2022 OFA Virtual Workshop

HatRPC: Hint-Accelerated Apache Thrift RPC over RDMA

Xiaoyi Lu, Assistant Professor

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

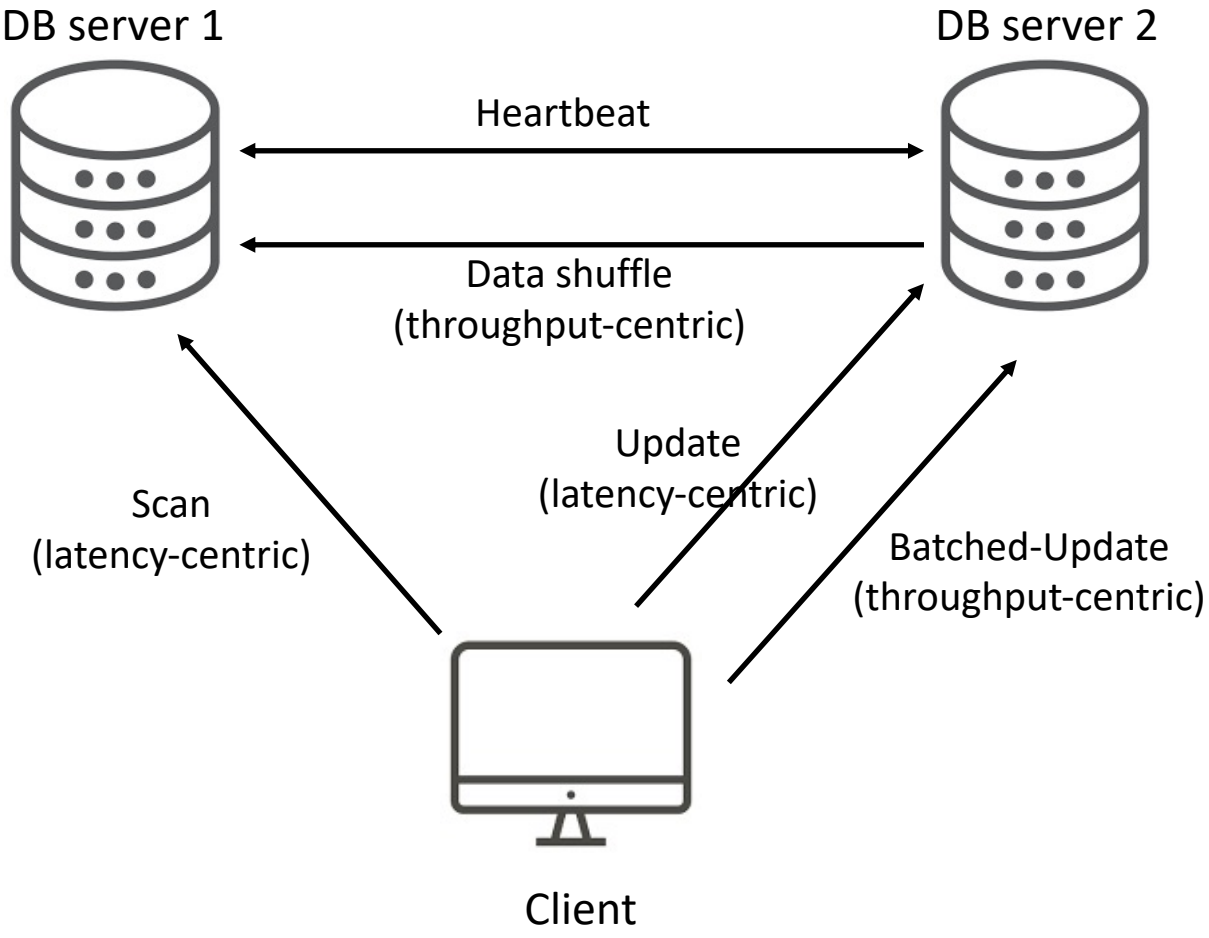
UNIVERSITY OF CALIFORNIA
MERCED

Overview

- **Introduction**
- Motivation
- HatRPC Design
- Evaluation
- Conclusion

Varied Communication Requirements in Apps

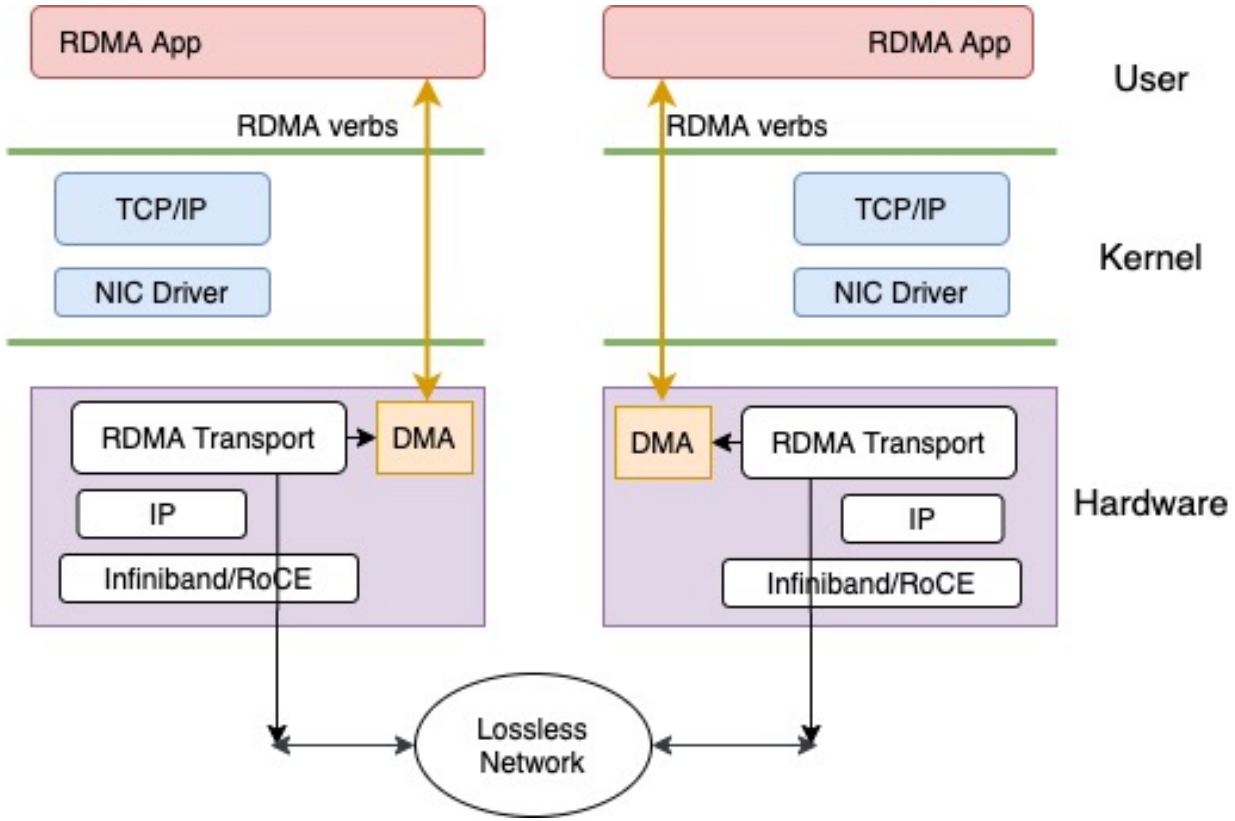
- Modern datacenters and clouds have many heterogeneous applications running simultaneously
- Different services and functions within the same server have different communication characteristics, communication patterns, and requirements



