The Reiser4 File System

An introduction to the path-breaking new file system, and some insights into the underlying philosophy.

Reiser4

New FS for Linux

- Open source

Currently in beta

Merged into kernel soon

Started by Hans Reiser of Namesys

Very fast

New innovations

Files

Files

- User - A collection of related data
  - Application abstracts actual storage at most times
  - Management is primitive
- Developer – A contiguous byte-stream
  - Random access, but…
  - Can't add to middle
  - Can only delete from end
- File system
  - Not necessarily contiguous
  - Divided into blocks
- Small files are expensive
  - 1 block per file, irrespective of file size
  - Tails

Directories

Directories allow the user to impose a hierarchy to storage

- Convenient
- Invaluable if number of files is large
- Simpler to implement than more generic interfaces

Actual implementation

- Linear arrangements are inefficient to search
- Trees make sorting faster
- B-Trees make access faster
  - More fanout, less depth
  - Price is complexity - justified by performance gain

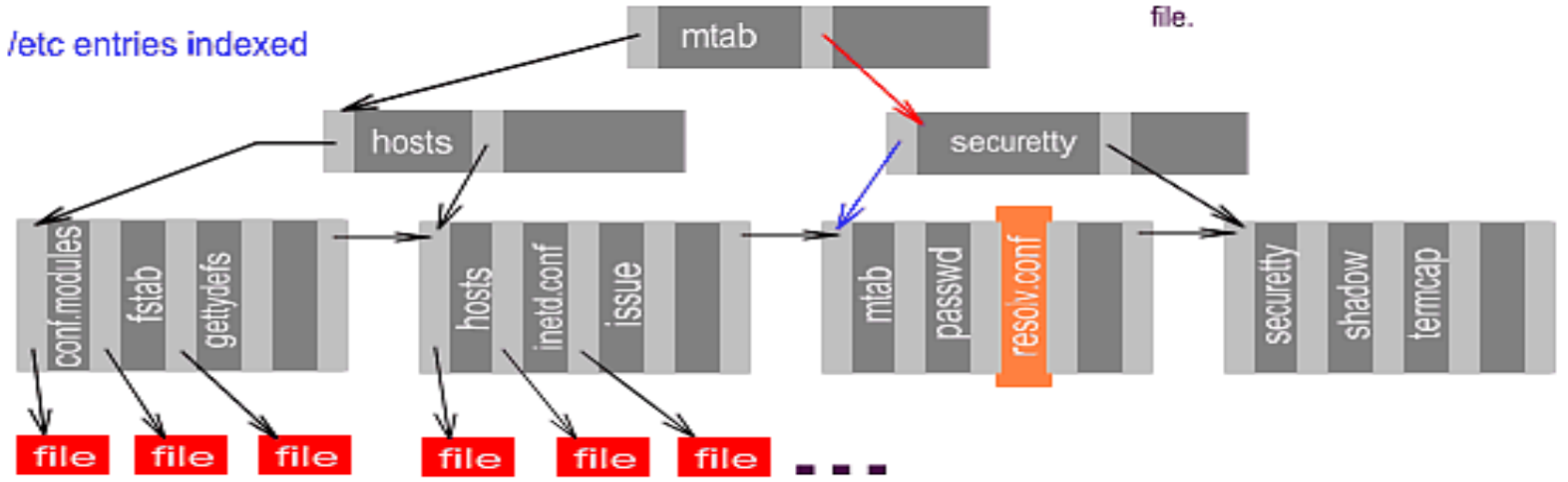
B+ Trees

Like B-Trees, but data *only* at **leaves**
Improves caching prospects

(1)to locate the file resolv.conf we begin at the tree's root. scan sequentially, and find that there is no key greater than resolv.conf, so we use the last pointer (in red)

(2) got directed to another internal node. Let's do the same. Scan through the node's keys and realise that "securetty" is a greater key than "resolv.conf". We use the accompanying pointer (in blue).

(3) we got the final leaf node. Now it's time to scan sequentially throughout the ascending ordered keys of the node. Finally, found the desired key, we should use the accompanying pointer to the "resolv.conf" named file.



Dancing Trees

XFS (by SGI) is very aggressive in caching

Reiser4 is too

The tree is updated on flush to disk *only*

What does this mean

- Repacking the tree repeatedly is intensive
- Do it at the last moment, before the flush

Tail Packing

Tails waste space

Reiser4 packs them into a single block

- The tree structure allows this

Saves about 5% disk space

- Reiser4 has 94% storage efficiency

Repacking is expensive

- Tail must be removed on append to file
- Rest must be repacked

Journaling 101

Traditionally

- Consider a power cut / server crash
- Must reboot, FS is unclean (inconsistent)
- Scan the disk, make sure all on-disk structures are valid
- Time consuming
- 500 GB server – 3 *hours!*

More Journaling

Journaling was first used in DBMSes

Make all “transactions” (disk updates) atomic

- Keep a log (journal) of all pending transactions
- Only remove from journal after completion
- An operation either occurs completely or doesn’t at all (*atomic*)
- No chance of inconsistency

Crash? No problem!

- On restart, *replay* the journal

Wandering Logs

Data journaling expensive

- Only metadata is done

Reiser4 introduces data journaling at reduced cost

- Don’t modify data in-place, modify disk data structures
- Some updates are better in-place – be intelligent about it
- Experimental

Layering

Semantic Layer

- Interfaces to user, developer
- Rich in expressive power

Storage Layer

- Limited interfaces
- Tuned for speed

Plugins

Plugins are the key to Reiser4's power

Extendible interfaces mean flexibility

- Not tied down to one set of interfaces or semantics any more
  - Addition of features like ACLs is a breeze
  - Changing user view of the FS is easy

Increased modularity

Every file has a plugin-id

- This is an offset into an array of plugin functions

Plugin Types

File plugins
Directory plugins
Hash plugins (directory -> file mapping)
Security plugins
Item plugins (balancing of the tree)
Key assignment plugins (for per file key)
Node search and Item search plugins
(For different kinds of nodes and items)

Software Engineering Miscellany

Break complex primitives into simpler ones

- Keeping only simple primitives maximises expressive power
- Can everything be done with files and directories alone?

Unify namespaces

- Too many different types of objects means every object needs to know how to talk to other objects
- Example, if attributes can be made files, don’t make a whole new object type for them

Putting it all together: /etc/passwd

Simple text file with ':' delimited fields

- Stores user name, id, password, shell etc…

Only *root* can modify it

Very coarse grained

User can't change own shell

Password is set by a privileged program

A better /etc/passwd

Make a plugin to give a simpler interface

- /etc/passwd/joe/shell accesses the shell field
- “passwd” is still a *file*, but looks like a directory
- Use finer grained security for the “files” that are actually fields

Alternatively, make passwd a directory, and let the plugin aggregate the contents

Other examples

Small files are efficient in Reiser4

Adding security attributes is simple

- ACLs are one example

Just make the attribute a file, internally

- It's small, but that's okay
- File is hidden to all higher layers
- The plugin handles all access to the attribute

Conclusion

Reiser4 and later are definitely worth keeping an eye on for

- Speed
- Features

WinFS is treading a similar path as well

References

www.namesys.com

- Documentation for Reiser4
- Hans Reiser's extensive Future Vision whitepaper

B-Trees can be looked up in any book on advanced data structures

The End

Thank you for coming!