Integrating OmSnapshotting with Incremental Checkpointing

The Reason for IC

The Ratis leader sends the "appendEntries" message to keep the followers up to date; however, it can only do so while it has a copy of the entries in its local Raft log. If the leader notices the follower needs expired entries, it sends the "installSnapshot" message. In response, the follower downloads a copy of the latest rocksdb checkpoint, then continues with the subsequent appendEntries messages.

However, when that rocksDb checkpoint gets large enough, the checkpoint transmission takes so long that subsequent raft logs have already expired before the transmission completes. Then the follower enters a vicious cycle of constantly receiving installSnapshot messages and never catching up.

The Current IC Fix

The current PR, https://github.com/apache/ozone/pull/4052 , resolves this issue by having the follower keep track of all the files it receives from the leader, until it catches up. Until then, the leader continues to notice it is behind and sends more installSnapshot requests. In response to each, the follower includes an "exclude" list of sst files that have been already received. That way sst files aren't sent twice. (Since they are immutable, there is no reason to send them a second time.)

Once the follower receives an appendEntries message which is consistent with its updated state, it knows that it has caught up and can stop tracking this particular checkpoint installation. It deletes the contents of the "candidate" directory which contains the in-progress installation and resumes updates through the appendEntries mechanism.

Since the contents of the candidate dir are deleted, the next time the follower receives the installSnapshot message, it will create a new candidate dir, (and return an empty exclude list.)

Also, since the exclude list only includes sst files, the "Manifest" file is always refreshed.

Associated with the candidate directory is the notion of a "currentLeader", i.e. the leader that was current at the time the directory was created. Before creating the exclude list, the follower checks to see if the actual leader matches the "currentLeader". If not, it deletes the candidate dir, and sends an empty exclude list to the new leader to start the process afresh.

Integrating OM Snapshotting into IC

A few weeks before the IC PR was created, I created a PR for integrating OM Snapshotting into the old non-incremental from of checkpointing/updating followers:
https://github.com/apache/ozone/pull/3980

Basically, that PR adds all the omSnapshot state files to the tarball while taking care to maintain internal consistency and preserve existing hard links. (The tarball doesn't contain the contents of any file considered a hard link. It just includes a list of hard links to be created after the tarball is installed.)

Now that IC PR exists, the OmSnapshotting PR will need to be expanded to take advantage of the exclude list. In order to generate that exclude list, the candidate dir will be expanded to include all the new omSnapshot state.

Issues

Removal of Unnecessary SST File Created by Compaction

As compaction continues to happen on the leader, some of the files on the follower may become obsolete. This shouldn't cause any problems, because the follower will receive the updated manifest and know to ignore the old files.

However it will lead to wasted disk space on the follower. We could address this by adding the notion of a "remove" list that the leader sends to the follower, (similar to the "hard link" list mentioned above.)

Hard Link Preservation

The current IC code copies the candidate dir into the proper rocksDb location when it is ready to be installed. This copy does not preserve hard links and will need to be modified to do so.

Convert to Http Post

The follower currently requests the tarball with an http Get that transmits the exclude-list as a url parameter. This will no longer be possible because the exclude list can get larger than what a url parameter supports.

So the request will have to be changed to a http Post, with the exclude list as a part of the Post body.

Handling the Different OmSnapshot State File Types

OmSnapshot Directories
These are immutable checkpoints from a particular point in time. Once installed on the follower they never need to be modified by the leader. Each directory will be added to the exclude list.

Snapdiff Compaction Logs
These are small text files that record the history of the compactions. These can be modified by the leader and will not be excluded.

Snapdiff SST Backup Directory
This directory of sst file hard links that are preserved to enable optimized snapdiff. Each of these files will be added to the exclude list.

There is a background process which deletes these files when no longer necessary. I haven't yet studied how it works, so I'll need to spend some time examining it to make sure IC doesn't cause any problems for it.

Background Delete Task State
The OmSnapshot subsystem has a background delete process. This process steps through the OmSnapshot directories and deletes sst files unused by the omBucket that has been snapshotted. It tracks its progress in a text file.

I don't think this simple text file can convey enough information to correctly deal with issues that IC creates. My thinking is that we should not copy this file over at all. Instead, we should let the followers instance of the task create it as it processes the snapshot directories naturally.