Database with Heterogeneous Applications

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

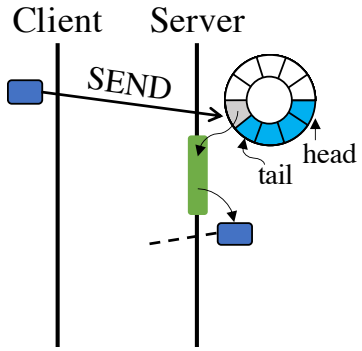
- 745 LOC for UCX-based hello world

  - ucp_hello_world.c - https://github.com/openucx/ucx/blob/master/examples/ucp_hello_world.c

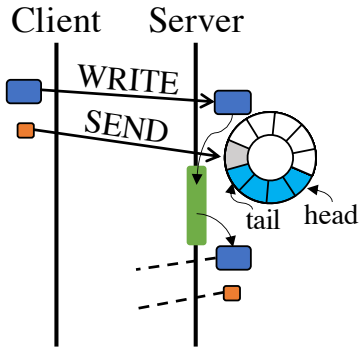
- 2335 LOC for Libfabric-based PingPong

  - pingpong.c - https://github.com/ofiwg/libfabric/blob/main/util/pingpong.c

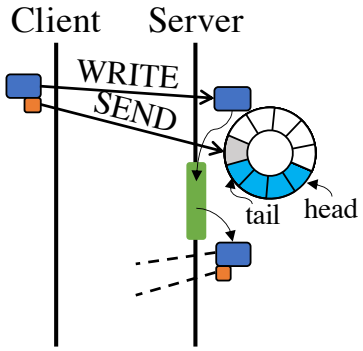
Varied RDMA Communication Schemes



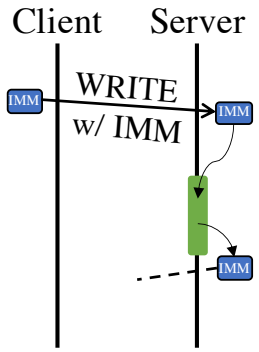
Eager-SendRecv



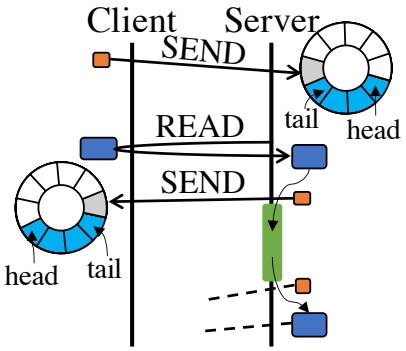
Direct-Write-Send



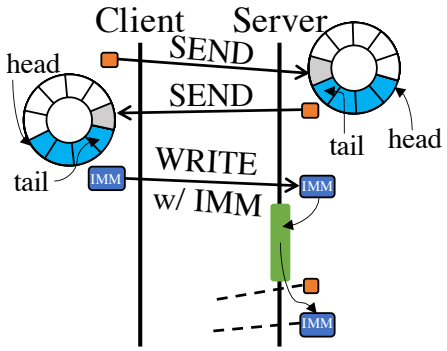
Chained-Write-Send



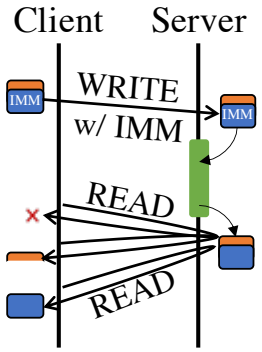
Direct-WriteIMM



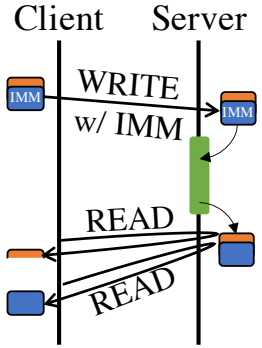
Read-RNDV



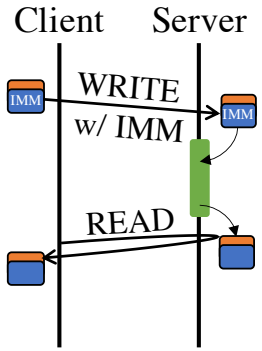
Write-RNDV



Pilaf



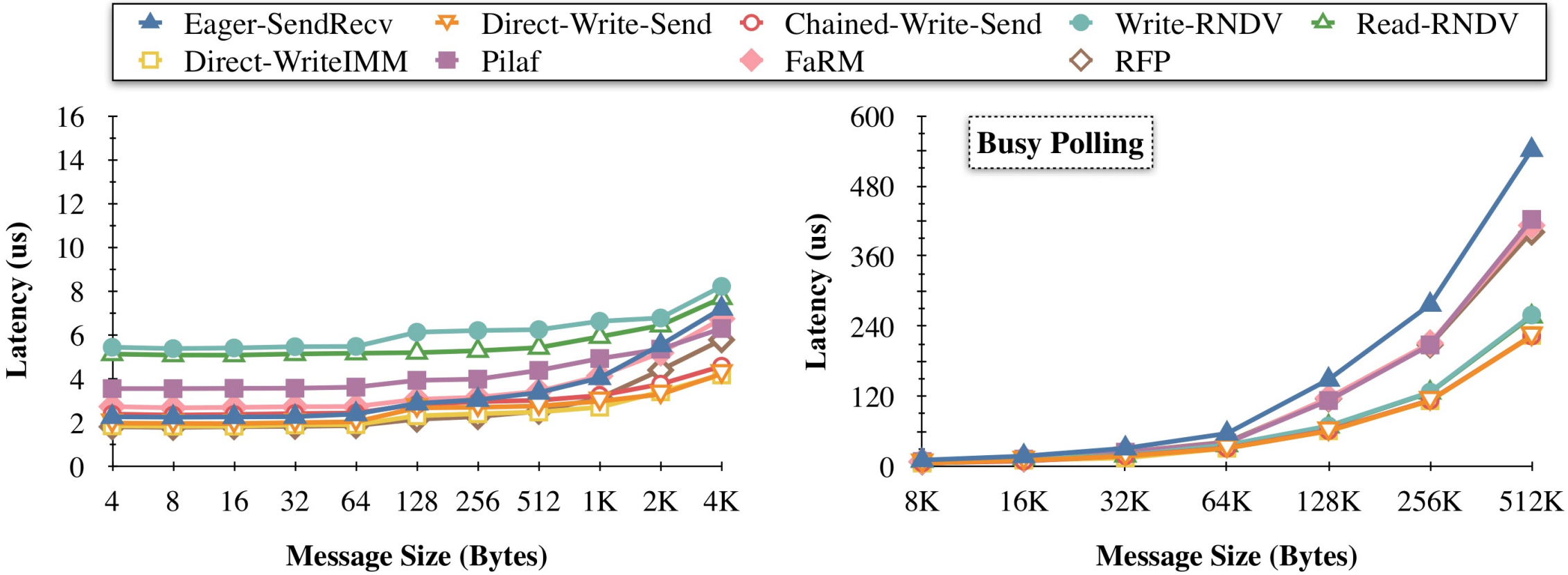
FaRM



RFP (Best Case)

