Network Working Group Matt Benjamin
Internet-Draft Linux Box Corporation
Intended status: Informational December 10, 2011
Expires: June 10, 2012

AFS Callback Extensions (Draft 14)

draft-benjamin-extendedcallbackinfo-02

Abstract

AFS cache-control strategy is callback (invalidate) based. The AFS callback design allows a client to know when an object it has cached is no longer consistent, but the callback notification message itself provides no specific information about the triggering event. This is a protocol inefficiency, as in several scenarios it results in unnecessary round-trips to file servers to verify file status information, file access information, or to fetch file data which has not changed. We propose an extension of the callback mechanism to provide information about the event(s) triggering a callback, in the payload of the callback notification message itself. The proposed mechanism eliminates most or all unnecessary round-trips imposed by the current callback mechanism, and simultaneously allows AFS implementations to (efficiently) provide correct semantics in several scenarios involving multiple writers (ie, where AFS currently provides incorrect semantics).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Table of Contents

1 Introduction

The AFS protocol provides a comprehensive framework for scalable, secure, wide-area file sharing over IP networks. The AFS system has historically distinguished itself through its emphasis on client-side caching[3, 7]. File data, file and directory metadata, and access control information may all be cached. Cache consistency is maintained through client registration and an associated asynchronous notification mechanism known as the callback.

The current AFS consistency model (which is of larger scope than the callback mechanism, eg, it includes AFS sync-on-close semantics) has allowed AFS to scale to large numbers of clients (tens of thousands today), and to perform well under the workloads for which AFS was originally designed.

However, AFS does not perform efficiently under other conditions, such as when more than one client is interested in a file which is changing--even if the file has only one writer, and many readers[footnote:
NFSv4.1 in particular efficiently supports this scenario with byte-range delegation, see[9].
]. In general, the AFS protocol arguably (still, considering improvements made between AFS-2 and AFS-3) places too little emphasis on efficient caching of mutable data. The current AFS consistency model is insufficient to correctly support single-file, multiple-writer scenarios, including those required for POSIX semantics, and therefore is insufficient to support many applications which may be run correctly on competing distributed file systems (e.g., CIFS, Novell Netware, or NFSv4).

The efficiency of the current AFS cache management algorithm can be substantially improved if specific triggering event information and current status are included in the payload of the callback notifications sent to clients. In particular, inclusion of the current DataVersion number and affected byte ranges in response to StoreData operations significantly reduces the need for cache revalidation and reconstruction traffic in response to callbacks--in many cases, altogether. These changes allow efficient support for single-writer updates on a file with multiple readers. More importantly, they permit AFS to correctly and efficiently support multiple writers updating disjoint ranges on a single file, a prerequisite for supporting granular file locking (and applications which require it) in future.

2 Conventions Used in this Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

3 The AFS Callback Mechanism

3.1 Description

When an AFS-3 client contacts a file server to perform any of several operations on a file, or explicitly to fetch its status, the file server includes in its RPC response an AFSCallBack structure, representing the server's promise to call back the client "if any modifications are made to the data in the file." (A key paper on AFS-2 has "before allowing a modification by any other workstation".) The AFSCallBack structure contains the callback expiration time, and two integer values treated as invariants.

When any client executes an operation which would change a file (e.g., StoreData), and in a variety of other situations, the file server invalidates the client's cached copy by executing a call to the CallBack procedure in the client's RPC interface. (The call includes in its arguments an AFSCallBack structure for each file being invalidated. However, the value of the passed AFSCallBack is unused [e.g., afs/afs_callback.c:643 ff., openafs-1.5.54]). Between the time of issue and either expiry or receipt of a callback, the client may consider any information it has cached on a file to be consistent with the file server's on-disk copy. Conversely, on receipt of a callback, the client must consider that it knows nothing about the file. Thus the client must re-establish a relationship with the file at the file server before executing any further operations on it.

The AFS callback mechanism obviates the need for clients to send frequent cache validation requests before performing operations on their locally cached copies of objects, reducing network traffic as well as file server workload[3, 7]. The callback innovation has been since taken up, with variations, by other distributed file system protocols[4, 6].

3.2 Analysis

The AFS callback mechanism reliably notifies clients when information they may have cached becomes invalidated, but omits to send information it trivially knows, ie, the triggering event, that could certainly be used by the client to more efficiently manage cache state.

For example, consider the case where 2 clients A and B are interested in a file F, each having read chunks 1-15 into cache. Now another client C initiates a change in the file, writing a new state to chunk 45. This event increments the dataversion of the file, and triggers a callback to A and B. (C, because it initiated the change, is not called back.) On receipt of the callback, A and B must issue FetchStatus requests on F to acquire its current status information, including its current data version. Since the data version of F has increased, any chunks of F which A or B has cached are invalidated, including 30 chunks correctly cached. Should A or B remain interested, it must refetch these chunks (up to 2 megabytes of data, in this case). This scenario will occur reasonably often in environments where mutable data is common, and a related scenario involving directory entries (omitted for brevity) is much more common. In these cases, an AFS callback mechanism capable of sending triggering event information with the callback would have facilitated a more efficient result, at small marginal cost. In another set of scenarios where a client A has changed data in a file invalidated by non-overlapping stores by B, a revised mechanism would be capable of delivering a correct result, whereas a correct result would be impossible with the mechanism in AFS today. (In the AFS-3 callback model, either As or Bs changes must be rejected. In the extended callback model, the range-based invalidate mechanism means that As and Bs changes will be merged, as they are disjoint.)

