              Network File System (NFS) Direct Data Placement

Abstract

   This document defines the bindings of the various Network File System
   (NFS) versions to the Remote Direct Memory Access (RDMA) operations
   supported by the RPC/RDMA transport protocol. It describes the use
   of direct data placement by means of server-initiated RDMA operations
   into client-supplied buffers for implementations of NFS versions 2,
   3, 4, and 4.1 over such an RDMA transport.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 5741.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   http://www.rfc-editor.org/info/rfc5667.

Table of Contents

1. Introduction

   The Remote Direct Memory Access (RDMA) Transport for Remote Procedure
   Call (RPC) [RFC5666] allows an RPC client application to post buffers
   in a Chunk list for specific arguments and results from an RPC call.
   The RDMA transport header conveys this list of client buffer
   addresses to the server where the application can associate them with
   client data and use RDMA operations to transfer the results directly
   to and from the posted buffers on the client. The client and server
   must agree on a consistent mapping of posted buffers to RPC. This
   document details the mapping for each version of the NFS protocol
   [RFC1094] [RFC1813] [RFC3530] [RFC5661].

1.1. Requirements Language

   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 [RFC2119].

2. Transfers from NFS Client to NFS Server

   The RDMA Read list, in the RDMA transport header, allows an RPC
   client to marshal RPC call data selectively. Large chunks of data,
   such as the file data of an NFS WRITE request, MAY be referenced by
   an RDMA Read list and be moved efficiently and directly placed by an
   RDMA Read operation initiated by the server.

   The process of identifying these chunks for the RDMA Read list can be
   implemented entirely within the RPC layer. It is transparent to the
   upper-level protocol, such as NFS. For instance, the file data
   portion of an NFS WRITE request can be selected as an RDMA "chunk"
   within the eXternal Data Representation (XDR) marshaling code of RPC
   based on a size criterion, independently of the NFS protocol layer.
   The XDR unmarshaling on the receiving system can identify the
   correspondence between Read chunks and protocol elements via the XDR
   position value encoded in the Read chunk entry.

   RPC RDMA Read chunks are employed by this NFS mapping to convey
   specific NFS data to the server in a manner that may be directly
   placed. The following sections describe this mapping for versions of
   the NFS protocol.

3. Transfers from NFS Server to NFS Client

   The RDMA Write list, in the RDMA transport header, allows the client
   to post one or more buffers into which the server will RDMA Write
   designated result chunks directly. If the client sends a null Write
   list, then results from the RPC call will be returned either as an
   inline reply, as chunks in an RDMA Read list of server-posted
   buffers, or in a client-posted reply buffer.

   Each posted buffer in a Write list is represented as an array of
   memory segments. This allows the client some flexibility in
   submitting discontiguous memory segments into which the server will
   scatter the result. Each segment is described by a triplet
   consisting of the segment handle or steering tag (STag), segment
   length, and memory address or offset.

```
    struct xdr_rdma_segment {
        uint32 handle;    /* Registered memory handle */
        uint32 length;    /* Length of the chunk in bytes */
        uint64 offset;    /* Chunk virtual address or offset */
    };

    struct xdr_write_chunk {
        struct xdr_rdma_segment target<>;
    };
```

```
      struct xdr_write_list {
         struct xdr_write_chunk entry;
         struct xdr_write_list  *next;
      };
```

The sum of the segment lengths yields the total size of the buffer,
which MUST be large enough to accept the result. If the buffer is
too small, the server MUST return an XDR encode error. The server
MUST return the result data for a posted buffer by progressively
filling its segments, perhaps leaving some trailing segments unfilled
or partially full if the size of the result is less than the total
size of the buffer segments.

The server returns the RDMA Write list to the client with the segment
length fields overwritten to indicate the amount of data RDMA written
to each segment. Results returned by direct placement MUST NOT be
returned by other methods, e.g., by Read chunk list or inline. If no
result data at all is returned for the element, the server places no
data in the buffer(s), but does return zeros in the segment length
fields corresponding to the result.