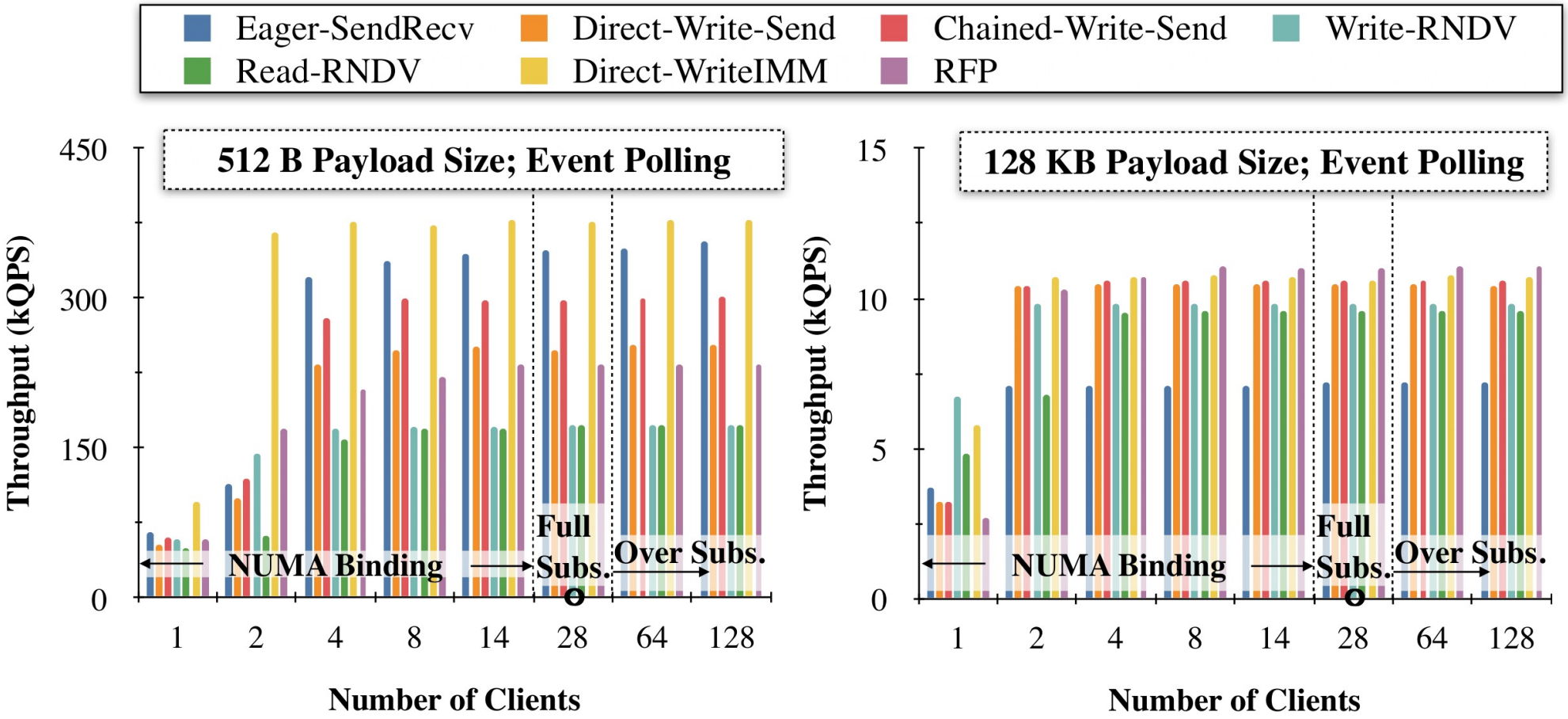
- Several RDMA communication schemes from previous works

