Finding the Needles in the Haystack: Multidimensional Extensions to a Distributed Filesystem

Presenter: Milo Polte, CMU

Mentors: John Bent, Meghan Quist, James Nuñez

Talk Outline

- What is a multidimensional filesystem and why do we need them?
- Examples of multidimensional datasets
- Why filesystems or databases alone aren't a solution
- Description of our hybrid system
- Evaluation of Overhead
- Remaining Challenges of Implementation
- Road Map

Why do we need a multidimensional filesystem?

- **Our ability to capture and store data is outpacing our ability to organize and analyze it**
  - Data Volumes are doubling each year
  - Scientific instruments are gaining greater precision
  - Automation is creating vast stores of data
- **Traditional filesystems allow one to access files along a single dimension: That of the filename and path**
  - Filenames are frequently irrelevant; analysis needs to be applied to all data with a certain set of attributes not a certain name
- **A multidimensional filesystem is one which also indexes and allows efficient access to files based on their meta-data tags**
  - Gives a more expressive way to describe and find files

Motivation continued

- **Lets you find the files you need quickly**
  - Must scale even when the file system contains billions of files
- **Allows you to define your own application specific search tags for your application**
  - Not just file type, owner, name, etc.
- **Already exists for desktop systems**
  - Google Desktop for Windows
  - Spotlight File System for OS X
  - Etc.
- **Our work is adding this functionality to a fast, parallel file-system**
  - Important for scientific computing
  - Extremely large number of files

Example of Multidimensional Datasets 1: Sloan Digital Sky Survey / SkyServer

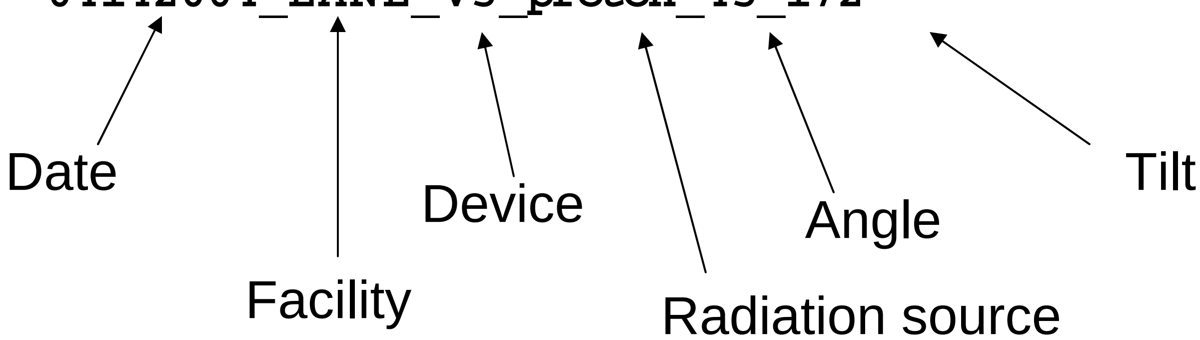
- Work between Jim Gray and the astronomy community.
  http://skyserver.sdss.org
- Information on roughly 230 million distinct photometric objects

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Example of Multidimensional Dataset 2: Effects of Radiation on Field Programmable Gate Arrays

- **Work by Heather M. Quinn and Sarah Michalak at LANL**
- **Studies effects of radiation on bit flip errors**
- **Groups samples into single files whose names are a concatenations of sample attributes:**

`04142004_LANL_V5_proton_45_172`



Sample Queries

- **Scientific:**
  - All satellite image files taken from a particular telescope and marked by an intelligent program as having a probability > 70% of being a Nebula.
  - All NMR results taken on the folding of a certain protein since Tuesday.
- **Administrative:**
  - Space saving: Show me the five largest files in the system that haven't be accessed in a month or more.
  - Security: Show me all system files whose content hash doesn't match a list of correct values.
  - Auditing: Show me all files accessed on Monday between 2 AM and 4AM

Why are filesystems approaches insufficient?

