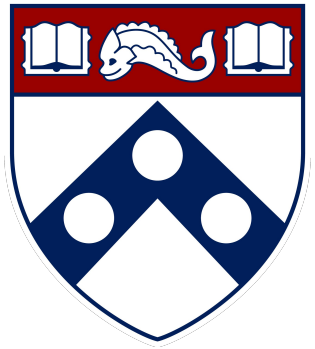


10th Feb 2022 - (2 weeks)

Literature Survey on RPC, RDMA

Shubhendra Pal Singhal

RPC is Not Dead: Rise, Fall and the Rise of Remote Procedure Calls

http://dist-prog-book.com/chapter/1/rpc.html#gridsolve1

The Rise: All Hail RPC (Early 1970's - Mid 1980's)

RPC started off strong. With RFC 674 (Postel & White, 1974) and RFC 707 (Postel & White, 1974; White, 1975) coming out and specifying the design of Remote Procedure Calls, followed by Nelson et. al (Birrell & Nelson, 1984) coming up with a first RPC implementation for the Cedar programming language, RPC revolutionized systems in general and gave rise to one of the earliest distributed systems.

1

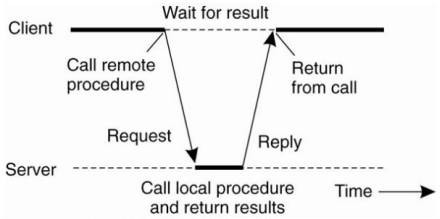


Figure 2 - Synchronous RPC.

[Image Source: (Norman, 2015)]

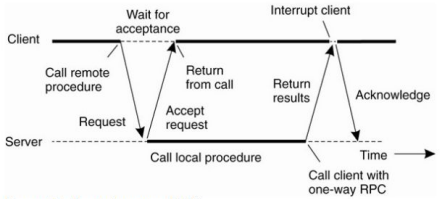


Figure 3 - Asynchronous RPC.

Asynchronous RPC. (2006). Microsoft. Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/aa373550(v=vs.85).aspx

- Multithreaded concept on server side to process the requests.

SUN ONC

- *Single threaded*
- *Transfer of objects using abstraction instead of using a IDL as done nowadays.*
- *Difficult to abstract all constructs of today's programming languages.*

2

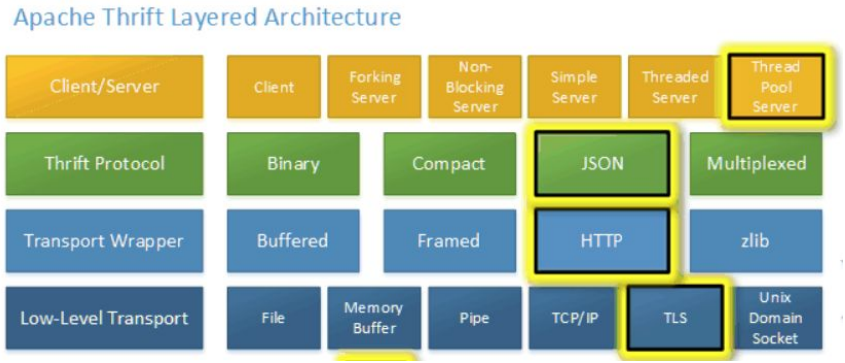
Comparison	Thrift	gRPC
License	Apache2	BSD
Sync/Async RPC	Both	Both
Supported Languages	C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, and OCaml	C/C++, Python, Go, Java, Ruby, PHP, C#, Node.js, Objective-C
Core Language	C++	C
Exceptions	Allows being built in the message	Implemented by the programmer
Authentication	Custom	Custom + Google Tokens
Bi-Directionality	Not straightforward	Straightforward
Multiplexing	Possible via `TMultiplexingProcessor` class	Possible via HTTP/2

In the post-2000 era, MAUI (Cuervo et al., 2010), Cap'n Proto (Kenton, n.d.), gRPC (Google, n.d.), Thrift (Prunicki, 2009) and Finagle (Eriksen, 2013) have been released, which have significantly boosted the widespread use of RPC.

Most of these newer systems include Interface Description Languages (IDLs). These IDLs specify the common protocols and interfacing languages that can be used to transfer information between clients and servers written in different programming languages, making these RPC implementations language-agnostic. Some of the most common IDLs are JSON, XML, and ProtoBufs.

