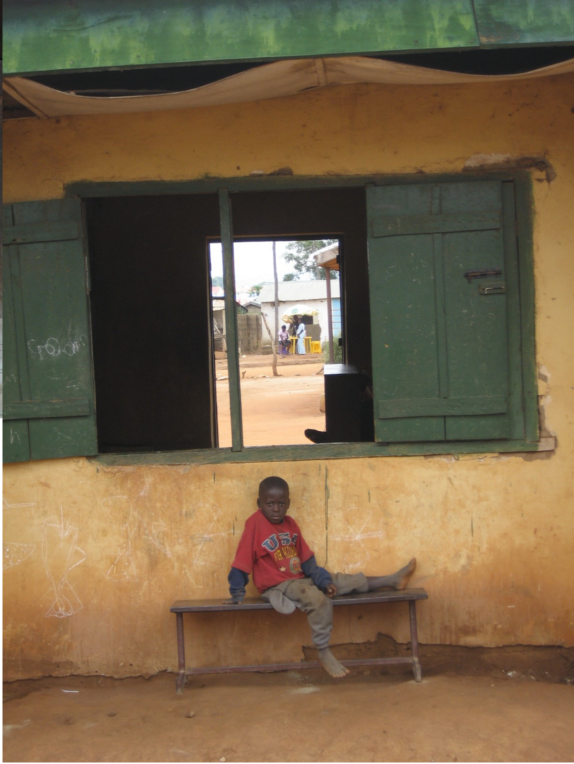
One Laptop per Child

# Algorithms and Data Structures

Dr. C. Scott Ananian <cscott@laptop.org>

# Talk Outline

- OLPC Overview
- Cool algorithms
  - Olpcfs
    - “Dancing trees” (Functional data structures)
    - Secure sharing
    - Content matching
  - Flash filesystems
    - PMA data structures
    - Cache-oblivious B-trees
- Open problems
  - Fully-persistent B-Trees
  - Set difference



# Sometimes the riskiest path is the status quo.

# A global transformation of education

- It's about giving children who don't have the opportunity for learning that opportunity: so it's about access; it's about equity; and it's about giving the next generation of children in the developing world a bright and open future.

# Children lack opportunity, not capability

1. High-quality education for every child is essential to provide an equitable and viable society;

2 A connected laptop computer is the most powerful tool for knowledge creation;

3 Access on a sufficient scale provides real benefits for learning.

ONE LAPTOP PER CHILD

# A vaccine is an agent of change.

Jonas Salk made the analogy between education reform and immunology: both require scale and reach in order to be successful.

# A connected laptop is not a cure

- ...but it is an agency through which children, their teachers, their families, and their communities can manufacture a cure.
- They are tools with which to think, sufficiently inexpensive to be used for work and play, drawing, writing, and mathematics.

# five principles

1. child ownership—use of the laptop at home;
2. low ages—ages 6 to 12—low floor, no ceiling;
3. saturation and
4. connection—collaborative and community;
5. free and open—the child is an active participant in a global learning community.

