Spyglass: Fast, Scalable Metadata Search for Large-Scale Storage Systems

Andrew W Leung • Ethan L Miller
University of California, Santa Cruz
Minglong Shao • Timothy Bisson • Shankar Pasupathy
NetApp

7th USENIX Conference on File and Storage Technology
February 24-27, 2009

Baskin Engineering
UC SANTA CRUZ

The data management problem

- Large-scale storage systems are difficult to manage
  - Management questions become difficult or impossible to ask
  - Must sift through billions of files

- Impacts users and administrators
  - User - "*Where are my recently modified presentation files?*"
  - Admin - "*Which apps and users are consuming the most space?*"

- Challenge: need fast answers to difficult questions
- Example -- smart data restore
  - A buggy script deletes a number of users' files
  - "*Which files should be restored from backup?*"
    - Search current and previous versions
    - Locate the affected files to restore
    - Find the files to restore first

What can be done?

- Need to quickly gather and search info from the storage system
  - Ad hoc queries over file properties
  - I.e., metadata search capabilities
- Metadata includes file inode and extended attributes
  - Formatted as <*attribute*, *value*> pairs

- Metadata search becoming more common
  - Desktop - Spotlight, Beagle, Windows search
  - Small-scale enterprise - IndexEngines, Google Enterprise, FAST

How can this be done at very large-scales?

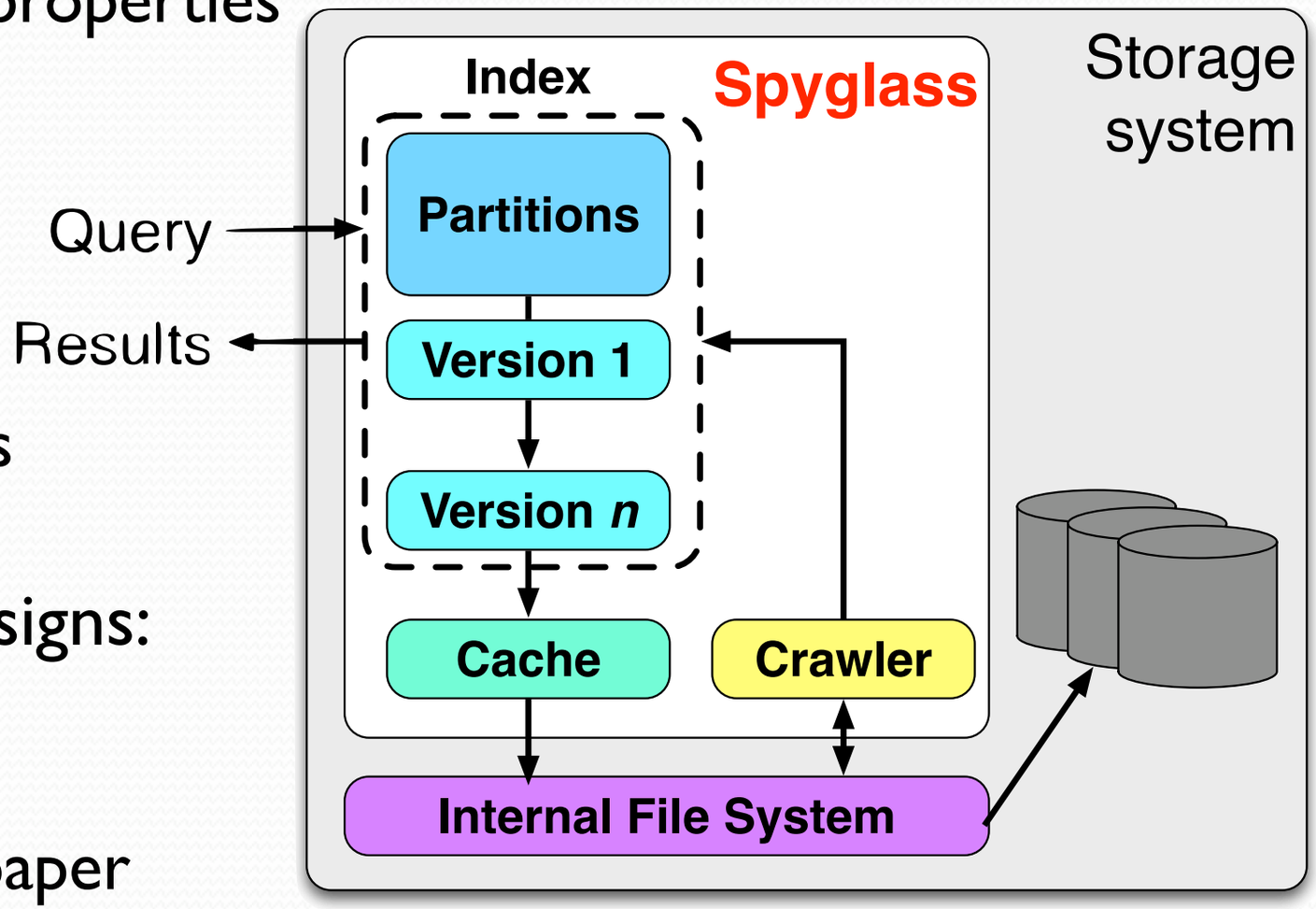
Large-scale search challenges

- Large-scale search challenges are not currently addressed
  - Requires a specialized solution

Challenge	Existing Solutions	Why Is This Difficult?
Cost & resources	Require dedicated hardware.	Can't afford dedicated hardware. Must share resources.
Metadata Collection	Slow to collect changes. Impacts the storage system.	Collect millions of changes.
Scale & performance	Use general-purpose DBMSs. Few storage optimizations.	Searching 10^{10} – 10^{11} pairs.

