IDS versus Direct File Access to Storage

Rolf Krahl

ICAT Meeting, Diamond Light Source, March 2015

Introduction

Why Direct File Access?

- IDS might turn out to be a bottleneck when many beam lines need to upload their data.
- There is some overhead in the internal processing in IDS: each uploaded file hits the disk three times.
- Need to access the files for analysis. Download and unpack the ZIP to local disk each time is cumbersome and not very efficient.

Introduction

Issues with Direct File Access?

- Need meaningful file and directory names.
- Concurrent and possibly conflicting file access.
- Permissions.

- Here: concentrate on concurrent file access.
- Assume an IDS with two-level storage and storage unit dataset.
- Use file locking to prevent conflicts.
- Consider some practical use cases.

File Locking

- File locking can be done in the storage plugin, even without modification of the IDS server.
- Use `fcntl` type file locking.
- For `ArchiveFileStorage`: acquire a shared lock on the ZIP file in method `get()`.
- For `MainFileStorage`, we would need to lock the entire dataset directory. Place a designated lock file for each dataset directory in its parent directory. Stille use `fcntl` file locking on this lock file.
    - acquire a shared lock in `get() method`.
    - acquire an exclusive lock in (each) `put()` and `delete()` methods.

Use Case: Ingest by Writing to Archive Storage

Use case: ingest a new dataset by writing the ZIP file directly into archive storage and creating all objects in ICAT.

Possible conflicts

- IDS could write the same ZIP file at the same time.
- IDS could try to restore a partly written ZIP file, leaving main storage in an inconsistent state.

Use Case: Ingest by Writing to Archive Storage

Solution

The ingestor uses the following procedure (order is important):

1. Verify that the dataset directory in main storage does not exist.
2. Open the ZIP in archive storage as a new file (use `O_CREAT` and `O_EXCL` flags) for writing and acquire an exclusive file lock on it.
3. Create the dataset including the datafiles in a single call (cascading) in ICAT.
4. Write the ZIP file.
5. Close the ZIP file (which implicitly releases the lock).

Use Case: Ingest by Writing to Archive Storage

Notes:

- Efficient: each file is only written to disk once.
- If the procedure fails in any of the first three steps, nothing serious has happend yet, worst case is leaving a spurious zero size ZIP file behind.
- If the creation of the dataset in step three succeeds, IDS will consider it as non-empty dataset in `ARCHIVED` state.
- In this case, any concurrent action in IDS on the dataset will first trigger a `RESTORE`, which in turn will be blocked until the file lock is released. ⇒ no access conflict can occur.

Use Case: Ingest by Writing to Main Storage

Use case: ingest datafiles to an existing dataset by writing the individual datafiles into main storage, creating all objects in ICAT and inciting IDS to create the ZIP in archive storage.

Possible conflicts

- Concurrent upload of files via IDS API could overwrite currently ingested files.
- An `ARCHIVE` operation could delete all files before the ZIP is created in archive storage.

Use Case: Ingest by Writing to Main Storage

Tentative Solution

The ingestor uses the following procedure:

1. Verify that the dataset is `ONLINE` and the directory in main storage does exist.
2. Acquire an exclusive lock on the dataset directory.
3. Write the files and create the Datafile objects in ICAT.
4. Release the lock.
5. Trigger a `WRITE` operation in IDS to create the archive file.

Use Case: Ingest by Writing to Main Storage

Notes:

- Simple and convenient: one could mount the dataset directory directly at the experiment and write directly into it.
- There is no IDS call to trigger the `WRITE` operation, can be forced crabwise though.
- Unfortunately: still not safe. There is no way to guarantee that the `WRITE` operation will be executed before a possible concurrent `ARCHIVE` operation. ⇒ show stopper.

Use Case: Read Access to Main Storage

Use case: read only access to the files of a dataset in main storage, such as doing analysis.

Possible conflicts

- Deleting individual files via IDS API while working on them.
- An `ARCHIVE` operation could delete all files while working on them.

Use Case: Read Access to Main Storage

Solution

Use the following procedure:

1. Make sure the dataset is `ONLINE`.
2. Lock the dataset directory.
3. Work on the files.
4. Release the lock.

Notes:

- Read only access is as simple as this. No further cooperation from the IDS server is needed.
- More efficient then downloading the files via IDS API.
- Works also if the storage is mounted read only.

Discussion and Conclusion

- With sufficient precautions, direct file access to the storage concurrently to the IDS server can be made safe.
- File locking can be implemented in the storage plugin, without need to modify the IDS server.
- Works for two relevant use cases: write access to archive storage and read only access to main storage.
- Write access to main storage cannot easily be made safe this way.
- Somewhat inefficient: acquiring and releasing the lock for each single file access from IDS.
- Further improvement would require changes in the IDS server and to the plugin API.