HatRPC: Hint-Accelerated Apache Thrift RPC over RDMA

Xiaoyi Lu, Assistant Professor

Department of Computer Science and Engineering (CSE)
University of California, Merced
Overview

- Introduction
- Motivation
- HatRPC Design
- Evaluation
- Conclusion
Varied Communication Requirements in Apps

- Modern datacenters and clouds have many heterogeneous applications running simultaneously.
- Different services and functions within the same server have different communication characteristics, communication patterns, and requirements.
Remote Direct Memory Access (RDMA)

- Remote Direct Memory Access (RDMA) can bypass CPU in transferring data across network.
- Delivers excellent performance in latency, bandwidth, throughput, etc.
- Reduces CPU involvements.
RDMA Programming Is Not Easy

• 600~1000 LOC for native verbs-based hello world
  - ~600, RDMA-CM-based example - [https://github.com/tarickb/the-geek-in-the-corner/tree/master/01_basic-client-server](https://github.com/tarickb/the-geek-in-the-corner/tree/master/01_basic-client-server)

• 745 LOC for UCX-based hello world
  - ucp_hello_world.c - [https://github.com/openucx/ucx/blob/master/examples/ucp_hello_world.c](https://github.com/openucx/ucx/blob/master/examples/ucp_hello_world.c)

• 2335 LOC for Libfabric-based PingPong
  - pingpong.c - [https://github.com/ofiwg/libfabric/blob/main/util/pingpong.c](https://github.com/ofiwg/libfabric/blob/main/util/pingpong.c)
Varied RDMA Communication Schemes

- Several RDMA communication schemes from previous works
RDMA Schemes in RPC - Latency

- Direct-WriteIMM provides the best performance in busy polling.
RDMA Schemes in RPC - Throughput

- Direct-WriteIMM with event polling is suitable for small payloads
- RFP with event polling is suitable for full- and over-subscription for large payloads
Problem Statements

• RDMA performance is high, but its productivity is low
  • Can we design an approach to automatically generate efficient RDMA-based communication substrates for data center applications?

• No one-size-fits-all RDMA scheme
  • How can the proposed approach satisfy different communication requirements on various RDMA schemes in datacenter applications?
  • How can we guarantee the effectiveness and efficiency of the generated RDMA-based communication schemes for heterogeneous applications?
Overview

• Introduction
• Motivation
• **HatRPC Design**
• Evaluation
• Conclusion
Key Ideas of HatRPC

P1: RDMA productivity is low

P2: No one-size-fits-all RDMA scheme

S1: Use RDMA enabled RPC frameworks

S2: Extend IDL files for application requirements.

Thrift

Hint

HatRPC

Overview of RPC Framework – Apache Thrift

- A communication framework that hides platform and hardware details for users
- Typically provide user-friendly Interface Definition Language for different services and applications
- Apache Thrift is a widely used RPC framework that adopts a hierarchical architecture and provides an IDL code generator
Proposed Hints in Thrift IDL

Service ::= 'service' Identifier ( 'extends' Identifier )?
     '{' HintGroup* Function* '}'
Function ::= 'oneway'? FunctionType Identifier '( Field* )'
     Throws? ListSeparator? FunctionHint?
FunctionHint ::= '[' HintGroup* ']
HintGroup ::= 'hint' ':' HintList ';'
     | 'c_hint' ':' HintList ';'
     | 's_hint' ':' HintList ';
HintList ::= Hint ',', HintList | Hint
Hint ::= key '=' value

HatRPC IDL Abstract Syntax Structure

- Extend from Thrift’s IDL syntax and allow users to insert key-value pairs as hints in IDL files
- Adopt a hierarchical architecture to specify hints in different services or functions. Use lateral partitioning to differentiate the client and server side
Code Generation

- Use flex to generate lexical analyser and Bison to generate parser for code generation
- Check hint validity and optimize hint map layout. Generate RDMA-based service and stub files with hints
HatRPC Architecture

- Implement various RDMA communication schemes, different polling mechanisms and memory management strategies in HatRPC’s RDMA engine

- TRdma layer is the counterpart for Thrift’s TSocket to bridge RDMA engine with Thrift
How to use HatRPC

**Step1:** Write HatRPC IDL file

```cpp
1 service latency {
2   hint: perf_pref=busy, concurrency=4;
3   c_hint: payload_size=128;
4   s_hint: payload_size=32;
5
6   void start()
7   string echo(1:string payload)
8   void finish()
9 }
```

