Taming Latency In Data Center Applications

Ph.D. Defense of Dissertation

Mohan Kumar          Advisor: Taesoo Kim
Motivation: Importance of Latency

Speed Matters for Google Web Search

Jake Brutlag
Google, Inc.
June 22, 2009

The Cost of Latency

March 13, 2015, jWitten by Farhan Khan, Director of Solution Architecture

Latency Critical In Data Center Applications

retail giant found that every 100ms of latency cost them 1% of sales. At the same time, a study by Tabb Group revealed that a broker lost $4 million in revenues per millisecond if its electronic trading engine was 5ms behind the competition.

Although low-latency connectivity has become more common, platforms like Amazon, Tabb and others began this conversation about 15 years ago, low latency is still a critical element of business success in the modern era. It’s practically a foregone conclusion that all companies these days will prioritize...
Contemporary Data Center Characteristics

- Optimized network – microsecond round-trip time
- Moving from 10/25 Gbps to 100/200 Gbps network
- Software running in servers induce high latency:
  - 66% of the inter-rack latency [1]
  - 81% of the intra-rack latency [1]

[1] Network requirements for resource disaggregation, OSDI’16
**Data Center Applications - Server Latency**

- **Protocol stack - 80% overhead**
- **TLB shootdown - 30% overhead**

**Key-value Stores**
- [redis](https://redis.io)

**Web Servers**
- [NGINX](https://nginx.org)
- [Apache](https://apache.org)
- [Cloud Spanner](https://cloud.google.com/spanner)

**Consensus - 82% overhead**
- [Apache ZooKeeper™](https://zookeeper.apache.org)

**Distributed Services**
- [bing](https://www.bing.com)
- [facebook](https://www.facebook.com)
- [amazon](https://www.amazon.com)
System abstractions and optimizations are needed at different levels of the software stack, from the software services running in the user space and the kernel to the software running on SmartNICs, to reduce the latency and improve the throughput of current data-center applications.
Data Center Applications - Server Latency

- Protocol stack - Xps
- TLB shootdown - LATR
- Key-value Stores
  - Redis
- Web Servers
  - Apache
  - NGINX
- Consensus - Dyad
- Distributed Services
  - Apache ZooKeeper™

Protocol stack - 80% overhead
TLB shootdown - 30% overhead
Consensus - 82% overhead
Taming Application Latency- Thesis

- **Xps - Extensible Protocol Stack:**
  - Abstraction in kernel and user-space protocol stacks, and SmartNICs
  - Reduces Redis latency by up to 73.3%

- **LATR - Lazy Translation Coherence:**
  - Kernel mechanism for free operations, page migration and swapping
  - Reduces Apache latency by up to 26.1%
Taming Application Latency - Thesis

- Dyad - Untangling Logically-Coupled Consensus:
  - Abstraction in SmartNIC for consensus
  - Reduces timestamp server latency by up to 79%
Dyad: Untangling Logically-Coupled Consensus
Motivation - Consensus Algorithms

The Amazon Outage in Perspective: Inevitable, Expensive, Risky

An endless stream of tweets last week’s Amazon Web was an indictment of public work at other cloud providers. Still others have to be sure to hammer negotiations, just to ensure...

The average total cost of unplanned application downtime per user is $1.25 billion to $2.5 billion.

The average cost of a critical application failure per hour is $500,000 to $1 million.
Consensus Algorithms

- Consensus Algorithms:
  - Provides high availability by state machine replication
  - Keeps data consistent - linearizable
  - Consensus algorithms:
    - Multi-Paxos/Viewstamp Replication (VR)
    - Raft and Zookeeper Atomic Broadcast (ZAB)
Consensus Algorithms - Applications

- Timestamp Servers
- Key-value stores
- Database
- Lock managers

Distributed Services
Dyad: Untangling Logically-Coupled Consensus

- Background
- Overview
- Design and Evaluation
- Conclusion
Consensus – VR Data Operation

