cloudera
Ask Bigger Questions

Keep Me in the Loop: Introducing HDFS INotify

by Colin P. McCabe

About Me

- I work on HDFS and related storage technologies at Cloudera.
- Committer on the HDFS and Hadoop projects.
- Previously worked on the Ceph distributed filesystem

Roadmap

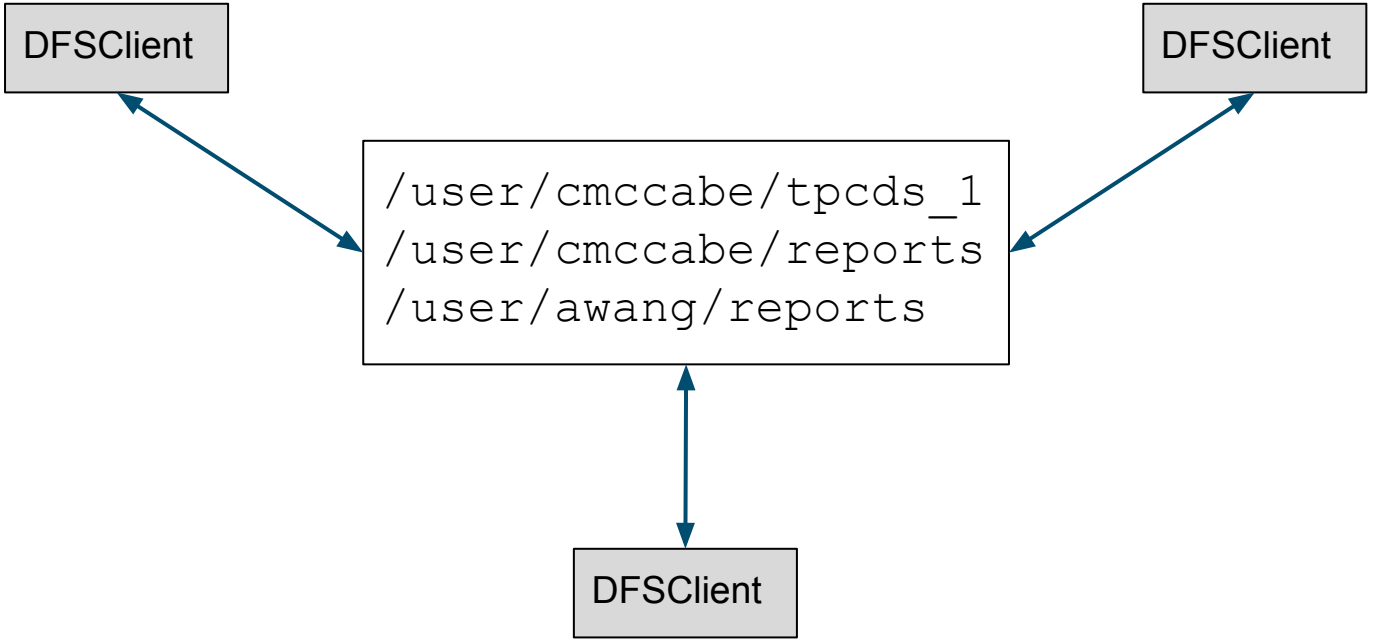
- Introduction
- Use Cases
- Design goals
- Architecture
- Future work
- Conclusion

HDFS: the Hadoop Distributed Filesystem

HDFS is the most popular distributed storage system for Hadoop.

Based on the concepts of robustness, fault-tolerance, and bringing the computation to the data

HDFS Provides a Shared Namespace



Systems Built on Top of HDFS

- Apache Hive
- Apache Solr
- Cloudera Impala
- Cloudera Manager
- and many more...

Most Systems Do Some Caching

- Apache Hive: Caches HDFS path names in the Hive metastore
- Apache Solr: Builds search indices
- Cloudera Impala: Caches HDFS block locations
- Cloudera Manager builds search indices for the files in HDFS using “Cloudera Headlamp”

Problems with Caching

- When to invalidate the cache?
  - Time-based invalidation
  - Manually triggered invalidation
- How to invalidate the cache?
  - Re-read all or part of the entire FS?

Problems with Caching (cont.)