The RDMA Write list allows the client to provide multiple result
buffers -- each buffer maps to a specific result in the reply. The
NFS client and server implementations agree by specifying the mapping
of results to buffers for each RPC procedure. The following sections
describe this mapping for versions of the NFS protocol.

Through the use of RDMA Write lists in NFS requests, it is not
necessary to employ the RDMA Read lists in the NFS replies, as
described in the RPC/RDMA protocol. This enables more efficient
operation, by avoiding the need for the server to expose buffers for
RDMA, and also avoiding "RDMA_DONE" exchanges. Clients MAY
additionally employ RDMA Reply chunks to receive entire messages, as
described in [RFC5666].

4. NFS Versions 2 and 3 Mapping

A single RDMA Write list entry MAY be posted by the client to receive
either the opaque file data from a READ request or the pathname from
a READLINK request. The server MUST ignore a Write list for any
other NFS procedure, as well as any Write list entries beyond the
first in the list.

Similarly, a single RDMA Read list entry MAY be posted by the client
to supply the opaque file data for a WRITE request or the pathname
for a SYMLINK request. The server MUST ignore any Read list for
other NFS procedures, as well as additional Read list entries beyond
the first in the list.

Because there are no NFS version 2 or 3 requests that transfer bulk
data in both directions, it is not necessary to post requests
containing both Write and Read lists. Any unneeded Read or Write
lists are ignored by the server.

In the case where the outgoing request or expected incoming reply is
larger than the maximum size supported on the connection, it is
possible for the RPC layer to post the entire message or result in a
special "RDMA_NOMSG" message type that is transferred entirely by
RDMA. This is implemented in RPC, below NFS, and therefore has no
effect on the message contents.

Non-RDMA (inline) WRITE transfers MAY OPTIONALLY employ the
"RDMA_MSGP" padding method described in the RPC/RDMA protocol, if the
appropriate value for the server is known to the client. Padding
allows the opaque file data to arrive at the server in an aligned
fashion, which may improve server performance.

The NFS version 2 and 3 protocols are frequently limited in practice
to requests containing less than or equal to 8 kilobytes and 32
kilobytes of data, respectively. In these cases, it is often
practical to support basic operation without employing a
configuration exchange as discussed in [RFC5666]. The server MUST
post buffers large enough to receive the largest possible incoming
message (approximately 12 KB for NFS version 2, or 36 KB for NFS
version 3, would be vastly sufficient), and the client can post
buffers large enough to receive replies based on the "rsize" it is
using to the server, plus a fixed overhead for the RPC and NFS
headers. Because the server MUST NOT return data in excess of this
size, the client can be assured of the adequacy of its posted buffer
sizes.

Flow control is handled dynamically by the RPC RDMA protocol, and
write padding is OPTIONAL and therefore MAY remain unused.

Alternatively, if the server is administratively configured to values
appropriate for all its clients, the same assurance of
interoperability within the domain can be made.

The use of a configuration protocol with NFS v2 and v3 is therefore
OPTIONAL. Employing a configuration exchange may allow some
advantage to server resource management through accurately sizing
buffers, enabling the server to know exactly how many RDMA Reads may
be in progress at once on the client connection, and enabling client
write padding, which may be desirable for certain servers when RDMA
Read is impractical.

5. NFS Version 4 Mapping

   This specification applies to the first minor version of NFS version
   4 (NFSv4.0) and any subsequent minor versions that do not override
   this mapping.

   The Write list MUST be considered only for the COMPOUND procedure.
   This procedure returns results from a sequence of operations. Only
   the opaque file data from an NFS READ operation and the pathname from
   a READLINK operation MUST utilize entries from the Write list.

   If there is no Write list, i.e., the list is null, then any READ or
   READLINK operations in the COMPOUND MUST return their data inline.
   The NFSv4.0 client MUST ensure in this case that any result of its
   READ and READLINK requests will fit within its receive buffers, in
   order to avoid a resulting RDMA transport error upon transfer. The
   server is not required to detect this.