- **The filename tag approach**
  - Filenames become concatenation of tags:
    <date>_<instrument>_<run#>.dat
  - Currently used on the FPGA data
  - Search is slow - Running `find`
  - Adding a new tag to a single file will require renaming all indexed files

- **The directory tag approach**
  - Files are stored in hierarchical directories based on tags:
    /<date>/<instrument>/<run#>.dat
  - Similar to how users organize personal files
  - Search can be slow depending on ordering decisions
  - Adding a new tag requires shuffling the entire hierarchy
  - Duplication is a problem, leading to either 2 copies or incompletion
    - Does a picture of your niece and your dog go in /Pictures/Niece or /Pictures/Dog ?

Why not a pure database system?

- **Scientific applications are usually based on a POSIX API**
  - Many tools are scripts or compiled programs that might be difficult to modify to use a database
- **Users are accustomed to a POSIX API**
- **Databases are good at storing structured data, but most don’t store large unstructured data well**
- **Distributed Filesystems already used in large scale clusters (PVFS, PanFS, LUSTRE, etc)**
- **Note: Google’s Bigtable is a pure database on top of GoogleFS**
  - Serves only Google apps

Design of our prototype system

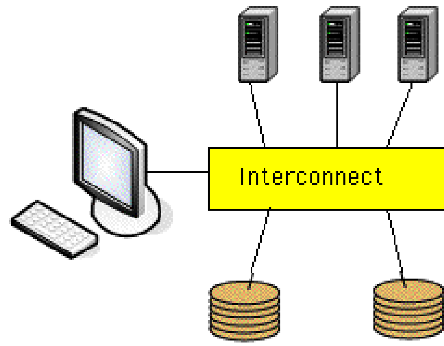
- **Built on top of the open source Parallel Virtual File System (PVFS) distributed filesystem**

<u>Traditional Filesystem:</u>

Single Machine:
Responsible for
Computation, metadata,
and data storage

<u>Distributed Filesystem (e.g. PVFS):</u>

Compute Node:
Mounts file system
Performs Computation



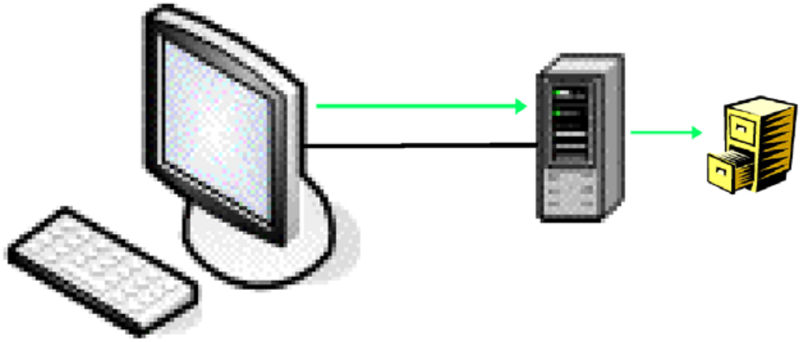
Metadata Servers:
Store and serve meta data:
Directory contents, permissions,
attributes, data locations, etc.

IO Servers:
Store and serve file data

- **PVFS has both standard attributes (owner, atime, etc.) and extended attributes (arbitrary key-val pairs)**
- **Integrates an sqlite3 SQL database on each of PVFS's meta-data servers. Sqlite3 databases used to index and query metadata. Called the 'Ledger'**
  - Embedded solution - low total cost of ownership
  - Indexes all 'normal' metadata (POSIX attributes, file sizes, etc.) stored at the MDS
  - Also allows application-specific metadata to be added as extended attributes for any file indexed by the MDS