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

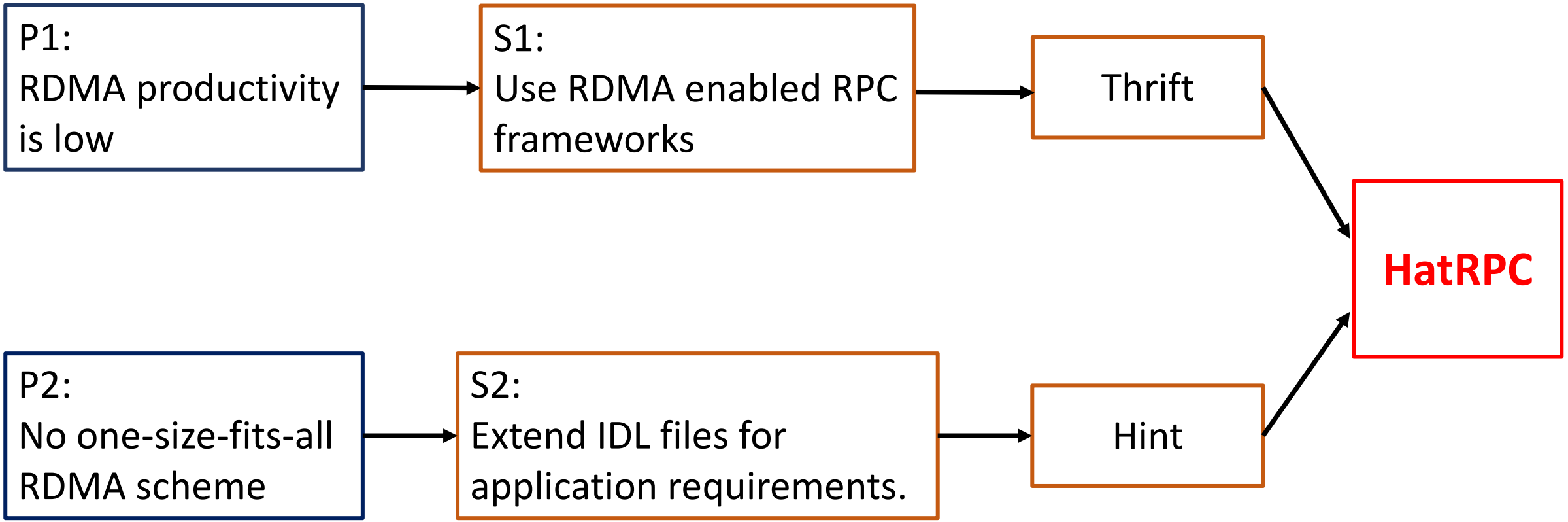
Problem Statements

- **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

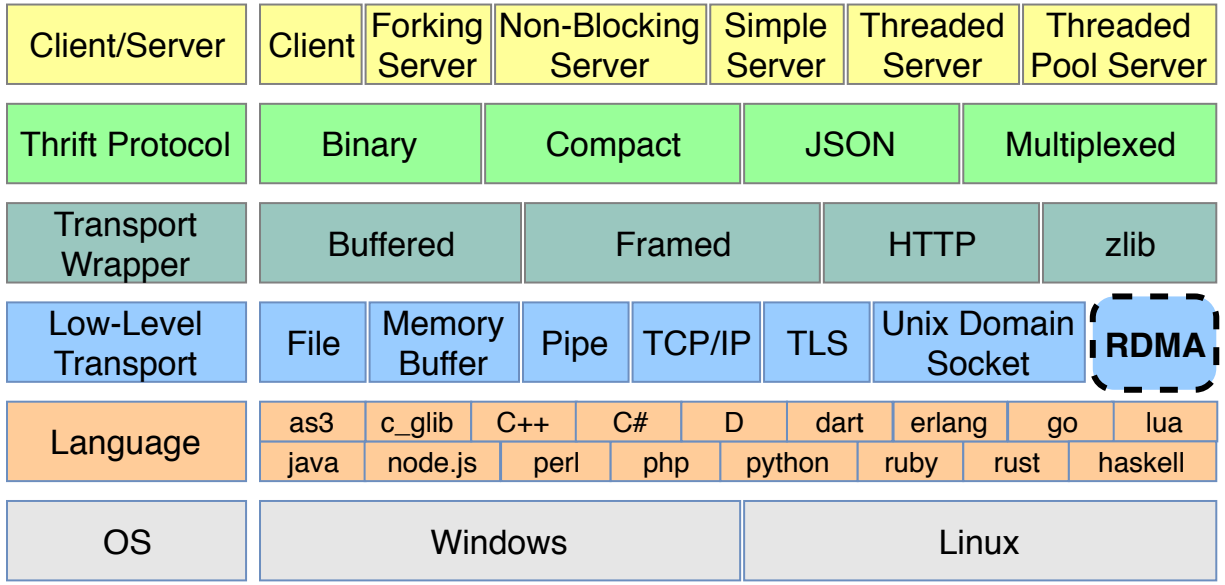
Key Ideas of HatRPC



Courtesy: Tianxi Li*, Haiyang Shi*, and Xiaoyi Lu. HatRPC: Hint-Accelerated Thrift RPC over RDMA. In Proceedings of the 34th International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2021. (*Co-First Authors)