Spyglass overview

- How should a solution address these challenges?
  - Leverage metadata search properties
- Metadata query properties
  - Conducted user survey
- File metadata properties
  - Analyzed 3 storage systems
- Spyglass uses new index designs:
  - *Hierarchical partitioning*
  - *Signature files*
  - *Partition versioning* - in the paper
  - *Snapshot-based metadata crawling*



Metadata query properties

- Surveyed over 30 users and administrators
  - Asked "how would you use metadata search?"

Property	Description	Example	Solution
Multiple attributes	Includes multiple metadata attributes.	Where are my recently modified presentations?	Use all attributes in query execution.
Localized	Includes a directory pathname.	Which files in my home directory can be deleted?	Include namespace knowledge.
Back-in-time	Searches multiple metadata versions.	Which files have grown the most in the last week?	Version index changes.

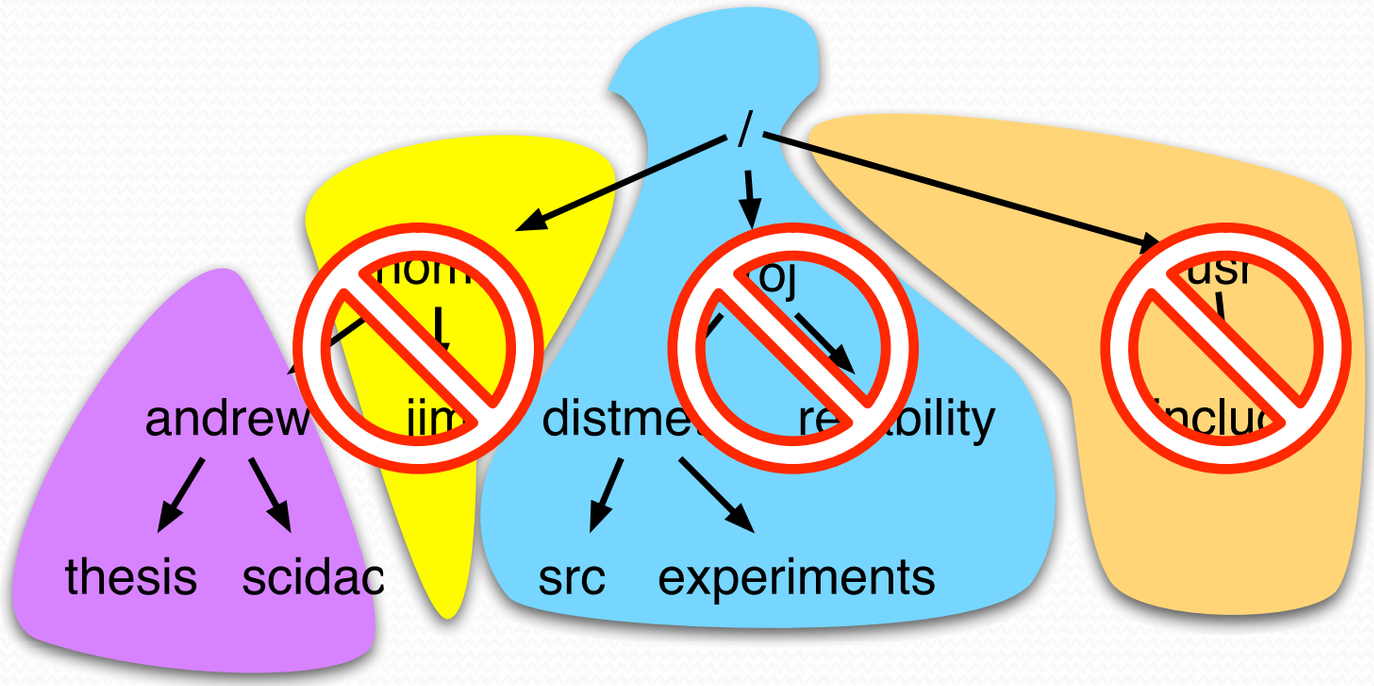
File metadata properties

- Analyzed 3 storage systems at NetApp
  - Web server -- 15 million files
  - Engineering server -- 60 million files
  - Home directory servers -- 300 million files

Property	Description	Example	Solution
Spatial locality	Values clustered in namespace.	Andrew's files mostly in /home/andrew	Allow index control using the namespace.
Skewed frequencies	A few values occur frequently. Intersections are more uniform.	Many PDFs or Andrew's files. Fewer Andrew's PDF files.	Query execution using intersections.

Hierarchical partitioning

- Partition the index using the namespace
- Parts of the namespace are indexed in separate partitions
  - Exploits spatial locality
  - Allows index control at the granularity of sub-trees
  - Uses a simple greedy algorithm
- Partitions are stored sequentially on disk
  - Fast access to each partition