1. Ordering
   - Request

2. Replication
   - Prepare
   - PrepareOK
   - Exec()

3. Ordered execution
   - Commit

Client
- Replica 1/Leader
- Replica 2
- Replica 3

Consensus

Application
Consensus – ZAB or Raft Data Operation

1. Ordering
   - Client request

2. Replication
   - Replica 1/Leader propose
   - TCP
   - Replica 2
   - Replica 3

3. Ordered execution
   - exec()
   - Disk
   - TCP
   - ack
   - commit
   - response

Consensus

Application
Replicas in a Data Center

Replica 1 - Leader
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Replica 2
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Replica 3
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Data Center Network (μs RTT)

Client Requests
Logically-Coupled Consensus

Application
Consensus
BSD socket
Linux epoll
Protocol Processing
PCIe
NIC
Network
Host

Replica

~0.8 μs [1]
~10 μs

[1] Understanding PCIe performance for end host, SIGCOMM’18
Consensus – VR Data Operation

1. Ordering
   - Client request
2. Replication
   - Leader prepare
   - Replica 1 prepare
   - Replica 2 prepare
3. Ordered execution
   - Leader prepareok
   - Replica 1 prepareok
   - Replica 2 prepareok
   - Commit

- PCIe
- Protocol processing
- Context switch
- Application
- ~11 μs
Consensus – Direct Cost of Latency

<table>
<thead>
<tr>
<th>System</th>
<th>Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR</td>
<td>67 μs</td>
</tr>
</tbody>
</table>
Consensus – Indirect Cost of Latency

Consensus - high system overhead due to direct and indirect cost
Consensus Latency - Increasing Replicas

Consensus latency is up to 82% of the end-to-end latency
Consensus Operations

● Data Operation:
  ➢ Critical path for handling a client request

● Control Operations:
  ➢ Recovery – application recovery after failure
  ➢ View Change – new replicas joining/leaving the group, new leader
  ➢ Heartbeats – health status messages exchanged across replicas
Cost of Consensus - Summary

- Every client request has high consensus overhead
- Consensus algorithms share resources with application
- Consensus overhead increases with increasing replicas
Consensus - Existing Research

- Network approaches:
  - NoPaxos, Speculative Paxos - relies on network to order requests
  - NetPaxos - proposal to execute Paxos in programmable switches

- Hardware approach:
  - Logically coupled consensus in hardware (FPGA)
  - Application is limited by the resources available on FPGA
Dyad: Untangling Logically-Coupled Consensus

- Background
- Overview
- Design and Evaluation
- Conclusion
Logically-Coupled Consensus

Application
Consensus - Control
BSD socket Linux epoll
Protocol Processing
PCIe
NIC
Network

Host
Dyad: Untangling Logically-Coupled Consensus

- Application
- Consensus - Control
- BSD socket
- Linux epoll
- Protocol Processing
- Consensus – Data
- SmartNIC
- PCIe
- Network
- Host
- Replica
Dyad: Untangling Logically-Coupled Consensus

Logically-Coupled Consensus

Dyad Consensus
Dyad: Classifying Consensus Operations

- **Data Operation - SmartNIC:**
  - Critical path for handling a client request

- **Control Operations - Host:**
  - Recovery – application recovery after failure
  - View Change – new replicas joining/leaving the group, new leader
  - Heartbeats – health status messages exchanged across replicas
Dyad: Untangling Logically-Coupled Consensus

● Background

● Overview

● Design and Evaluation

● Conclusion
Dyad – Viewstamp Replication (VR)

1. Ordering
   - Client request to Replica 1/Leader
   - Latency: 2.6 μs

2. Replication
   - Replica 1/Leader to Replica 2
   - Latency: 3.5 μs
   - Replica 1/Leader to Replica 3
   - Latency: 1.7 μs

3. Ordered execution
   - Replica 2 to Replica 1/Leader
   - Latency: 3 μs
   - Replica 3 to Replica 1/Leader
   - Latency: 3 μs