The justification for sending minimal information with the callback is presumably to minimize the execution cost of the callback procedure. The increased cost of sending a limited but informative callback notification to clients, relative to sending an uninformative one, is small. Analysis of the OpenAFS file server code reveals that the file server always has the information that would logically be sent as extended callback information in response to file operations (e.g., file ranges affected by StoreData operations, or changed entries for various directory modification operations).

For these reasons, enhancement of the AFS callback interface to supply triggering event information seems likely to improve both correctness and performance of AFS implementations, and experimental implementation and profiling appear justified.

4 Extended Callback Interface

4.1 Backward Compatibility

AFS clients will indicate their preference to receive extended callback notifications through a new client capability flag:

```
const CLIENT_CAPABILITY_EXT_CALLBACK = 0x0002;
```

4.2 Interface Changes

4.2.1 Procedures

We propose a new procedure ExtendedCallBack in the client's RPC interface. The ExtendedCallBack procedure arguments consist of a HostIdentifier containing the UUIDs of the sending fileserver and of its cell, and a (variable-length) sequence of AFSXCBInvocation structures. And AFSXCBInvocation represents a (variable-length) sequence of AFSExtendedCallBack events on one AFSFid at Server. One invocation of the ExtendedCallBack procedure can thus deliver up to AFSXCBMAX event notifications on each of up to AFSXCBMAX fids. An OUT-direction sequence of variant AFSExtendedCallBackResult structures is added for future callback notification styles (e.g., locks, delegations) which may return structured data on receipt of notifications:

```
proc ExtendedCallBack(

    IN HostIdentifier *Server,

       AFSXCBInvocationSeq *Invocations_Array,

    OUT AFSXCBRInvocationSeq *Result_Array

) multi = 65540;
```

4.2.2 Constants<sub:Constants>

All Sequences

The AFSXCBMAX constant is the maximum allowed length for AFSXCBInvocation and AFSExtendedCallBack sequences:

```
const AFSXCBMAX = 512;
```

AFSXCBInvocation

As detailed in section [sub:Data-Types], an AFSXCBInvocation is a structure representing a sequence of XCB events on one Fid. The following constants are flag values are used as flag values on AFSXCBInvocation instances:

```
const AFSCB_IFlag_SOrigin = 1; /* callbacks on this invocation
have a single origin host */

const AFSCB_IFlag_Release = 2; /* this invocation was triggered
by a volume release */
```

AFSExtendedCallBack

As detailed in section [sub:Data-Types], an AFSExtendedCallBack is a structure representing an XCB event, and is principally constituted by an instance of an XDR union, discriminated on the callback event type. The following callback event types are defined:

```
const AFSCB_Event_Cancel = 1; /* extended break callback */

const AFSCB_Event_StoreData = 2; /* data in file changed */

const AFSCB_Event_StoreACL = 3; /* ACL changed on vnode */

const AFSCB_Event_StoreStatus = 4; /* status stored on vnode */

const AFSCB_Event_CreateFile = 5; /* file created in directory
vnode */

const AFSCB_Event_MakeDir = 6; /* dir created in directory
vnode */

const AFSCB_Event_Symlink = 7; /* symlink created in directory
vnode */

const AFSCB_Event_Link = 8; /* hard link created in directory
vnode */

const AFSCB_Event_RemoveFile = 9; /* file removd from directory
vnode */

const AFSCB_Event_RemoveDir = 10; /* dir removed from directory
vnode */
```

```
const AFSCB_Event_Rename = 11; /* object renamed (moved) */

const AFSCB_Event_Deleted = 12; /* object no longer exists, ex
object */

const AFSCB_Event_ReleaseLock = 13; /* traditional AFS lock
released */
```

A flag constant is provided to indicate callback cancellation along with an extended notification message of any of the above types:

```
const AFSCB_Flag_Cancel = 1; /* Callback promise is cancelled
*/
```

The following constants indicate reasons for cancellation, when (Flags & AFSCB_Flag_Cancel):

```
const AFSCB_Cancel_Shutdown = 1;

const AFSCB_Cancel_CallbackGC = 2;

const AFSCB_Cancel_VolumeOffline = 3;

const AFSCB_Cancel_VolumeMoved = 4;

const AFSCB_Cancel_LostMyMind = 5;

const AFSCB_Cancel_IHateYou = 6;
```

The following flag constants (Flags) indicate the beginning or end of a callback bracketing sequence (each is orthogonal to each other and to AFSCB_Flag_Cancel):

```
const AFSCB_Flag_OpenBracket = 2; /* Callback syncrhony is
interrupted */

const AFSCB_Flag_CloseBracket = 4; /* Callback syncrhony is
restored */
```

The following constants indicate direction (from or to called back FID) in the atomic AFSCB_Event_Rename notification:

```
const AFSCB_Rename_From = 1;

const AFSCB_Rename_To = 2;
```

As detailed in section [sub:Data-Types], an AFSRXCBInvocation is a structure representing the sequence of results of XCB events delivered in one AFSXCBInvocation.