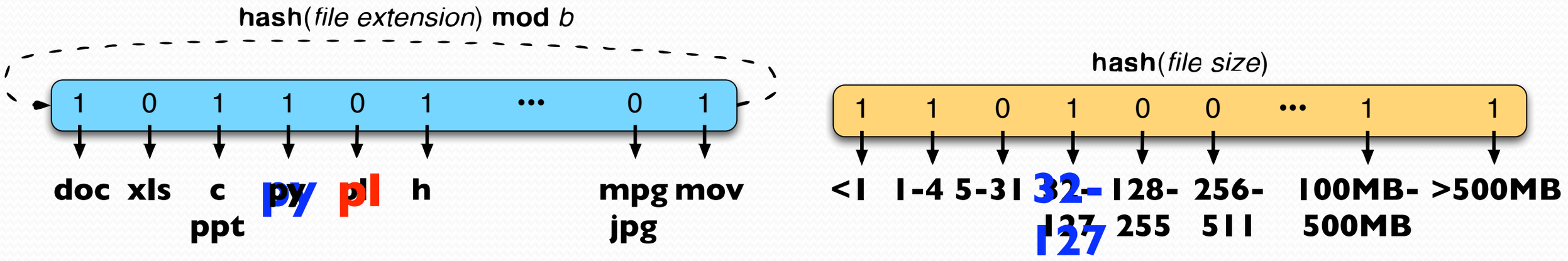
Query execution

- Performance is tied to the number of partitions searched
  - How do we determine which partitions to search?

- Users can localize their queries
  - Users control the size of the search space

- Signature files
  - Automatically determines which partitions to search
  - Exploits attribute value intersections
    - Searches only partitions containing all query values

Signature files



- Compactly summarizes a partition's contents
  - A signature file for each attribute in the partition
  - Small enough to fit in memory
  - Created as files are inserted
- Only search a partition if *all* tested bits are 1

- False positives can be reduced by
  - Increasing signature size
  - Changing the hashing function

Partition design

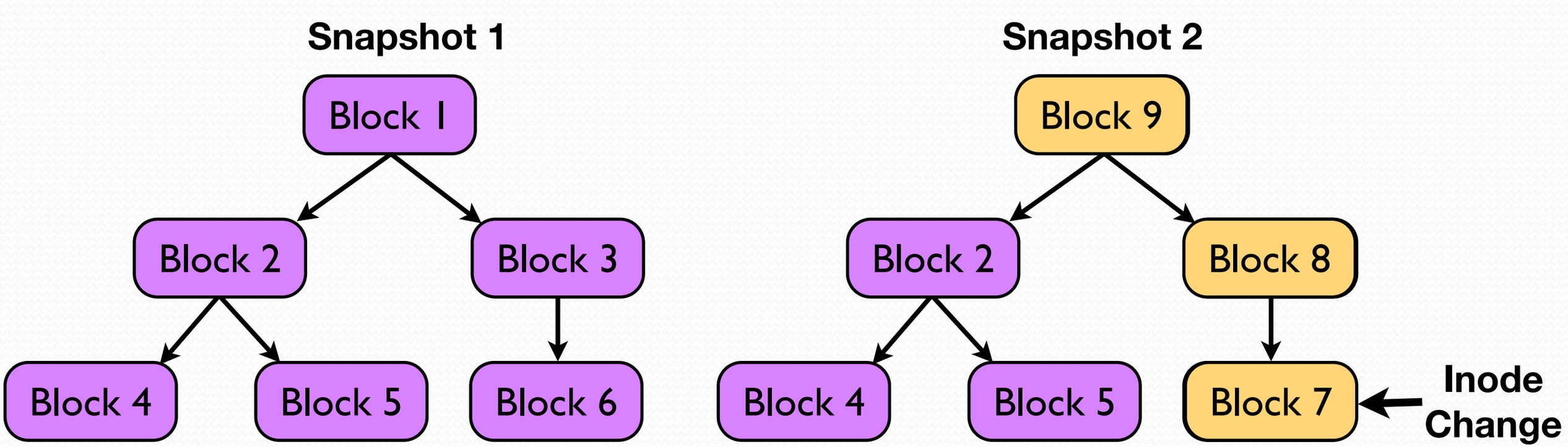
- Each partition stores metadata in a KD-tree
- Not explicitly tied to a particular index structure

- KD-trees
  - A multi-dimensional binary tree
  - Provides fast, multi-dimensional search
  - Allows a single index structure to be used

- Performance is bound by reading partitions from disk
- Partitions are managed by a caching sub-system
  - Uses LRU
  - Captures the *likely* Zipf-like query distributions
  - Ensures popular partitions are in-memory

Metadata collection

- Metadata collection must
  - Scale to millions of changes
  - Not degrade storage system performance
- Calculates the difference between to two snapshots
  - Leverages the *inode file* in WAFL snapshots
  - Only re-crawls metadata for changed files



Evaluation

- Evaluate performance, scalability, and efficiency
  - Using our real-world traces from NetApp