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

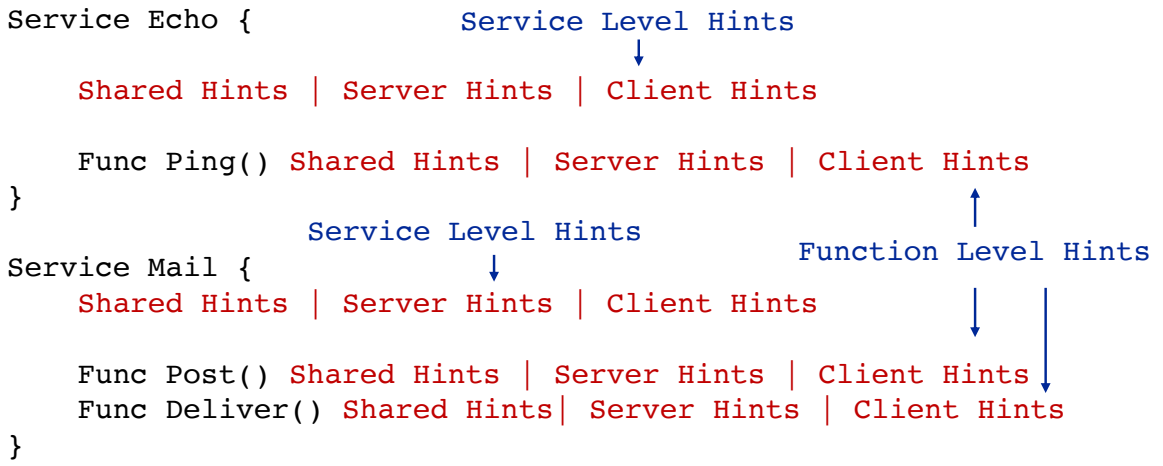


Apache Thrift Architecture

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

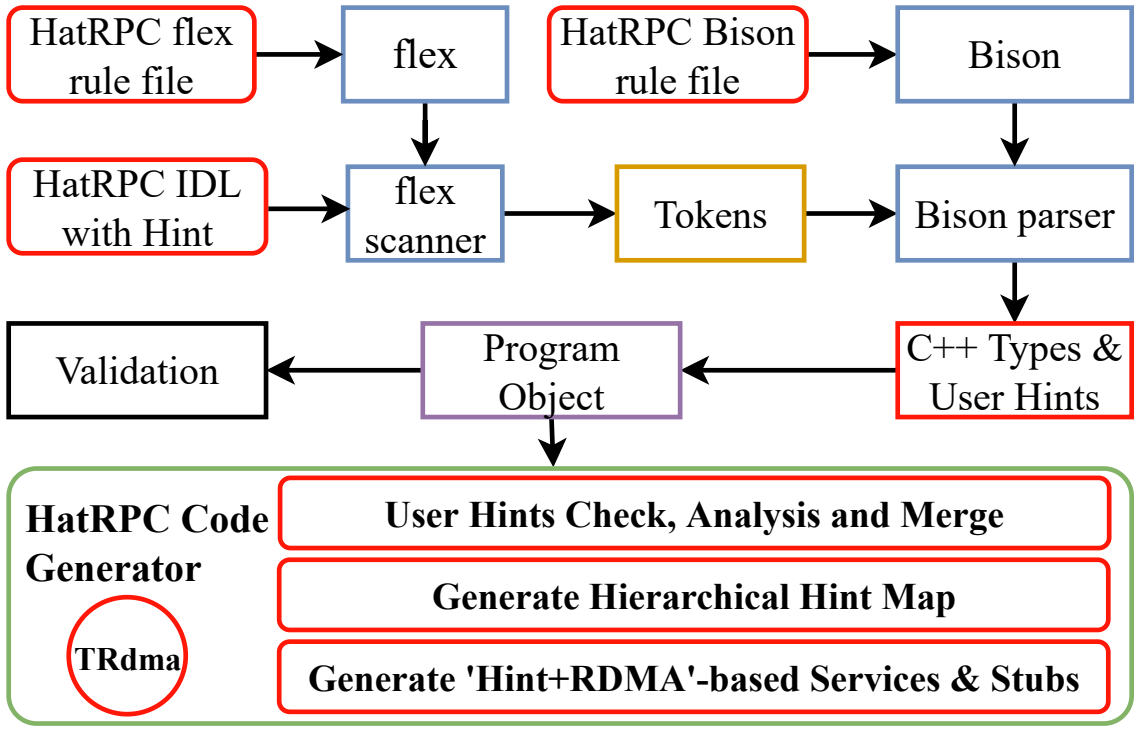


HatRPC Hint Hierarchy

- 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

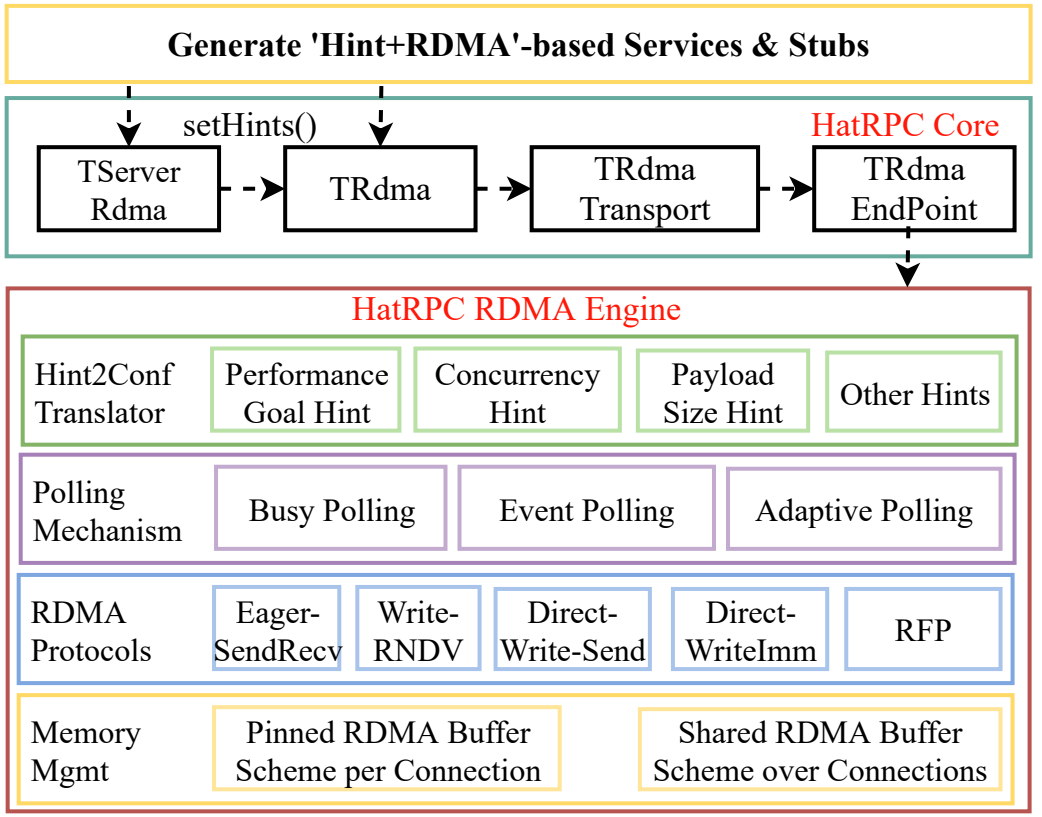
- Use flex to generate lexical analyser and Bison to generate parser for code generation
- Check hint validity and optimize hint map layout. Generate RDMA-based service and stub files with hints



HatRPC Code Generation

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



HatRPC Architecture

How to use HatRPC

Step1: Write HatRPC IDL file

```
service latency {
    hint: perf_pref=busy, concurrency=4;
    c_hint: payload_size=128;
    s_hint: payload_size=32;

    void start()
    string echo(1:string payload)
    void finish()
}
```

Step2: Use HatRPC compiler to generate source code.

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

Step3: Implement server handler

```
class latencyHandler : virtual public latencyIf {
public:
    latencyHandler() = default;

    ~latencyHandler() = default;

    void start(const int64_t iter) {
        std::cout << "RPC call: start" << std::endl;
    }

    void echo(std::string &_return, const std::string
&payload) {
        _return = payload;
    }

    void finish() {
        std::cout << "RPC call: finish" << std::endl;
    }
};
```

How to use HatRPC (cont.)

```cpp
#include <thrift/transport/TServerRdma.h>
#include ...

int main(int argc, char const *argv[])
{
  shared_ptr<latencyHandler> handler(new
    latencyHandler());
  shared_ptr<TProcessor> processor(new
    latnecyProcessor(handler));
  shared_ptr<TServerTransport> serverTransport(new
    TServerRdma(port));
  shared_ptr<TTransportFactory> transportFactory(new
    TBufferedTransportFactory());
  shared_ptr<TProtocolFactory> protocolFactory(new
    processor, serverTransport, transportFactory,
    protocolFactory);

  server->serve();
}
```

Server Example

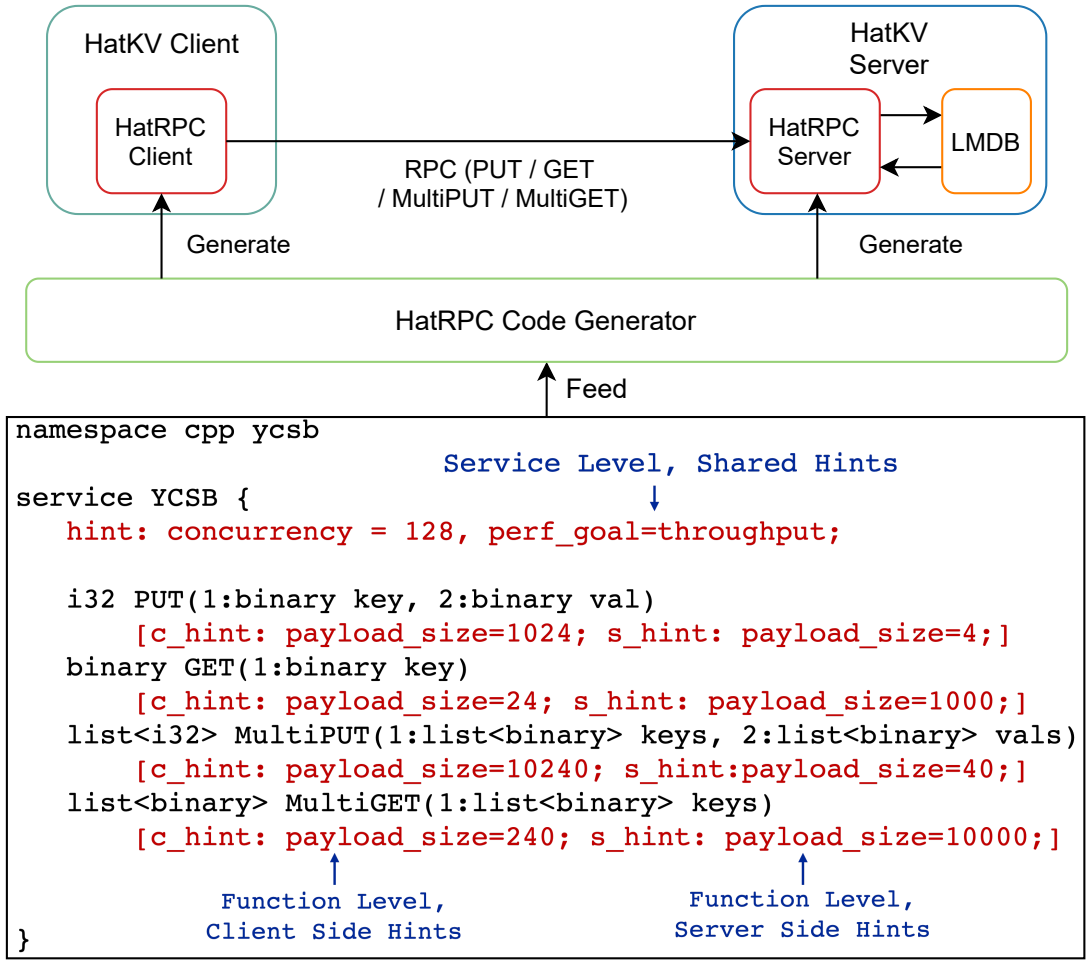
```cpp
#include <thrift/transport/TRdma.h>
#include ...

int main(int argc, char const *argv[])
{
  shared_ptr<TRdma> rdma_socket(new TRdma(ip, port));
  shared_ptr<TTransport> transport(
    new TBufferedTransport(socket));
  shared_ptr<TProtocol> protocol(
    new TBinaryProtocol(transport));

  latencyClient client(protocol);
  transport->open();

  }
  client.finish();

  transport->close();
}
```