- 0.1 μs
- 3 μs
- 1.5 μs

Timelines:
- Preparing: 3 μs
- Preparing ok: 1.5 μs
- Committing: 1.5 μs

Legend:
- PCie
- Protocol processing
- Context switch
- Application
- SmartNIC
Dyad – Direct Cost

1. Ordering
2. Replication
3. Ordered execution

Client
Replica 1/Leader
Replica 2
Replica 3

SmartNIC

<table>
<thead>
<tr>
<th>System</th>
<th>Direct</th>
<th>Indirect</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR</td>
<td>67 μs</td>
<td>85 μs</td>
</tr>
<tr>
<td>Dyad</td>
<td>12.8 μs</td>
<td></td>
</tr>
<tr>
<td>% Reduction</td>
<td>81%</td>
<td></td>
</tr>
</tbody>
</table>
Dyad – Indirect Cost

1. Ordering
2. Replication
3. Ordered execution

Direct and indirect cost reduced by Dyad

% Reduction
81% 92%

PCIe Protocol processing Context switch Application SmartNIC

commit
1.5 μs

0.1 μs
Dyad: SmartNIC Primitives

- **Hardware Filtering:**
  - Specify packet format in domain-specific language (P4)
  - Filter messages based on the header and payload
  - Filters are applied to messages coming from the network and the host
Dyad: SmartNIC Primitives

- Packet Processing:
  - Filtered messages invoke request/consensus/response handlers
  - Handlers drop/forward/modify a packet
  - Generate new packets
Dyad: SmartNIC Primitives

Ingress
H/W filter
(P4)

C Handlers

Egress
H/W filter
(P4)

PCIe

Network
Dyad - Leader Data Operations

1. Ordering
   - Client request

2. Replication
   - Leader prepare
   - Replica 1 prepare
   - Replica 2 prepare
   - Leader prepareok
   - Replica 1 prepareok
   - Replica 2 prepareok
   - To host response

3. Ordered execution
   - Commit

- PCIe
- Protocol processing
- Context switch
- Application
- SmartNIC
Dyad: Ordering on Leader SmartNIC

1. Request
   - Client

2. Assign sequence number and Log
   - Leader SmartNIC
   - Replica

3. Prepare 2
   - Replica

PCIe

Ordered Log

Consensus - Data

Request Handler

2, 3

2, 3

2, 3
Dyad: Replication on Leader SmartNIC

1. Prepareok 1
2. Majority prepareok for request 1
3. Request 1
Dyad: Reordered Consensus Message

1. Prepareok 2
   - Replica 2

2. Majority prepareok for request 2
   - Leader SmartNIC
   - Replica 2

3. Request not sent to host
   - PCIe

Ordered Log:
- 2
- 1
- 3
- 2, 3

Consensus - Data

Prepare Handler
Dyad: Reordered Consensus Message

1. Leader SmartNIC
2. Majority prepareok for request 1
3. Request 1 & 2

Prepared Handler

Consensus - Data

Ordered Log

PCIe

Network
Dyad: Response and Commit

1. Response
2. Update log meta-data
3. Response

Leader
SmartNIC

Response Handler

Consensus - Data

Client

Replica

Commit

Ordered Log

PCIe
Dyad: Timestamp Server with 5 replicas

➢ Reduce latency by up to 76%, Improves throughput by 5.8x

~2 Million messages processed on the NIC
Dyad – Replica Data Operations

1. Ordering
Client request to Leader

2. Replication
Leader prepare
Replica 1 prepare
Replica 2 prepare
Leader prepareok
Replica 1 prepareok
Replica 2 prepareok
Leader commit

3. Ordered execution
Leader response to host
Replica 1 response
Replica 2 response

Legend:
- PCie
- Protocol processing
- Context switch
- Application
- SmartNIC
Dyad: Ordering on Replica SmartNIC

