HAMMER File System

Efficient Log-less Master to Multi-Slave Replication

Michael Neumann <mneumann@ntecs.de>

Oustanding Features

- Fine-grained automatic snapshots
- Master to Multi-Slave Replication

Fine-grained automatic snapshots

- Every flush to disk (about every 30 seconds) creates a new version of the file system.
- FS never forgets data unless explicitly told.
- `undo` can be used to restore old versions of files.
- Files can be accessed „as-of“ by appending `@@transaction-id` to the filename (transaction_id via `hammer synctid`).

Master-Slave Replication

- A master filesystem can be incrementally replicated locally or remote (via ssh) to one or more read-only slave filesystems:

```
masterhost> hammer pfs-master /www # create master filesystem
masterhost> hammer -b 100k mirror-stream /www slavehost:/www &
slavehost> hammer -b 1m mirror-stream masterhost:/www /www &
```

Limit bandwidth

- `mirror-stream` runs in the background and transfers changes as soon as they happen.

Live Replication Demo

Upload

Replicate in real-time

You

B+Tree Representation

B-Tree Leaf Node

`+obj_id: int64_t`
FS-wide unique identifier

`+key: int64_t`
data offset or dir-entry namekey hash

`+rec_type: uint16`
record type (e.g. INODE, DIRENTRY, DATA)

`+obj_type: uint8_t`
DIR, REGFILE, SYMLINK, CDEV, BDEV, FIFO

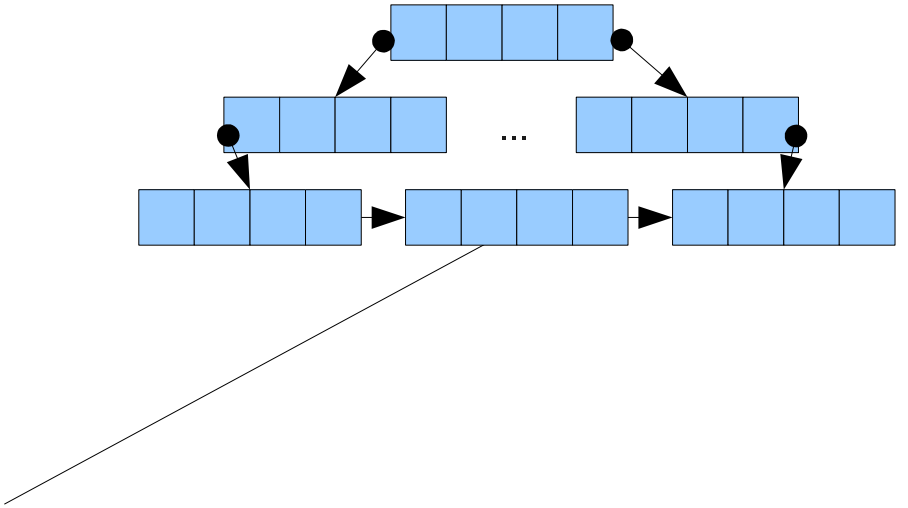
`+data_offset: hammer_off_t`
Pointer to data block

`+data_len: int32_t`
Length of data

`+create_tid: hammer_tid_t`
Record creation transaction ID

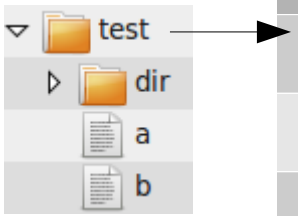
`+delete_tid: hammer_tid_t`
Record deletion transaction ID

`+localization: int32_t`
Used to store inodes of the same directory close to each other on disk.