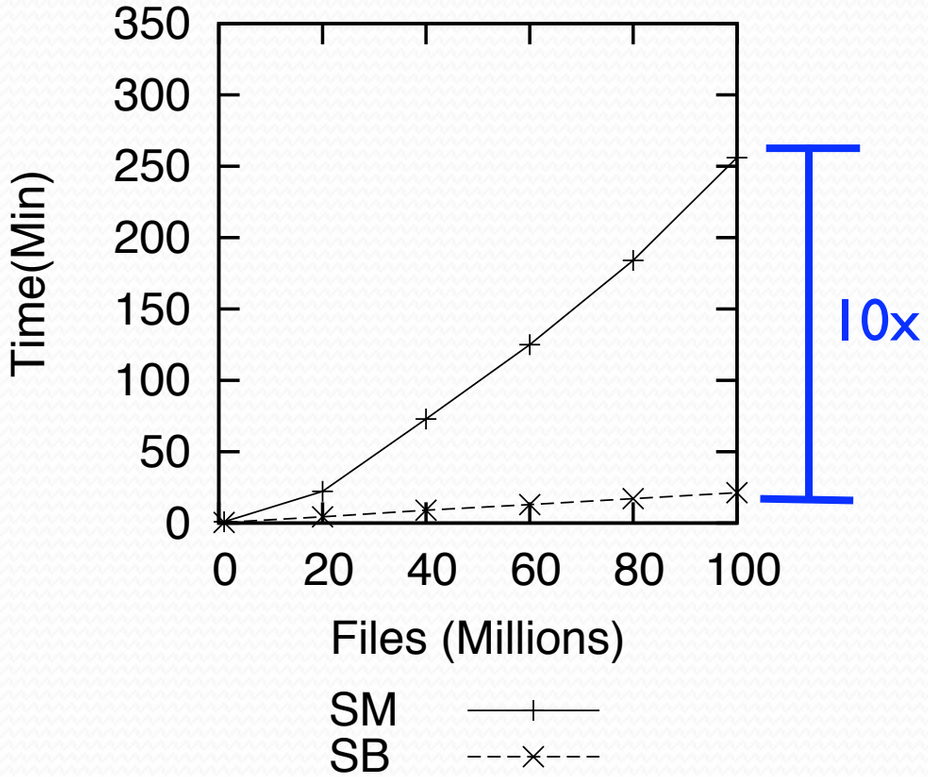
- In the talk we evaluate
  - Metadata collection
    - Compare performance to a simple straw-man
  - Search performance
    - Compare performance to PostgreSQL and MySQL

- In the paper we also evaluate
  - Update performance
  - Space requirements
  - Index locality
  - Versioning overhead

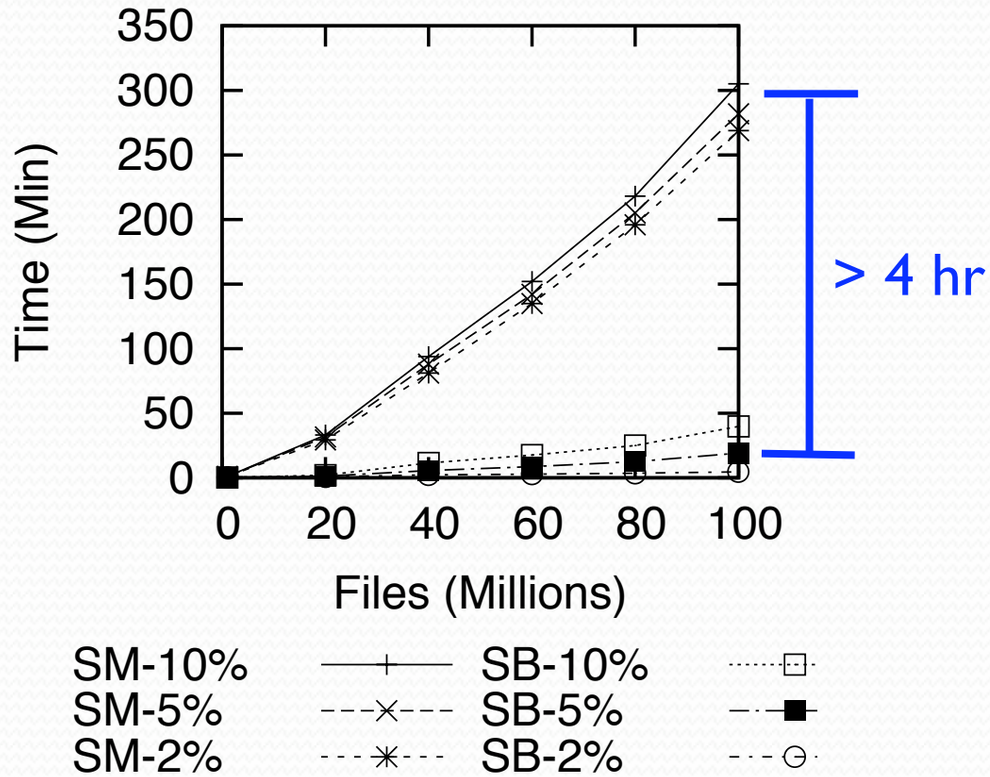
Collection performance

- Compare snapshot-based collection (SB)
  - To a parallel file system crawl (SM)