● Ordering and Logging:
  ➢ Logs ordered by the sequence number in prepare message
  ➢ Prepare message are processed and dropped on the SmartNIC

● Ordered Execution:
  ➢ Commit messages forwarded to the host processor
  ➢ The request is appended to the commit message by SmartNIC
Dyad: Logging on Replica SmartNIC

1. Prepare 2
   - Leader
   - Replica SmartNIC

2. Log request using sequence number
   - Leader
   - Leader

3. Prepareok 2
   - Replica SmartNIC

Consensus - Data

Ordered Log

PCIe
Dyad: Ordered Execution on the Replica

1. Commit 1
2. Leader
3. Replica
4. SmartNIC
5. PCIe
6. Ordered Log
7. Consensus - Data
8. Commit Handler
9. Verify order of received commit
Dyad: Timestamp Server with 5 replicas

➢ Reduce latency by 30 μs
## Dyad: Consensus Latency

<table>
<thead>
<tr>
<th>System</th>
<th>Consensus latency (μs)</th>
<th>% reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR</td>
<td>350</td>
<td>N/A</td>
</tr>
<tr>
<td>VR-batching</td>
<td>409</td>
<td>N/A</td>
</tr>
<tr>
<td>Dyad-Leader</td>
<td>48</td>
<td>86%</td>
</tr>
<tr>
<td>Dyad-All</td>
<td>17</td>
<td>95%</td>
</tr>
</tbody>
</table>

Timestamp server - 5 replicas
Dyad: CPU Usage Timestamp Server

➢ Reduce CPU usage by up to 70% on the leader
Dyad: Application Failures

92% catastrophic failure - due to software [1]

Dyad: Detecting Application Failures

![Diagram showing the flow of data through various layers including Application, Consensus - Control, Protocol Processing, BSD socket, Linux epoll, Consensus - Data, and SmartNIC. The diagram also highlights the Host RTT and network connections between Replica and Host.]
Dyad: Detecting Application Failures

- Measure host RTT for each request
- Computed weighted average of host RTTs
- Detect failure - response not within host RTT threshold
Application Recovery - VR

Replica 1 - Leader

- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Replica 2

- Application Restart
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Log Transfer

Replica 3

- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- PCIe
- NIC

Data Center Network (μs RTT)

Client Requests
Dyad: Application Recovery

● Recovery using logs on SmartNIC

● Two stage recovery:
  ➢ Recover logs from the SmartNIC
  ➢ Recover remaining logs from other replicas
Dyad: Application Recovery

➢ Dyad reduces recovery time by up to 67%
Dyad: SmartNIC Failure

Replica

- Protocol Processing
  - BSD socket
  - Linux epoll
  - Consensus - Control
  - Application

Host

- PCIe
- Network

SmartNIC

Consensus – Data

X

60
Dyad: System Failure

8% - hardware faults, misconfigs [1]

Simple Testing Can Prevent Most Critical Failures, OSDI’14
Dyad: System Recovery

● SmartNIC Failure:
  ➢ Detected on the host using heartbeat/client messages
  ➢ Existing VR recovery: fetch remaining logs from other replicas

● System Failure:
  ➢ Existing VR recovery: fetch logs from other replicas
  ➢ Dyad supports logging to disk from host (Raft)
Dyad: Reliable Connection

- Dyad Supports Raft:
  - Using TCP connection to replicas
  - TCP stack specifically decode Raft headers and payload
  - Host application logs client commands to disk for persistence
Dyad: Raft Latency

➢ Improves latency by up to **62%**
Dyad: Ease of Use

- Memcached:
  - Enable consensus for Memcached
    - ~100 lines of code for data operations on replica
  - Evaluate impact on latency and throughput
Dyad: Memcached Throughput

➢ Provides consensus with ~7% reduction in throughput
Dyad: Memcached Latency

➢ Provides consensus with \(~16\%\) increase in latency
Dyad: Untangling Logically-Coupled Consensus

