Distributed Procedure Call
A Primitive for High-Performance “Multi-Hop” Communication

Collin Lee [June 12, 2020]
What communication framework should I use?
RPC is the de facto standard. Can we do better?
Distributed Procedure Call
A Primitive for High-Performance “Multi-Hop” Communication
Distributed Procedure Call (DPC)
A Primitive for High-Performance “Multi-Hop” Communication

- Support *multi-hop* communication pattern
  - Eliminates or moves messages out of the critical path
- Complete reference DPC framework implementation, *Roo*
  - Lightweight for use in low-latency environments
  - Solutions for detecting DPC completion and DPC failure
- Lower latency than RPC in many use cases
  - 20-59% reduction in end-to-end latency
Example “Complex” Distributed Service
A real life *multi-hop* example

- **Complex service**: Request processing requires visiting multiple consecutive service nodes
- Need more *supplies* from the *store*
- Need *membership card* from *office*
- Execution of *getToiletPaper*
  1. Go to *office* to get *membership card*
  2. Go to *store* to get *supplies*
  3. Return *home*
Example “Complex” Distributed Service

Why not go home between trips to office and store?

- Need more supplies from the store
- Need membership card from office
- Execution of getToiletPaper
  1. Go to office to get membership card
  2. Return home
  3. Go to store to get supplies
  4. Return home
Example RPC Pattern
Key-value store lookup by secondary index

- Need **object** from the **object store**
- Need **index info** from **index server**
- Execution of **getObjectByIndex**
  1. Go to **index server** to get **index info**
  2. Return to **client**
  3. Go to **object store** to get **object**
  4. Return to **client**
RPC vs Multi-Hop Pattern

RPC pattern can be less efficient than Multi-Hop pattern

- Increased end-to-end latency (2.0 RTT vs 1.5 RTT)
- Increased message processing (is sent and handled twice vs once)
- Performance impact grows with the number of hops and messages
Features of the DPC Abstraction

The two features needed to support any multi-hop pattern

Direct Communication (hops)

Parallel Invocation (fan-out)
The DPC Abstraction
Supporting the multi-hop pattern with a simple abstraction

Similarities with RPC

• Request-Response model:
  • requests sent to DPC servers to be processed
  • responses sent and received by DPC clients
  • Associates request and response messages

Differences from RPC

• Delegated requests (multi-hop):
  • DPC servers can delegate a request to other servers

• Multiple requests and responses (fan-out):
  • DPC clients and servers can send multiple requests
  • DPC clients can receive multiple responses
The DPC API

Client API

send_request()

Send a request to a server

receive_response()

Return a response, if available

wait()

Block until all responses have arrived

Server API

receive()

Receive request, if available

delegate()

Send a request to another server

reply()

Return a response back to the client
Example: DPC in a Distributed Graph DB

Find all Friends and Friends of Friends named “Jane”

Client::send_request() to server with my list of friends

Server::reply() with friends of friends named “Jane”

Server::delegate() request to servers with my friends’ list of friends

Server::reply() with friends named “Jane”
Implementing the Roo framework
A complete reference DPC framework implementation

• Designed for maximum application flexibility
  • Allows DPC communication pattern to be defined dynamically
  • Applications defined communication pattern during DPC execution

• Challenges for Roo implementation:
  1. Detecting when a DPC execution has completed
  2. Detecting when a DPC execution has failed
Detecting DPC Completion
Roo's implementation challenge

- DPC complete when all expected responses are received by the client
- **Challenge: How does the client know what responses to expect?**
  - Roo allows dynamic definition of communication pattern at run-time
  - No a priori knowledge of multi-hop structure
- Roo needs a dynamic way to build the expected response set
How responses are generated?
Dynamic DPC communication patterns

- Request generated during request processing
- Request generated via delegated requests processing
Manifest Messages
Collecting DPC metadata

Manifest Message

Request ID

Delegated Request IDs

Response IDs
Detecting DPC Completion with Manifests
How the Roo client uses manifests

Known Requests
C1:1

Expected Responses
C1:1

S1:1
S1:2
S1:3

S2
S3
S4

C1:1
S1:[1,2]
S1:[1,3]
Detecting DPC Completion with Manifests

How the Roo client uses manifests

Known Requests
C1:1

Expected Responses
S1:1

C1:1
S1
S1:1
S2
S1:2
S1:3

C1:1
S1:[1,2]
S1:[1,3]

