Moving from Extreme Scale Data to Extreme Scale Metadata Concerns: It’s About Time!

Gary Grider
HPC Division Leader, LANL/US DOE
2018

Subset of LA-UR-18-24612
What is Los Alamos
Eight Decades of Production Weapons Computing to Keep the Nation Safe

Maniac  IBM Stretch  CDC  Cray 1  Cray X/Y  CM-2
CM-5  SGI Blue Mountain  DEC/HP Q  IBM Cell Roadrunner  Cray XE Cielo
Cray Intel KNL Trinity  Ising DWave  Cross Roads

Los Alamos
National Laboratory
EST. 1943

CROSSROADS
An APEX Collaboration
Mr. Seymour Cray  
Cray Research, Inc.  
P. O. Box 169  
Chippewa Falls, WI 54729  

Dear Mr. Cray:

This is to advise you that the Los Alamos Scientific Laboratory of the University of California is interested in acquiring the first Cray-1 computer, scheduled for delivery in November 1975, to handle calculational requirements beyond the capability of our presently-installed computers.
Some Storage Products You May Not Realize Were Funded/Heavily Influenced by DOE/LANL

- Panasas
- Lustre
- Data Warp
- IBM GPFS
- Unistress
- Ceph
- DataTree
- CFS
- HPSS
- DDN Storage
- Infinite Memory Engine
- Takutek

IBM Photo-store
An example of metadata scaling: MarFS Scaling

Scaling test on our retired Cielo machine:
835M File Inserts/sec Stat single file < 1 millisecond
> 1 trillion files in the same directory

Striping across 1 to X Object Repos
Hopefully we have whipped the scalable parallel data into submission on to Metadata pursuits

**DeltaFS**
A File System Service for Simulation Science
Best Paper SC18

**HXHIM**
Indexing for Scientific Data

**GUFI**
Fast Userspace Metadata Query
R&D100 Award Disruptor
A dynamically loadable namespace – DeltaFS
Lets make metadata scale with the application!

DeltaFS
A File System Service for Simulation Science

HXHIM
Indexing for Scientific Data

GUFI
Fast Userspace Metadata Query
Brief VPIC Overview

- Particle-in-cell MPI code (scales to ~100K processes)
  - Fixed mesh range assigned to each process
  - 32 – 64 Byte particles
  - Particles move frequently between 10’s of thousands of processes
  - Million particles per node (Trillion particle in target simulation)
  - Interesting particles identified at simulation end

![Diagram of particle-in-cell simulation process](image)
Brief DeltaFS Overview

Every process:
• Runs a linkable KVS in the app that looks like a file system (IndexFS) (LevelDB)
• “Checks Out” its namespace for the particles files it will hold – loads a LevelDB SSTable with hundreds of thousands of “particle files” time stamp records.
• When Storing article records are sent to the appropriate “file”
• This is writing data to a 10’s of thousands distributed KVS
Tracking the Highest Energy Particles
Recall the intent is 1 Trillion particles
These thousand particles are interesting, where have they been?

VPIC Particle Dump Size

Collaboration of CMU, LANL, ANL, HDF Group
(papers at PDSW 15, PDSW 17, SC18)

Application thought it was writing/reading from 1 file per trillion particles
but really was writing records to massive parallel distributed KVS!
Today we are getting like 8 Billion Particle File Ops/Sec. (yes Billion)
Isn’t 8 Billion Metadata ops/sec good enough? Well maybe, but that was low dimensional Metadata. What about higher dimensional Metadata?

Now that I know “where the interesting particles were” what was going on around those interesting particles?

MDHIM -> XDHIM (Thank you to DoD and DoE ECP funding)

DeltaFS
A File System Service for Simulation Science

GUFI
Fast Userspace Metadata Query

HXHIM
Indexing for Scientific Data
An application linkable parallel KVS Framework
KVS is linked in the application, bulk put/get ops, uses key range sharding and server side stored procedures, X Dimensional Sharded Index (Hexastore 6 dimensional linkable KVS is currently in use)
• How do you store/represent an AMR mesh?
• (What is AMR and Why Do We Care?)
  - In memory, dynamic tree and nested list structures are common

How many rows are in each of these columns?
(For that matter, how many columns are in each of these columns?!)

How do you store this kind of time series data in a usable form?
• Key-value exposes the data structures underlying most FS
• Key-value allows fine-grained data annotation
• Need to add some HPC research to make efficient for HPC platforms
  - Mercury RPC and Margo (lightweight IO threads) for platform services
  - Multidimensional Hashing Indexing Middleware
HXHRM Mesh Storage Example

- If “position” in the mesh is the key, and you keep subdividing the key, how do you have a reasonable key structure
- Old trick using hierarchy of keys (borrow from Farsite FS - Microsoft)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh</td>
<td>name</td>
<td>“My Mesh”</td>
</tr>
<tr>
<td>sim</td>
<td>timestep</td>
<td>3.0</td>
</tr>
<tr>
<td>c0</td>
<td>position</td>
<td>[0.0,0.0]</td>
</tr>
<tr>
<td>c1</td>
<td>position</td>
<td>[0.1,0.0]</td>
</tr>
<tr>
<td>c2</td>
<td>position</td>
<td>[0.0,0.1]</td>
</tr>
<tr>
<td>c3.0</td>
<td>position</td>
<td>[0.1,0.1]</td>
</tr>
<tr>
<td>c3.1.0</td>
<td>position</td>
<td>[0.15,0.1]</td>
</tr>
<tr>
<td>c3.1.1</td>
<td>position</td>
<td>[0.175,0.1]</td>
</tr>
<tr>
<td>c3.1.2</td>
<td>position</td>
<td>[0.125,0.15]</td>
</tr>
<tr>
<td>c3.1.3</td>
<td>position</td>
<td>[0.125,0.125]</td>
</tr>
<tr>
<td>c3.2</td>
<td>position</td>
<td>[0.1,0.15]</td>
</tr>
</tbody>
</table>
Sample Query: Tracking a Wave thru Time

