Once upon a time

- 80’s: Ingres, System R, SyBase, ...
Once upon a time

- 80’s: Ingres, System R, SyBase, …
- Common Recommendations: *Use raw discs for transaction logs* …
Once upon a time

- 80’s: Ingres, System R, SyBase, …
- Common Recommendations: Use raw discs for transaction logs …
  - And for data areas as appropriate
Once upon a time

- 80’s: Ingres, System R, SyBase, ...
- Common Recommendations: *Use raw discs for transaction logs* …
  - *And for data areas as appropriate*
- Stripe table space across spindles
Typical storage stack

- syscalls to userspace
- VFS
- XFS
- ext4
- swap
- page cache
- volume
- volume
- volume
- LVM
- RAID controller
- High Speed Disk
- High Speed Disk
- High Speed Disk
- High Speed Disk
Typical storage stack

- Userspace doesn’t understand disc layout — can’t optimise.
- Filesystems can’t see disc layout – can’t optimise
What’s a file system?

(name)

(metadata)

(file content)

(symlinks)
What’s a file system?

- Writes ordered for **Consistency**
- Writes to different files are **Isolated** from each other
- After `fsync()` data written survives crashes: it is **Durable**
Database ACID

- Atomic
- Consistency
- Isolated
- Durability
But wait there’s more

- Use a RDBMS that handles replication (e.g., Galera MariaDB) — distributed FS for ‘free’
- Easy to add attributes for experimentation
- No need for `fsck`
But:
But:

Modern RDBMS rely on FS.
But:

Modern RDBMS rely on FS.

(At least, Postgres does: discovered last night MySQL does not)

Decided to try proof of concept anyway.
Sam Li

Tsukasa Hamano and Michal Ludvig

https://sourceforge.net/projects/mysqlfs/

(Not touched since 2009)
Schema

- **tree**
  - inode
  - parent
  - name

- **inodes**
  - possible joins

- **data blocks**
  - inode
  - seq
  - data
namei() is funky join: namei("/a/b/c") →

SELECT t3.inode,
   (SELECT COUNT(inode) FROM tree AS t4 WHERE t4.inode=t3.inode)
   AS nlinks
FROM tree AS t3
JOIN tree AS t2 ON t3.parent = t2.inode
JOIN tree AS t1 ON t2.parent = t1.inode
JOIN tree AS t0 ON t1.parent = t0.inode
WHERE t0.parent IS NULL AND
   t1.name = 'a' AND
   t2.name = 'b' AND
   t3.name = 'c';
Functionality

- Passes https://www.tuxera.com/community/posix-test-suite/
Functionality

- Passes [https://www.tuxera.com/community/posix-test-suite/](https://www.tuxera.com/community/posix-test-suite/)
  - Except ctime updates to directories
Functionality

- Passes [https://www.tuxera.com/community/posix-test-suite/](https://www.tuxera.com/community/posix-test-suite/)
  - Except ctime updates to directories
  - And `rename()` integrity checks
Functionality

- Passes [https://www.tuxera.com/community/posix-test-suite/](https://www.tuxera.com/community/posix-test-suite/)
  - Except ctime updates to directories
  - And `rename()` integrity checks
  - And multi-user mounts
Functionality

- Passes [https://www.tuxera.com/community/posix-test-suite/](https://www.tuxera.com/community/posix-test-suite/)
  - Except ctime updates to directories
  - And `rename()` integrity checks
  - And multi-user mounts
  - And silent truncations of long file names
Functionality

- Passes [https://www.tuxera.com/community/posix-test-suite/](https://www.tuxera.com/community/posix-test-suite/)
  - Except ctime updates to directories
  - And `rename()` integrity checks
  - And multi-user mounts
  - And silent truncations of long file names
  - ...
Performance

450x slower than XFS on postmark
Performance

450x slower than XFS on postmark

20x slower than XFS for general read/write ops
Performance

450x slower than XFS on postmark
20x slower than XFS for general read/write ops
(machine with slow disk)
Performance

40x slower than XFS on postmark
Performance

40x slower than XFS on postmark

100x slower than XFS for general read/write ops
Performance

40x slower than XFS on postmark
100x slower than XFS for general read/write ops
(machine with fast disk)
Fixes

- Move to fuse version 3 — better caching
- Fix ctime updates where it’s cheap
- Reduced number of queries in `getattr()` (for `stat()`)
- Fix off-by-one errors for ENAMETOOLONG
- Refuse to rename over a non-empty directory; refuse to unlink non-empty directory
  - needs extra query in `unlink()` to check
- Use `allow_user` and `use_ino` flags to fuse
Non-Posix Semantics

- . and .. are fake: don’t contribute to nlinks
- ctime not updated on directories
- Directories have zero size
- statvfs() returns zero size and usage.
fuse bug?

every now and then sync hangs.
9x slower than XFS on postmark
Performance

9x slower than XFS on postmark
7x slower than NFS for general read/write ops
Performance

9x slower than XFS on postmark
7x slower than NFS for general read/write ops
2.5x CPU utilisation (Mostly in mariaDB engine)
Performance

$ time git clone /usr/src/linux-5.x

<table>
<thead>
<tr>
<th></th>
<th>mysqlfs</th>
<th>XFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>23m53.192s</td>
<td>9m2.217s</td>
</tr>
<tr>
<td>user</td>
<td>8m13.445s</td>
<td>6m20.768s</td>
</tr>
<tr>
<td>sys</td>
<td>1m46.557s</td>
<td>1m19.244s</td>
</tr>
</tbody>
</table>
### Performance

```bash
$ time git clone /usr/src/linux-5.x
```

<table>
<thead>
<tr>
<th></th>
<th>mysqlfs</th>
<th>XFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>23m53.192s</td>
<td>9m2.217s</td>
</tr>
<tr>
<td>user</td>
<td>8m13.445s</td>
<td>6m20.768s</td>
</tr>
<tr>
<td>sys</td>
<td>1m46.557s</td>
<td>1m19.244s</td>
</tr>
</tbody>
</table>

Hides CPU time used by DBMS – about another 10m User, 2m Sys.
Where’s all the time going?

Worst operations: creat(), rename(), unlink(), write(), stat()
Where’s all the time going?

Worst operations: creat(), rename(), unlink(), write(), stat()

These use stat() internally.
Where’s all the time going?

Worst operations: `creat()`, `rename()`, `unlink()`, `write()`, `stat()`

These use `stat()` internally.

three queries per block written.
More improvements possible

- Cache type and mode in tree table
- Cache recently used inodes
- Double block size
New things possible

- Replication (with MariaDB Galera Cluster) — works, but slow.
- Fast `find` using SQL query.
- Full-text-search if index on content.
- Fast `fsck`
- Easy to add other features (e.g., resource fork, HFS style)
Summary

• Get it from
  https://github.com/samzyy/DB-based-replicated-filesystem

• More-or-less works
• Performance not too painful
• May serve as experimental platform for FS features
• Pull requests welcome.