AFSExtendedCallBackResult

As detailed in section [sub:Data-Types], an AFSExtendedCallBackResult is a structure describing the result of an XCB event. The following constants a provided as descriminator for the AFSCB_ResultData union:

```
const AFSCB_Result_NoResult = 1;

const AFSCB_Result_Diag = 2;

const AFSCB_Result_Generic = 3; /* all currently defined
messages */
```

Sequence Types

The following sequences are defined, and are used to construct the input and output arguments for the ExtendedCallBack procedure:

```
typedef AFSXCBInvocation AFSXCBInvocationSeq<AFSXCBMAX>;

typedef AFSExtendedCallBack AFSExtendedCallBackSeq<AFSXCBMAX>;

typedef AFSXCBRInvocation AFSXCBRInvocationSeq<AFSXCBMAX>;

typedef AFSExtendedCallBackResult
AFSExtendedCallBackRSeq<AFSXCBMAX>;
```

4.2.3 Data Types<sub:Data-Types>

HostIdentifier

A HostIdentifier structure contains the unique server and cell UUIDs of a specific host in some AFS cell.

```
struct HostIdentifier {

    afsUUID ServerUuid;

    afsUUID CellUuid;

};
```

AFSXCBInvocation

The AFSXCBInvocation data type represents a sequence of 0 or more callback events on one fid. The enclosed AFSExtendedCallBack objects MUST be in DataVersion order.

  Fid

Fid is the fid object of the callback sequence.

  Flags

Flags provide specializing information about the invocation.

  lowDV

The lowest data version of Fid at all events in the sequence.

  highDV

The highest data version of Fid at all events in the sequence.

  ExpirationTime

ExpirationTime indicates a new expiration time for the receiving client's callback on fid. And ExpirationTime of 0 indicates no change in ExpirationTime.

  CallBacks_Array

A sequence of 0 or more AFSExtendedCallBack notifications on FID.

```
struct AFSXCBInvocation {

    AFSFid Fid;

    afs_uint32 Flags;

    afs_uint64 lowDV; /* lowest DV at invocation */

    AFSTimestamp highDV; /* highest */

    AFSTimestamp ExpirationTime; /* new expiration, or 0 if
unchanged */

    AFSExtendedCallBackSeq CallBacks_Array;

};
```

  AFSExtendedCallBack

The AFSExtendedCallBack data type represents a single callback event on some fid, that of its containing AFSXCBInvocation when sent with an ExtendedCallBack RPC.

Flags and ExtraFlags

Flags and ExtraFlags (added for future expansion) provide possibly event-specific information.

Origin

Origin is the AFS UUID of the host or server which originated the event, ie, the client whose operation on fid triggered some event, in the typical case. If the origin is unknown to the server or would not be meaningful, it MAY send the null UUID.

NCoalesced

As specified later in this document, certain operations (ie, StoreData, StoreStatus) MAY be regarded by the file server as idempotent and sent as one callback. NCoalesced indicates the number of equivalent or combined operations coalesced on the event, or 0 if the event is singular.

DataVersion

DataVersion is the (possibly updated) data version of fid at the completion of the operation which triggered the event. Considering coalescing, DataVersion is the data version at the completion of the first event in the coalesced sequence.

Data

Data is an object of the discriminated union type AFSCB_NotificationData:

```
struct AFSExtendedCallBack {

    afs_uint32 Flags;

    afs_uint32 ExtraFlags;

    afsUUID Origin; /* originator of changes */

    afs_uint32 NCoalesced; /* calls combined on this */

    afs_uint64 DataVersion;

    AFSCB_NotificationData Data;
```

```
};
```

A non-zero value in Flags for the AFSCB_Flag_Cancel bit indicates cancellation of the callback upon receipt of the message. In that event, a non-zero value of ExtraFlags indicates the reason for the cancellation.

AFSXCBRInvocation

An AFSXCBRInvocation is a structure describing the result of an XCB invocation (a sequence of extended callback events at one Fid).

```
struct AFSXCBRInvocation {

    AFSExtendedCallBackRSeq Xcb_Result_Array;

};
```

AFSExtendedCallBackResult

An AFSExtendedCallBackResult is a structure describing the result of a single XCB event.

```
struct AFSExtendedCallBackResult {

    afs_uint32 Flags;

    afs_uint32 ExtraFlags;

    AFSCB_ResultData Data;

};
```

AFSCBFileStatus

The AFSCBFileStatus structure is a reduced-footprint AFSFetchStatus replacement intended to communicate changed vnode information in response to StoreData operations:

```
struct AFSCBFileStatus {

    afs_uint32 LinkCount;

    afs_uint64 ClientModTime;

};
```

AFSCBDirStatus

The AFSCBDirStatus structure is a reduced-footprint AFSFetchStatus replacement intended to communicate changed vnode information in response to directory change operations:

```
struct AFSCBDirStatus {

    afs_uint32 LinkCount;

    afs_uint64 ClientModTime;

};
```

AFSCB_NotificationData

AFSCB_NotificationData is a union discriminated by callback event type, ie, its value may be any of the constants defined in section [sub:Constants].

```
ext-union AFSCB_NotificationData switch (afs_uint32 Event_Type)
{

case AFSCB_Event_StoreData:

    AFSCB_Data_StoreData u_store_data;

case AFSCB_Event_StoreACL:

    void;

case AFSCB_Event_StoreStatus:

    AFSCB_Data_StoreStatus u_store_status;

case AFSCB_Event_CreateFile:

    AFSCB_Data_CreateFile u_create_file;

case AFSCB_Event_MakeDir:

    AFSCB_Data_MakeDir u_make_dir;

case AFSCB_Event_Symlink:

    AFSCB_Data_Symlink u_symlink;

case AFSCB_Event_Link:

    AFSCB_Data_Link u_link;
```

```
case AFSCB_Event_RemoveFile:

    AFSCB_Data_RemoveFile u_remove_file;

case AFSCB_Event_RemoveDir:

    AFSCB_Data_RemoveDir u_remove_dir;

case AFSCB_Event_Rename:

    AFSCB_Data_Rename u_rename;

case AFSCB_Event_Deleted:

    void;

case AFSCB_Event_ReleaseLock:

    AFSCB_Data_Lock u_lock;

case AFSCB_Event_Cancel:

    void;

};
```

The types for the variant member u_data are enumerated and discussed in detail in section [sub:Callback-Invocations].