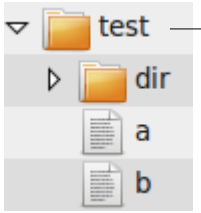
ORDER BY (localization, **obj_id**, **key**, create_tid)

B+Tree Representation



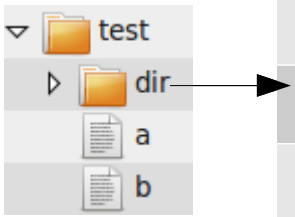
obj_id	key	rec_type	obj_type	data
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}
1	hash(„a“)	DIRENTRY	-	{obj_id: 3, name: „a“}
1	hash(„b“)	DIRENTRY	-	{obj_id: 3, name: „b“}
2	-	INODE	DIR	{uid, gid, ...}
3	-	INODE	REGFILE	{uid, gid, ...}
3	0	DATA	-	Data @offset 0
3	65536	DATA	-	Data @offset 64k

B+Tree Representation



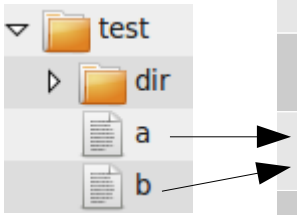
obj_id	key	rec_type	obj_type	data
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}
1	hash(„dir“)	**DIRENTRY**	-	{obj_id: 2, name: „dir“}
1	hash(„a“)	**DIRENTRY**	-	{obj_id: 3, name: „a“}
1	hash(„b“)	**DIRENTRY**	-	{obj_id: 3, name: „b“}
2	-	INODE	DIR	{uid, gid, ...}
3	-	INODE	REGFILE	{uid, gid, ...}
3	0	DATA	-	Data @offset 0
3	65536	DATA	-	Data @offset 64k

B+Tree Representation



obj_id	key	rec_type	obj_type	data
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}
1	hash(„dir“)	DIRENTRY	-	**{obj_id: 2, name: „dir“}**
1	hash(„a“)	DIRENTRY	-	{obj_id: 3, name: „a“}
1	hash(„b“)	DIRENTRY	-	{obj_id: 3, name: „b“}
2	-	INODE	DIR	{uid, gid, ...}
3	-	INODE	REGFILE	{uid, gid, ...}
3	0	DATA	-	Data @offset 0
3	65536	DATA	-	Data @offset 64k

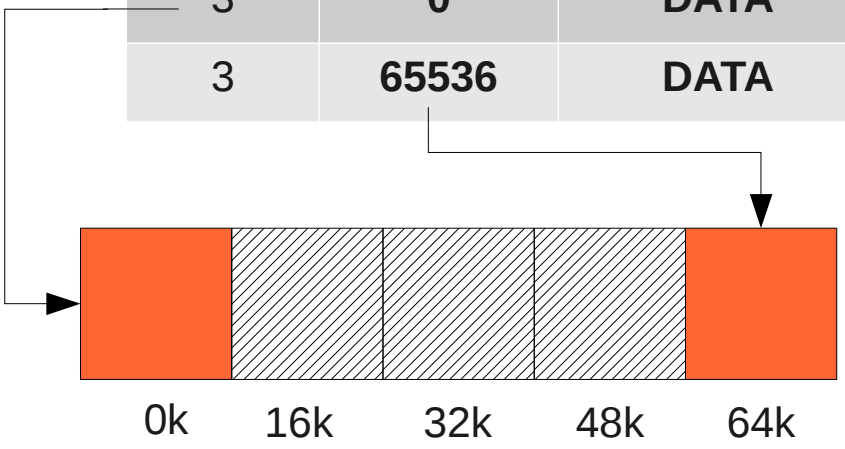
B+Tree Representation



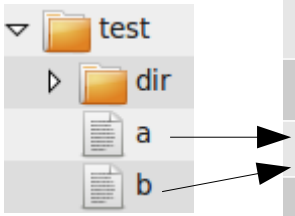
obj_id	key	rec_type	obj_type	data
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}
1	hash(„a“)	DIRENTRY	-	**{obj_id: 3, name: „a“}**
1	hash(„b“)	DIRENTRY	-	**{obj_id: 3, name: „b“}**
2	-	INODE	DIR	{uid, gid, ...}
3	-	INODE	REGFILE	{uid, gid, **linkcnt=2**, ...}
3	0	DATA	-	Data @offset 0
3	65536	DATA	-	Data @offset 64k

