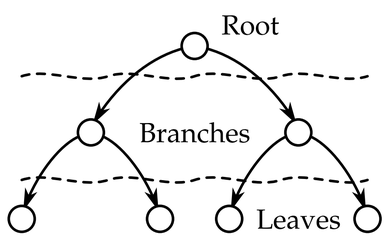


Overview:
An overview of Hindsight. Each snapshot is represented by an database mapping file paths to data-blocks. The hash database knows where each file-block is located on the back-end. A snapshot database keeps track of all the snapshots. In this figure three snapshots named `home` was taken.

Crash safety

Hindsight uses concurrent b-trees [3] for all its internal state. This includes its two main databases: one maintaining files in the snapshots; and one maintaining known (i.e. previously saved) file-blocks.

For Hindsight to be crash-safe, these databases need to be crash-safe. We achieve this in a very simple, yet elegant way: due to the b-tree structure, we can simply write the trees button up: from leaf to root.



After a crash due to a failure (e.g. no more battery), a recovery mechanism sets in to recover as much of the prior state as possible. This allows Hindsight to resume a snapshot after a crash, and quickly skip to the stage at which it crashed or was forced to stop.

Hindsight can work on very hostile systems, that shuts down often. Tests show that Hindsight can work with just a few minutes between crashes, and still complete the snapshot eventually.

Examples

Below we give a few examples of how to list file structures with Hindsight. The snapshot contains a total of 104391 files and are located on Amazon S3. Before each test, we clear the local cache.

First, we list all 104391 files in the snapshot:

```
jos@laptux ~ > time hindsight list diku | wc -l
104391

real   26m00.28s
```

Next, we list only the 14597 files in the folder "speciale", including all files in all subfolders:

```
jos@laptux ~ > time hindsight list diku:speciale | wc -l
14597

real    3m20.21s
```

Last, we list just the files of the "speciale" folder, excluding subfolders:

```
jos@laptux ~ > time hindsight listdir diku:speciale
speciale/.git
speciale/.gitmodules
speciale/INSTALL
speciale/README
speciale/TODO.org
speciale/code
speciale/literature
speciale/poster
speciale/report
speciale/salt

real    0m19.56s
```

We see that the time needed for a `list` command varies greatly. This is because we try to only download the needed part of the directory structure. The same approach is used for partial checkouts.

[1] Udi Manber. *Finding similar files in a large file system*. In Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference, pages 2–2, Berkeley, CA, USA, 1994. USENIX Association.

[2] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. *The security impact of a new cryptographic library*. 2011.

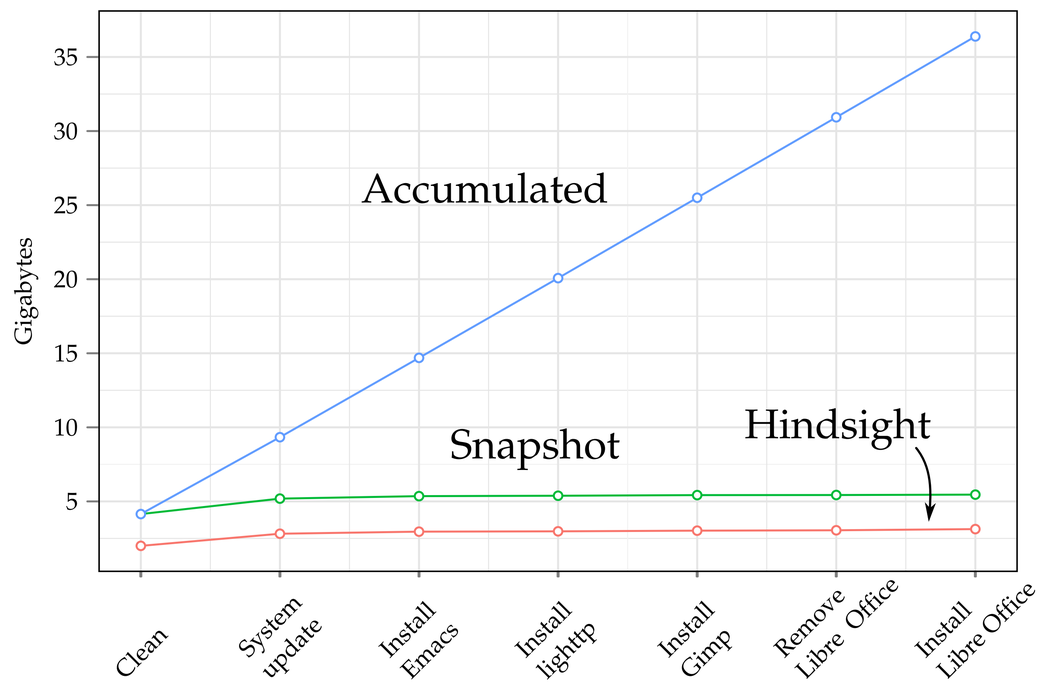
[3] Kim S. Larsen and Rolf Fagerberg. *B-trees with relaxed balance*. In Proceedings of the 9th International Parallel Processing Symposium, pages 196–202. IEEE Computer Society Press, 1993.

[4] Amazon Web Services. *Amazon Simple Storage Service*, 2012. http://goo.gl/wXJ1r.

[5] J. Weizenbaum. *Knotted list structures*. Commun. ACM, 5:161–165, March 1962. ISSN 0001-0782. http://goo.gl/PlmHU.

[6] U. Maheshwari and B.H. Liskov. *Fault-tolerant distributed garbage collection in a client-server object-oriented database*. In Proceedings of the Third International Conference on Parallel and Distributed Information Systems, pages 239–248. IEEE, 1994.