4.3 Semantic Changes

A file server MAY send traditional callback messages, with traditional semantics, to any AFS client in response to any event. A file server MAY send extended callback notifications to any client which has announced the capability to use the extended interface, with the following semantics:

* extended callback notification messages, in general, preserve the file server's callback promise to send further notifications for the called-back FID

* the file server MAY cancel a client's registration (callback promise) with any extended callback notification message, by setting AFSCB_Flag_Cancel in the Flags member of the AFSExtendedCallBack structure

* the AFSCB_Event_Cancel message is similar to a traditional AFS callback, breaking the callback promise, and requesting the client not request further status on the FID

* the file server MAY suspend its obligation to deliver callback notifications synchronously without revoking an object's registration, by setting AFSCB_Flag_OpenBracket in the Flags member of the AFSExtendedCallBack structure

* a file server which as previously suspended synchronous notification on an object (with AFSCB_Flag_OpenBracket) MAY restore it by sending AFSCB_Flag_CloseBracket in Flags in a subsequent message

  * callback bracketing, if enabled, terminates automatically on ordinary cancellation or expiration of the corresponding callback registration

4.3.1 DataVersion Rule

The various extended callback notification messages include information a client may use to selectively invalidate or reconstruct its cache. In interpreting each message, the client MUST observe the dataversion rule, which states:

If the client's cached DataVersion is DataVersion or (DataVersion-1), the client MAY invalidate or update its cache using the type-dependent information contained in the message. In all other cases, the client MUST regard the message as equivalent to a traditional AFS callback.

The semantics of specific callback events are enumerated in section [sub:Callback-Invocations].

4.3.2 Callback Synchrony and Callback Bracketing

The AFS prototol regards client data and metadata operations as atomic and synchronous. A legacy AFS callback registration establishes a domain of synchrony between the registering file server(s) and each registered client. A client operating on any object in the file server interface is assured that each eligible client has been notified of its operation when the operation completes.

The extended callback mechanism carries forward these semantics in full, with the following modification, referred to as callback bracketing: A synchronous notification may suspend the sending server's obligation to deliver synchronous notifications without otherwise removing the receiving client's registration on a Fid. The motivation for callback bracketing is to permit more efficient processing of sequential operations at one or several clients, through callback coalescing, without violating any client's expectation of synchrony. Since an AFSCB_Flag_OpenBracket indication is delivered synchronously to any client which would be assured to receive an ordinary legacy or extended callback notification with synchrony, in any circumstance in which a server would be obligated to send callback breaks or notifications, the callback bracketing mechanism does not violate any client's expectation of synchrony.

4.3.3 Callback Workload Considerations

The preference to preserve the fileserver's callback promises to clients across file operations is a significant behavioral change. In particular, file servers configured with traditional small callback databases may be vulnerable to callback exhaustion, which in turn may lead to thrashing or other undesirable behavior. Sites are recommended to explicitly increase provisioned callbacks, likely to at least 1 million. The OpenAFS viced implementation will increase the number of callbacks in the "large" configuration (-L) to a new, large value. In general, a file server is expected take appropriate action to shed workload (e.g., break callbacks) whenever appropriate.

4.4 Callback Invocations<sub:Callback-Invocations>

The various extended callback notification types generally respond to specific events at the file server, but present a view of it relevant to a specific callback promise at one client. In one case (ie, AFSCB_Event_Rename), the file server is sending notification of an event which effects two FIDs, either or both of which may be cached by the receiving client. A structure of type AFSExtendedCallBack is sent with each extended callback notification message, as noted above. Unless otherwise noted, FID is the FID of the object that is the subject of the callback.

4.4.1 AFSXCBInvocation

Origin

A file server MAY omit to send extended callback notifications triggered by a file operation to the client host which originated the change. (Omission to send such callbacks has been the general behavior of AFS file servers.) A client MUST be prepared to appropriately process (or ignore) callbacks for which its own UUID is the Origin.

ExpirationTime

The new expiration time asserted for the server's callback promise, not necessarily different from the existing expiration cached by the client.

4.4.2 AFSExtendedCallBack

The members of the AFSExtendedCallBack structures are to be interpreted as follows:

Flags

If AFSCB_Flag_Cancel is set, the notification effects a callback break. The client may make use of the information sent with the message. If AFSCB_Flag_OpenBracket is set, the notification (if actionable with respect to DataVersion) suspends the server's obligation to send synchronous notifications on FID. The client may make use of the information sent with the message. The new state persists until restored by a subsequent AFSCB_Flag_OpenBracket notification or client-initiated callback-granting operation.

ExtraFlags

If (Flags & AFSCB_Flag_Cancel), a non-zero value for ExtraFlags indicates the reason for cancellation.

DataVersion

The value of DataVersion at completion of the event of which the client is being notified. Considering coalescing, the new data version after completion of all events summarized at this callback is (DataVersion+NCoalesced).

Data

The message-specific data for this notification.

4.4.3 AFSXCBRInvocation

The AFSXCBRInvocation structure contains only an array of AFSExtendedCallBackResult.

Xcb_Result_Array

A sequence of AFSExtendedCallBackResult objects dimensioned by the matching Callbacks_Array argument of an ExtendedCallBack invocation.