- Cap n proto : RPC Chains
- gRPC: Stream bulk data
- Thrift: Low latency for small exchanges.(Read More..)

Understanding RPC stack



The Heir to the Throne: gRPC or Thrift

Although there are many candidates to be considered as top contenders for RPC throne, most of these are targeted for a specific type of application. ONC is generally specific to the Network File System (though it's being pushed as a standard), Cap'n Proto is relatively new and untested, MAUI is specific to mobile systems, the open-source Finagle is primarily being used at Twitter (not widespread), and the Java RMI simply doesn't even come close due to its transparency issues.

Implementing Remote Procedure Calls by ANDREW D. BIRRELL and BRUCE JAY NELSON 1984 (Basic SUN rpc architecture with primitive style in Cedar language)

- *Transport Layer* : Current RPCs are not dependent on transport layer.
- Using RDMA as transport layer is the current trend ? [WHY]

- What's new??

Use different serialisation formats like JSON, Protocol Buffer

Application use

gRPC - HTTP2
Thrift - HTTP1
REST
SOAP
ONC

Using RDMA as transport layer is the current trend?

Datacenter RPCs can be General and Fast (CMU) - Anuj Kalia 2019
https://www.usenix.org/system/files/nsdi19-kalia.pdf

- Herd, FAast RPC over RDMA for key-store and data center applications.
- eRPC

on these functions. eRPC's speed comes from prioritizing common-case performance, carefully combining a wide range of old and new optimizations, and the observation that switch buffer capacity far exceeds datacenter BDP. eRPC delivers performance that was until now believed possible only with lossless RDMA fabrics or specialized network hardware. It allows unmodified applications to perform close to the hardware limits. Our ported versions of LibRaft and Masstree are, to our knowledge, the fastest replicated key-value store and networked database index in the academic literature, while operating end-to-end without additional network support.

RFP: When RPC is Faster than Server-Bypass with RDMA
https://dl.acm.org/doi/pdf/10.1145/3064176.3064189

- In-bound vs Out-bound RDMA
- Hybrid between bypass vs kernel reply.
- *Wherever by-pass RDMA deals with problem of data race by multiple requests, server-reply is better.*
  No of threads/client, data size to transfer.

on the above two observations. First, server should process the request rather than totally bypassed, so that no application-specific data structure or redesign is needed. Second, results should be remotely fetched by the client through *RDMA_Read* instead of being sent by server, hence the server only handles in-bound RDMA operations.

The first is **in-bound vs. out-bound asymmetry**. Specifically, issuing a one-sided RDMA operation (i.e. out-bound RDMA) has much higher overhead than that of serving one (i.e. in-bound RDMA). This is because, taking *RDMA_Read* as an example, the issuing side needs to maintain certain context and involve both software as well as hardware to ensure the operation is sent and completed, while the serving side is purely handled by hardware. As a result, taking the RDMA Network Interface Card (RNIC) from InfiniBand we have in hand as an example, the peak IOPS (Input/output Operation Per Second) of in-bound RDMA (about 11.26 MOPS[1]) is about 5× higher than that of out-bound RDMA (about 2.11 MOPS). This explains why *server-reply* is sub-optimal: besides not bypassing server's CPU, it also requires server to issue RDMA operations, which is bounded by the limit of out-bound RDMA. The latter is quickly saturated while the in-bound IOPS are far from being saturated by client requests. In the above example, there is only one *RDMA_Read* operation issued by the client and handled by the server. It is called to be out-bound RDMA on the issuing side and called to be in-bound RDMA on the other side.