**Step2:** Use HatRPC compiler to generate source code.

```
1 hatrpc_gen -out gen --gen cpp latency.thrift
```

**Step3:** Implement server handler

```cpp
1 class latencyHandler : virtual public latencyIf {
2   public:
3     latencyHandler() = default;
4     ~latencyHandler() = default;
5
6     void start(const int64_t iter) {
7         std::cout << "RPC call: start" << std::endl;
8     }
9
10    void echo(std::string &_return, const std::string &payload) {
11        _return = payload;
12    }
13
14    void finish() {
15        std::cout << "RPC call: finish" << std::endl;
16    }
17 }
18 ```
How to use HatRPC (cont.)

An RDMA-enabled Hello World example only needs ~70 LOC!

### Client Example

```cpp
#include <thrift/transport/TSocket.h>
#include <thrift/transport/TTransport.h>
#include <thrift/transport/TBinaryProtocol.h>

int main(int argc, char const *argv[])
{
    std::string response = client.echo(payload);
    client.finish();
    transport->close();
}
```

### Server Example

```cpp
#include <thrift/transport/TSocket.h>
#include <thrift/transport/TTransport.h>
#include <thrift/transport/TBinaryProtocol.h>

int main(int argc, char const *argv[])
{
    server->serve();
}
```
Co-designed HatKV and YCSB Example

- Build HatKV, a KV Store atop HatRPC with LMDB as the storage backend
- Set PUT and GET to be latency centric and the corresponding multi-operations to be throughput centric, set payload size and concurrency level accordingly
- Hints are also passed to LMDB to tune configurations

```
namespace cpp ycsb

service YCSB {
  hint: concurrency = 128, perf_goal=throughput;

  i32 PUT(1:binary key, 2:binary val)
    [c_hint: payload_size=1024; s_hint: payload_size=4;]

  binary GET(1:binary key)
    [c_hint: payload_size=24; s_hint: payload_size=1000;]

  list<i32> MultiPUT(1:list<binary> keys, 2:list<binary> vals)
    [c_hint: payload_size=10240; s_hint:payload_size=40;]

  list<binary> MultiGET(1:list<binary> keys)
    [c_hint: payload_size=240; s_hint: payload_size=10000;]
}
```

HatKV and IDL Example for YCSB Workloads
Overview

• Introduction
• Motivation
• HatRPC Design
• **Evaluation**
• Conclusion
## Experimental Setup

<table>
<thead>
<tr>
<th></th>
<th>Cluster A</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Intel Skylake Gold6132 (2.6 GHZ)</td>
</tr>
<tr>
<td><strong>RAM (DDR)</strong></td>
<td>192 GB</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>720 GB SSD</td>
</tr>
<tr>
<td><strong>Interconnect</strong></td>
<td>ConnectX-5 IB-EDR (100 Gbps)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>CentOS Linux 7.6.1810</td>
</tr>
<tr>
<td><strong>OFED</strong></td>
<td>OFED-5.0-2.1.8</td>
</tr>
<tr>
<td><strong>Scale</strong></td>
<td>10 nodes</td>
</tr>
</tbody>
</table>
Microbenchmark - Latency

- HatRPC can select the best scheme, Direct-WriteIMM for the latency goal, achieving up to 54% improvement over other schemes.
Microbenchmark - Throughput

- HatRPC switches from Direct-WriteIMM to RFP when over-subscription (32), yielding up to 20% improvement for 512 B messages and up to 56% for 128 KB messages
For YCSB workload A, HatRPC-Service and HatRPC-Function reduce latency by up to 73% and 80%, respectively.
YCSB Workload A Evaluation - Throughput

- For YCSB workload A, HatRPC-Service and HatRPC-Function gain a speedup of 2.7x and 3.8x, respectively.
YCSB Workload B Evaluation - Latency

- For YCSB workload B, HatRPC-Service and HatRPC-Function improve the performance by up to 84% and 85%, respectively.
YCSB Workload B Evaluation - Throughput

- For YCSB workload B, HatRPC-Service and HatRPC-Function can be up to 6.4x and 7.4x faster than other schemes
Overview

- Introduction
- Motivation
- HatRPC Design
- Evaluation
- Conclusion
Conclusions

• Re-examine many existing RDMA schemes and their performance in RPC systems

• Propose HatRPC, a hint-accelerated RPC based on Apache Thrift over RDMA
  • Leverage hints and RDMA to improve the performance for varied communication requirements in applications

• Co-design a HatRPC-based key-value store (HatKV) with LMDB as the backend
  • Achieve up to 85% improvement for YCSB workloads over other state-of-the-art RDMA schemes

• Acknowledgement
  • Tianxi Li and Haiyang Shi
  • NSF

Thank you!

http://faculty.ucmerced.edu/luxi
http://padsys.org/