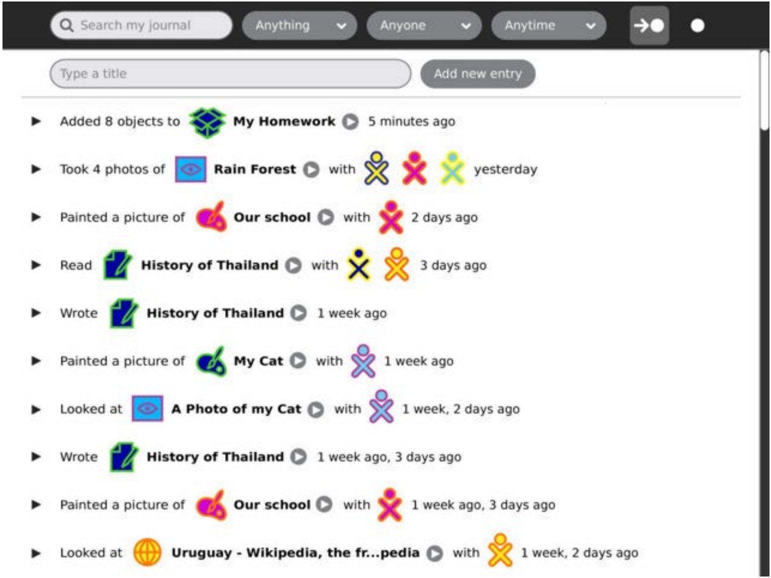
# Datastore: olpcfs

# End-user goals

- Support journal and bulletin board abstractions
- Provide Bitfrost P_SF_RUN and P_SF_CORE protections

# Journal: objects & actions

- Action view



▶ Added 8 objects to My Homework 5 minutes ago

▶ Took 4 photos of Rain Forest with yesterday

▶ Painted a picture of Our school with 2 days ago

▶ Read History of Thailand with 3 days ago

▶ Wrote History of Thailand 1 week ago

▶ Painted a picture of My Cat with 1 week ago

▶ Looked at A Photo of my Cat with 1 week, 2 days ago

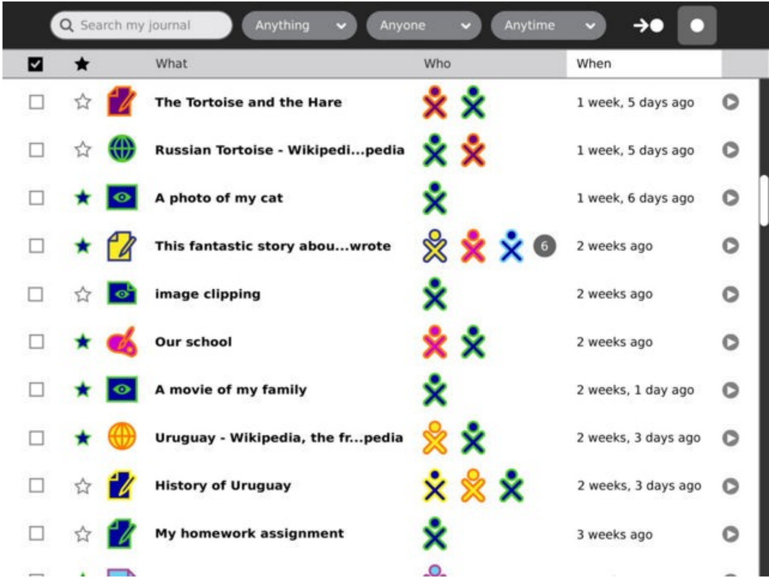
▶ Wrote History of Thailand 1 week ago, 3 days ago

▶ Painted a picture of Our school with 1 week ago, 3 days ago

▶ Looked at Uruguay - Wikipedia, the fr...pedia with 1 week, 2 days ago

# Journal: objects & actions

- Object view



| ☑ | ★ | What | Who | When |
|---|---|---|---|---|
| ☐ | ☆ | The Tortoise and the Hare | | 1 week, 5 days ago |
| ☐ | ☆ | Russian Tortoise - Wikipedi...pedia | | 1 week, 5 days ago |
| ☐ | ★ | A photo of my cat | | 1 week, 6 days ago |
| ☐ | ★ | This fantastic story abou...wrote | 6 | 2 weeks ago |
| ☐ | ☆ | image clipping | | 2 weeks ago |
| ☐ | ★ | Our school | | 2 weeks ago |
| ☐ | ★ | A movie of my family | | 2 weeks, 1 day ago |
| ☐ | ★ | Uruguay - Wikipedia, the fr...pedia | | 2 weeks, 3 days ago |
| ☐ | ☆ | History of Uruguay | | 2 weeks, 3 days ago |
| ☐ | ☆ | My homework assignment | | 3 weeks ago |

# Design goals

- Filesystem w/ POSIX semantics
  - This is the codeword for “standard filesystem”. Windows, UNIX, and MacOS in various flavors have (more-or-less) POSIX-compliant filesystems.
  - Our first generation design was a “simple” proprietary wrapper: let's move forward!
  - Aim to provide ***best possible*** support for legacy applications

# Design goals

- Content-addressable
  - Lots of attempts out there to create global distributed filesystems with unified namespaces – *let's not try this!*
  - Local arrangement & organization of documents is up to the individual user; all we need is an opaque tag to call it by.
  - Commercial support: XAM/Honeycomb (Sun)/Jackrabbit (Apache), etc, etc.