Client Example

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

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



HatKV and IDL Example for YCSB Workloads

Overview

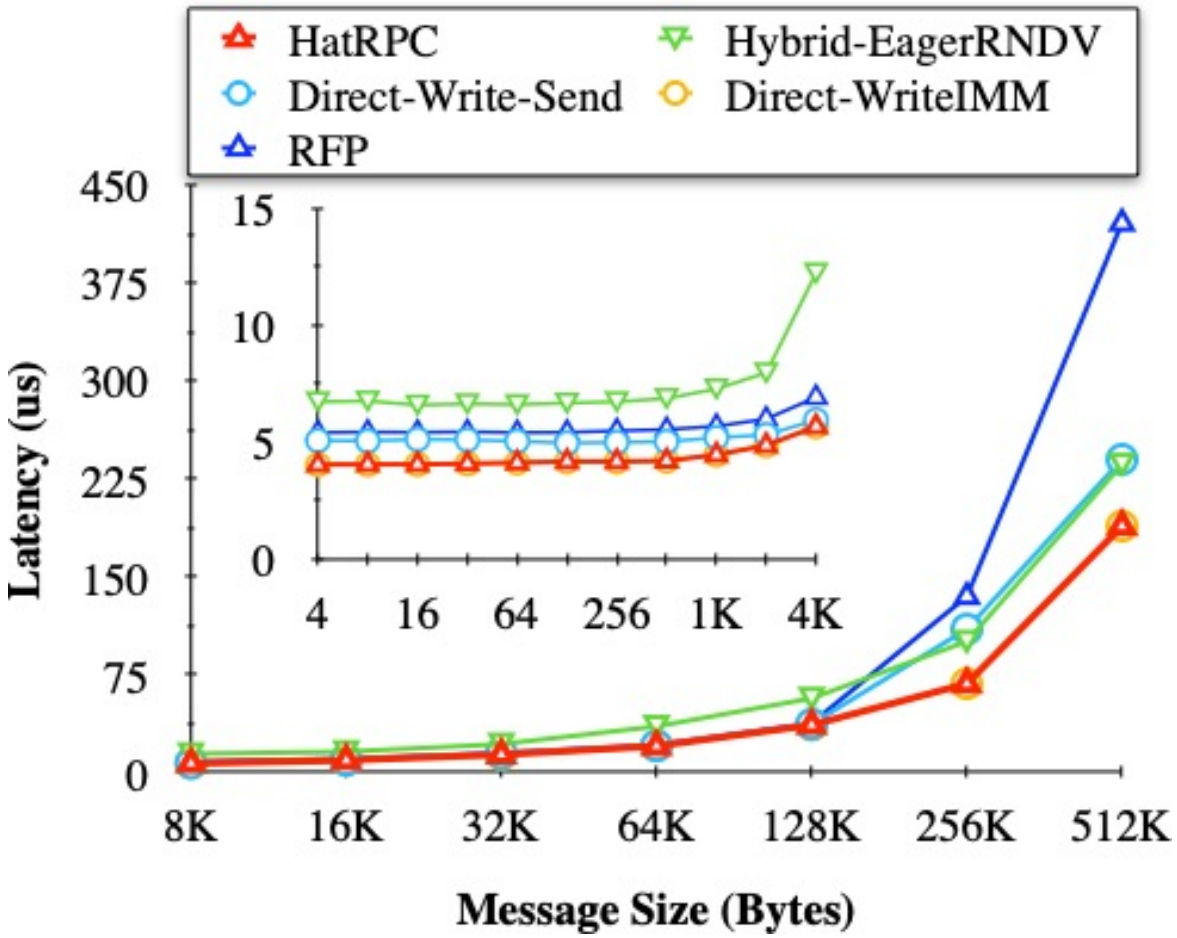
- Introduction
- Motivation
- HatRPC Design
- **Evaluation**
- Conclusion

Experimental Setup

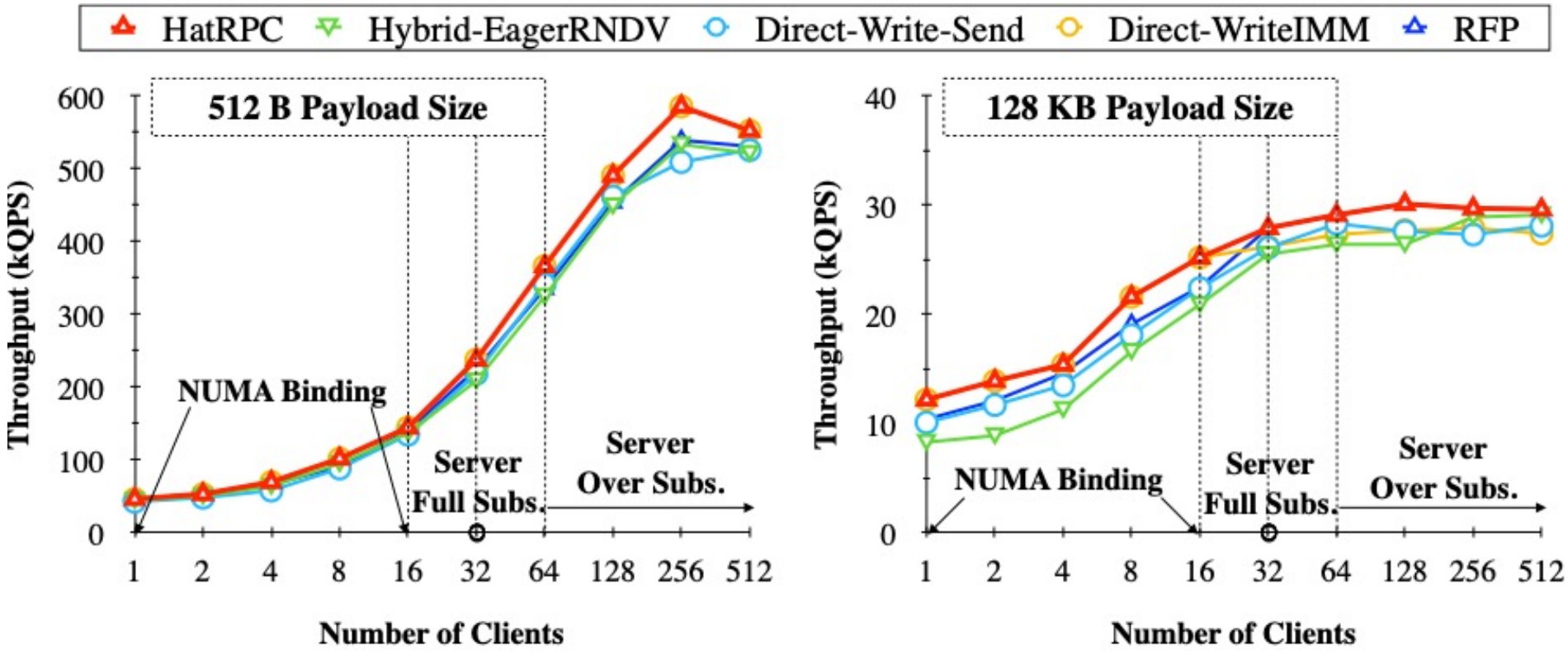
	Cluster A
Processor	Intel Skylake Gold6132 (2.6 GHZ)
RAM (DDR)	192 GB
Storage	720 GB SSD
Interconnect	ConnectX-5 IB-EDR (100 Gbps)
OS	CentOS Linux 7.6.1810
OFED	OFED-5.0-2.1.8
Scale	10 nodes