B+Tree Representation

obj_id	key	rec_type	obj_type	data
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}
1	hash(„a“)	DIRENTRY	-	{obj_id: 3, name: „a“}
1	hash(„b“)	DIRENTRY	-	{obj_id: 3, name: „b“}
2	-	INODE	DIR	{uid, gid, ...}
3	-	INODE	REGFILE	{uid, gid, linkcnt=2, ...}
3	**0**	**DATA**	-	Data @offset 0
3	**65536**	**DATA**	-	Data @offset 64k

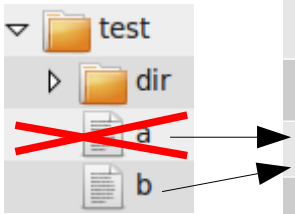


Adding History



obj_id	key	rec_type	obj_type	data	create_tid	delete_tid
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}	100	0
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}	100	0
1	hash(„a“)	DIRENTRY	-	**{obj_id: 3, name: „a“}**	100	0
1	hash(„b“)	DIRENTRY	-	**{obj_id: 3, name: „b“}**	100	0
2	-	INODE	DIR	{uid, gid, ...}	100	0
3	-	INODE	REGFILE	{uid, gid, **linkcnt=2**, ...}	100	0
3	0	DATA	-	Data @offset 0	100	0
3	65536	DATA	-	Data @offset 64k	100	0

Adding History

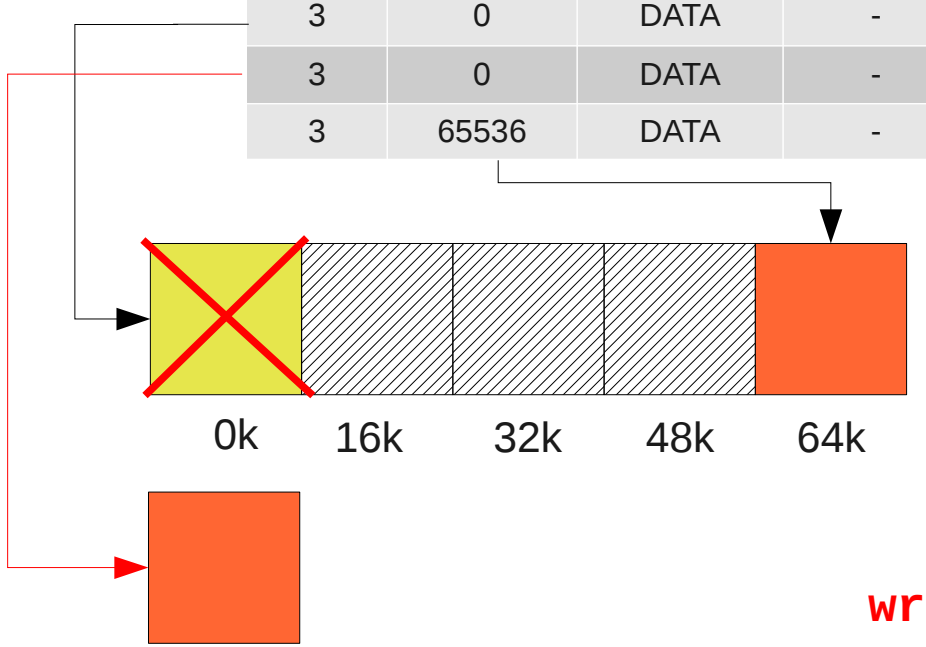


obj_id	key	rec_type	obj_type	data	create_tid	delete_tid
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}	100	0
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}	100	0
1	hash(„a“)	DIRENTRY	-	**{obj_id: 3, name: „a“}**	100	**101**
1	hash(„b“)	DIRENTRY	-	**{obj_id: 3, name: „b“}**	100	0
2	-	INODE	DIR	{uid, gid, ...}	100	0
3	-	INODE	REGFILE	{uid, gid, **linkcnt=2**, ...}	100	**101**
3	-	INODE	REGFILE	{uid, gid, **linkcnt=1**, ...}	101	0
3	0	DATA	-	Data @offset 0	100	0
3	65536	DATA	-	Data @offset 64k	100	0