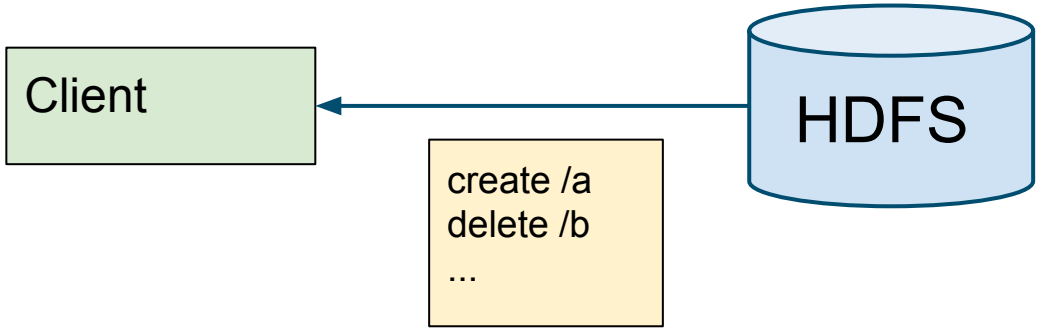
- Time-based invalidation either leaves the cache stale for long periods, or puts a heavy load on the system.
- Manually triggered invalidation makes the system more complex to administer.
- Re-scanning a large part of the filesystem is slow and wasteful.

Caching Data from HDFS

- How do we know when to rebuild the cache?
- Periodic scan?
  - Slow, consumes a lot of resources
- Manual trigger?
  - Burdens the admin, still slow

Introducing HDFS INotify

- Allows clients to receive asynchronous notifications when files or directories in HDFS change



Design Goals for the HDFS INotify API

- Provide a stable and easy to use API
  - Newer versions should be backwards compatible
  - Avoid parsing text files such as log files… it is messy and slow
    - The format of log files often changes… even the HDFS audit log.

Design Goals for HDFS INotify (cont.)

- Allow clients to see the order in which events occurred
- Don’t lose events
  - If an HDFS client restarts, it should be able to pick up reading INotify events where it left off

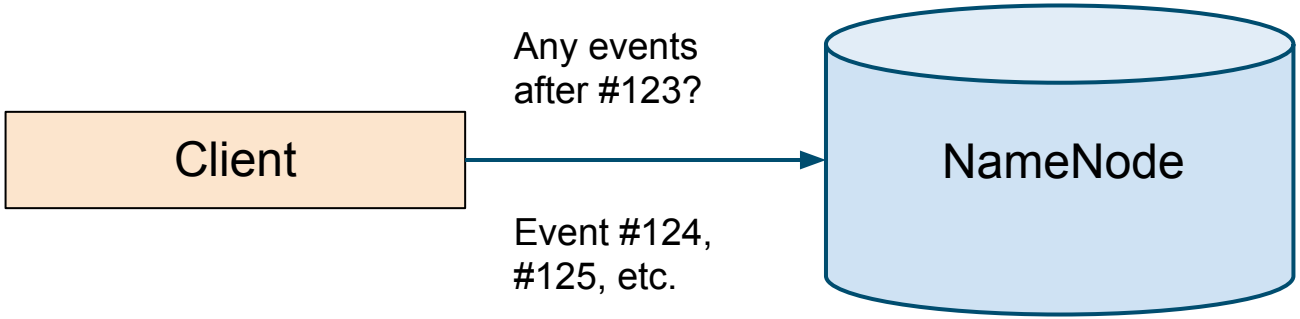
Design Goals for HDFS INotify

- Don’t require external systems
  - Don’t require Flume, Kafka, etc.
  - All great systems which can add a lot of value, but not every HDFS cluster has them

Non-Goals for HDFS INotify

- Synchronous replication of events
  - We can't block the NameNode waiting for a client
- Notifications about individual writes to DataNodes
- Notifications about files being opened or read, or other read-only operations

INotify Architecture



Client caches highest event number that it's seen.

