Updating RPC-over-RDMA:
Next steps

David Noveck
nfsv4wg meeting at IETF96
July 19, 2016
Contents

• Overall goals
• Individual documents
  • draft-dnoveck-nfsv4-rpcrpdma-xcharext-00
  • draft-dnoveck-nfsv4-rpcrpdma-rtissues-00
  • draft-dnoveck-nfsv4-rpcrpdma-rtrext-00
    • Send-based DDP
    • Message Continuation
    • New Optional header types
• Where do we go from here?
Overall Goals

Use the extension framework provided in draft-cel-nfsv4-rpcrdma-version-two in order to improve RPC-over-RDMA performance.

1) Recover valuable things that were mentioned in RFC5666 but unfortunately had to be dropped in the transition to rfc5666bis.
   • Transport characteristics reporting/exchange
   • Direct Placement without explicit RDMA ops (was RDMA.MSGP).

2) Get the performance-related features that have been proven in other similar transports but are not available in Version One:
   • Remote invalidation
   • Message continuation
• Provides framework to describe transport characteristics
• Set of initial characteristic provided but set is extensible
• Enables Performance Improvements:
  • Determination of remote invalidation support
  • Enables larger (than 4K) receive buffers.
  • Is built upon (in draft-dnoveck-nfsv4-rpcrdma-rtrext) to enable:
    • Message Continuation
    • Send-based DDP
Summary

- Addresses issue of round trips within RPC/RDMA v1
  - Focused on simple common cases such as 8K IOs
  - Wanted to know how bad the problem is and whether it can be fixed without writing off v1 and taking a completely new direction in a vN, for N >= 2.

- Conclusions:
  - Three round-trips when explicit RDMA operation is used
    - But some of those don’t contribute to latency
  - There is a latency issue in the WRITE (RDMA read) case
    - Two nested round-trip latencies.
    - Also, two instances of server-side interrupt latency (one addl. one)
  - Performance issues can be addressed within current v2 framework
8K WRITE: explicit RDMA vs. send-receive into large buf

Explicit RDMA operation
- Register Memory
- RPC call sent
- After internode latency, RPC Call received; ack sent, but nobody waits for it
- After server interrupt latency, start processing RPC Call
- RDMA read operation started
- After internode latency, data retrieved from client
- After further internode latency, data received is stored on server and server interrupted
- After interrupt latency, processing of RPC Call resumed
- RPC Reply sent
- After internode latency, RPC Reply in client mem; client interrupted;
- Adapter undoes registration
- Ack sent but nobody waits for it
- After client interrupt latency, RPC Reply processed

Send-receive only
- Registration not needed
- Same
- Same (incl. IN L)
- Same (incl. SRV INT L)
- RDMA op Not needed
- IN L Not needed
- IN L Not Needed
- SRV INT L Not Needed
- Same
- Same (incl IN L)
- Deregistration not needed
- Same
- Same (incl CL INT L)
8K READ: explicit RDMA vs. send-receive into large buf

Explicit RDMA operation

- Register Memory
- Send RPC Call
- After inter-node latency, RPC Call received; ack sent, but nobody waits for it
- After server interrupt latency, server starts processing RPC Call
- When data ready, RDMA write operation started
- After internode latency, data stored on client and ack sent (not waited for)
- Response sent
- After internode latency, RPC Call in client mem; client interrupted;
- Adapter undoes registration;
- Ack sent but nobody waits for it
- After client interrupt latency, RPC Reply processed

Send-receive only

- Registration not needed
- Same
- Same (incl. IN L)
- Same (incl. SRV INT L)
- RDMA op Not needed
- 1-way latency Not needed
- Same
- Same (incl. IN L)
- Deregistration not needed
- Same
- Same (incl. CL INT L)
Observations

• There is a bunch of work involved in providing DDP using explicit RDMA operations.
  • Not as much work as copying data but it isn’t free.

• It is best to avoid copying without using explicit RDMA operations
  • On send side, implementation can avoid using send buffers. Just write data from where it is, using SG lists when necessary.
  • On receive side, need DDP without explicit RDMA ops
    • So RDMA_MSGP was on the right track ☝
    • Too bad it had to be dropped 😞
    • The essence of that approach can be obtained in a different form ≜
      • Pursued in draft-dnoveck-nfsv4-rpcrma-rtrext.
Motivation

• Based on results of rtissues investigation.
• Additional round trip involved in doing an explicit RDMA operation in addition to basic sends/receives for RPC.
  • In addition to round-trip, latency to interrupt server.
  • Acks (because of use of reliable datagram) don’t add to latency.
• Initial investigation showed explicit RDMA operation not needed in many common cases
Basic Approach

• Can we get rid of explicit RDMA operations while retaining benefits?
  • Yes, in most cases.