- **Baseline crawl**
  - Reads entire inode file
  - 10x faster that SM



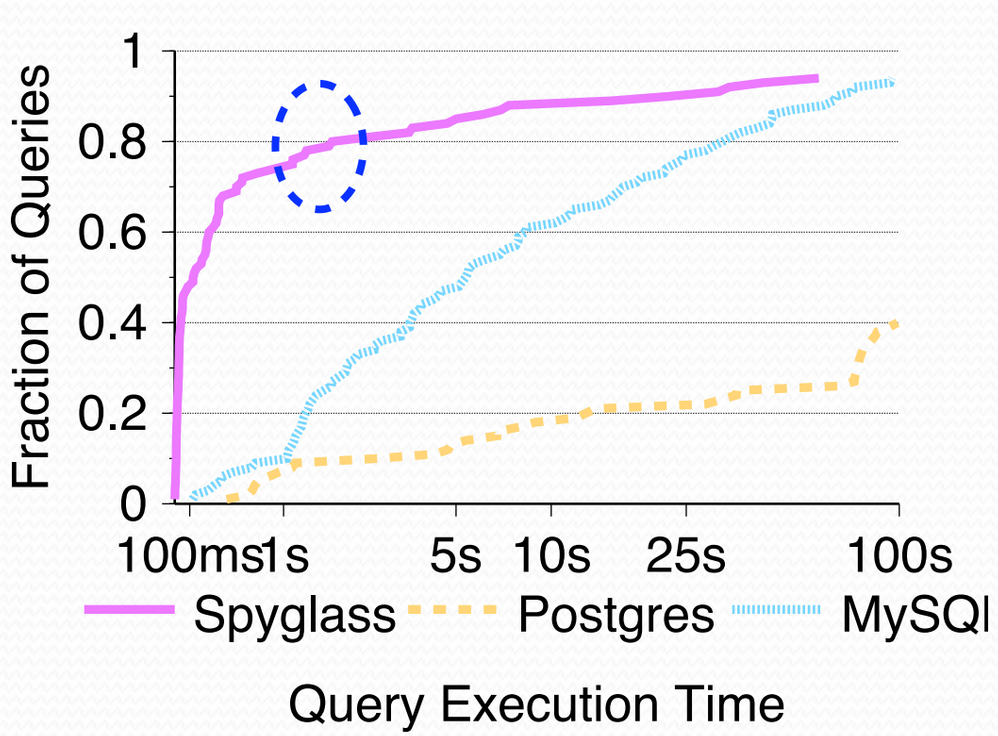
- **Incremental crawl**
  - 2%, 5%, and 10% change
  - Finishes in less than 45 min

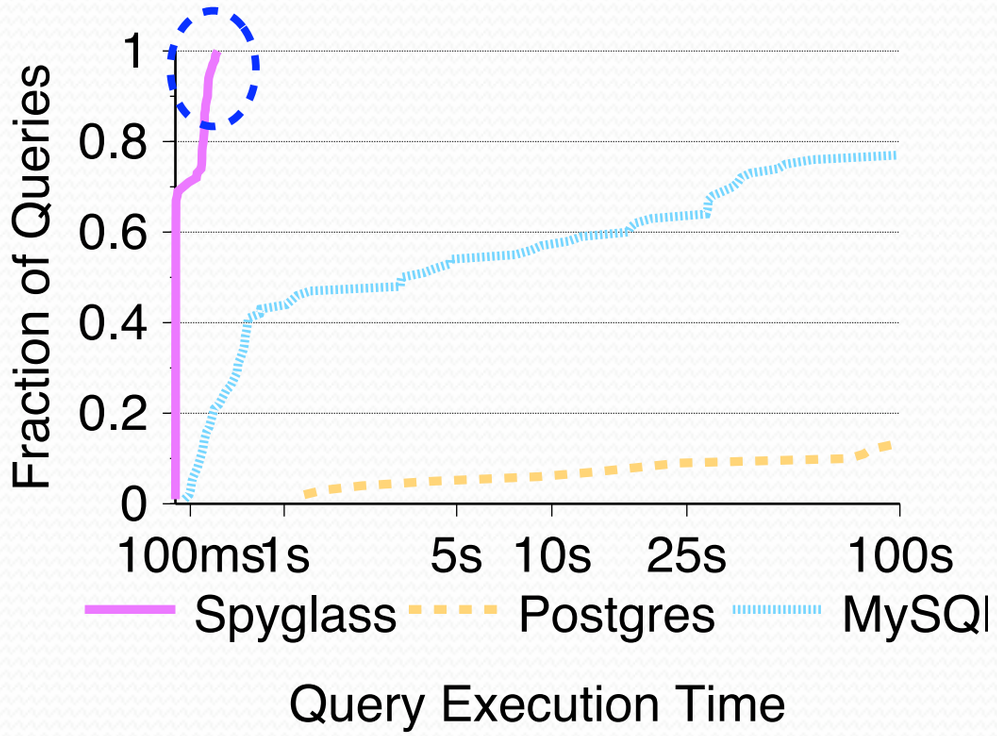


Search performance

- Evaluate 100 queries generated from 300 million file trace

Set	Search	Metadata Attributes
Set 1	Which user and application files consume the most space?	Sum sizes for files using owner and ext
Set 2	How much space in this part do files from query 1 consume?	Use query 1 with an additional directory path.





- Set 1: 75% queries finish in less than 1 second
  - Many partitions are eliminated from search
- Set 2: Localization significantly improves performance

Conclusions

- Metadata search can greatly improve how we manage data
- Large-scale storage systems present unique challenges
  - Cost & resources, metadata collection, performance & scalability

- There are opportunities to leverage query and file properties
  - Conducted a survey of real users and administrators
  - Analyzed real-world large-scale storage systems

- Spyglass is a new metadata search design
  - Hierarchical partitioning
  - Partition versioning
  - Snapshot-based metadata collection

Thank you! Questions?

NetApp™

Thanks to our sponsors:

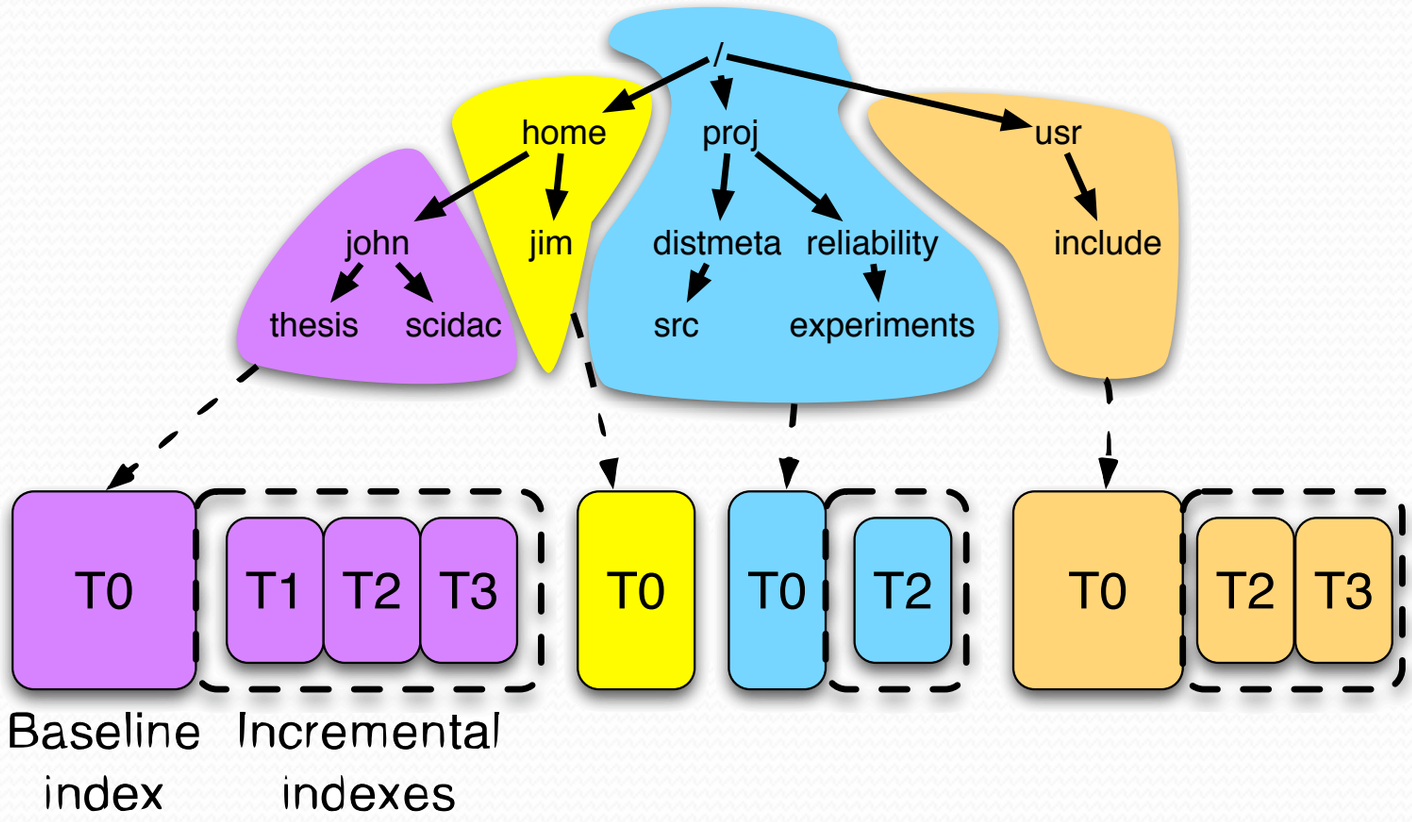
Partition versioning

- Index versioning provides
  - Back-in-time search capabilities
  - Fast, out-of-place index updates

- Each partition manages its own versions with a version vector
  - Exploits file update locality

- Each version is a batch of index updates
  - Represents the state of metadata at a given time
  - Absorbs frequent file re-modifications
  - However, creates a stale index