- A fast multi-dimensional index
  - Time is discretized separately (indexing not required)
  - Energy and position must both be indexed (and not trivially)
    - Energy extrema search is worse than VPIC example!
- Efficient filtering for contiguity!
  - We could probably work around most of these problems, but level arrays will always convert spatially contiguous workloads into disjoint query sets
  - Neighbor lists won’t limit the pointer chasing
- Why do I think a Key-Value organization can do better?
Range-based Iteration with Stored Procedures

• **Advantages of Key-Value Organization**
  - Decouples file size, I/O size from data set size (efficient I/O)
  - Keyspace *dimension* can change dynamically
    - Leverage naming technique described by Farsite FS
  - Supports iteration across multiple dimensions simultaneously
  - In-situ rather than post-hoc

• **Advantages of client-server architectures**
  - Even with the above we can’t accomplish what we need
  - Stored procedures to identify extrema in-situ
How do we ever find anything in our trillions of files?

GUFI Grand Unified File Index

DeltaFS
A File System Service for Simulation Science

HXHIM
Indexing for Scientific Data

GUFI
Fast Userspace Metadata Query
Motivation

• Many layers of storage at LANL
  • By design – users would have us only buying storage if we used HSMs
• Data management by users is driven by need, sporadically
  • Users go find unneeded data and delete, if prodded
  • Users have no easy way to find particular datasets unless they have a good hierarchy or they remember where they put it
  • Users have bad memories and bad hierarchies...(you can see where this leads)
• ...lower (longer) tiers of storage systems accumulate cruft over time
GUFI Goals

- **Unified** index over home, project, scratch, campaign, and archive
- **Metadata only** with extended attribute support
- Shared index for **users** and admins
- **Parallel search** capabilities that are very fast (minutes for billions of files/dirs)
- Search results can appear as a **mounted File System**
- Full/Incremental update from sources with reasonable update time/annoyance
- Leverage **existing tech** as much as possible both hdwr and software: flash, threads, clusters, sql as part of the interface, commercial db tech, commercial indexing systems, commercial file system tech, threading/parallel process/node run times, src file system full/incremental capture capabilities, posix tree attributes (permissions, hierarchy representation, etc.), open source/agnostic to leveraged parts where possible.
- **Simple** so that an admin can easily understand/enhance/troubleshoot

- **Why not a flat namespace?**
  - Performance is great, but...Rename high in the tree is terribly costly
  - Security becomes a nightmare if users/admins can access the namespace
GUFI Prototype

SystemA-namespaceA
/search/scratch2/ProjectA

- DirA
  - db.db
  - entries
  - dir summary
  - tree summary

- DirB
  - db.db
  - entries
  - dir summary
  - tree summary

SystemA-namespaceB
/search/scratch2/ProjectB

- DirA
  - db.db
  - entries
  - dir summary
  - tree summary

- DirB
  - db.db
  - entries
  - dir summary
  - tree summary

SystemB-namespaceA
/search/campaign/ProjectB

- DirA
  - db.db
  - entries
  - dir summary
  - tree summary

- DirB
  - db.db
  - entries
  - dir summary
  - tree summary

- DirC
  - db.db
  - entries
  - dir summary
  - tree summary

- Tree-Summary
  - DB with summary of the tree below optional and can be placed anywhere in the tree

- Entries
  - DB with name/stat/linkname/xattr info for each file or link

- Dir-Summary
  - DB with summary of this directory

Process/Node Parallelism for different parts of the tree, within each system-namespace combination use thread based parallelism

Los Alamos National Laboratory
EST. 1943
Programs Included / In Progress

- DFW – depth first walker, prints pinode, inode, path, attrs, xattrs
- BFW – breadth first walker, prints pinode, inode, path, attrs, xattrs
- BFWI – breadth first walker to create GUFI index tree from source tree
- BFMI – walk Robinhood MySQL and list tree and/or create GUFI index tree
- BFTI – breadth first walker that summarizes a GUFI tree from a source path down, can create treesummary index of that info
- BFQ – breadth first walker query that queries GUFI index tree
  - Specify SQL for treesummary, directorysummary, and entries DBs
- BFFUSE – FUSE interface to run POSIX md tools on a GUFI search result
- Querydb – dumps treesummary, directorysummary, and optional entry databases given a directory in GUFI as input
- Programs to update, incremental update (in progress):
  - Lustre, GPFS, HPSS
Early performance indicators

- All tests performed on a 2014 Macbook (quad core + SSD)
- No tree indexes used
- ~136k directories, mostly small directories, 10 1M entry dirs, 20 100K size dirs, and 10 20M size dirs
- ~250M files total represented
- Search of all files: 2m10s (~1.75M files/sec)
- Search of all files and dirs: 2m19s (~1.63 M entries/sec)
- Search of all files and dirs, but exclude some very large dirs: 1m18s
- Search of all files and dirs, but exclude all < 1000 file directories: 1m59s
- ...on a laptop!
Open Source
BSD License
Partners Welcome

https://github.com/mar-file-system/marfs
https://github.com/pftool/pftool
https://github.com/mar-file-system/GUFI
https://github.com/mar-file-system/erasureUtils

Thanks to all that participated in this work

Thanks For Your Attention