4.4.4 AFSExtendedCallBackResult

An AFSExtendedCallBackResult is a structure describing the result of a single XCB event.

Flags

Provided for future extension.

ExtraFlags

Provided for future extension.

  Data

The result, expressed as a union of nil, diagnostic/string, and generic result types. The generic result type (or future extension) should be used in general, as it allows for per-message acknowledgement.

4.4.5 ExtendedCallBack Procedure

Extended callbacks are delivered through a new ExtendedCallBack procedure.

```
proc ExtendedCallBack(

    IN HostIdentifier *Server,

       AFSXCBInvocationSeq *Invocations_Array,

    OUT AFSXCBRInvocationSeq *Result_Array

) multi = 65540;
```

ExtendedCallBack provides for flexible event notification, including bulk notification support by Fid and per Fid, supports per-message acknowledgement, and uniquely identifies the issuing server host.

4.4.6 Callback Coalescing

A server implementation MAY coalesce sequences of effectively-simultaneous notifications to a single client, in accordance with rules of composition of specific notifications, and provided doing so would not violate any client's expectation of synchrony.

  Call Consolidation (Sequences of Notifications)

A server implementation MAY coalesce any sequence of effectively simultaneous notifications into sequences of AFSExtendedCallBack objects enclosed in one AFSXCBInvocation object, provided doing so would not violate any client's expectation of synchrony. Any number of such callbacks may be combined, up to the limit of AFSXCBMAX.

  Coalescing of Equivalent Notifications

A server implementation MAY coalesce a sequence of effectively simultaneous and equivalent notifications to the same client into a single callback in a notification message, provided doing so would not violate any client's expectation of synchrony. The following combinations of operations are explicitly permitted:

* sequences of AFSCB_EventStoreAcl notifications on FID from a single Origin MAY be delivered as a single notification

* sequences of AFSCB_EventStoreStatus notifications on FID from a single Origin MAY be delivered as the single notification of the most recently stored status

* sequences of AFSCB_Event_StoreData notifications on FID from a single Origin at adjacent or overlapping byte ranges MAY deliver a single notification at the consolidated range

Implementation Note

"Effectively simultaneous" is left intentionally unspecified. An adaptive window expanding from 100ms to a small number of seconds appears to work well with commonly available switched networks and multi-core fileservers, in 2008. The current OpenAFS implementation uses an adaptive per-Fid window, which extends on repeated events, and closes when client contention is detected. Peer RTT or other considerations may be added to the windowing algorithm in future.

4.4.7 AFSCB_Event_Cancel

The AFSCB_Event_Cancel notification indicates that the client's callback promise on the corresponding Fid is cancelled. It is therefore equivalent to a legacy AFS break call back indication, but uses the extended interface. A cancel indication may include an optional reason for cancellation in the Flags member of the corresponding AFSExtendedCallBack message.

Reasons for Cancellation

The following reasons for cancellation are defined:

AFSCB_Cancel_Shutdown

The server or service is shutting down.

AFSCB_Cancel_CallbackGC

Callback has been disposed during periodic garbage collection.

AFSCB_Cancel_VolumeOffline

The volume associated with FID is now offline.

AFSCB_Cancel_VolumeMoved

The volume associated with FID has moved.

AFSCB_Cancel_LostMyMind

The server may be having problems related to provisioning an insufficient number of callback structures.

AFSCB_Cancel_IHateYou

Callback has been administratively revoked.

4.4.8 AFSCB_Event_StoreData

The notification is sent in response to a successful StoreData RPC on FID. A structure of type AFSCB_Data_StoreData is sent with the message.

```
struct AFSCB_Data_StoreData {

    afs_uint64 StoreOffset;

    afs_uint64 StoreLength;

    afs_uint64 Length;

    AFSCBFileStatus FileStatus;

};
```

StoreLength bytes were stored starting at position StoreOffset in FID. Length is the current file length and FileStatus contains the modification time of FID following the operation. The client must regard cached file data in the range [StoreOffset, StoreOffset+StoreLength) as invalidated, and may regard data outside that range as up-to-date. The client MUST discard undirtied cached data in the invalidated range. The client MAY send dirtied data in the invalidated range to the file server prior to discarding (as allowed in current AFS semantics).

4.4.9 AFSCB_Event_StoreACL

ACL and/or access information cached by the client for FID, if any, is invalidated.

4.4.10 AFSCB_Event_StoreStatus

A StoreStatus RPC was successfully executed on FID. A structure of type AFSFetchStatus is sent with the message.

```
struct AFSCB_Data_StoreStatus {

    struct AFSFetchStatus Status;

};
```

Status is the new AFSFetchStatus of FID, ie, the message communicates the current status information of FID.[footnote: This is changed from earlier drafts.
]

4.4.11 AFSCB_Event_CreateFile

A file has been created in the vnode corresponding to FID. A structure of type AFSCB_Data_CreateFile is sent with the message.

```
struct AFSCB_Data_CreateFile {

    string Name<AFSNAMEMAX>;

    AFSFid Fid;

    AFSFetchStatus FidStatus;

    AFSCBDirStatus DirStatus;

};
```

Name and Fid are, respectively, the name and FID of the created file. FidStatus is the AFSFetchStatus of the created file, and DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.12 AFSCB_Event_MakeDir

A directory has been created in the vnode corresponding to FID. A structure of type AFSCB_Data_MakeDir is sent with the message.

```
struct AFSCB_Data_MakeDir {

    string Name<AFSNAMEMAX>;

    AFSFid Fid;

    AFSFetchStatus FidStatus;

    AFSCBDirStatus DirStatus;

};;
```