# Design goals

- Versioned
  - Support exploratory learning by always allowing user to undo his most recent mistake.
    - “Continuous” versioning.
    - Snapshots don't work for this.
  - Groups of files may have independently modifiable *tagged versions* (“full persistence”)
    - Gives us our P_SF_CORE/P_SF_RUN support
    - Also very useful when importing collaborative work

# *The* `olpcfs` *filesystem*

- Transparent **versioning**
  - Reach back and study the past – then change it!
- Rich **metadata** via POSIX xattrs
  - Enhanced by mechanisms to treat metadata as 1st-class files
- Integrated metadata **indexing**
  - Unifies “Journal” and “files & folders” views

# Demo

- http://wiki.laptop.org/go/Olpcfs has pointers to the source
- 2,500 lines of Python
  - Bdb and python-fuse packages
- First impressions (of FUSE):
  - I prefer managing directory objects, rather than being given full pathnames

# Journal integration

- All documents live in `~olpc/Documents`
- Sugar-aware activities add `action_id` xattrs for file grouping
- Add'l journal properties are directly implemented as xattrs

# Journal, cont.

- Journal search built on native indexing
- Journal versions built on native versions
  - But additional attributes may be used for richer merge semantics, etc.
  - “Keep stars” in Journal correspond to landmark versions

# Sync'ing & sharing

- All objects have “XUID”
  - Content-addressable
- Distributed indexes of various scopes on top of local index
- Not all local objects may appear in filesystem tree!
  - Some may be imported into index only

# Sync'ing & sharing

- XUID encapsulates *object plus metadata*
  - “Who's got this XUID?”
  - “I'll tell you which XUIDs I don't have if you'll tell me your XUIDs.”
- Independently-modified documents may result in tagged versions in filesystem after import

# Cool algorithms

How in the world do we implement this?

# Functional trees (aka “Dancing trees”)

- Persistent data structures
  - Not persistent: A.put(k, v) destroys old A
  - Fully persistent: A' = A.put(k, v)
  - *Partially* persistent: mutates A, but allows for queries on past state.
    - t is an monotonically increasing *timestamp*
    - t = A.put(k, v)
    - A.get(k, t)
  - Also, confluently persistent

# Functional trees, part 2

- More formal definitions:
  - J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. *Making data structures persistent.* Journal of Computer and System Sciences, 38(1):86–124, 1989.
  - A. Fiat and H. Kaplan. *Making data structures confluently persistent.* In Proc. 12th Ann. Symp. Discrete Algorithms, pages 537–546, Washington, DC, January 2001.

# Functional trees, part 3

- An example of a fully-persistent map
- Our filesystem is really just a collection of such maps
  - Path -> inode
  - Inode -> XUID & metadata
  - XUID -> contents
  - Indexing (xattr->XUID)

# Partially persistent B-trees

- Keys are (key, timestamp)
- Do range query for (key, ∞)
- Use special “deleted” value
- For more efficiency, tweak split policy
  - Keep “latest values” on one side of split
- Reference: Lanka & Mays 1991

# Flash filesystems

- Unlike filesystems for hard drives
- Fast random access for read
  - No seek time
- Writing is hard
  - Erase blocks vs. smaller write blocks
  - WLOG erase to FF, can only write 1->0
  - Need to erase entire erase block for 0->1

# How to maintain indices?

- “Log structured” (jffs2)
  - Just write new data at end w/ a (monotonic) timestamp
  - Scan all entries to find out true value
  - Generally keep cached results of scan in memory
- “Dancing tree” (logfs)
  - Use functional trees
  - Now I just have to keep track of the root!
- Periodic garbage collection needed

# New algorithms for Flash filesystems

- PMA data structures
  - Maintain a sorted array in memory
  - Write patterns a good match for flash
  - APMA: Bender/Hu 2007
- Cache-oblivious B-trees (Bender, Demaine, et al)
  - PMA needs index
  - van Emde Boas layout
  - O(ln N/B) vs O(ln N)

# If time permits

- Secure sharing
- Content matching

# Open problems

- Fully-persistent B-Trees
  - Do they exist?
    - Lanka & Mays paper is often incorrectly cited.