• Look at where explicit RDMA operations are used:
  • To do DDP
    • Send-based DDP is a viable alternative.
  • Because message is too long for buffer
    • Use multiple messages (i.e. message continuation)

• Result:
  • Two OPTIONAL features in single extension:
    • Message continuation, Send-based DDP
    • Implementations may support either or both
Send-based DDP

Overview

• Same basic idea as RDMA_MSGP
  • Place data in area with required size and alignment.
  • As opposed to directing it at a pre-specified address

• Major Differences from RDMA_MSGP
  • Treated as an instance of DDP and governed by ULB.
    • Allows operations in COMPOUND past the READ or WRITE
    • And can also allow multiple IO operations in single COMPOUND
  • Sender has knowledge of receiver’s buffer structure.
  • Supports DDP on the request (for WRITE data) and not just for response data.
Send-based DDP
Buffer Structure

• Built on idea receive buffers will use SG lists
  • Typically, a smaller area for payload stream plus an aligned buffer as DDP target.
  • Some likely buffer structures:
    • 1K for payload segment plus an 8K DDP-targetable buffer segment
    • 1K for payload segment plus a 4K DDP-targetable buffer segment
      • Could use msg continuation to read 8K
    • Could have multiple DDP-targetable buffer segments
  • Buffer structure available to partner as transport characteristic.
    • Can compute necessary fill/padding to get the DDP-eligible data to required alignment in DDP-targetable buffer segment
Send-based DDP
New DDP-related Data Structures

• New message types do not use existing read and write chunks
  • But provisions made for old-style DDP with explicit RDMA ops
• New message type for requests includes optional reply chunk
  • Needed to support cases for which msg continuation support is not present or cannot be taken advantage of.
• New approach to DDP
  • Each message, whether request or response, indicates where DDP-eligible data, in that message, is located.
  • Request indicates how DDP-eligible data in response should be placed
    • Provides more flexibility than current chunk-based approach
  • With msg continuation, DDP info only present in first SEND of message
New DDP-related Data Structures

DDP-eligible Data Locations (one of three)

• Requests and responses each have an array of `xmddp_mitem`
  • In a request, there is one for each (DDP-eligible) data item directly placed
  • In a response, there is one for each response-direction element in request
    • Includes those for which no direct placement actually occurred

• Each `xmddp_mitem` contains:
  • The displacement the item would have in the XDR stream as whole
  • The length of the item
  • Location information, which can have a number of forms, as described in the next slides
New DDP-related Data Structures
DDP-eligible Data Locations (two of three)

• To accommodate old- and new-style DDP, a switched union is used

• Allows new header types to be used by those that don’t support send-based DDP
  • Implementations that only support the msg continuation feature
  • Implementations that only need the more flexible DDP structures and don’t support either new feature.
New DDP-related Data Structures
DDP-eligible Data Locations (three of three)

• In the XMDTYPE_EXRW case:
  • Contains an array of `rpcrma1_segments` indicating where the data is located

• In the XMDTYPE_TBSN case, contains
  • Offset of start of item in first DDP-targetable buffer segment
  • An array of buffer segment numbers of DDP-targetable buffer segments where the data is located

• A few cases (with void) only useful in the response case
  • XMDTYPE_NOITEM is for response direction item which had no data item
  • XMDTYPE_TOOSHORT is for response direction item where data item is too short to merit DDP
  • These cases are semantically invalid in request.
New DDP-related Data Structures
DDP Response Direction (one of two)

• One response direction item for each *potential* DDP-eligible data item
  • Can be organized into sets, based on the associated region of request, so ...
    • Each op in a COMPOUND with DDP-eligible data items can have a separate set.

• Each `xmddp_rsdset` contains:
  • A range of positions in the request to which this set applies
  • An array of `xmddp_rsditems`

• Each `xmddp_rsditem` contains:
  • A minimum length for direct placement
    • Any item, that is not at least this length, is placed inline
  • Information about how item is to be placed, if it is placed, in an `xmddp_rsdloc`
    • Details on this in next slide
New DDP-related Data Structures
DDP Response Direction (two of two)

• `xmddp_rsdloc` is a switched union
  • Three cases contain an array of `rpcrdma1_segments`
    • `XMDTYPE_EXRW` directs the data to these segments
    • `XMDTYPE_CHOICE` allows responder to use those segments to DDP-targetable buffer segments in response
    • `XMDTYPE_BYSIZE` tells responder to choose explicit RDMA only above a certain size.
  • `XMDTYPE_TBSN` (with void) tell responders to use DDP-targetable buffer segments in response

• Mapping from type in `xmddp_rsdloc` to `xmddp_loc` in response
  • `EXRW`, `TBSN` typically come over as is
  • `CHOICE`, `BYSIZE` are converted to type of placement actually used
  • Any entry type can be mapped to `TOOSHORT` or `NOITEM`
Message Continuation
Overview

• Allows a single request or response to be split into multiple SENDs
• There is less need to use it when receive buffers are larger, but ...
  • There are important feature synergies with send-based DDP
    • For example, when 64K (e.g.) IOs are being done.
  • It is also valuable to have available, when it is hardly ever used
    • To avoid need for reply chunk when large reply is just barely possible
    • Can avoid registration overhead when it serves no real purpose.

