BeeGFS®

Large Filesystems Indexing & iRODS

Peter Braam
2021-06-09

HPC wire
2020 Readers' Choice Awards
Top 5 New Products or Technologies to Watch
BeeGFS
Award winners for 3 years!

Background - 1

I act as the CTO for ThinkParQ which develops BeeGFS and HPC Parallel File System.

Some of our larger users want to use iRODS, but have large, very fast file systems.

- scans take weeks
- BeeGFS has no persistent changelog (instead a socket with “events”)

What should we do?

Many related questions: HSM (without suffering), storage pools (without asking me to change the file system code too much)...

The problem is not new

Background - 2

Around 2015 a subject called “**Campaign Storage**” was explored, led by Los Alamos.

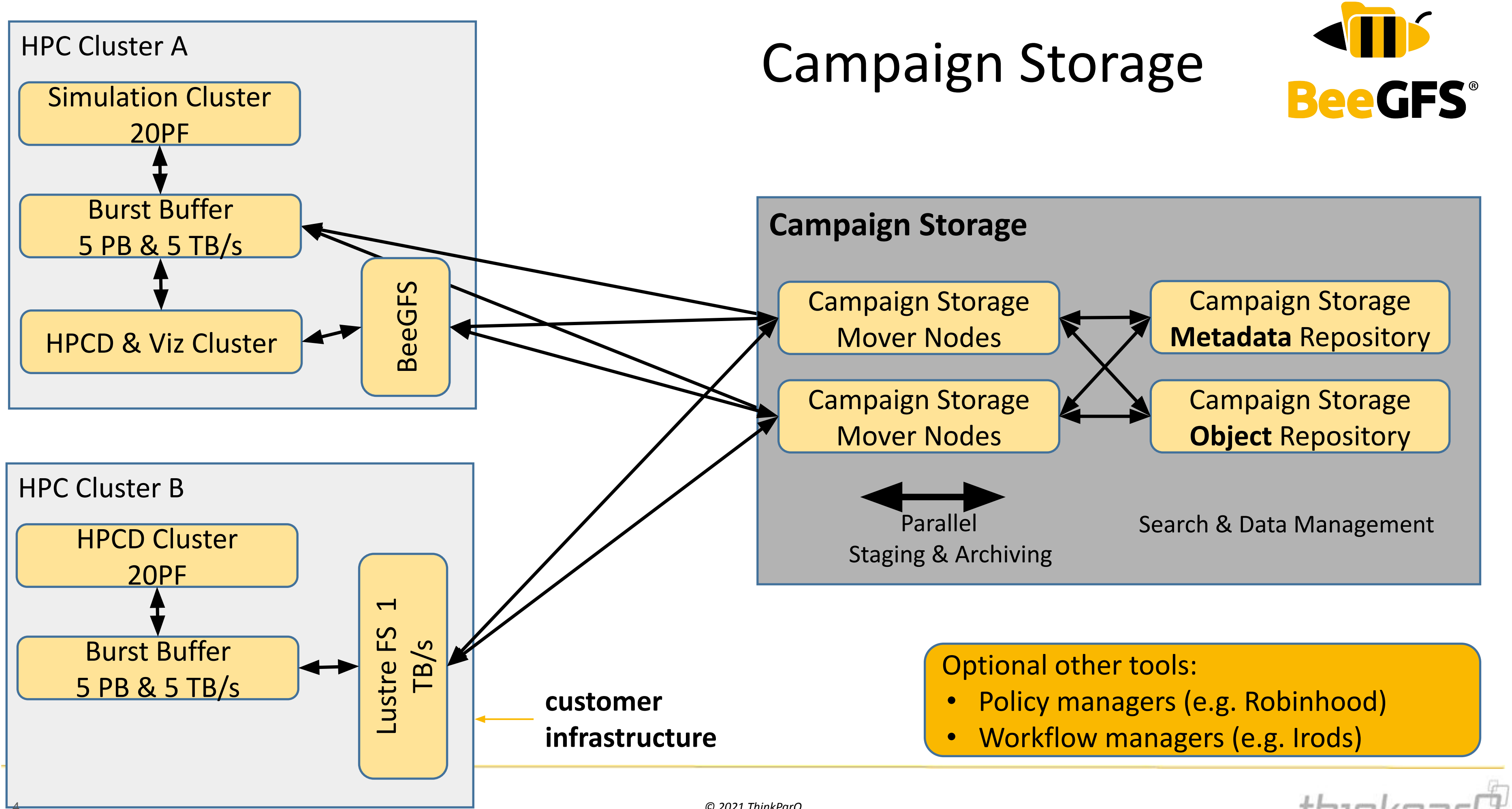
This Campaign Storage file system was even **larger and even slower to crawl**, while **data management** was expected.