Microbenchmark - Latency

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

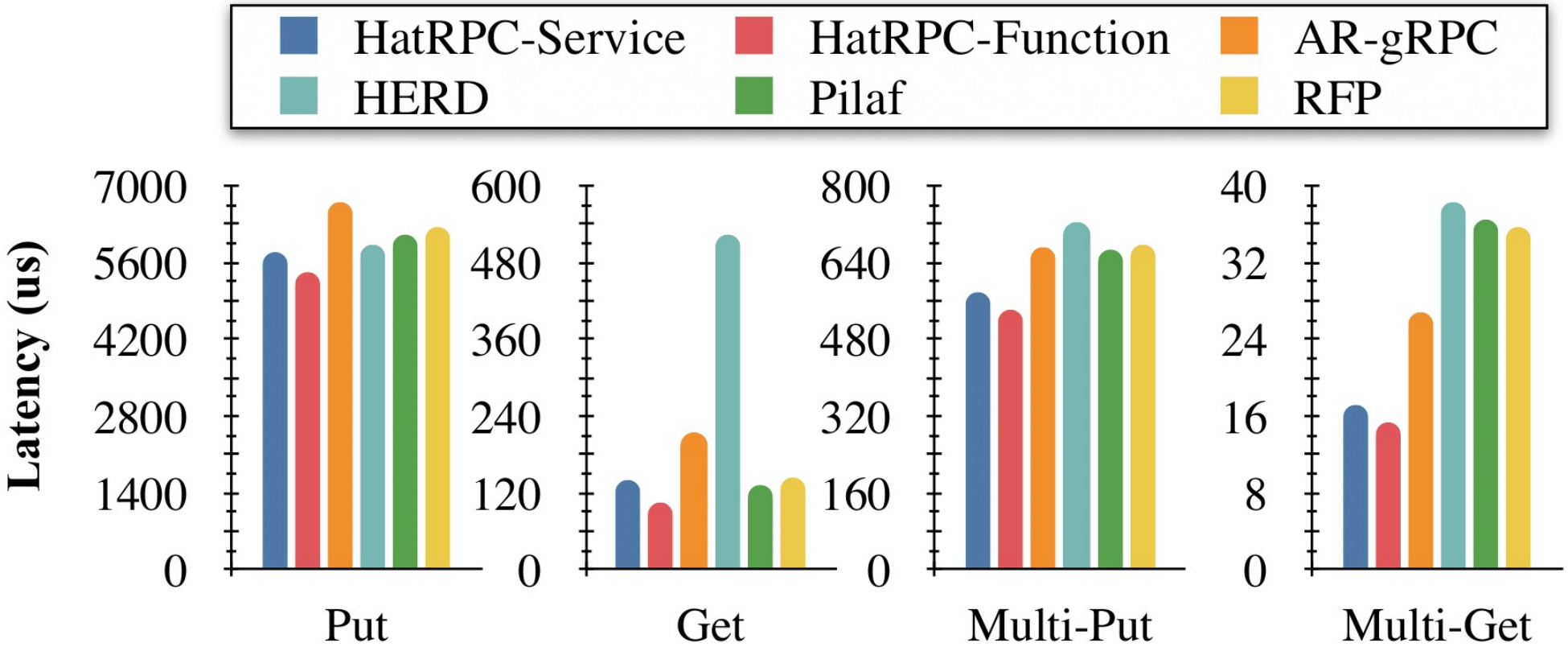


Microbenchmark - Throughput



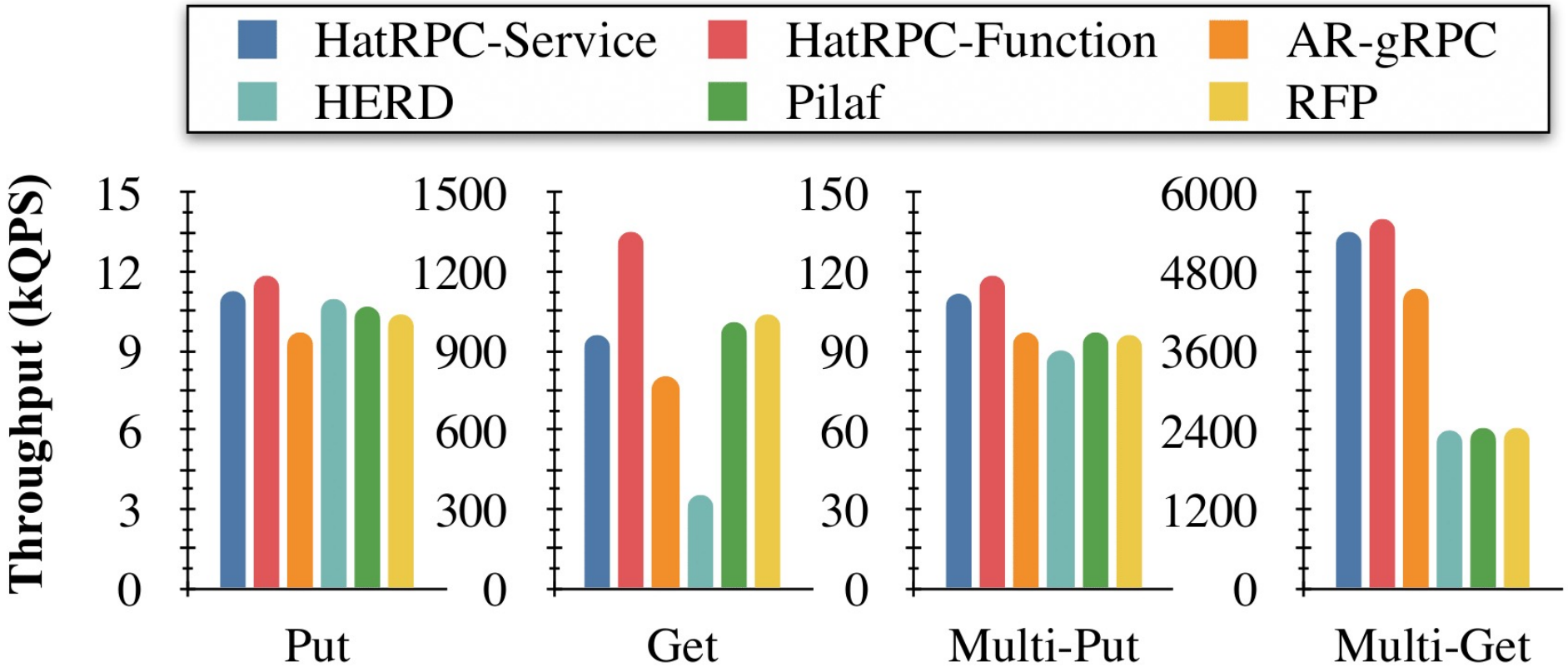
- HatRPC switches from Direct-WriteIMM to RFP when over-subscription (32), yielding up to 20% improvement for 512 B messages and up to 56% for 128 KB messages