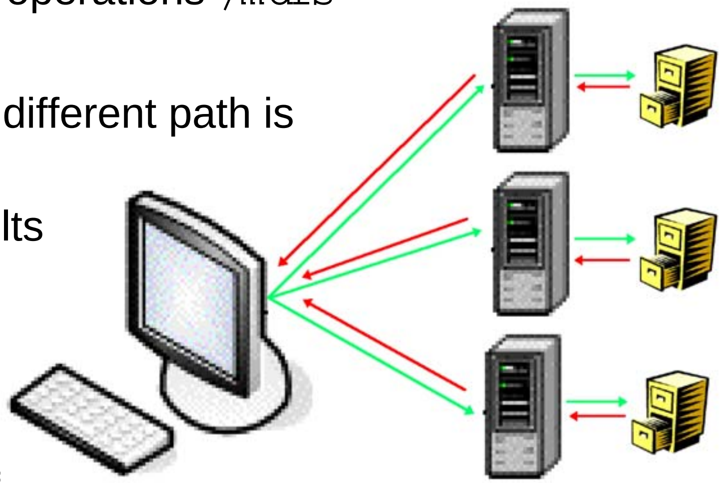
PVFS Integration - Replication of Attributes

- **Client behavior remains the same**
- **Server state machines set-attr.sm and set-eattr.sm (responsible for normal attributes and extended attributes) have new states to store information in the Ledger**
- **Attributes asynchronously written to SQLite DB**
- **EAttrs sufficient for testing**
  - Eventually separate application specific schema into distinct tables



PVFS Integration - Querying

- **Queries are SQL style query strings. Expressiveness limited only by application metadata tags**
- **Current design: New client program distinct from POSIX**
  - Issues query to each MDS in parallel
  - MDSes search their ledger and respond
  - Clients collate and report results
- **Eventually needs semantic integration with the file-system**
  - Plan: Special top-level directory for semantic operations '/mdfs'
  - mkdir /mnt/pvfs/mdfs/query/"<query string>"
  - If the the client detects this special directory, different path is taken than normal mkdir
  - Populate directory with symbolic links to results of queries