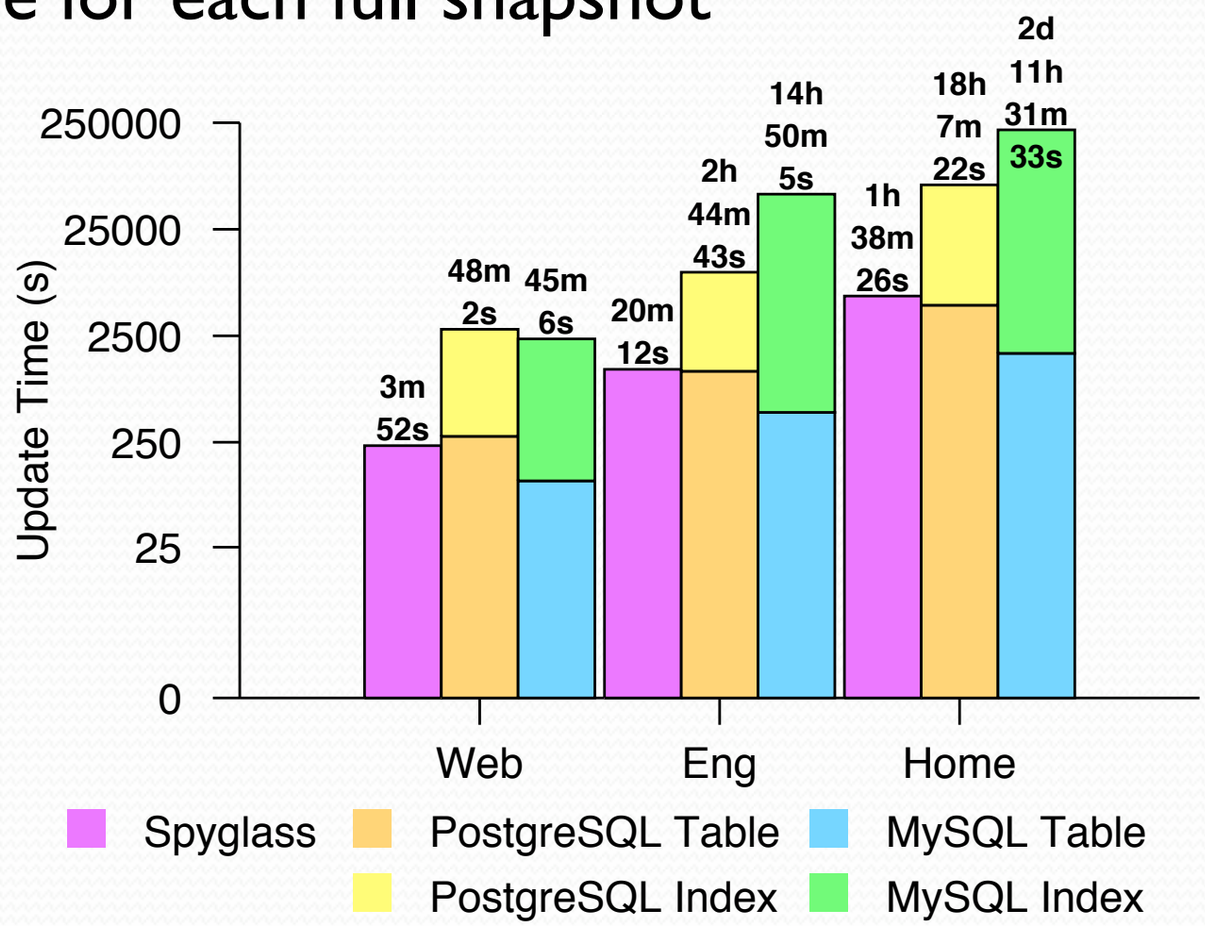
Versioning design



- Versions contain incremental metadata *changes*
  - Changes roll results forward
- Stored sequentially with the partition
  - Updates are fast - small sequential writes
  - Search overhead is low - Longer sequential reads