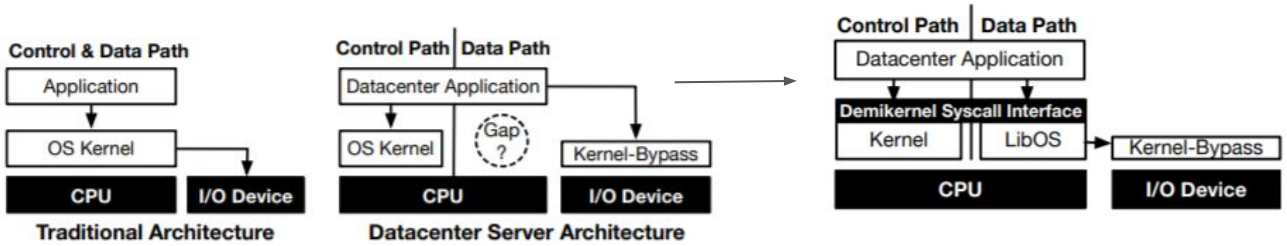
The second observation is **bypass access amplification**, which leads to a significant gap between expected performance and measured performance of *server-bypass*. The former corresponds to the ideal case where only one RDMA operation is required to complete a request, which is usually not true in reality. The root cause is that CPU processing on server is bypassed and multiple clients need to coordinate their access to avoid data access conflict with more RDMA operation rounds! Moreover, they may also need additional RDMA operations for meta-data probing to find where the data is stored on server. Thus, the measured performance of *server-bypass* is typically much lower than its expected one. For example, Pilaf uses 3.2 RDMA operations for each GET request on average even with read-intensive workloads. The

Using RDMA as transport layer is the current trend?

Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store - Pilaf (2013)
https://www.usenix.org/system/files/conference/atc13/atc13-mitchell.pdf

- Performs better than IP over InfiniBand (IPoIB)

I'm Not Dead Yet! The Role of the Operating System in a Kernel-Bypass Era 2018
https://irenezhang.net/papers/demikernel-hotos19.pdf



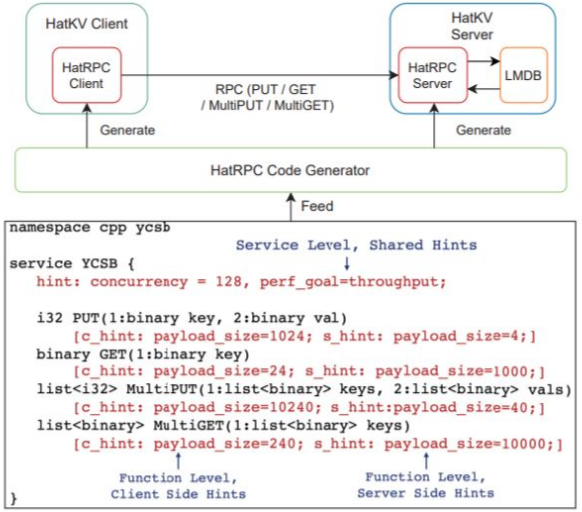
- OS abstractions like sockets are difficult to re-create!

HatRPC: Hint-Accelerated Thrift RPC over RDMA 2021
https://dl.acm.org/doi/pdf/10.1145/3458817.3476191

HatRPC: Hint-Accelerated Thrift RPC over RDMA 2021

https://dl.acm.org/doi/pdf/10.1145/3458817.3476191

accelerated RPC framework is too broad, and there is no single design and optimization can beat all others. Prior studies like AR-gRPC [18], HERD [36], FaRM [23], and RFP [59], are optimized towards specific types of applications or adopt relatively generic design approaches to serve some common workloads. However, as an RPC framework, optimizing for specific types of applications or adopting balanced design approaches for generic use cases are far from the desired RPC design. The desired RPC framework should perform well in not only homogeneous services/functions but also heterogeneous ones. For instance, an RPC framework in



RDMA knobs inside which differs according to application.

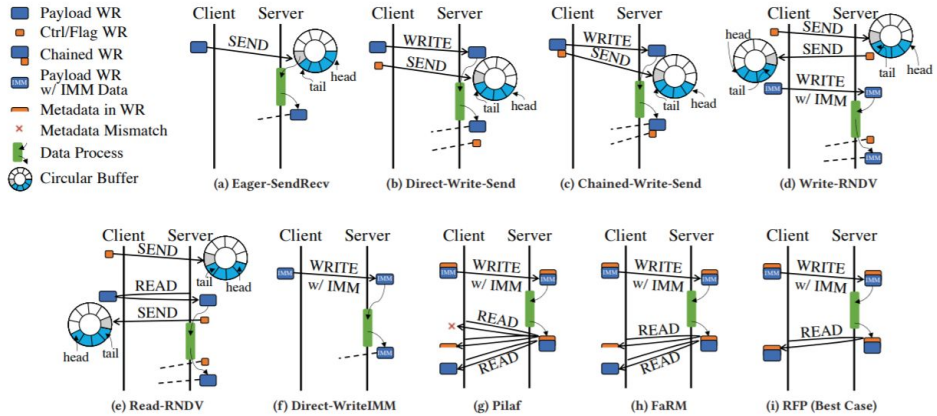


Figure 3: RDMA Protocols. WR: Work Request. Dashed lines indicate that servers repeat the same procedures as what clients have done.