YCSB Workload A Evaluation - Latency



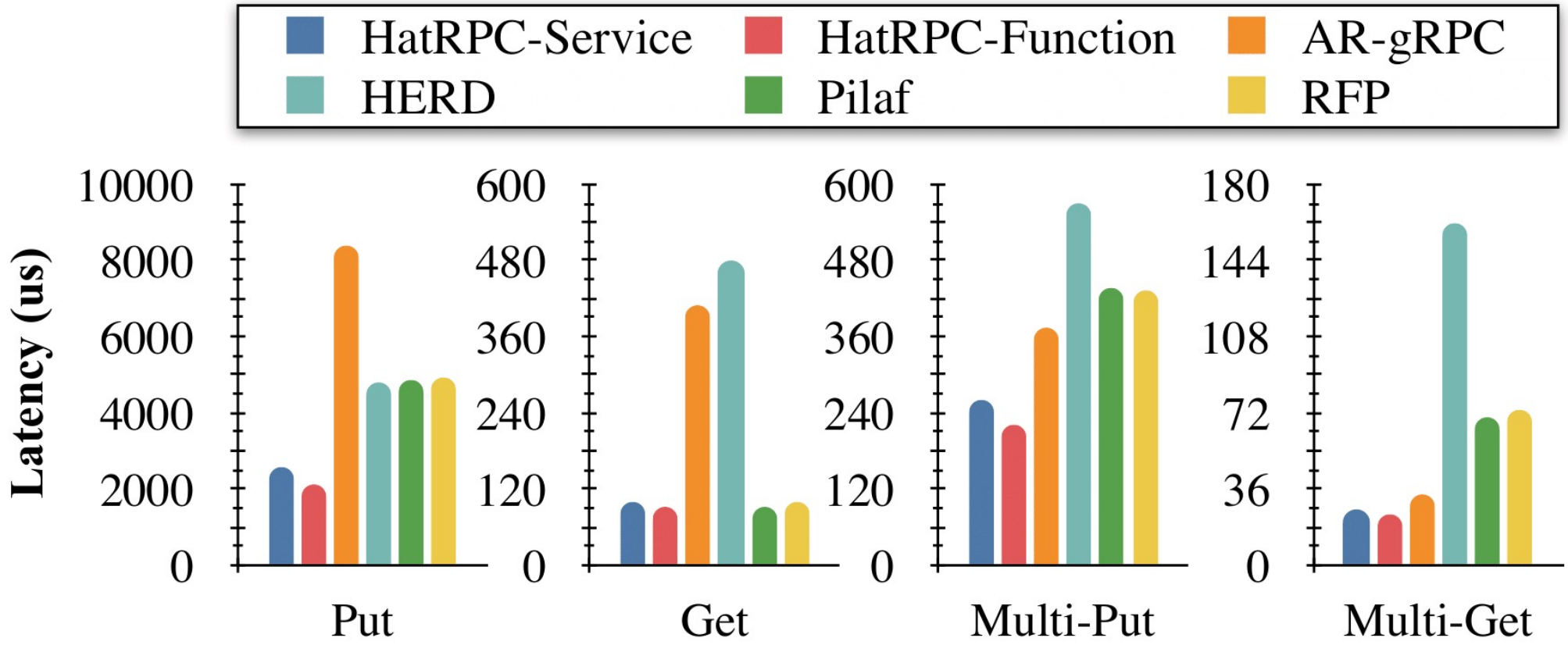
- For YCSB workload A, HatRPC-Service and HatRPC-Function reduce latency by up to 73% and 80%, respectively

YCSB Workload A Evaluation - Throughput



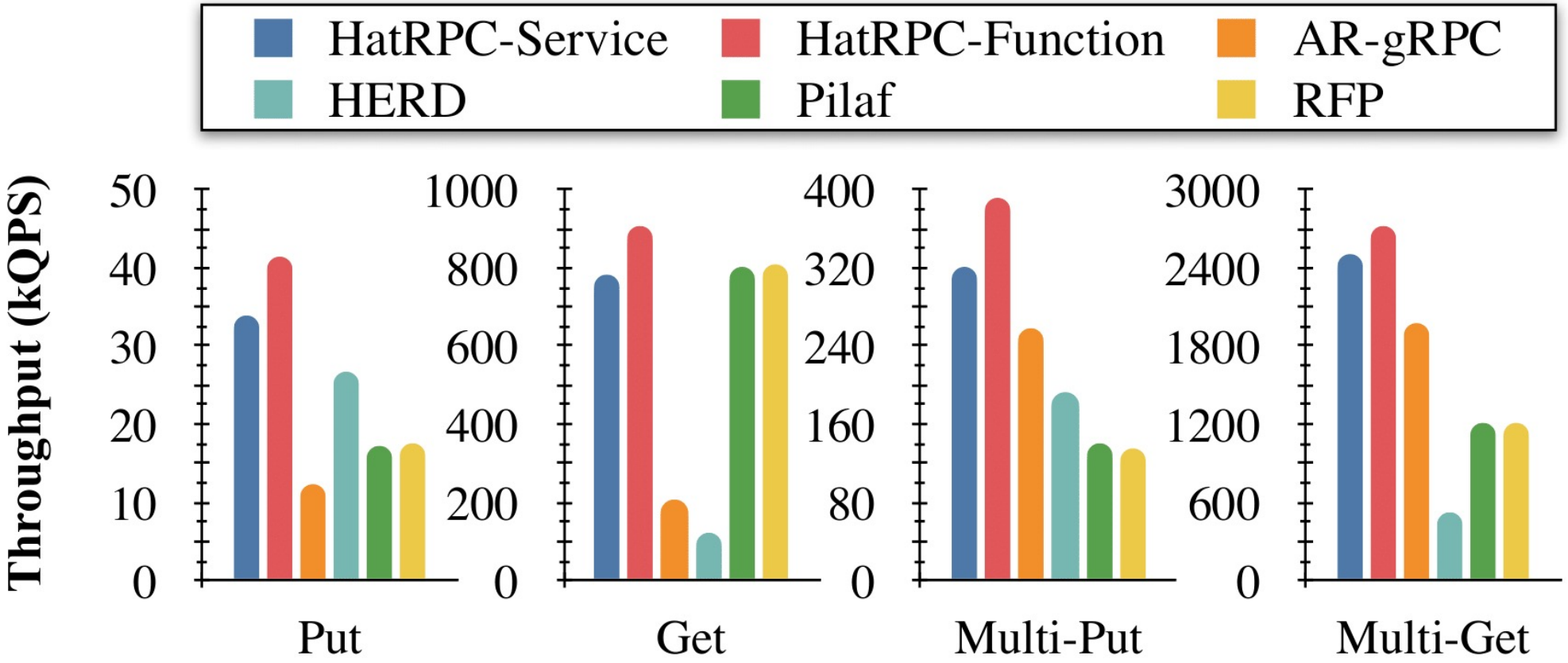
- For YCSB workload A, HatRPC-Service and HatRPC-Function gain a speedup of 2.7x and 3.8x, respectively

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

Courtesy: Tianxi Li*, Haiyang Shi*, and Xiaoyi Lu. HatRPC: Hint-Accelerated Thrift RPC over RDMA. In Proceedings of the 34th International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2021. (*Co-First Authors)

Thank you!

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

UNIVERSITY OF CALIFORNIA
MERCED