• Approach taken is to number segments of message
  • Initial message has number of segments
  • Count has to be known in advance to support credit management:
Message Continuation
Credit Management (one of two)

• Msg continuation requires one credit per RDMA transmission
  • Despite earlier language tying credits to RPC messages or RPCs

• When sending a multi-transmission request:
  • Enough credits need to be available on responder to receive complete request.
  • If they aren’t, position-zero read chunk can be used
Message Continuation
Credit Management (two of two)

• When sending a request which might need a multi-transmission response
  • Requester need to prepost enough buffers to receive the maximum size response
  • If that’s not possible, reply chunk needs to be provided
  • Number of posted receives sent with request.
    • When requester cannot receive or responder cannot send XMOPT_CONT, that number will be one.
• First transmission of response will have actual number of SENDs in response
  • When that is less than original maximum, excess receives become available for credits or may be recycled for future long responses.
New Message Types
Overview

• Three new OPTIONAL message types:
  • XMOPT_REQ to send request (or initial segment of request)
  • XMOPT_RESP to send response (or initial segment of response)
  • XMOPT_CONT to send later segments of requests or responses

• Can be supported even if Send-based DDP is not supported

• XMOPT_{REQ,RESP} can be supported even if msg continuation is not supported.

• Can determine peer support for these message types by trying these or by looking at a transport characteristic.
Where do we go from here?

Overview

• First assess where we are.
  • I’ll present my own assessment
  • Want to hear others’

• Make some decisions about directions for RDMA
• Address near-term document issues
• Better understand RDMA performance
Where do we go from here?
My Assessment about where we are now.

• Clarifying Version One is now pretty much complete.
  • It now seems that this was a necessary chore.
  • I thought the focus on Version One was excessive, but now that it has been done, it doesn’t matter.
  • It certainly was a chore.
  • I want to thank Chuck, Tom, and Bill for getting this chore done, and done well.

• But now, the XDR shackles are off, and we can look at what is necessary to proceed further.
  • Important to not interfere with ongoing Version One implementation work.
  • I think the best approach is to use the Version Two framework already established.
    • That will allow Version Two implementations to interoperate with existing Version One implementations
Where do we go from here?
Decisions that need to be made

• Working group needs to decide future RDMA directions:
  • Has decided rfc5667bis is needed
    • That work can proceed while other RDMA work goes on.
    • If that work uses rfc5666bis framework, it will be compatible with extensions proposed.
  • No wg decision has yet been made on a Version Two.
  • This talk has assumed extensible Version Two is a good vehicle for further work

• Implementers need to decide on the focus of their efforts:
  • Those focusing on Version One could easily support Version Two with no extensions, if the current Version Two approach is adopted.
  • Those with a post-Version-One focus need to decide what existing extensions are important to them and whether to propose others.
Where do we go from here?
Near-term Document Issues

• We now have a tower of I-Ds
  • I-D draft-dnoveck-nfsv4-rpcrdma-rtext is built on I-D draft-dnoveck-nfsv4-rpcrdma-xcharext
  • Which, in turn, is built on I-D draft-cel-nfsv4-rpcrdma-version-two

• Issues:
  • None of these documents has had much working group discussion and review so far.
  • Now that the Version One documents are done, urge people to look at the documents related to Version Two.
  • As that process proceeds, working group needs to consider making the lower levels of this tower into working group documents.
Where do we go from here?
Understanding RDMA Performance Issues

• Had a situation in which our performance issues could have been:
  • Protocol weaknesses
  • Implementation problems

• That uncertainty made it hard to make progress
  • Can’t tackle big implementation issues if protocol might be to blame
  • Hard to tackle protocol weaknesses if the issue might “really” be implementation

• Believe this set of extensions addresses the protocol issues
  • Want to hear from anyone who disagrees
  • If they do, it’s now time to tackle implementation overhead and need for trunking.

• Some cases in which protocol and implementation are both at fault.
  • Interrupt latency: send-based DDP has made this less critical but it still is an issue