Name and Fid are, respectively, the name and FID of the created directory. FidStatus is the AFSFetchStatus of the created directory, and DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.13 AFSCB_Data_Symlink

A symbolic link has been created in the vnode corresponding to FID. A structure of type AFSCB_Data_Symlink is sent with the message.

```
struct AFSCB_Data_Symlink {

    string Name<AFSNAMEMAX>;

    AFSFid Fid;

    string LinkContents<AFSPATHMAX>;

    AFSFetchStatus FidStatus;

    AFSCBDirStatus DirStatus;

};
```

Name is the name of the symbolic link. Fid is its AFSFid. The link points to LinkContents. FidStatus is the AFSFetchStatus of the created symbolic link, and DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.14 AFSCB_Event_Link

A hard link has been created in the vnode corresponding to FID. A structure of type AFSCB_Data_Link is sent with the message.

```
struct AFSCB_Data_Link {

    string Name<AFSNAMEMAX>;

    AFSFid LinkTarget;

    AFSFetchStatus FidStatus;

    AFSCBDirStatus DirStatus;

};
```

Name is the name of the link. The link is a synonym for LinkTarget. FidStatus is the AFSFetchStatus of the created link, and DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.15 AFSCB_Event_RemoveFile

A file has been removed from the vnode corresponding to FID. A structure of type AFSCB_Data_RemoveFile is sent with the message.

```
struct AFSCB_Data_RemoveFile {

    string Name<AFSNAMEMAX>;

    AFSCBDirStatus DirStatus;

};
```

Name indicates the removed entry. DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.16 AFSCB_Event_RemoveDir

A directory has been removed from the vnode corresponding to FID. A structure of type AFSCB_Data_RemoveDir is sent with the message.

```
struct AFSCB_Data_RemoveDir {

    string Name<AFSNAMEMAX>;

    AFSCBDirStatus DirStatus;

};
```

Name indicates the removed entry. DirStatus the current modification time and link count of FID, at the completion of the call.

4.4.17 AFSCB_Event_Rename

A file or directory has been renamed, ie moved, from or to the vnode corresponding to FID. A structure of type AFSCB_Data_RemoveDir is sent with the message.

```
const AFSCB_Rename_From = 1;

const AFSCB_Rename_To = 2;



struct AFSCB_Data_Rename {
```

```
    afs_uint32 Direction;

    string OldName<AFSNAMEMAX>;

    string NewName<AFSNAMEMAX>;

    AFSCBDirStatus FromStatus;

    AFSCBDirStatus ToStatus;

};
```

Direction indicates whether FID is the source or the destination directory of the move. OldName is the name of the object in its old location, NewName the name of the object in its new location. FromStatus is the current modification time and link count of the source directory vnode, and ToStatus is the current modification time and link count of the destination directory vnode, and FidStatus the at the completion of the call.

To preserve atomicity, the AFSCB_Data_Rename message is constructed so that changes to cached copies of both the source and directory vnodes may be recovered from a single notification. If a client owns callbacks for both the source and destination FIDs, a file server MAY elect to send only one notification, for either the source or the destination FID.

4.4.18 AFSCB_Event_Deleted

The object corresponding to FID not longer exists, and so may no longer be cached. It is an ex-object. (I.e., the client MUST discard any information it has cached about FID.)

4.4.19 AFSCB_Event_ReleaseLock

A traditional AFS whole-file lock has been released on FID. A structure of type AFSCB_Data_Lock is sent with the message. LockType is the type of the lock released.

```
struct AFSCB_Data_Lock {

    afs_uint32 LockType;

};
```

Receipt of an AFSCB_Event_ReleaseLock notification does not imply that a lock on FID will be immediately available to the receiving client (i.e., it is not a reservation). Non-receipt of a notification of this type does not imply non-release of locks that may be (may have been) held on FID. A file server SHOULD send notifications of this type only to clients which have indicated probable interest in the event, e.g., by having recently requested a lock on FID.

4.5 Callbacks And Read-Only Volume Replicas

Callbacks associated with read-only volume replicas have traditionally been handled specially in AFS. When any file in an RO volume is accessed the AFS file server establishes a single callback promise considered to be on the entire volume. Any event which updates the replica (e.g., vos release) triggers a whole-volume callback break. The whole-volume callback optimization significantly reduced file server memory utilization, which was at a premium in 1988. However, the whole-volume callback is less of an optimization in OpenAFS in 2008:

* modern AFS file servers have sufficient memory to track millions of callbacks (and do track up to 1 million callbacks at one site we know of, with up to 3 million callback structures available)[8]

* whole-volume callback semantics require clients (and the file server) to potentially expend considerable effort re-establishing cache consistency, and so whole-volume callbacks are necessarily a considerable protocol inefficiency for sites relying heavily on AFS replication (in particular, incremental replication now possible in OpenAFS)

For these reasons, we propose that the scope of extended callback information include notifications concerning changes that originate in the release of a volume. We provide the option for the file server to provide whole-volume or per file notifications, at its discretion. We provide the option for the file server to track client interest in specific files (ie, issue per-file callbacks on files in RO volumes), and speculate that this implementation would be preferred, but do not mandate it.

4.5.1 Constants

The following flag constants are added:

```
const AFSCB_Flag_Release = 2;

const AFSCB_IFlag_Release = 2;

const AFSCB_Release_WholeVolumeCancel = 1;
```

AFSCB_Flag_Release

In an AFSExtendedCallBack instance, (Flags & AFSCB_Flag_Release) indicates a notification in response to the (possibly incremental) release of a read-only replica.

