HLD for Readdir

Wang Di

2006/01/25

1 Requirement

Make readdir POSIX compliant especially for split dir nodes. According to the requirement of readdir, the dir entries should be read out according to some order, no matter what will happen in this process, then tell-seek interface could be implemented based on this. But our current implementation can not assure this, especially when the splitting happens in-between of readdir. Understanding of ext3 htree code and splitting dir in CMD is assumed in this HLD.

2 Definitions

Here are some definitions used in this HLD:

hash value: it is computed by some kind of hash algorithm according to dir entry itself.

hash order: it means accessing the dir object according to the hash value.

hash interval: it means an interval of hash value, which is correspondent to each MDS for the splitted object.

hash position: it means current position of dir, which is indicated by hash value, since the dir entry is sorted by hash value.

splitting rule: it means the splitting way which MDS will use to split the dir. Current implementation use name hash to locate the object in a splitting dir.

3 Functional Specification

In MDS, the htree enable flag is set default, so all the dir entries should be stored as a hash tree.

3.1 Current readdir implementation has some dis-advantages especially for splitted dir.

- In readdir, the client will access the entry of the dir sequentially, i.e. reading the page of the dir from MDS sequentially. But if the dir is a hash tree dir, this sequence order will bring us some troubles, since it is different with the one used by local ext3 filesystem, which use hash order.
- Further more, if the dir is split in-between of readdir, the position of those splitted entries might be lost, then some entries might be gotten two times.

3.2 readdir in hash order

To keep readdir fully POSIX compliant when splitting, readdir and splitting must follow the same order. Since the splitting is just to scatter the dir entries to multiple MDS for load balance, so the splitting must follow some kind of hash order. Luckily, the dir is stored as a hash tree in MDS and readdir in the bottom filesystem(ext3) also follow the hash order. So we decide readdir and splitting will follow this hash order. According to this splitting rule, the hash interval will be defined for each MDS, then in splitting or lookup, the hash of the dir entry will be calculated to locate the right MDS. Since dir entries with same hash value will be stored in the same MDS, so the ext3 will help to resolve the hash collision. An new RPC will be defined to handle this new kind of readdir.

Since POSIX did not specify that whether readdir and those modify ops(create/unlink) could happen concurrently, so we do not permit that to keep things simple, although ext3 permit that. Note: here the concurrent means modify ops just happened in the same time as readdir in MDS.

4 Use case

4.1 Readdir in a no-splitted dir

- Client: issue readdir req, in which the current position(hash value) is included, to the MDS where dir is located at.
- Server: Retrieve entries from the dir according to the current position.
- Server: Return them back to client, and client will submit them to the upper layer caller.

4.2 lookup in a splitted dir

- Client: compute the hash value according to the name.
- Client: send the req to the right MDS according to the hash value.
- Server: use this hash value to locate the entry in the dir, and return it back to client.

4.3 readdir in a splitted dir

- Client: send readdir req to the first MDS according to the hash interval.
- Server: execute readdir in hash order and fill the request buffer with the dir entries.
- Client: get all the entries of the MDS, the client will resort to the next MDS in the hash interval, Until it get all the entries of the dir.

4.4 Readdir and splitting happened in the same time

- One process is in-between readdir, while another process splitting the dir at the same time.
- The readdir process will continue to send read_dir request to the original MDS, until it can not find the next entries according to the current position(hash value).
- Client will check whether it reach the EOF of the dir, if not, it will revalidate this dir and check whether the dir is split, if it is split, then decide where the following req should go by current hash position and hash interval of this inode(discussed this in Logical specification) and send the req to the right MDS.
- In the new MDS it will locate the entry by the current hash position, and return the following entries to client.

5 Logic Specifications

5.1 splitting dir with hash order

According to 0.2.2, the splitting should also follow the hash order used by ext3 htree. In the htree of ext3, the index entries are stored in hash order. In the splitting, for the splitted dir, we will define one hash interval for each MDS, then the split will be executed according to these hash interval. The split process should be

- Check the dir whether it should be split. Only when the count of the entries reached the upper limit number, it could be split.
- Divide total _hash_ range into equal intervals and assign each interval to separate MDS.
- Iterate over the index entries of the dir and scatter index entries and the blocks they point to each MDS according to the hash interval.

5.2 Readdir with hash order

In ext3 readdir, the current entry hash value is stored in f_pos to indicate the current position. To assure lustre client also know it, MDS should return it to client and client should record it in its f_pos. Then client could retrieve the entries from MDS by this hash position, not by offset as the original. The process of readdir in client should be

- Client first check where the req should go according to the current hash position in f_pos and the hash interval.
- The dir entries are returned in reply buffer according to the current hash position.
- The client retrieve the entries from the reply buffer, and reply them to the caller.
- Calculate the last entry hash value of the retrieve, then record it in the f_pos to indicate current hash position.

A new readdir rpc handler will be defined in MDS. The process of readdir in MDS should be

- Unpack the req, and get the current position from the request.
- Got and lock the dir. Here the lock should prevent deleting and creating the entry in this dir at the same time.
- Locate the current position of dir and read entries from the bottom filesystem.
- pack and return the entries back to the client.

Note: The above discussions are all about h-tree dir, since the htree enable flag is set default for MDS.

6 State Management

6.1 splitting dir lock

As for dir, those modified operations are not permitted when reading, so it should be locked by LCK_PW mode to prevent any modification before reading. While when those modified operations being executed on the dir, it could not be read as well.

6.2 Recovery for splitting dir

Since several MDSes are involved into the splitting of the dir, we need some cluster rollback mechanism to implement the recovery of splitting. When splitting

- correspondent records of this dir will be written into some kinds of logs in the Master MDS.
- Then splitting the dir.
- When each MDS finish splitting, it will send cancel log cookie req to the Master MDS to cancel these logs.

In MDS recovery, it will check these llog, if it found some, the dir will be rollback to the original state by some cluster rollback mechanism, which is discussed in other HLD.

7 Alternative

7.1 Not permit splitting in progress of readdir

Another alternative way to avoid conflicts between splitting and readdir is that splitting is not permitted when the dir is opened, which means some other processes are in progress of readdir. So if the dir is open, the splitting would be delayed until the dir is closed.

7.2 Another splitting rule

There is another splitting rule which can be used for defining the hash interval for each MDS.

1. Get the total count of the index entry of the dir. For 2-level hash trees (the max level of current ext3 hash tree), we should get the count of the second level. There may need some patches in ext3 to export the entries count to MDS.
2. Divide the count of index entries by the number of MD servers, the result is D.
3. For the hash interval in MDS_1, seek and get the hash value of D_th entry in the dir, and which is the upper limit of the dir's hash interval in MDS_1, and the lower limit is 0.
4. For the hash interval in MDS_n, its lower value is MDS_(n-1)'s upper value, and its upper value is the hash value of n*D_th entry. If this value is same as the lower value, we should resort to next entry, until we get different upper value for this hash interval.

5. If the hash interval of the splitted dir in MDS_n is (lower_n, upper_n). then for all entries of the splitted dir in the MDS_n, their hash value are <= upper_n and > lower_n.

6. These hash interval and corresponding number of MD servers should be stored in splitted dir EA. Then when client do lookup, it could get the right MDS number according to these hash interval and the hash value of the name.

Since this hash interval definition will need store another EA for the splitted dir. And the tea-hash method, which ext3 used, could also distribute the whole entries evenly across whole hash value field (hash value is 31 bit for ext3, so the hash value field should be (0, 0x7fffffff)).

8 Focus of Inspection

1. Is splitting policy reasonable?

9 Inspection Summary