`remove(„a“)`

Adding History

obj_id	key	rec_type	obj_type	data	create_tid	delete_tid
1	-	INODE	DIR	{uid, gid, mode, mtime, ...}	100	0
1	hash(„dir“)	DIRENTRY	-	{obj_id: 2, name: „dir“}	100	0
1	hash(„a“)	DIRENTRY	-	{obj_id: 3, name: „a“}	100	101
1	hash(„b“)	DIRENTRY	-	{obj_id: 3, name: „b“}	100	0
2	-	INODE	DIR	{uid, gid, ...}	100	0
3	-	INODE	REGFILE	{uid, gid, linkcnt=2, ...}	100	101
3	-	INODE	REGFILE	{uid, gid, linkcnt=1, ...}	101	0
3	0	DATA	-	Data @offset 0	100	**102**
3	0	DATA	-	Data @offset 0	**102**	0
3	65536	DATA	-	Data @offset 64k	100	0




```
write(file=„b“, pos=0, some_bytes);
```

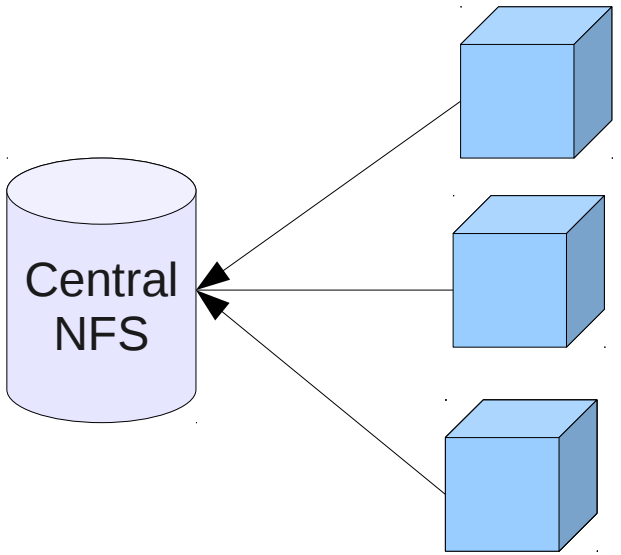
Ultimate Goal of DragonFlyBSD

- *Single-System-Image Clustering*
  - Cluster of machines that appears to be a single system
  - → Single process space
  - → **Single view of the file system**