AFSCB_IFlag_Release

In an AFSXCBInvocation instance, (Flags & AFSCB_IFlag_Release) indicates a notification in response to the (possibly incremental) release of a read-only replica.

AFSCB_Release_WholeVolumeCancel

If additionally (ExtraFlags & AFSCB_Release_WholeVolumeCancel), then the callback invalidates the entire volume, otherwise it is a selective invalidation of just the FIDs in Fids_Array.

4.5.2 Semantic Changes

An AFS file server MAY send selective or whole-volume extended callback notifications. The file server MAY choose to regard files in RO volumes equivalently to files in RW volumes, ie, effectively maintain callback state on them. Alternatively it MAY send selective notifications on any FIDs changed, removed, or added in the volume without regard to client cache state. The AFS client must handle such notifications gracefully.

5 Security Considerations

Extended callback information messages that only invalidate information that may be cached at clients have equivalent security implications to AFS-3 callback messages. This class of messages includes AFSCB_Event_Cancel and probably AFSCB_Event_StoreData. The remaining extended callback information messages (most of them) contain explicit metadata information which could potentially be used by an attacker impersonating a file server to introduce malicious information into a client cache. Rx security extensions in development (eg, rxgk) include provisions for secure transmission of callback messages.

Edinburgh Consensus

Implementations should ensure that extended callbacks which send explicit metadata use a secure communication channel. Cancellation and StoreData messages may be sent over any channel.

6 IANA Considerations

This document makes no request of the IANA.

7 Acknowledgements

Thanks to Jeffrey Altman, Tom Keiser, Jeffrey Hutzelman, Derrick Brashear, and Steven Jenkins for their feedback and suggested improvements from previous drafts. Thanks to participants at the 2009 Hackathon in Edinburgh and 2011 Hackathon in Pittsburgh for their feedback and assistance.

