# Metadata Stat-ahead HLD

Lai Siyao <lsy@clusterfs.com>

2007.03.16

## 1 Introduction

This document describes metadata stat-ahead, which is a part of metadata improvements. The client will perform metadata stat-ahead when it detects readdir and sequential stat of dir entries therein.

## 2 Requirements

- Perform client-side metadata stat-ahead when the client detects readdir and sequential stat of dir entries therein.
- An ‘ls -l‘ in a directory with 1 million files takes at most 50% of the pre-feature time.
- Stat-ahead shouldn’t have obvious impact for applications other than ‘ls -l‘.

## 3 Functional specification

### 3.1 ‘ls -l‘ scenario

Herein we will describe what’s happening in ‘ls -l‘, firstly the syscall chain looks like this:

1. open(“.”)
2. getdents()
3. stat(“file1”)
4. ...

5. getdents()
6. stat...
7. close(“.”)

One thing we will notice is that the result of ‘ls -l‘ is sorted, however this is done before close(“.”), and in user space after all the stat results are collected in a list. And in this design we will start stat-ahead in step 3, just before doing stat(“file1”).

## 3.2 Feature control

For each mount point, the feature of metadata stat-ahead can be turned on or off with a /proc/fs/lustre/llite variable. In this way we can turn on/off stat-ahead in run-time, and this is more flexible than other ways and can test more conveniently. Another reason for this is that IO read-ahead did this way.

## 3.3 Stat-ahead detection

To trigger metadata stat-ahead, client needs to detect readdir and sequential stat of dir entries. Upon client opening this dir, the pid will be logged. And later if client stat the first dir entry, and the the pid is the same as the one opening its parent dir, the stat-ahead of the following dir entries will begin. Herein we check the first dir entry is stated, because we will only do stat-ahead for ‘ls -l‘ case.

## 3.4 Stat-ahead thread

Once client assumes it needs do stat-ahead, it will start a stat-ahead thread for this dir. This is a llite level thread, which keeps sending new stat requests for each entry in this dir until:

- dir is closed, this thread will exit.
- it hits the end of the dir, just like above this thread will exit.
- it is ***statahead_count*** entries ahead of the stat process. The ***statahead_count*** value should be dynamic, and the default number for ***statahead_count*** should probably be something quite small like 3, once client stat from local cache, as means stat-ahead has hit, then this value will be increased gradually, and the maxium value can reach 50 so that we don’t drop entries from the DLM LRU.
- the hit rate is too low, this happens when client do stat, but misses it from local cache, generally this means we are not doing ‘ls -l‘, instead user might have issued command like ‘ls -l file1 file10‘, or ‘ls -l a*‘. As stated above, we don’t do stat-ahead for these cases.

## 3.5 Stat-ahead process

To make stat-ahead effective, stat-ahead for one dir entry should save the result to a dentry. Because only in this way the following stat can just obtain data from local cache, otherwise it will issue an RPC to getattr anyway. So the process is as follows:

1. if d_lookup(parent, name) return NULL, call ll_sa_lookup() to lookup this dentry.
2. else if d_lookup() finds a dentry, call ll_sa_revalidate() to revalidate this dentry.

## 3.6 Async stat

To improve stat-ahead performance, we should do it asynchronously. To achieve this, we need lookup/revalidate asynchronously(see 3.5).

MDC layer will send all stat-ahead requests asynchrously by ptlrpcd. In this way we can stat in parallel up to max_rpcs_in_flight in MDC.

A new callback function ll_statahead_interpret() will be set for md_statahead_info.sai_cb. And this function will be called to finish lookup after aync MDC enqueue lock.