Src: www.staff.tugraz.at/mlang/cluster

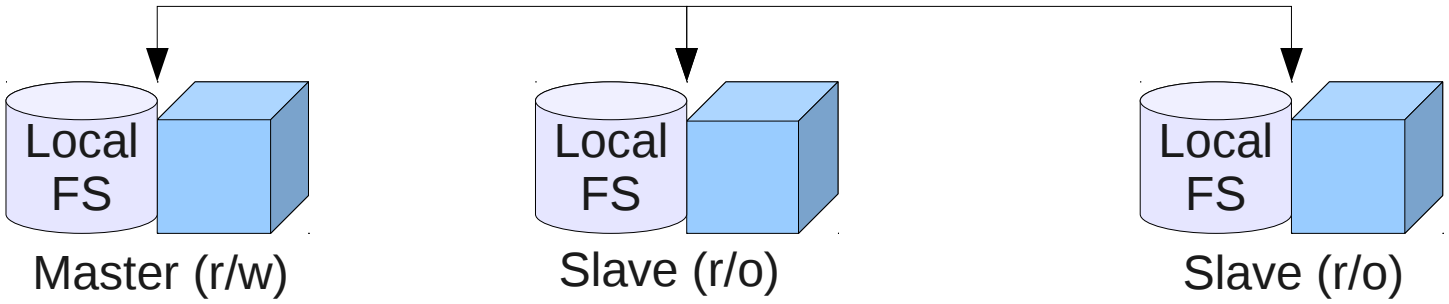
Single View of File System

- Central Networked File System
  - Single Point of Failure
  - Not Scalable



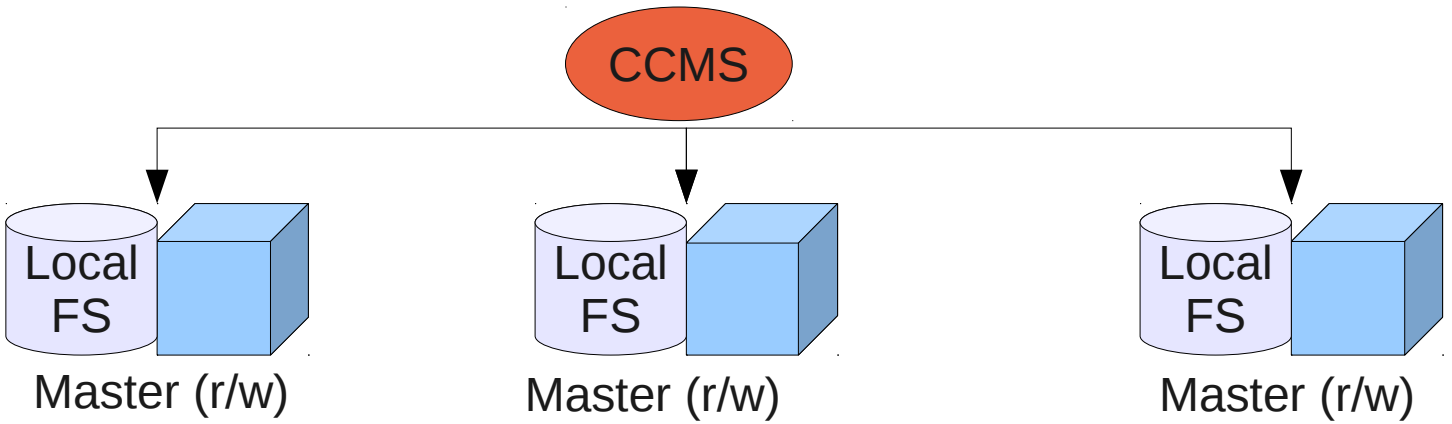
Single View of File System

- Master-Slave Replicated File System
  - HAMMER supports this.
  - Does not allow true SSI operation.



Single View of File System

- Multi-Master Replicated File System
  - Scalable and Fault-Tolerant
  - Not trivial to implement Cache Coherency Management System (CCMS).
  - HAMMER is designed to support this in the future!



Literature

- http://kerneltrap.org/HAMMER
- http://www.dragonflybsd.org/hammer/

Backup Slides

Explicit snapshots

- In order to remove „old“ data, HAMMER must be told which snapshots to keep.
- A snapshot is a special B-Tree node containing the transaction-id and a description.
  - Creating a snapshot as-of-now:

    `hammer snap /fs „description“`
  - Listing available snapshots:

    `hammer snapls /fs`

Cleaning up

- HAMMER filesystems must be large enough (40+ GB) in order to not fill up too quickly.
- `hammer cleanup` will get rid of all versions in-between explicit snapshots. It involves:
  - Pruning: Removing old versions from the B-Tree; does not actually delete data!
  - Rebalance the B-Tree
  - Reblock: Defragment bigblocks
- The frequency and duration of each operation, e.g. *rebalance 5 minutes every day*, can be set up for each pseudo-filesystem individually.

Incremental Replication

- Each slave FS stores last_seen_tid.
- SELECT * FROM Records WHERE create_tid > last_seen_tid;

  UPDATE last_seen_tid;
- This is further optimized by storing max(create_tid, delete_tid) of the left and right subtrees in each internal node, allowing to skip whole subtrees.