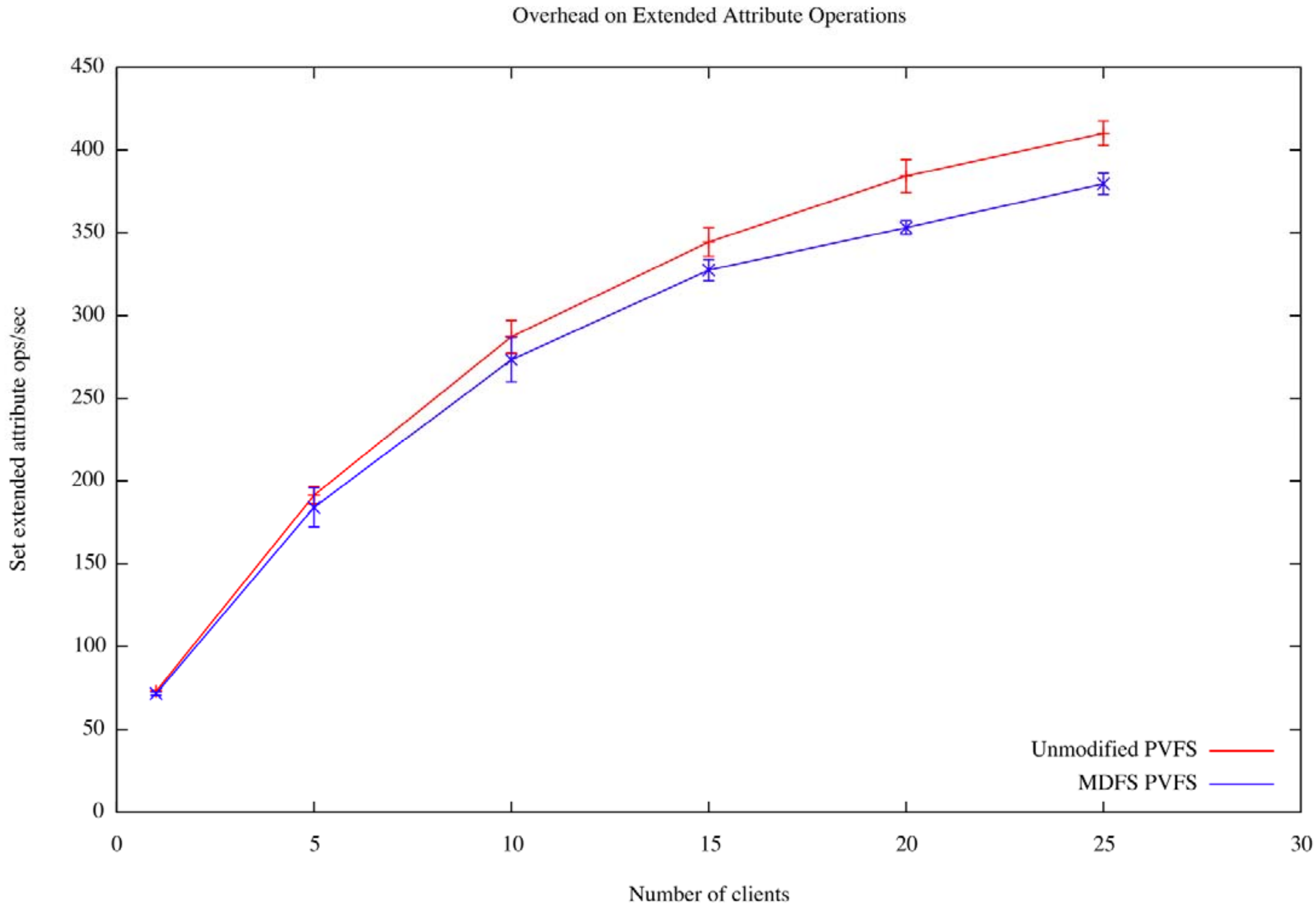
Evaluation of Overhead

- Measurement of overhead imposed on PVFS by the MDFS extension
- Measures the sustained throughput of typical filesystem operations (create, seteattr, etc.) on unmodified PVFS versus multidimensional PVFS
- Graph of aggregate operation throughput versus number of clients (keeping number of MDSes constant at 1)
- Want lines to be close and symmetrical (I.e. small overhead, normal behavior)