Space usage:
The graph shows the space required to hold seven snapshots of a KVM virtual machine image going through various changes. The blue line shows the spaces needed to make a new copy for each snapshot, the red line shows the space needed by Hindsight, and the green shows the size of the virtual machine image at each snapshot.



Deletion

Global de-duplicity comes at a price. In order to share blocks between one file and another, we have to re-arrange the data and keep track of who's using what.

When a snapshot is deleted, we cannot simply delete all the blocks it is using. What if another snapshot is using them as well? Instead a garbage collector must analyse the block usage and remove the file-blocks that are referenced by no-one.

A classic approach for keeping track of references is called reference counting [5]. Here, each file-block is associated with a count of those referencing it. When the count goes to zero, the block can be safely deleted.

But reference counting requires transactions, which are incompatible with our b-trees. Instead, we solve the problem using reference sets [6]. Like reference counting, a reference set keeps track of references to a file-block. But instead of just maintaining the count, the set keeps track of everyone using the block. Thus adding or removing a reference several times is no longer harmful, which means transactions are no longer necessary.

The cost of maintaining reference sets — as opposed to reference counts — is more than made up for by a faster system, and a smaller space overhead. Our b-trees split naturally into multiple files, making it trivial to take advantage of the usually large amount of redundancy between b-trees belonging to different snapshots. This again means a faster system with a smaller per-snapshot space overhead.

Performance

The graph shows the amount of storage required to back up a virtual machine image (figure: space usage). The image is a KVM image of a machine running Ubuntu Linux 11.10. The first snapshot was taken right after the operating system finished installing. Then the system was updated, and software was installed and removed. A snapshot was performed after each such change.

The red line shows the space used by Hindsight and the blue shows the accumulated space requirement (i.e. the space required if a full file copy was performed for each snapshot — this would be the case if for example Apples Time Machine had been used, because it doesn't support inter-file de-duplication). The green line is the size of the virtual machine image at the time of each snapshot.

We see that after the initial backup, space requirements are almost constant. Even though the virtual machine image changes from snapshot to snapshot Hindsight can see that most of the file hasn't changed and thus skip those parts. In this case the Hindsight back-up uses around 9% of the naïve approach.

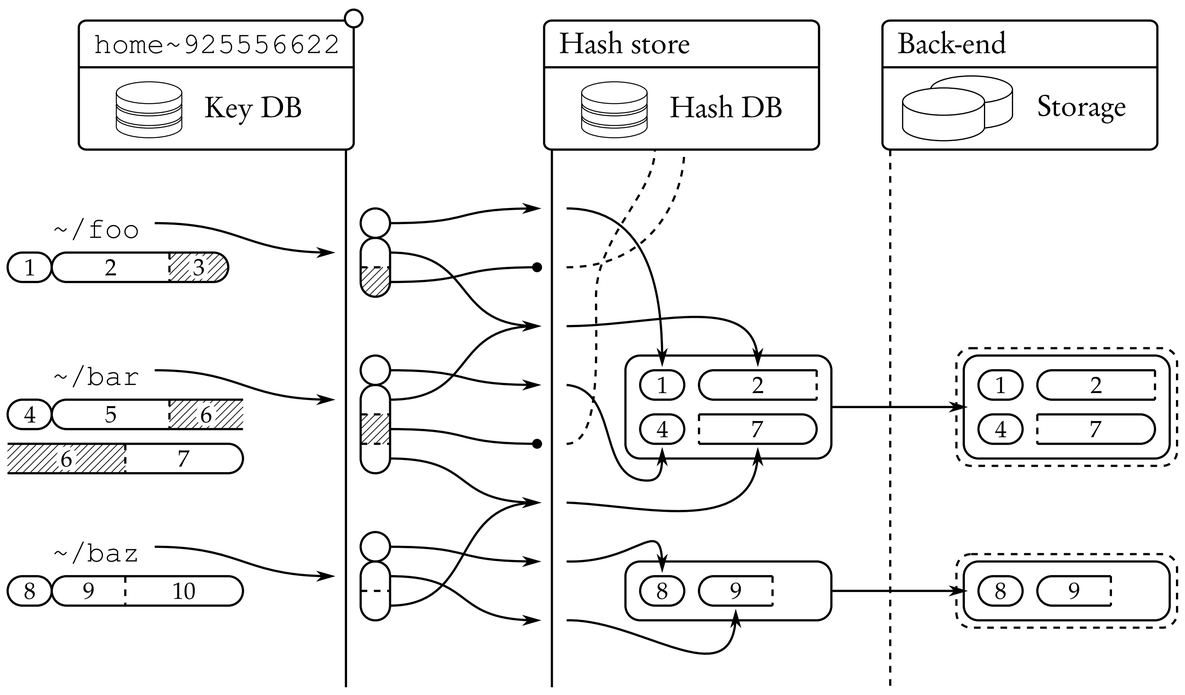
Internals

The figure below shows how data is stored internally in Hindsight. On the left are three files. The first block of each file is its metadata which always constitutes a single block. The rest of the files is split up into several blocks.

The key database maps each file path to references to the blocks it consists of. These references are in turn stored in the hash database, which maps them to the location of the block data inside blobs. Finally the blobs are wrapped in a layer of authentication and encryption before being stored on the back-end.

The shaded blocks in the figure are blocks which have been saved at an earlier time. Notice also that some blocks are equal and thus share the same reference.

To be able to recover the hash and key databases in the event of a disk crash, they must also be saved on the back-end. We use Hindsight itself to save these databases. This yields a new set of databases but these are much smaller, which enables us to save them as tarballs.



The internals of Hindsight:
On the left are the files being backed up, and on the right is the back-end. The vertical lines represents mappings; the solid are databases stored locally as b-trees, and the dashed is the back-end API.