Feed Use Cases and API

Sun Microsystems

1/25/07

1 Use Cases

1.1 Audit

A particular site policy requires audit logs of filesystem usage (access, create, delete, write, etc) and errors (specifically permission failures, perhaps quota overrun). Files A and B are on MDT0001, where the sysadmin has set up a Lustre audit feed with mask ALL_FILES|ALL_EVENTS. User on client 1 opens file A, reads file A, removes file A. User on client 2 attempts to open file B, but fails with permission denied. The audit feed, presented as a file under /mnt/MDT0001/.lustre/audit, is updated synchronously with event records A/open/ok, A/read/ok, A/delete/ok, B/open/EACCESS. The feed is read, translated into human-readable form, and piped into a regular file in /var/logs/audit where it is periodically purged by a logging daemon.

The requirements of this scenario are:

- audit log, including not only filesystem changes but also access events, atime changes, request failures.
- feed based on audit log
- feed set up (filter, retention policy)

1.2 Database

An external database is to be updated with filesystem changes for customer-specific purposes (audit, query, HSM, etc.). An audit feed is set up on each server; the feed consumer sends entires to the database backend. The filesystem events are integrated into database, even in the event of power loss and recovery.

Entries are removed from the audit feed only after the feed consumer has indicated completion (database integration) of the entry.

Cross-server synchronization required is not required; if a single event results in two changelog entries on two servers, these need not be reconciled/recombined before submission to the consumer. However, a common identifier will indicate linked entries. For example, renaming a file from MDT0001 to MDT0002 will result in a changelog entry on each server; these will share a UUID so that the consumer (database) can act appropriately.

The requirements of this scenario are:

- audit log, including not only filesystem changes but also access events, atime changes
- feed based on audit log
- feed set up (filter, retention policy)
- shared UUID for compound transaction
- full recovery semantics on feed

2 Feed API

2.1 Feed API

Feeds provide userspace access to a specific changelog. Multiple feeds may all be based on the same server changelog (e.g. using different filters). Multiple user processes (consumers) may access a feed. Feeds are transactional and persistent; feed entries are guaranteed to be replayable in the event of a server restart, from the point where the consumer last indicated completion.

2.1.1 Feed content

Feed entries will be packed binary data, with the form

```
struct feed_entry {
    _u32 fe_len;        total record length
    _u32 fe_type;       transaction type
    _u64 fe_seq;        local feed sequence number
    _u64 fe_cookie;     synchronization cookie (for distributed events)
    _u64 fe_time;       event time, server-local
    _u32 fe_result;     return code (0=success)
    void *fe_data;      transaction type-specific struct
```

Transaction types:

```
enum {create, unlink, open, close, read, write, attrib, rename, link, admin}
```

Transaction type-specific struct contains event-specific data. For example:

```
struct feed_entry_open {
    ll_fid fid;          (see lustre_idl.h)
    _u32 fsuid;
    _u32 fsgid;
    _u32 cap;
    _u32 flags;
    _u32 mode;
    _u32 filename_len;
    _u32 clientname;
    char *filename;
    char *clientname;
```

Feed content examples (expressed in human-readable form, produced by a Sun provided feed consumer demo utility):

```
logid=1 cookie=0 type=OPEN rc=0 name=/etc/passwd fid=23a87346:003d source=cli1@tcp0
logid=2 cookie=0 type=UNLINK rc=-EACCESS fid=23a87346:003d source=cli2@tcp0 uid=joe
```

2.1.2 Feed setup

New feeds are defined through lfs or a direct call to the liblustre c library (llapi).

```
int llapi_audit_init(char *fileset, struct audit_policy *policy)
```

starts a new audit feed. <fileset> is a previously defined fileset or a server name. Filesets are defined via the Fileset API. If <fileset> is not available locally, ENOENT is returned. Special user permissions are required to start an audit log; else EPERM is returned. The new feed is created as $MNT/.lustre/audit/<feedname>, where <feedname> is <fileset>[_xx], with increasing numerical xx if the name already exists.

<policy> is a structure containing

```
struct audit_policy {
```

```
_u32 ap_filtermask;
_u32 ap_entry_timeout;
_u32 ap_abort_timeout;
int  ap_canceltype;
int  ap_flags;
```

<filtermask> is a bitwise event mask:

mask bit	description
AF_CREATE	new file creation
AF_WRITE	file modify/append
AF_READ	file read
AF_OPEN	open/close
AF_ATTRIB	file attribute / EA change
AF_DELETE	file removal
AF_LINK	soft/hard link
AF_RENAME	rename
AF_FILE	all of the above (shortcut)
AF_ADMIN	administrative event
AF_ERR	report failed requests also

Retention policies (see Feed I/O below) may include:

- entry_timeout X - automatically cancel each feed record after X seconds (0=off).
- abort_timeout X - abort recording and destroy the feed after X second timeout (0=off).
- canceltype - define the cancellation policy of the feed entries; i.e. when an entry can be removed from the feed.
  - AC_ONESHOT (default) - after an entry has been read, it may be cancelled when the consumer starts reading the *next* entry. (The continuing read implies the previous read has been fully processed). There should only be a single reader with this policy (a second open(2) call should return EALREADY.)
  - AC_BATCH - make all feed entries available to consumers as they come in. One of the consumers at some point explicitly batch cancels records (see below). (Entries must be locked during read.)
- flags
  - AFG_RATELIMIT - don't report multiple consecutive similar entries.

The feed setup remains persistent across reboots (in e.g. a feed database), until it is explicitly destroyed:

```
int llapi_audit_destroy(char *feedname)
```

Feed setup info is available for retrieval from an existing feed:

```
int llapi_audit_getinfo(char *feedname, struct audit_policy *policy)
```

2.1.3 Feed I/O

The feed output data stream looks like a regular file under $MNT/.lustre/audit/<feedname>. A feed may be open(2)ed by one or more readers. Feed entries are retrieved using read(2) on the file. Feed entries are removed from the feed depending on the cancellation policy:

AC_ONESHOT: the filesystem will indicate that only a single entry's worth of data is available. When that entry has been read, a subsequent read(2) or file close(2) indicates that the feed consumer has completed processing of the previous entry, and that entry is removed (the reference to the changelog is dropped).

AC_BATCH: the filesystem will indicate all available entries. The entries are not removed until explicitly cancelled by write(2)ing the last committed logid value (followed by a newline) back into the file descriptor. Multiple readers are allowed; the largest last-committed value written controls entry removal.

Upon recovery, we restart the feed from the first uncancelled entry. The feed consumers are responsible for skipping any replayed feed entries they may have already processed (identified by repeated logid).