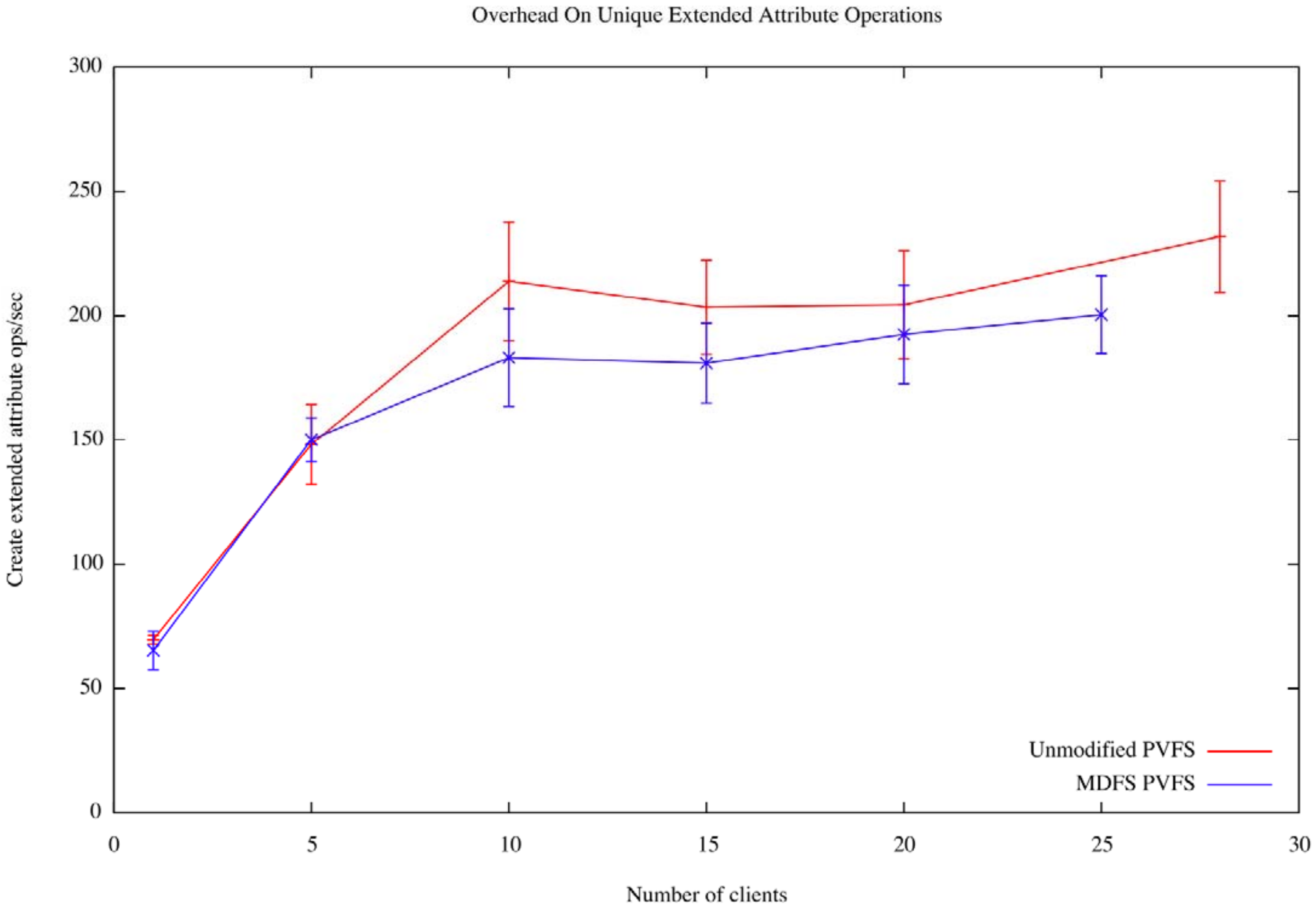
Experimental Methodology

- **Experiments run on lambda cluster**
  - Dual Pentium III nodes connected by Gig-E ethernet
- **For each experiment single server was used for both MDS and IO/Server**
  - Performance should scale as the number of MDSes increases
  - Will test, but not today
- **On each trial a clean file system was created with one directory per client and one file per client**
- **Client machines chosen at random for every trial**
- **Tests were one hundred operations per client**
  - Throughput calculated as Num_Clients * 100 / Time
- **Ten trials used to generate each data point**