8 Appendix A: XDR Grammar

```
/* Cache Manager Capability Flags */

const CLIENT_CAPABILITY_EXT_CALLBACK = 0x0002;



/* Host Tracking/Extended Information */

struct HostIdentifier {

    afsUUID ServerUuid;

    afsUUID CellUuid;

};



/* Extended Callback Information */

/* callback event types, predominantly events on the vnode for

* which the callback is being made, but also (e.g., Deleted)
side

* effects of operations on related vnodes */

const AFSCB_Event_Cancel = 1;

const AFSCB_Event_StoreData = 2;

const AFSCB_Event_StoreACL = 3;

const AFSCB_Event_StoreStatus = 4;

const AFSCB_Event_CreateFile = 5;

const AFSCB_Event_MakeDir = 6;

const AFSCB_Event_Symlink = 7;
```

```
const AFSCB_Event_Link = 8;

const AFSCB_Event_RemoveFile = 9;

const AFSCB_Event_RemoveDir = 10;

const AFSCB_Event_Rename = 11;

const AFSCB_Event_Deleted = 12;

const AFSCB_Event_ReleaseLock = 13;


/* for use in AFSExtendedCallBack Flags */

const AFSCB_Flag_Cancel = 1;

const AFSCB_Flag_Release = 2;


/* intended for use in AFSExtendedCallBack ExtraFlags,

 * when (flags & AFSCB_Flag_Cancel), to indicate reason for

 * cancellation */

const AFSCB_Cancel_Shutdown = 1;

const AFSCB_Cancel_CallbackGC = 2;

const AFSCB_Cancel_VolumeOffline = 3;

const AFSCB_Cancel_VolumeMoved = 4;

const AFSCB_Cancel_LostMyMind = 5; /* ran out of callbacks? */

const AFSCB_Cancel_IHateYou = 6; /* callback administratively
revoked */


/* for use in AFSXCBInvocation Flags */

const AFSCB_IFlag_SOrigin = 1;

const AFSCB_IFlag_Release = 2;
```

```
/* flags intended for use in AFSExtendedCallBack ExtraFlags

* to indicate RO volume callback events */

const AFSCB_Release_WholeVolumeCancel = 1;


/* callback result types */

const AFSCB_Result_NoResult = 1;

const AFSCB_Result_Diag = 2;

const AFSCB_Result_Generic = 3; /* all currently defined
messages */


/* differential status to be sent with StoreData msgs */

struct AFSCBFileStatus {

    afs_uint32 LinkCount;

    afs_uint64 ClientModTime;

};


/* differential status to be sent with directory change msgs */

struct AFSCBDirStatus {

    afs_uint32 LinkCount;

    afs_uint64 ClientModTime;

};


/* variant data types for AFSCB_Notification_Data */

struct AFSCB_Data_StoreData {
```

```
    afs_uint64 StoreOffset;
    afs_uint64 StoreLength;
    afs_uint64 Length;
    AFSCBFileStatus FileStatus;
};

struct AFSCB_Data_StoreStatus {
    struct AFSFetchStatus Status;
};

struct AFSCB_Data_CreateFile {
    string Name<AFSNAMEMAX>;
    AFSFid Fid;
    AFSFetchStatus FidStatus;
    AFSCBDirStatus DirStatus;
};

struct AFSCB_Data_MakeDir {
    string Name<AFSNAMEMAX>;
    AFSFid Fid;
    AFSFetchStatus FidStatus;
    AFSCBDirStatus DirStatus;
};

struct AFSCB_Data_Symlink {
```

```
    string Name<AFSNAMEMAX>;
    AFSFid Fid;
    string LinkContents<AFSPATHMAX>;
    AFSFetchStatus FidStatus;
    AFSCBDirStatus DirStatus;
};

struct AFSCB_Data_Link {
    string Name<AFSNAMEMAX>;
    AFSFid LinkTarget;
    AFSFetchStatus FidStatus;
    AFSCBDirStatus DirStatus;
};

struct AFSCB_Data_RemoveFile {
    string Name<AFSNAMEMAX>;
    AFSCBDirStatus DirStatus;
};

struct AFSCB_Data_RemoveDir {
    string Name<AFSNAMEMAX>;
    AFSCBDirStatus DirStatus;
};

const AFSCB_Rename_From = 1;
```

```
const AFSCB_Rename_To = 2;

struct AFSCB_Data_Rename {
    afs_uint32 Direction;
    string OldName<AFSNAMEMAX>;
    string NewName<AFSNAMEMAX>;
    AFSCBDirStatus FromStatus;
    AFSCBDirStatus ToStatus;
};

struct AFSCB_Data_Lock {
    afs_uint32 LockType;
};

union AFSCB_NotificationData switch (afs_uint32 Event_Type) {
case AFSCB_Event_StoreData:
    AFSCB_Data_StoreData u_store_data;
case AFSCB_Event_StoreACL:
    void;
case AFSCB_Event_StoreStatus:
    AFSCB_Data_StoreStatus u_store_status;
case AFSCB_Event_CreateFile:
    AFSCB_Data_CreateFile u_create_file;
case AFSCB_Event_MakeDir:
    AFSCB_Data_MakeDir u_make_dir;
```

```
case AFSCB_Event_Symlink:

    AFSCB_Data_Symlink u_symlink;

case AFSCB_Event_Link:

    AFSCB_Data_Link u_link;

case AFSCB_Event_RemoveFile:

    AFSCB_Data_RemoveFile u_remove_file;

case AFSCB_Event_RemoveDir:

    AFSCB_Data_RemoveDir u_remove_dir;

case AFSCB_Event_Rename:

    AFSCB_Data_Rename u_rename;

case AFSCB_Event_Deleted:

    void;

case AFSCB_Event_ReleaseLock:

    AFSCB_Data_Lock u_lock;

case AFSCB_Event_Cancel:

    void;

};


struct AFSExtendedCallBack {

    afs_uint32 Flags;

    afs_uint32 ExtraFlags;

    afsUUID Origin; /* originator of changes */

    afs_uint32 NCoalesced; /* calls [StoreData] combined on
this */

    afs_uint64 DataVersion;
```

```
    AFSCB_NotificationData Data;
};


const AFSXCBMAX = 512;


struct AFSXCBInvocation {
    AFSFid Fid;
    afs_uint32 Flags;
    afs_uint64 lowDV; /* lowest DV at invocation */
    AFSTimestamp highDV; /* highest */
    AFSTimestamp ExpirationTime; /* new expiration, or 0 if
unchanged */
    AFSExtendedCallBackSeq CallBacks_Array;
};


typedef AFSExtendedCallBack AFSExtendedCallBackSeq<AFSXCBMAX>;


/* Forward-looking union for callback results */
struct AFSCB_Result_Data_Generic {
    afs_int32 code;
};


union AFSCB_ResultData switch (afs_uint32 Result_Type) {
case AFSCB_Result_NoResult:
    void;
```

```
case AFSCB_Result_Diag:

    string msg<30>;


case AFSCB_Result_Generic:

    AFSCB_Result_Data_Generic u_generic;

};


/* extended callback result structure */

struct AFSExtendedCallBackResult {

    afs_uint32 Flags;

    afs_uint32 ExtraFlags;

    AFSCB_ResultData Data;

};


typedef AFSExtendedCallBackResult
AFSExtendedCallBackRSeq<AFSXCBMAX>;


struct AFSXCBRInvocation {

    AFSExtendedCallBackRSeq Xcb_Result_Array;

};


typedef AFSXCBRInvocation AFSXCBRInvocationSeq<AFSXCBMAX>;


proc ExtendedCallBack(
```

```
    IN HostIdentifier *Server,

       AFSXCBInvocationSeq *Invocations_Array,

    OUT AFSXCBRInvocationSeq *Result_Array

) multi = 65540;
```

9 Informative References

References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M. "Scale and Performance in a Distributed File System" ACM Transactions on Computer Systems, February 1988

[4] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.

[5] Edward R Zayas, "AFS-3 Programmer's Reference: File Server/Cache Manager Interface", Transarc Corporation, FS-00-D162, 20th August 1991

[6] Paul J. Leach, Dilip C. Naik. A Common Internet File System (CIFS/1.0) Protocol [http://www.tools.ietf.org/html/draft-leach-cifs-v1-spec-01], 1997.

[7] Kazar, Michael Leon, "Synchronization and Caching Issues in the Andrew File System," USENIX Conference Proceedings, USENIX Association, Berkeley, CA, Dallas Winter 1988, pages 27-36.

[8] Alistair Ferguson. OpenAFS and the Dawn of a New Era. AFS and Kerberos Best Practices Workshop, 2008.

[9] Trond Myklebust. Byte Range Delegations. [https://www3.ietf.org/proceedings/05nov/slides/nfsv4-3.pdf], November 2006.

Author's Address

Matt Benjamin
Linux Box Corporation
206 S. Fifth Ave, Ste 150
Ann Arbor, MI 48104
USA
Phone: +1 734 761 4689

Email: matt@linuxbox.com