Scalable Full-Text Search for Petascale File Systems

Andrew W. Leung • Ethan L. Miller
University of California, Santa Cruz

3rd Petascale Data Storage Workshop (PDSW '08)
November 17th, 2008

Need scalable file management

- Today's file systems contain
  - Petabytes of data, billions of files, and thousands of users

- File systems have focused on scaling
  - I/O and metadata throughput, latency, fault-tolerance, cost
  - Limited work on scaling organization and retrieval

- File system organization largely unchanged for 30 years

- File organization and retrieval has ***not*** kept pace with file systems

Problems with current approach

- Files are organized into a single hierarchy
  - Possibly billions of files and directories
- Slow and inaccurate
  - Users must carefully organize and name files and directories
    - Tedious and time consuming
  - Users must manually navigate huge hierarchies
    - Wastes time and is inaccurate
  - Files only have a single classification
- Does not scale to petascale file systems

Scalable file retrieval with search

- File system search has been researched for decades
  - Focused on full-text (aka keyword) search

- Organizing and retrieving files with search
  - Files have many automatic classifications
    - Organization becomes much simpler
  - Files can be retrieved with any feature/keywords
    - No more slow namespace navigation
    - Reduces the chances of lost data

Petascale search challenges

- **Cost**
  - Very expensive - often requires dedicated hardware
- **Performance**
  - Tough to scale - often trade-off search and update performance
  - File system search should efficiently do both
- **Ranking**
  - Limited file ranking algorithms
- **Security**
  - Can significantly degrade search performance

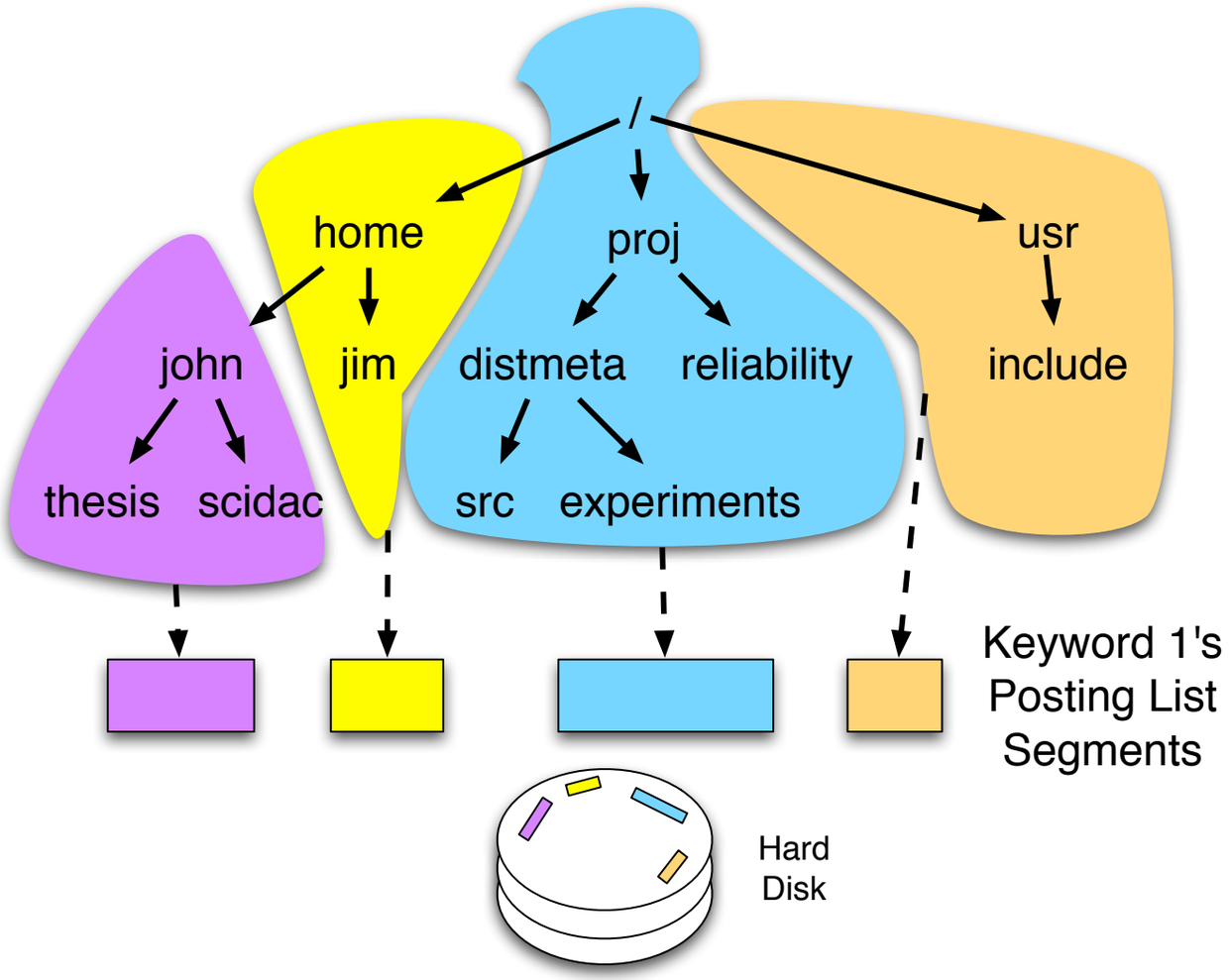
A specialized petascale search design

- Exploits file system properties
- Can be integrated within the file system

- Leverage namespace locality with *hierarchical partitioning* [Leung09]