S3

S4
Detecting DPC Completion with Manifests
How the Roo client uses manifests

Known Requests
- C1:1
- S1:1
- S1:2

Expected Responses
- S1:1
- S1:2
- S1:3

Diagram:
- C1
  - C1:1
  - S1
    - S1:1
    - S1:2
    - S1:3
  - S2
- S3
- S4

Manifests:
- C1:1
- S1:[1,2]
- S1:[1,3]
Detecting DPC Completion with Manifests

How the Roo client uses manifests
Detecting DPC Completion with Manifests
How the Roo client uses manifests

Known Requests
C1:1
S1:1
S1:2
S3:1

Expected Responses
S1:1
S1:2
S1:3
S2:1
S4:1

DPC complete when all manifests and responses have been received

Request ID
S3:1
S4:1

S1:1
S2:1
S3:1
S4:1

S1:1
S2:1
S3:1
S4:1

S1:1
S2:1
S3:1
S4:1

Detecting DPC Completion with Manifests

DPC complete when all manifests and responses have been received
Optimizing Manifest Usage
Eliminate the need for manifests in the common case

With single response, treat response as manifest

With single delegation, the delegated request inherits the parent ID
Evaluating DPC

• DPC vs RPC Comparison

• Roo – implementation of DPC abstraction
  • C++ user-space library
  • Built with user-space implementation of Homa
  • Kernel-bypass with DPDK

• Comparable RPC framework
  • Shares mostly the same code as Roo
  • Optimized/simplified for RPC where possible
  • Allows for direct comparison of DPC vs RPC

• Hardware Setup (CloudLab x1170)
  • NIC: Two Dual-port Mellanox ConnectX-4 25Gbps NIC
  • Switch: 40Gbps HP FlexFabric 12910 switch
  • RTT: ~6us
Latency for various hop counts

DPC vs RPC Benchmark Communication Pattern
Latency for various hop counts
Median latency for 100B message

End-to-end latency (µs)

Number of consecutive nodes visited ("hops")

Roo

RPC

Approaches 50% reduction in end-to-end latency

Roo is only slower at 1-hop (8% slowdown)
Latency for various degrees of fan-out
DPC vs RPC Benchmark Communication Pattern
Latency for various degrees of fan-out

Median latency for 100B message

End-to-end latency (µs)

Number of parallel nodes visited ("fan-out")

Roo
RPC

Uniform improvement from 2-hop execution
Graph DB Interactive Query Benchmark
A set of "real-would" graph query patterns

- LDBC Social Network Benchmark
  - Industry standard graph database benchmark
  - Models graph representation of a social network application
- Interactive Short (IS) Queries
  - Common user interactions
Latency for Graph DB Interactive Queries

LDBC Social Network Benchmark Interactive Short (IS) Queries

End-to-end latency (µs)

Normalized Roo latency

Interactive Short Query Type

Single Hop Pattern

Greater reduction than predicted due to client bottleneck

Ex: 41 RPCs vs Roo with 21 messages

Latency Comparison:
- IS 1: Roo 0.41, RPC 1.10
- IS 2: Roo 0.80, RPC 1.06
- IS 3: Roo 0.79, RPC 0.79
- IS 4: Roo 0.70, RPC 0.70
- IS 5: Roo 0.65, RPC 0.65
- IS 6A: Roo 0.53, RPC 0.53
- IS 6B: Roo 0.53, RPC 0.53
- IS 7: Roo 0.53, RPC 0.53
CPU time for Graph DB Interactive Queries
LDBC Social Network Benchmark Interactive Short (IS) Queries

Roo uses less total CPU time in multi-hop cases
Topics of Current and Future Work

Interesting issues that I didn’t cover today

• Detecting DPC failure
• Reliability semantics
• Consistency semantics
• More applications and use cases
• DPC effect on throughput
• Optimizing metadata transmission
• Serialization and message structure
Distributed Procedure Call (DPC)
A Primitive for High-Performance “Multi-Hop” Communication

• Support multi-hop communication pattern
  • Eliminates or moves messages out of the critical path

• Complete reference DPC framework implementation, Roo
  • Solutions for detecting DPC completion and DPC failure
  • 20-59% reduction in end-to-end latency

Roo [https://github.com/PlatformLab/Roo]
Homa [https://github.com/PlatformLab/Homa]

Feedback? Use cases?
Email: cstlee@cs.stanford.edu