● Motivation
● Background
● Overview
● Design and Evaluation
● Conclusion
Dyad: Conclusion

- SmartNIC abstraction for consensus
- Data operations performed on the SmartNIC
- Control operations performed on the Host
- Enables consensus as a service on SmartNICs
System abstractions and optimizations are needed at different levels of the software stack to reduce the latency and improve the throughput of current data-center applications.
Thank you!
Backup Slides
Arrakis
Redis comparison with Arrakises
Latr - Apache
Latr - Apache latency

Latency (Milli seconds)

- Apache
- Latr-Apache
User-Space Stacks
# User Space: Protocol processing

<table>
<thead>
<tr>
<th>Systems</th>
<th>Latency (μs)</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTCP</td>
<td>~23</td>
<td>Batching</td>
</tr>
<tr>
<td>IX</td>
<td>~12</td>
<td>Batching</td>
</tr>
<tr>
<td>Arrakiss</td>
<td>~2.6 - 6.3</td>
<td>None</td>
</tr>
</tbody>
</table>
VR: IX batching with 3 Replicas

![Graph showing latency vs packets per second for VR, VR-batching, Dyad-leader, Dyad-all, and IX-batching.]
Context Switch
VR - Leader Context Switch

![Graph showing the relationship between Packets Per Second (PPS) and Context Switch. The graph indicates a linear increase in Context Switch as PPS increases.]
Dyad - Parallelism
Dyad: Application Parallelism

• Without SmartNIC:
  ➢ Sequence numbers are available in prepareok message
  ➢ Multi-thread execution by using the sequence number

• Dyad:
  ➢ Request are ordered without containing the sequence number
  ➢ SmartNIC appends the sequence number to the client request
Dyad: Parallelism Timestamp Server

➢ Improves throughput by up to 2.1x
Reading Logs
Dyad: Log Read Throughput

➢ Log read throughput ~256 MB with 16 threads
Direct Cost Formula
Cost of Consensus - Direct and Indirect

\[
Latency_{direct} = \frac{n-1}{2} \times RTT + (n-1) \times TX + \frac{n-1}{2} \times RX
\]

Leader prepare

\[
\text{Leader prepare}
\]

Replicas prepare

Consensus overhead increases with increasing replicas

\[
Latency_{indirect} = \frac{n-1}{2} \times RX + (n-1) \times TX
\]

Replicas commit

\[
+ (n-1) \times RX + t_{processing}
\]
VR Recovery Data Transfer
### Application Recovery - VR data transfer

<table>
<thead>
<tr>
<th>Replicas</th>
<th>Log Size (MB)</th>
<th>Data transferred (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>600</td>
</tr>
</tbody>
</table>
False Positives RTT
Dyad: False Positives with Timestamp Server

➢ RTT = ~96 μs
SmartNIC - Netronome
SmartNIC: Memory Hierarchy and Latency
Recovery Example
Dyad - Recovery Phase 1

Replica 1 - Leader
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- SmartNIC

Replica 2
- Application Restart
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- SmartNIC

Replica 3
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- SmartNIC

Data Center Network (μs RTT)

Client Requests

1, 2
Dyad - Recovery Phase 2

Data Center Network (μs RTT)

Replica 1 - Leader
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- NIC
- PCIe

Client Requests

Replica 2
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- Log Transfer
- NIC
- PCIe

Replica 3
- Application
- Consensus
- BSD socket
- Linux epoll
- Protocol Processing
- NIC
- PCIe

Log Transfer

Replica 1

Replica 2

Replica 3
Raft - Logging to Disk
Dyad: Raft Latency with disk logging

➢ Improves latency by up to 46%
Dyad - Future Work
Dyad: Future Work

- Logging to disk from SmartNIC:
  - Possible with NVMe over fabric
  - Possible over PCIe? - ARM, FPGA, or NPU

- Optimize request handling:
  - Sending parsed requests to host