- Namespace influences
  - File access patterns [Leung08, Vogel99]
  - File properties [Agrawal07, Leung09]
  - Who accesses them [Agrawal07, Leung08]

Index partitioning



- Traditional file system search uses an inverted index
  - Consists of a dictionary that points posting lists
- Our approach partitions the index based on the namespace
  - Posting lists are broken into *segments*

Benefits of our design

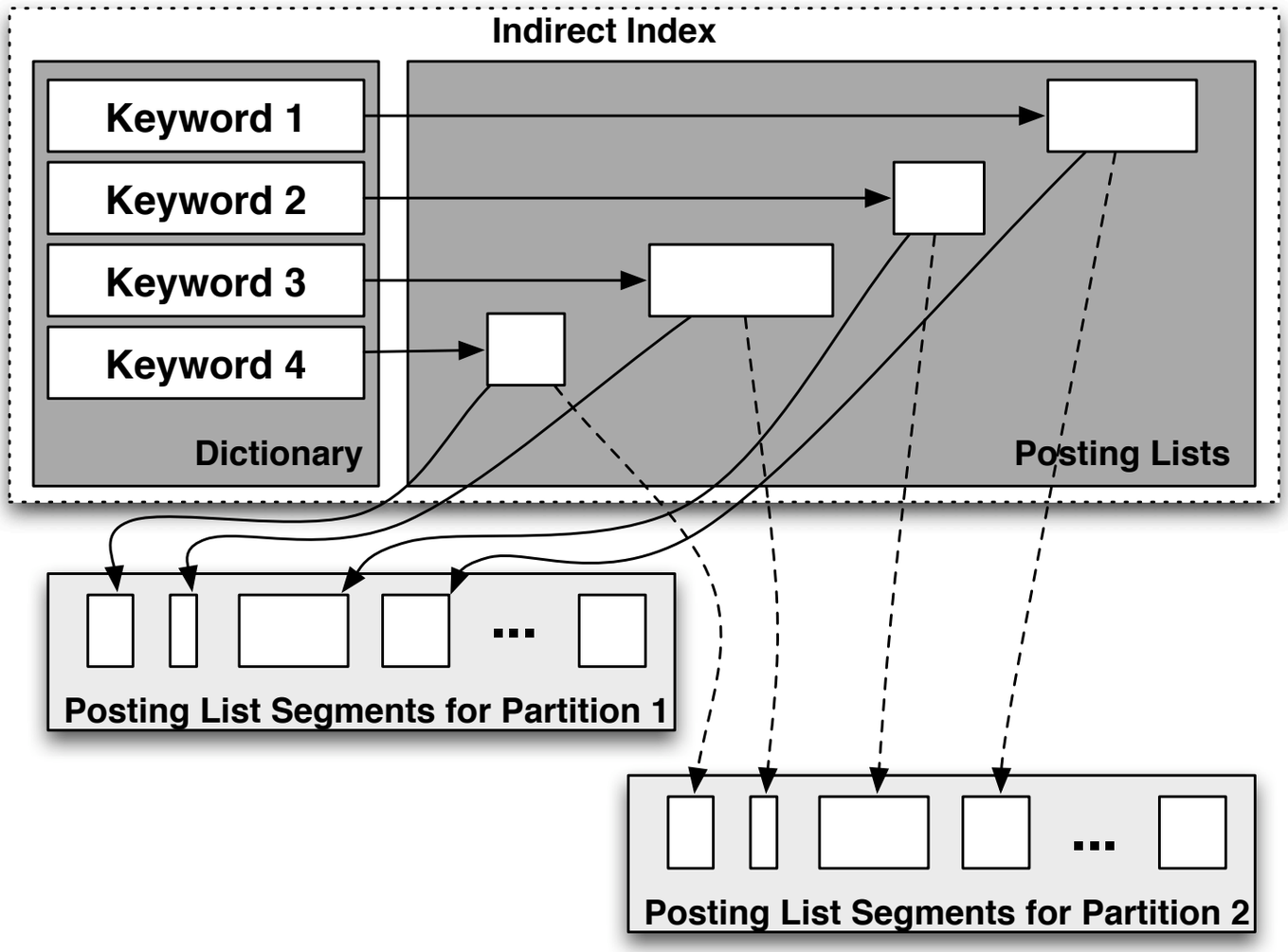
- Flexible, fine-grained index control
  - Search and update can be controlled at sub-tree granularity
  - Critical for index with billions of files

- Reducing the search space
  - Eliminate partitions that do not match search criteria
  - Allows users to control scope and performance of queries

- Efficient index updates
  - Smaller posting lists are easier to update and keep sequential on-disk

- Better resource utilization

The indirect index



- An inverted index that points to partition locations
  - Stores the dictionary
  - Posting lists store partition segment locations

Other possible extensions

- Security
  - Eliminate restricted sub-trees from search space
  - No extra space required and reduces permission check

- Ranking
  - Utilize namespace locality to improve search result ranking
  - Employ different ranking algorithms for different sub-trees

- Cost efficiency
  - Exploit Zipf-like sub-tree query patterns
  - Compress or migrate rarely searched sub-tree segments to lower-tier

Current and future work

- We are currently working on...

- Collecting and analyzing keyword data sets
  - Crawl real-world large-scale file systems
  - No current file system search keyword collections exist

- Completing the index and algorithm designs

- Implementation and evaluation within the Ceph petascale file system
  - Allows realistic integration and benchmarking

Thank you!

- Thanks to:
  - Minglong Shao, Timothy Bission, Shankar Pasupathy and NetApp's ATG
  - SSRC faculty and students

- Come see us at the poster session!
  - Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems

- Questions?