Thrift over RDMA addressing ***heterogeneous RPC*** as application, Using diff RDMA based on hints!

- Saw why RDMA as transport layer as more research potential. Support for other transport layers like QUIC using UDP, TCP (all proven to be slower than RDMA)

- Idea : HatRPC was for heterogeneous applications which did not consider *geo-distributed* applications!! So what if we use the RPC style for this application using RDMA hybrid as built in HatRPC with *more RDMA considerations*!! [checked citations for this paper below.]

RPC Chains: Efficient Client-Server Communication in Geodistributed Systems
https://www.usenix.org/legacy/event/nsdi09/tech/full_papers/song/song.pdf

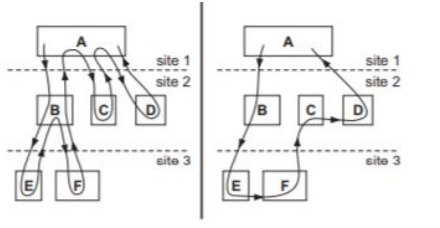
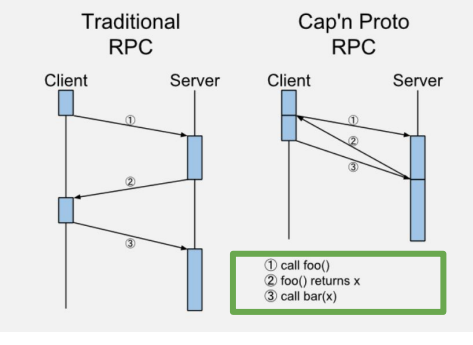


Figure 1: (**Left**) Standard RPCs. (**Right**) RPC chain.

More to read to get better RDMA and RPC design for geo-distributed :

1. CrossStitch: An Efficient Transaction Processing Framework for Geo-Distributed Systems - 2015
2. Near-Optimal Latency Versus Cost Tradeoffs in Geo-Distributed Storage 2020
3. Exploiting RDMA to create a geographically dispersed storage network over the WAN with real-time access to data 2021
4. Communicating Efficiently on Cluster-Based Remote Direct Memory Access (RDMA) over InfiniBand Protocol 2018
5. Software architecture and algorithm for reliable RPC for geo-distributed mobile computing systems 2018
6. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing 2019



Cap n proto
(Based on gRPC)

Some papers related to RPCs….

- Accelerating TensorFlow with Adaptive RDMA-based gRPC 2018 (exists!!)
  - https://par.nsf.gov/servlets/purl/10112767

TwirPHP: A modern RPC framework for PHP
https://sagikazarmark.hu/blog/twirphp-a-modern-rpc-framework-for-php/

The most important differences between Twirp and gRPC are HTTP 1.1 support and JSON serialization (in addition to protobuf) which really proves to be useful during

Thrift: Scalable Cross-Language Services Implementation
https://thrift.apache.org/static/files/thrift-20070401.pdf

Combining, all this we realise that TCP(original)/UDP(QUIC), RDMA on thrift, gRPC are engineered. Also for different applications : Data center, distributed GraphQL, geo-distributed, mobile.

Insight of why gRPC over Thrift

https://www.alluxio.io/blog/moving-from-apache-thrift-to-grpc-a-perspective-from-alluxio/

Might help..

1. SpecRPC: A General Framework for Performing Speculative Remote Procedure Calls
2. *Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing 2019 - helps in deciding features/knobs (Think of features like atmost-once semantics, idempotency etc. as resource constraint.)*
3. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing
4. M-RPC: Remote Procedure Call Service for Mobile Clients

MISTAKE : RDMA vs. RPC for Implementing Distributed Data Structures

Considered RTT for RPC, but RDMA was bypass!!