# Open Problems

- Set difference
  - Alice has a small set of documents, *A*
  - Bob has a much bigger set of documents, *B*
  - We want to determine *A-B*, so that we can efficiently back up Alice's stuff
  - Traditionally we tag Alice's documents as she backs them up
    - But what if Bob forgets?
  - Solution idea: use *treaps*

# Extra slides

Abandon hope all ye who pass here!

# Bitfrost

- P_SF_CORE: no system files may be modified
- P_SF_RUN: the “working copies” of the system can't be modified

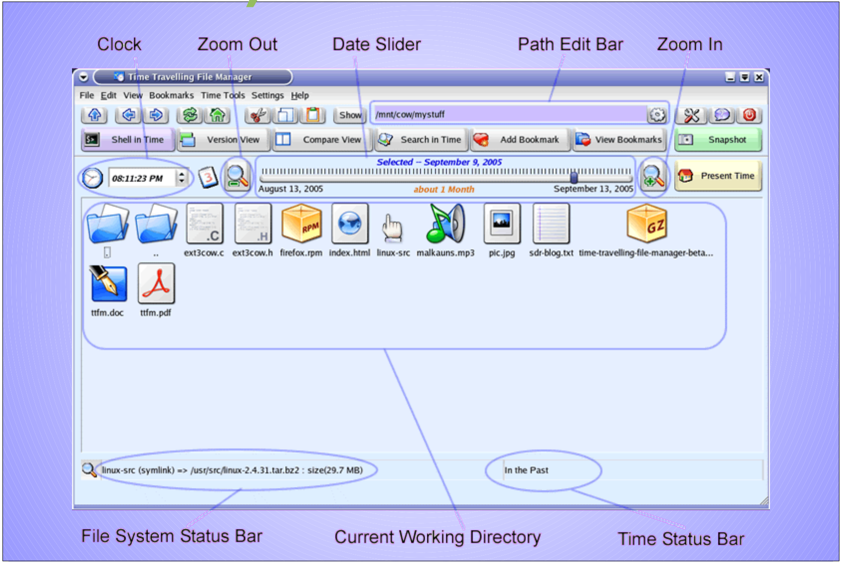
# Bitfrost: it gets interesting!

- When #P_SF_RUN is disengaged... [i]nstead of loading the stored files directly, a COW (copy on write) image is constructed from them, and system files from that image are initialized as the running system. ... These modifications persist between boots, but only apply to the COW copies: the underlying system files remain untouched.

# Bitfrost: turning P_SF_RUN back on.

- If #P_SF_RUN is re-engaged after being disabled, the boot-time loading of system files changes again; the system files are loaded into memory directly with no intermediate COW image, and marked read-only.
- End result:
  - Hacking is safe again!

# Time-travelling file manager (an aside)



linux-src (symlink) => /usr/src/linux-2.4.31.tar.bz2 : size(29.7 MB)

In the Past

# Implementation scope

- Lots of fallback/alternative implementations possible
  - Currently writing proof-of-concept to test APIs and unblock journal & other work
  - Non-versioned implementations with look-aside metadata easy using FUSE, lufs, 9P, etc.
- Flash filesystems are hard.
  - But the datastructures used here are very flash-friendly!

# Olpcfs directions

- Even if it's not as ambitious as this, our datastore should look like a filesystem!
- Lots to learn from BeOS & the BSDs; they rock!
  - Even NTFS

# Questions?

- OLPC mailing list: devel@laptop.org
- Or ask me: cscott@laptop.org