   The first entry in the Write list MUST be used by the first READ or
   READLINK in the COMPOUND request. The next Write list entry is used
   by the next READ or READLINK, and so on. If there are more READ or
   READLINK operations than Write list entries, then any remaining
   operations MUST return their results inline.

   If a Write list entry is presented, then the corresponding READ or
   READLINK MUST return its data via an RDMA Write to the buffer
   indicated by the Write list entry. If the Write list entry has zero
   RDMA segments, or if the total size of the segments is zero, then the
   corresponding READ or READLINK operation MUST return its result
   inline.

   The following example shows an RDMA Write list with three posted
   buffers A, B, and C. The designated operations in the compound
   request, READ and READLINK, consume the posted buffers by writing
   their results back to each buffer.

       RDMA Write list:

          A --> B --> C

       Compound request:

          PUTFH LOOKUP READ PUTFH LOOKUP READLINK PUTFH LOOKUP READ
                        | | |
                        v   v   v
                        A   B   C

If the client does not want to have the READLINK result returned
directly, then it provides a zero-length array of segment triplets
for buffer B or sets the values in the segment triplet for buffer B
to zeros so that the READLINK result MUST be returned inline.

The situation is similar for RDMA Read lists sent by the client and
applies to the NFSv4.0 WRITE and SYMLINK procedures as for v3.
Additionally, inline segments too large to fit in posted buffers MAY
be transferred in special "RDMA_NOMSG" messages.

Non-RDMA (inline) WRITE transfers MAY OPTIONALLY employ the
"RDMA_MSGP" padding method described in the RPC/RDMA protocol, if the
appropriate value for the server is known to the client. Padding
allows the opaque file data to arrive at the server in an aligned
fashion, which may improve server performance. In order to ensure
accurate alignment for all data, it is likely that the client will
restrict its use of OPTIONAL padding to COMPOUND requests containing
only a single WRITE operation.

Unlike NFS versions 2 and 3, the maximum size of an NFS version 4
COMPOUND is not bounded, even when RDMA chunks are in use. While it
might appear that a configuration protocol exchange (such as the one
described in [RFC5666]) would help, in fact the layering issues
involved in building COMPOUNDs by NFS make such a mechanism
unworkable.

However, typical NFS version 4 clients rarely issue such problematic
requests. In practice, they behave in much more predictable ways, in
fact most still support the traditional rsize/wsize mount parameters.
Therefore, most NFS version 4 clients function over RPC/RDMA in the
same way as NFS versions 2 and 3, operationally.

There are however advantages to allowing both client and server to
operate with prearranged size constraints, for example, use of the
sizes to better manage the server's response cache. An extension to
NFS version 4 supporting a more comprehensive exchange of upper-layer
parameters is part of [RFC5661].

5.1. NFS Version 4 Callbacks

The NFS version 4 protocols support server-initiated callbacks to
selected clients, in order to notify them of events such as recalled
delegations, etc. These callbacks present no particular issue to
being framed over RPC/RDMA, since such callbacks do not carry bulk
data such as NFS READ or NFS WRITE. They MAY be transmitted inline
via RDMA_MSG, or if the callback message or its reply overflow the

negotiated buffer sizes for a callback connection, they MAY be
transferred via the RDMA_NOMSG method as described above for other
exchanges.

One special case is noteworthy: in NFS version 4.1, the callback
channel is optionally negotiated to be on the same connection as one
used for client requests. In this case, and because the transaction
ID (XID) is present in the RPC/RDMA header, the client MUST ascertain
whether the message is in fact an RPC REPLY, and therefore a reply to
a prior request and carrying its XID, before processing it as such.
By the same token, the server MUST ascertain whether an incoming
message on such a callback-eligible connection is an RPC CALL, before
optionally processing the XID.

In the callback case, the XID present in the RPC/RDMA header will
potentially have any value, which may (or may not) collide with an
XID used by the client for a previous or future request. The client
and server MUST inspect the RPC component of the message to determine
its potential disposition as either an RPC CALL or RPC REPLY, prior
to processing this XID, and MUST NOT reject or accept it without also
determining the proper context.