Two index systems were explored:

- LANL created **GUFI - grand unified file index**
- I described a **Hierarchical Subtree Index (**MSST 2017)

- Both are a collection of databases, one database for each directory in the file tree
- Each of them can be stored embedded in the file-tree or outside the file-tree
- With scalable metadata service, this database can scale

Campaign Storage


HPC Cluster A
Simulation Cluster 20PF
Burst Buffer 5 PB & 5 TB/s
HPCD & Viz Cluster
BeeGFS
HPC Cluster B
HPCD Cluster 20PF
Burst Buffer 5 PB & 5 TB/s
Lustre FS 1 TB/s
Campaign Storage
Campaign Storage Mover Nodes
Campaign Storage Mover Nodes
Campaign Storage **Metadata** Repository
Campaign Storage **Object** Repository
Parallel Staging & Archiving
Search & Data Management
customer infrastructure

Optional other tools:

- Policy managers (e.g. Robinhood)
- Workflow managers (e.g. Irods)

Why do we want an index?

iRODS perspective:

To create a faster, scalable feed into iRODS

Independently (or in conjunction) as a basic solution for:

- **Archiving:** which files were modified or accessed after a particular date?
- **Creating space:** where are the files that are very large?
- **Managing projects/users:** how much space is used by a user or project?
- **Managing tiers/HSM:** which files were accessed recently?
- **Managing pools:** which files have components in a pool or on a device?

GUFI

grand unified file index
on GitHub from LANL

How does GUFI work?

For each directory, there are 3 tables:

entries: has records with name & attributes for each inode in that directory

dir summary: has one record with **hierarchical** summary info

e.g. no. of files, minimum file size, maximum file size, etc.

tree summary: has one record with summary of subtree

could be used to summarize subtrees (appears not to be done)

GUFI scans the entire file system to populate the databases

- tables can be stored **in the tree** or in a **mirrored directory tree**.
- in tree: any user can do some searches
- in a mirrored directory tree: for data management. Typically smaller & faster

GUFI use

1 record is around 1KB: 1B files fit on 1TB. Nice for a set of NVME drives

Sample queries

- Find files larger than 1GB in subtree
- Find files not accessed since T
- Find large files modified before MTIME and not accessed after ATIME

Performance

- Most queries >100x faster than “find”
- Avoids descent based on tree summary
- Avoids RPC’s, exploits data locality

Administrative Commands:

- Create GUFI index
- Update GUFI index
- Give statistics of GUFI index

Future desirable searches:

- Find files larger than 1GB in storage pool P
- Find files not accessed for T in a storage pool
- Find files with data on server S
- Find files with data on device D in server S

Hierarchical Subtree Index

MSST 2017

Histogram with Subtree Information

Alternative to GUFI.

Store only a histogram with “bars” like:

- the number of files
- the bytes consumed
- the number of files bigger than something
- the number of files using a particular device, group, project

What is special about these summary quantities?

- **additive:** wrt files in the directory and the subtrees
- **Merkl tree** property (similar to GIT trees)
  e.g. #(files in subtree) =
  # files in dir + # files in subdir’s trees

Place a single histogram in every subdirectory that has a choice of “bars”.

The histogram data:

- summarizes relevant data management
- guides searches into the subtree
- scales with the number of directories
- can be updated real-time, through additivity

Conclusions

Origin and Problems to be Addressed

Wanted:

- Happy HPC archives
- Indexing must scale with file system capabilities
- Purpose: feed into other tools
  - data movers
  - policy and data management databases

Industry Solutions - how widely adopted are they:

- Policy databases: Robinhood, HPSS, DMF, Komprise, …..
- Faster scanning tools (avoid the file system)
- Apple File System implements subtree information

iRODS perhaps easily the happiest and most successful

Difficulties to be Addressed

- Search times too high
- Scan times increase
- Complexity
- Changelogs: huge slowdown for parallel file systems
- A full policy database is too big
- Index object stores

ThinkParQ

Maintains BeeGFS parallel file system

- user base includes several customers with >1B files

ThinkParQ to provide an index and data moving tool.

Further described in a blog post in January 2021

- long term: some tight integrations with file system
- short term: provide basic data management tools

BeeGFS®

Thank You