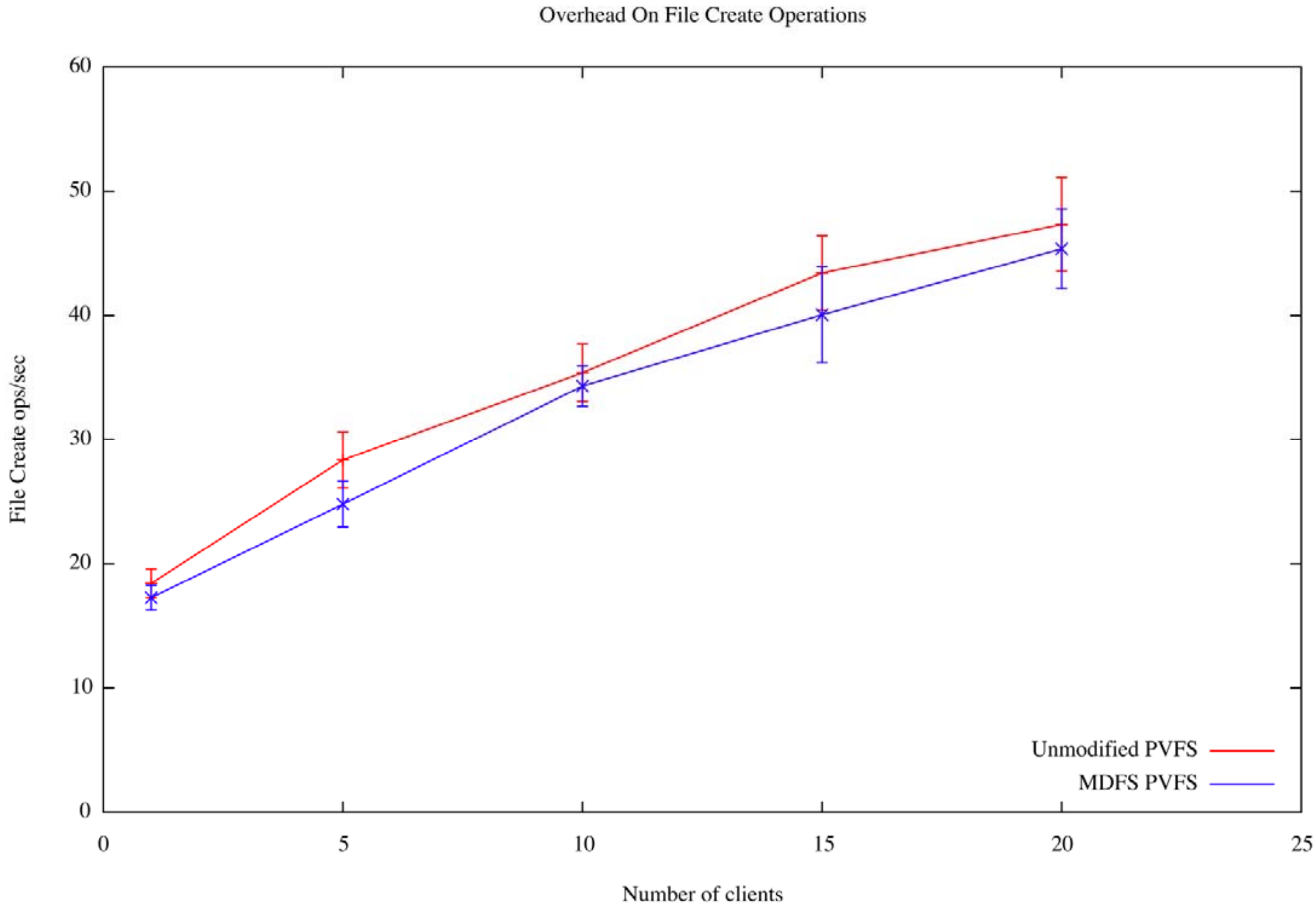
Updating extended attributes



Creating extended attributes



Creating Files



Further Evaluation Plan: Performance of Queries

- **Compare performance of MDFS to traditional approaches (I.e. `find`, `locate`, `ls`) on a variety of queries using the FPGA dataset.**
- **The overhead time to extract the tags and add them to the database will be included in the measurement of MDFS and locate performance**
- **Graph of query time versus size of dataset, on various structures (sparse queries, bushy directories, etc.)**
- **Graph of query time versus number of MDSes on various structures**
- **Expect performance of `find` and `ls` to fall off much more rapidly than the MDFS interface**

PVFS Integration Challenges

- **File names**
  - Returns from queries are file handles
  - Want names/paths, dynamically created symbolic links
  - Keep name and parent of each object in its extended attributes and reconstruct?

- **Size attribute**
  - Not stored on MDSes. Requires querying of data servers
  - Who does it?
    - Clients? (Normally not responsible for coherence)
    - MDS? (May not see data operations)
  - Related to transducer challenges

- **More PVFS-like behavior**
  - Extensibility: Needs abstraction layer for different ledger implementations
  - `fsck` extension

Roadmap

- **Milestone 0: Overhead**
  - Extend overhead evaluation out to 50+ nodes
- **Milestone 1: Querying system**
  - Finish querying system
  - Semantic behavior
  - Symbolic link creation
- **Milestone 2: Evaluation of queries**
  - FPGA dataset
  - Other datasets
    - NIST Face/Iris Recognition datasets?
    - Interested in your ideas!
- **Milestone 3: Transducers**
  - Transducers for things like content hashes
  - Trade off between time-to-coherence and performance