6. Port Usage Considerations

   NFS use of direct data placement introduces a need for an additional
   NFS port number assignment for networks that share traditional UDP
   and TCP port spaces with RDMA services. The iWARP [RFC5041]
   [RFC5040] protocol is such an example (InfiniBand is not).

   NFS servers for versions 2 and 3 [RFC1094] [RFC1813] traditionally
   listen for clients on UDP and TCP port 2049, and additionally, they
   register these with the portmapper and/or rpcbind [RFC1833] service.
   However, [RFC3530] requires NFS servers for version 4 to listen on
   TCP port 2049, and they are not required to register.

   An NFS version 2 or version 3 server supporting RPC/RDMA on such a
   network and registering itself with the RPC portmapper MAY choose an
   arbitrary port, or MAY use the alternative well-known port number for
   its RPC/RDMA service. The chosen port MAY be registered with the RPC
   portmapper under the netid assigned by the requirement in [RFC5666].

   An NFS version 4 server supporting RPC/RDMA on such a network MUST
   use the alternative well-known port number for its RPC/RDMA service.
   Clients SHOULD connect to this well-known port without consulting the
   RPC portmapper (as for NFSv4/TCP).

   The port number assigned to an NFS service over an RPC/RDMA transport
   is available from the IANA port registry [RFC3232].

7. Security Considerations

The RDMA transport for RPC [RFC5666] supports all RPC [RFC5531]
security models, including RPCSEC_GSS [RFC2203] security and link-
level security. The choice of RDMA Read and RDMA Write to return RPC
argument and results, respectively, does not affect this, since it
only changes the method of data transfer. Specifically, the
requirements of [RFC5666] ensure that this choice does not introduce
new vulnerabilities.

Because this document defines only the binding of the NFS protocols
atop [RFC5666], all relevant security considerations are therefore to
be described at that layer.

8. Acknowledgments

The authors would like to thank Dave Noveck and Chet Juszczak for
their contributions to this document.

9. References

9.1. Normative References

   [RFC1094]   Sun Microsystems, "NFS: Network File System Protocol
               specification", RFC 1094, March 1989.

   [RFC1813]   Callaghan, B., Pawlowski, B., and P. Staubach, "NFS
               Version 3 Protocol Specification", RFC 1813, June 1995.

   [RFC1833]   Srinivasan, R., "Binding Protocols for ONC RPC Version 2",
               RFC 1833, August 1995.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2203]   Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol
               Specification", RFC 2203, September 1997.

   [RFC3530]   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.

   [RFC5531]   Thurlow, R., "RPC: Remote Procedure Call Protocol
               Specification Version 2", RFC 5531, May 2009.

   [RFC5661]   Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed.,
               "Network File System (NFS) Version 4 Minor Version 1
               Protocol", RFC 5661, January 2010.

9.2. Informative References

   [RFC3232] Reynolds, J., Ed., "Assigned Numbers: RFC 1700 is Replaced
              by an On-line Database", RFC 3232, January 2002.

   [RFC5040] Recio, R., Metzler, B., Culley, P., Hilland, J., and D.
              Garcia, "A Remote Direct Memory Access Protocol
              Specification", RFC 5040, October 2007.

   [RFC5041] Shah, H., Pinkerton, J., Recio, R., and P. Culley, "Direct
              Data Placement over Reliable Transports", RFC 5041,
              October 2007.

   [RFC5666] Talpey, T. and B. Callaghan, "Remote Direct Memory Access
              Transport for Remote Procedure Call", RFC 5666, January
              2010.

Authors' Addresses

   Tom Talpey
   170 Whitman St.
   Stow, MA 01775 USA

   EMail: tmtalpey@gmail.com

   Brent Callaghan
   Apple Computer, Inc.
   MS: 302-4K
   2 Infinite Loop
   Cupertino, CA 95014 USA

   EMail: brentc@apple.com