LUG13

Using Changelogs for Efficient Search and Content Discovery

Ashley Pittman

Changelogs

- Not end-user feature
  - Great building block for components layered on top of Lustre
- Not actually new
  - Although not widely used yet

LUG13

Changelogs, what you get.

- Configuring/using changelogs.
  1. Enable
  2. Read
  3. Mark consumed (delete)
- Example changelogs

```
52 02MKDIR 15:36:12.485458611 2013.04.15 0x0 t=[0x200000400:0x6c6:0x0] p=[0x200000400:0x6af:0x0] osd-ldiskfs
53 01CREAT 15:36:12.487463260 2013.04.15 0x0 t=[0x200000400:0x6c7:0x0] p=[0x200000400:0x6c6:0x0] osd_oi.h
54 15XATTR 15:36:12.493459818 2013.04.15 0x0 t=[0x200000400:0x6c7:0x0]
55 01CREAT 15:36:12.564463988 2013.04.15 0x0 t=[0x200000400:0x6c8:0x0] p=[0x200000400:0x6c6:0x0] osd_iam.h
56 15XATTR 15:36:12.570466507 2013.04.15 0x0 t=[0x200000400:0x6c8:0x0]
57 01CREAT 15:36:12.660468456 2013.04.15 0x0 t=[0x200000400:0x6c9:0x0] p=[0x200000400:0x6c6:0x0] osd_iam_lvar.c
58 15XATTR 15:36:12.666467542 2013.04.15 0x0 t=[0x200000400:0x6c9:0x0]
59 01CREAT 15:36:12.738461715 2013.04.15 0x0 t=[0x200000400:0x6ca:0x0] p=[0x200000400:0x6c6:0x0] osd_iam.c
60 15XATTR 15:36:12.744461576 2013.04.15 0x0 t=[0x200000400:0x6ca:0x0]
61 01CREAT 15:36:12.852466656 2013.04.15 0x0 t=[0x200000400:0x6cb:0x0] p=[0x200000400:0x6c6:0x0] osd_internal.h
62 15XATTR 15:36:12.859466307 2013.04.15 0x0 t=[0x200000400:0x6cb:0x0]
```

Lifetime of a file

- created... written… closed.
- From the point where it's closed to when it's subsequently opened again it's contents are static.
- Not dissimilar to a object.
  - "Immutable"
  - Has attributes
    - Can think of name and directory as a attribute, rather than a way of accessing the file.

LUG13

Why search?

- If you can’t find it you may as well not have it.
  - Larger filesystems
  - Tiering/HSM make traditional search even harder
  - Maps well onto certain workflows

ElasticSearch

- "ElasticSearch is a distributed, RESTful, free/open source search server based on Apache Lucene."
  - Scalable
  - Resilient
  - Matches the write-once, read-many model
    - Or rather – matches the "single producer", "multiple consumer" model

Elastic Search internals

Multiple indexes

Multiple types per index

Create documents of a type

- Documents are returned from search queries

Types are schema-less

- Documents are normally represented in JSON

FileSystem River

The Filesystem river plugin helps to index documents in local filesysems.

- Creates indexes automatically.
- Creates type automatically.
- Imports contents of a POSIX filesystem as documents
  - Uses file metadata to define a schema.

FileSystem River without Lustre.

- Periodically scans filesystems
  - Controllable frequency.
- Stores all posix metadata so any field is searchable
- Uses Apache Tika for indexing common file types
  - .doc, .html, .pdf
- File contents are searchable by keywords extracted by Tika.

Using Changelogs.

1. Modify fsriver “scan” function to consume changelog rather than scan
   1. Only need to intercept write-metadata IOPS
   2. Leaving scan function in place for import/recovery
2. Change the frequency of scans.
3. Real-time searching of file metadata in Lustre with almost zero code.

Lustre specific extensions required

- To be really useful searching by OST should work.
  - Query FID and stripe information from file and import into json document.
- Parsing “lfs getstripe” in java.
- Disable content-search for files which are archived.
  - Danger of archiving the file, then elasticsearch trying to read it immediately afterwards

The Downsides

- Scalability isn't free
  1. Works well with the write once, read many workload
  2. Read-delay on writing. Files written to Lustre aren't imported immediately. Neither changelogs or elasticsearch import are synchronous
  3. Possible false positives
  4. Better to create/delete than to modify
- Most data is static most of the time, great for HSM or long-term storage. Less good for "home" filesystems.

Making it usable.

- Permissions.
  - Checking file permission in isolation is not enough
    - Need to walk the POSIX namespace checking permission for each directory
  - Means you have to build a layer on top of elastic search
    - Which is normal but beyond the scope of this work
- Everything we've done is just interacting with the search engine directly.
  - Command line queries, manually reading JSON replies.
- Content scanning on archive is too late.

Conclusions on searching

1) Suitable backend exists and is easy to use & configure

   Very flexible in configuration/scalability

2) Not dissimilar in functionality to robin-hood

   Different performance characters

3) Flat, object-based model of filesystem contents

   True for archived files, less true for active files

4) POSIX-like construct on top of it

   POSIX semantics however are probably impossible

5) Would be better implemented behind policy-engine