Update performance

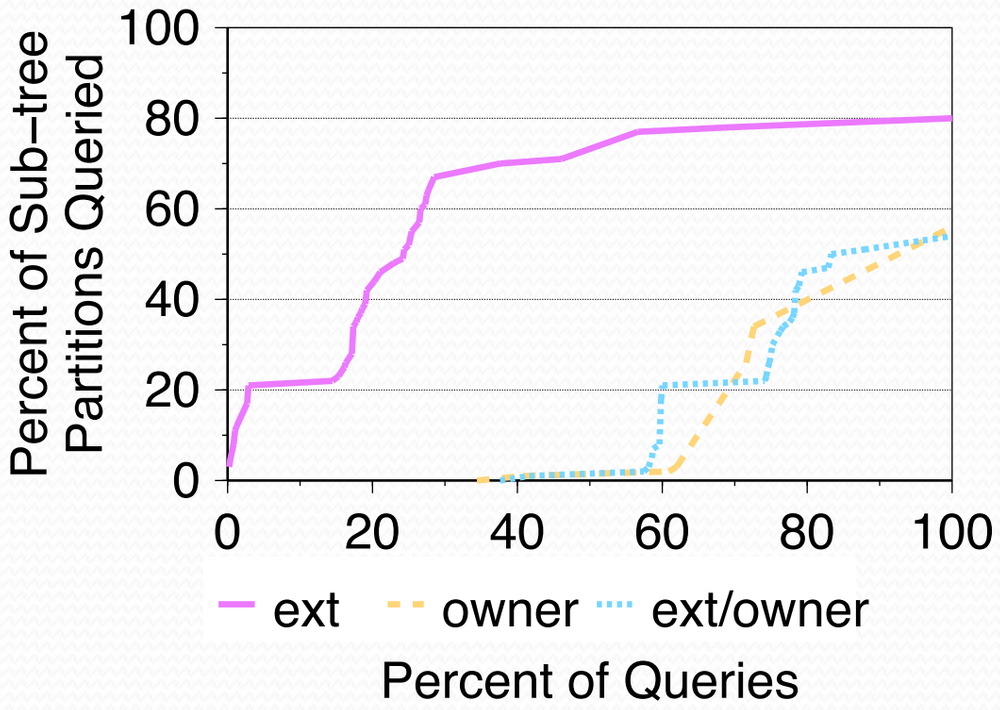
- Build baseline for each full snapshot

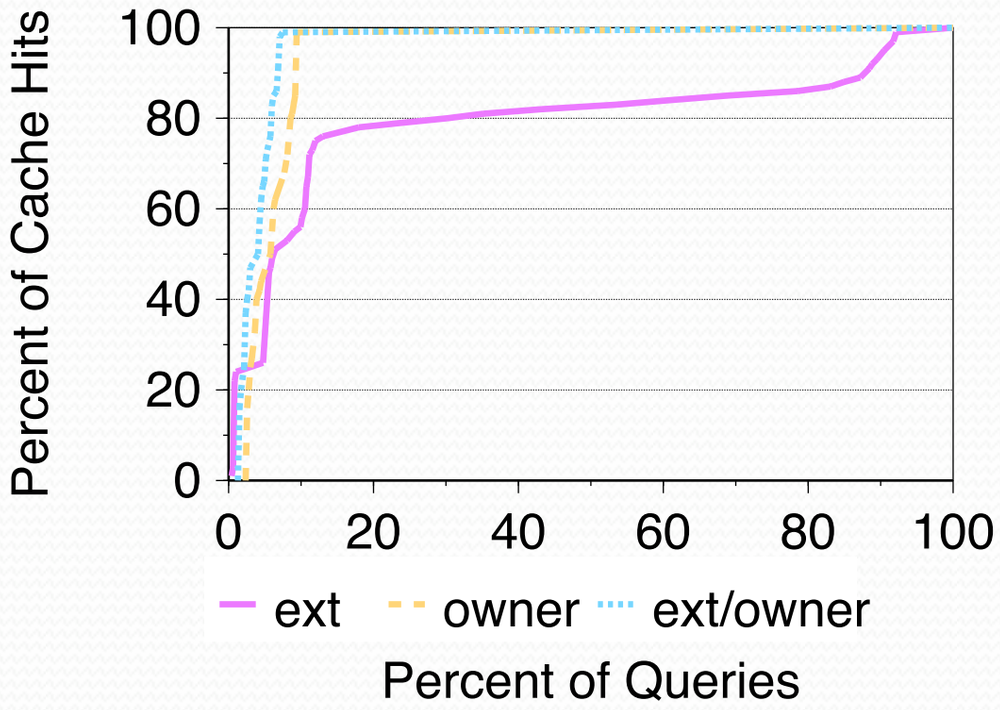


- Between 8x and 44x faster than DBMSs
  - DBMS load table and build indexes
- Scales linearly

Index locality

- Evaluate how partitions are queried and cached
- Generate queries based on attribute distributions

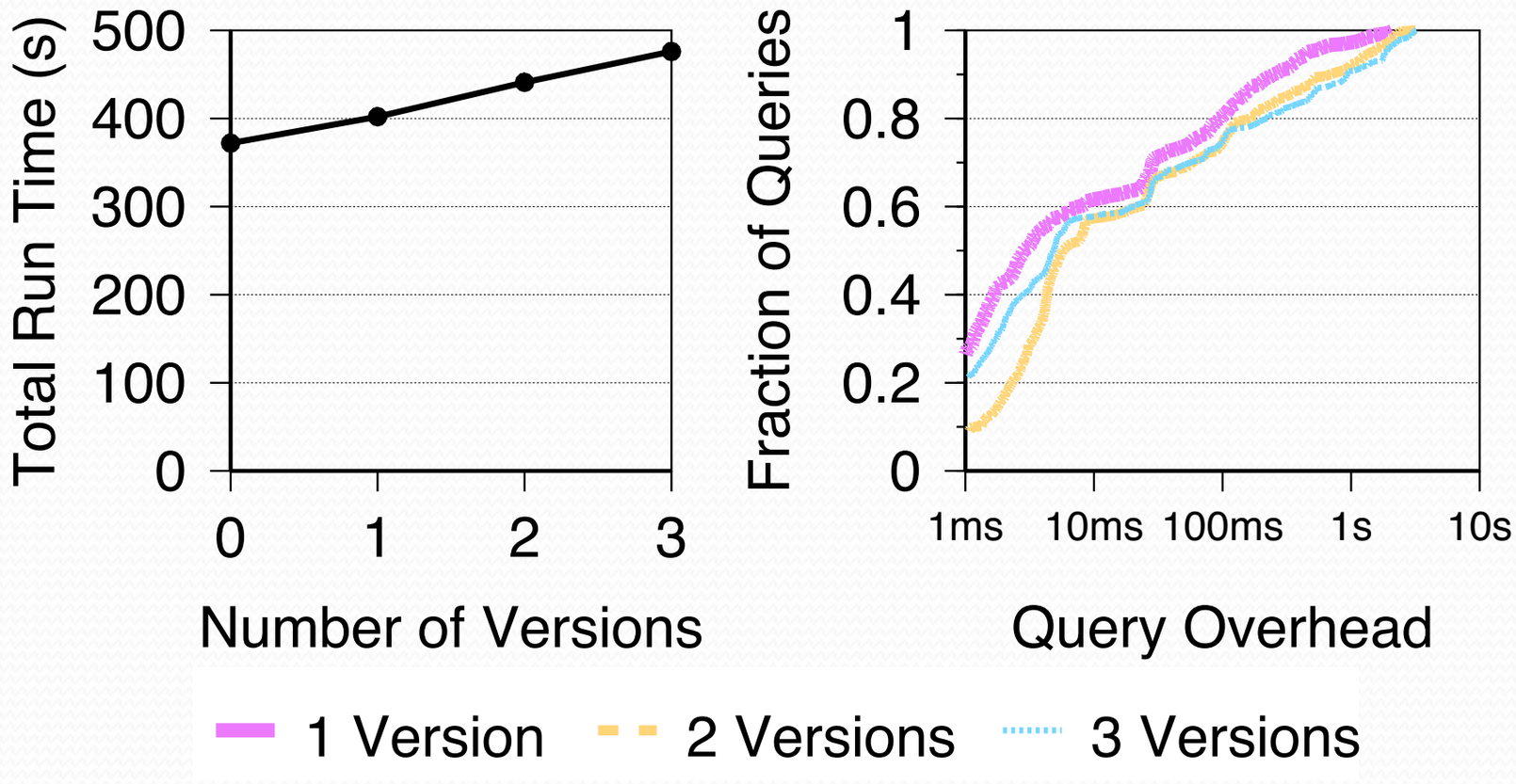




- Attribute intersections reduce the search space
  - 50% of queries access less than 2% of partitions
- Selective attributes improve cache hit ratio
  - 95% of queries have 95% cache hits

Versioning overhead

- Evaluate overhead of 1 to 3 days of changes



- Each versions adds 10% runtime overhead
  - Overhead is not evenly distributed
- 50% of queries have less than a 5 ms overhead
  - A few queries contribute most to overhead