## 3.7 Avoiding double stat

While the stat-ahead thread is doing stat, client might be doing a stat() for an entry that is still in the wire, so two stat RPCs are sent for the same dir entry simultaneously. To avoid this, the stat process will check stat-ahead thread before doing real stat, if the stat-ahead count is more than 0(which means stat-thread has got some results), it will continue with stat, otherwise it will put itself in a waitqueue of stat-ahead thread; while for stat-ahead thread, once it gets a result of stat, it will wake all processes who are waiting in the queue. The process which is doing ll_lookup_it() will check dentry->d_inode after woken up, if it's not NULL, it will return current dentry immediately (as means stat-ahead thread has filled inode informations), else it will continue getattr from MDS.

## 3.8 Hit ratio track

Stat-ahead thread will exit when the hit ratio is too low, which generally means user is not doing 'ls -l'. But this hit accounting might not be accurate: firstly the IT_UPDATE lock fetched by stat-ahead thread might be canceled by MDS, or client already has the UPDATE lock in local cache before stat-ahead. To solve this, for stat-ahead thread, the dirent stated count includes both those stated by it, and those already in local cache. And for client lookup, each result it gets in local cache, it regards it as hit.

# 4 Use cases

## 4.1 user issues ‘ls -l‘

1. the process of ‘ls -l‘ open(“.”), and in this place the pid is logged.
2. It then calls getdents(), which finally calls ll_readdir().
3. stat(“file1”) is called, which calls ll_lookup_it(), herein it checks current pid, if it’s the same as the pid of that in openning parent dir, and this entry is the first entry of its parent dir, a ll_sa thread is created to do metadata stat-ahead for its parent dir.
4. ll_sa thread keeps stat the remaining dir entries, until it is max_statahead ahead of the ‘ls -l‘ process.
5. the next stat(“file2”) is called by ‘ls -l‘ process, if it finds a ‘ll_sa’ thread has been started, and the stated count is 0 yet, it will queue itself in the waitqueue of ‘ll_sa‘ thread.
6. later ll_sa thread gets reply, and wake up the stat(“file2”) process, ll_lookup_it() will check whether dentry->d_inode is NULL, if not, it returns immediately(as means ll_sa thread has filled inode informations there), otherwise it will go on sending RPC to getattr as before.
7. step 5 is repeated until dir is closed, or the ll_sa thread hits the end of the parent dir, and at this moment the ‘ll_sa‘ thread will exit spontaneously.

# 5 Logic specification

## 5.1 struct md_statahead_info

A new structure md_statahead_info is introduced to store stat-ahead related infomations:

```
struct md_statahead_info {
        struct inode             *sai_inode;          /* inode */
        spinlock_t                 sai_lock;
        unsigned int              sai_count;          /* stat-ahead count */
        atomic_t                   sai_stated;         /* stated count */
        atomic_t                   sai_hit;              /* stat hit count */
        atomic_t                   sai_miss;           /* stat miss count */
        struct ptlrpc_thread   sai_thread;        /* stat-ahead thread */
        void                          *sai_cb;              /* callback for async enque
};
```

This structure is part of ll_inode_info, and is created for dir inode when client start to do stat-ahead:

```
struct ll_inode_info {
        ...
        pid_t                               lli_opendir_pid;
        struct ll_statahead_info *lli_sai;
};
```

### 5.2 ll_sa thread

This thread will stat dir entries in a loop, and all the statistics are stored in struct ll_statahead_info.

### 5.3 MDC support async enqueue lock

Add struct mdc_enqueue_args for async enqueue lock in MDC:

```
struct mdc_enqueue_args {
        struct obd_export            *ma_exp;
        struct obd_enqueue_info  *ma_ei;
        struct lookup_intent          *ma_it;
        struct lustre_handle           ma_lockh;
        struct mdc_op_data         *ma_data;
        union  {
                struct md_statahead_info *sai;
        } u;
};
```

Since ldlm_cli_enqueue already support async enqueue, we just need to rearrange mdc_intent_lock() and mdc_enqueue_lock() code, and reuse most part of their code. And at last the request will be added to ptlrpcd set.

## 6 State management

N/A

## 7 Alternatives

- Async getattr could use a separate ptlrpc set instead of ptlrpcd to send requests.

# 8 Focus for inspections

N/A