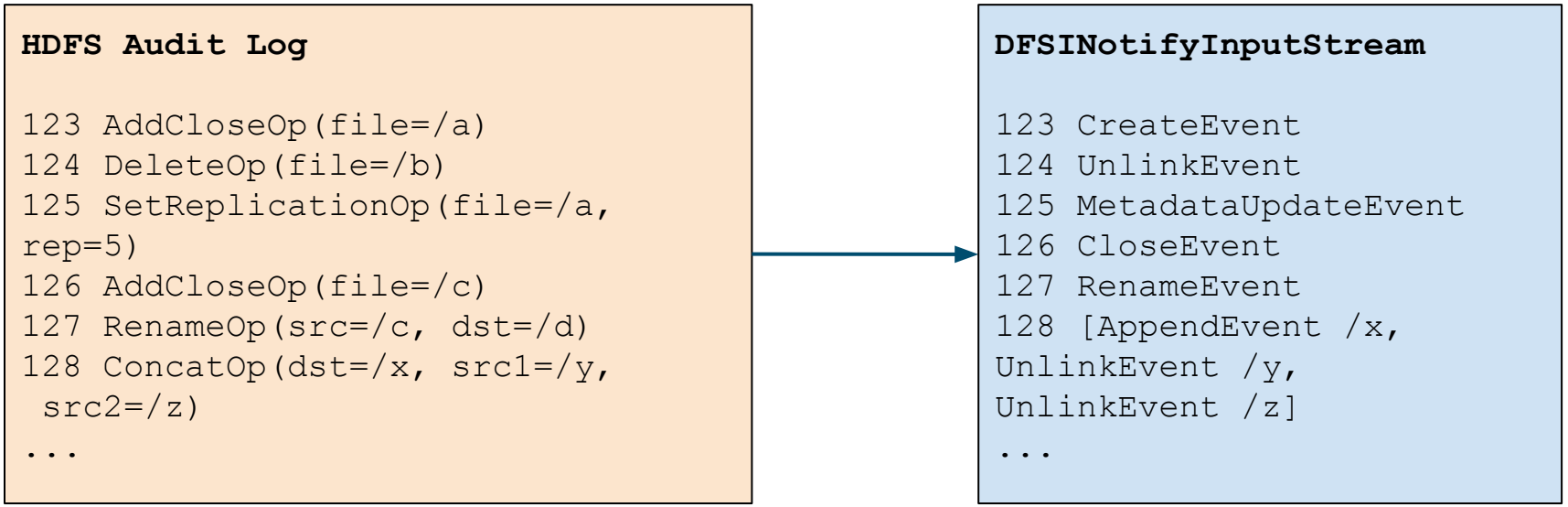
Client polls periodically.

NameNode uses monotonically increasing 64-bit event IDs

The HDFS Edit Log

- A write-behind log which contains every modification which was made to the filesystem.
- Each HDFS edit log entry has a transaction ID. They are monotonically increasing and unique.
- In high availability setups, HDFS edit log entries are stored on multiple JournalNodes for extra safety

INotify Maps Edit Log Entries to Events



Mapping Edit Log Entries to Events

- A single edit log entry can map to multiple events, or to no events at all.
- The internal representation of the edit log changes often.
  - Recently added a length field to all edit log ops.
  - Often add new fields when new features are added
- The edit log uses custom serialization.

INotify Events are stored in Protobuffers

Sample INotify event:

```
message RenameEventProto {
  required string srcPath = 1;
  required string destPath = 2;
  required int64 timestamp = 3;
}
```

HDFS INotify API Example

```
DFSInotifyEventInputStream stream =
  dfs.getInotifyEventStream(prevHighestTxId);
EventBatch batch = stream.take();

long newTxid = batch.getTxid();
switch (batch.getEvents()[0].getEventType()) {
  case Event.EventType.RENAME:
…
}
stream.close();
```

Polling versus Pushing

- Why polling and not a push model?
  - Polling fits into Hadoop RPC better.
  - Don’t have to maintain a list of clients to push changes to, or open sockets for those clients.
  - Polling will always be needed anyway if push connections drop.
- But a push model could be slightly more efficient
  - Might implement push later as an optimization.

NameNode RPC versus JournalNode RPC

- Why make INotify a NameNode-side feature?
  - It's much easier to implement on the NameNode because the client already talks to the NameNode for most operations.
- Can we offload INotify to the JournalNodes?
  - This would decrease RPC load on the NameNode
  - But not all installations use JournalNodes
  - We might implement this later as an optimization

Security and INotify

- Right now, using HDFS INotify requires superuser privileges.
- Can we make it accessible to normal users?
- Yes… can filter events by path names
- Can’t use standard permission model because we have no FSImage at the point in time of an edit.

Previous Work

- INotify in Linux
  http://www.linuxjournal.com/article/8478
- INotify as a third-party service built on top of HDFS
  https://www.youtube.com/watch?v=7KumMKqBtr8
  - by Benoit Perroud and Hariprasad Kuppuswarmy

Users

- Cloudera Headlamp now uses INotify!
- We no longer have to rely on parsing logs to follow along with HDFS.
- Stable API for the future.

Future Plans

- Support subtree watches
- Support access by non-superusers
- New event types as we add features to HDFS
- Efficiency improvements

Conclusion

- HDFS INotify is a great way to build caches and indexes on top of HDFS, without resorting to inefficient and costly full scans or manual cache invaldation
- Production-ready in CDH5.4 and later
- Production-ready in Hadoop 2.7

Thanks for Listening!

http://www.cloudera.com/careers