Towards a Redundancy-Aware Network Stack for Data Centers

Ali Musa Iftikhar (Tufts)
Fahad R. Dogar (Tufts)
Ihsan Ayyub Qazi (LUMS)
The Problem of Tail Latency in Data Centers!
The Problem of Tail Latency in Data Centers!
The Problem of **Tail Latency** in Data Centers!

- Load imbalance
- Background tasks
- Failures, etc.

**High fan-out**

**Straggler**

- Load imbalance
- Background tasks
- Failures, etc.
The Problem of **Tail Latency** in Data Centers!

- Load imbalance
- Background tasks
- Failures, etc.

High fan-out + Stragglers → Long tail latency
How to avoid stragglers?

- Reactively
- Proactively
How to avoid stragglers?

- **Reactively**
  - Hopper (SIGCOMM’15)
  - C3 (NSDI’15)
  - Sinbad (SIGCOMM’13)

  **PRO:** Low overhead

- **Proactively**

  **CON:** Requires straggler detection (slow and inaccurate)
How to avoid stragglers?

**Reactively**
- Hopper (*SIGCOMM’15*)
- C3 (*NSDI’15*)
- Sinbad (*SIGCOMM’13*)

**PRO**: low overhead
**CON**: requires straggler detection (slow and inaccurate)

**Proactively**
- Dolly (*NSDI’13*)
  - Low latency via redundancy (*CoNext’13*)

**PRO**: fast and accurate
**CON**: requires determining threshold load (non-trivial)
How to avoid stragglers?

**Reactively**
- Hopper *(SIGCOMM’15)*
- C3 *(NSDI’15)*
- Sinbad *(SIGCOMM’13)*

**PRO:** low overhead

**CON:** requires straggler detection (slow and inaccurate)

**Proactively**
- Dolly *(NSDI’13)*
  - Low latency via redundancy *(CoNext’13)*

**PRO:** fast and accurate

**CON:** requires determining threshold load (non-trivial)

Can we achieve the benefits of both without their limitations?
Overview

• Duplicate-Aware Scheduling Framework
  Generic framework

• Redundancy-Aware Network Stack
  New network stack for DC

• Preliminary Results
Duplicate-aware scheduling
1. Priority Queues
Duplicate-aware scheduling

1. Priority Queues

Client → Request → Replica 1

Client → Request → Replica 2
Duplicate-aware scheduling

1. Priority Queues

request

Client

Replica 1

Replica 2
Duplicate-aware scheduling

1. Priority Queues

[Diagram showing a client sending requests to two replicas, with priority queues indicated by red and blue bars.]

Request
Client

Replica 1
high
low

Replica 2
high
low
1. Priority Queues

Duplicate-aware scheduling

Client

request

Replica 1

Replica 2

P

high

B

low

low

high
Duplicate-aware scheduling

1. Priority Queues

Client

Request response (P)

Replica 1

Replica 2

request
Duplicate-aware scheduling

1. Priority Queues
2. Purging

Client

Request response (P)

Replica 1

high
low

Replica 2

high
low

request

purge

18
Need for **Priority Queuing**

- Duplication has an overhead!

![Diagram with primary and backup sections, high and low priorities]
Need for **Priority Queuing**

- Duplication has an overhead!

Properties required:

- Strict priorities
- Work conservation
- Preemption
Need for **Priority Queuing**

- Duplication has an overhead!

Properties required:

- Strict priorities
- Work conservation

**PQ** makes the overhead of duplication **low**.
Need for Priority Queuing

- Duplication has an overhead!

Properties required:
- Strict priorities
- Work conservation

PQ makes the overhead of duplication low.

essential
Importance of Purging

- Stale requests block new requests.

![Diagram showing stale requests blocking new requests.]

req1

req2

high

low

stale
Importance of Purging

- Stale requests block new requests.

Purging makes the system more efficient!
Importance of Purging

➢ Stale requests block new requests.

Purging makes the system more efficient!
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC

Network

Compute

Memory

File system / Database

HDFS GFS BigTable

redis memcached
Realizing Duplicate-Aware Scheduling

at every **potential bottleneck** resource in a DC
Realizing Duplicate-Aware Scheduling

at every potential bottleneck resource in a DC

challenges

Network
Compute
Memory
File system / Database
Storage

In-network purging

Prioritization

Purging + preemption
Redundancy Aware Network Stack (RANS)

Layer | New Role
--- | ---
Application | Duplicate-Awareness
Transport | Point to multipoint + purging
Network | Priority Queues + purging
Link | Same as before
Physical | Same as before

Expressive Interface
Redundancy Aware Network Stack (RANS)

Layer          New Role
Application    Duplicate-Awareness

Expressive Interface

Transport
Point to multipoint
+ purging

Network
Priority Queues
+ purging

Link
Same as before

Physical
Same as before

Applications need to be modified.

Challenge

Applications need to be modified.
Redundancy Aware Network Stack (RANS)

- **Layer**
  - Application
  - Transport
  - Network
  - Link
  - Physical

- **New Role**
  - Duplicate-Awareness
  - Point to multipoint + purging
  - Priority Queues + purging
  - Same as before

- **Expressive Interface**

- **Challenge**
  - Applications need to be modified.

- **Opportunity**
  - Expressive interface allows rich communication b/w App and Transport.
  - E.g. DAG

- **Diagram**
  - Expressive interface allows rich communication b/w App and Transport.
  - E.g. DAG
  - get(<10ms,...)

- **Applications**
  - Applications need to be modified.
  - Expressive interface allows rich communication b/w App and Transport. E.g. DAG
  - get(<10ms,...)
  - Expressive Interface
Redundancy Aware Network Stack (RANS)

- **Layer**
  - Application
  - Transport
  - Network
  - Link
  - Physical

- **New Role**
  - Duplicate-Awareness
  - Point to multipoint + purging
  - Priority Queues + purging
  - Same as before
  - Same as before

- **Expressive Interface**

- **Applications need to be modified.**

- **Opportunity**
  - Expressive interface allows rich communication b/w App and Transport.
  - E.g. DAG

- **Challenge**
  - Hard to implement per packet purging.

**Applications** need to be modified. The expressive interface allows rich communication between the Application and Transport. For example, DAG. Challenges include:

- Hard to implement per packet purging.
Redundancy Aware Network Stack (RANS)

- **Layer**: Application
  - New Role: Duplicate-Awareness
- **Layer**: Transport
  - New Role: Point to multipoint + purging
- **Layer**: Network
  - New Role: Priority Queues + purging
- **Layer**: Link
  - Same as before
- **Layer**: Physical
  - Same as before

**Applications need to be modified.**

Expressive interface allows rich communication b/w App and Transport. E.g. DAG

- **Challenge**: Hard to implement per packet purging
- **Opportunity**: Adds support for existing PQs in DC switches.
Redundancy Aware Network Stack (RANS)

Layer

Application
- Duplicate-Awareness

Transport
- Point to multipoint + purging
- Priority Queues + purging

Network

Link

Physical
- Same as before
- Same as before

Expressive Interface

Applications need to be modified.

Expressive interface allows rich communication b/w App and Transport.
E.g. DAG

Hard to implement per packet purging

Challenge

Adds support for existing PQs in DC switches.

Opportunity
RANS Transport: Point to Multi-point

- Enables: Rich transport
  - Multipath
  - Multi-destination
    e.g. Improved fault tolerance
RANS Transport: Byte Aggregation

- Opportunity: Receiver driven transport
  - Two or more response streams
  - Aggregate bytes at receiver side
    - e.g. More efficient congestion control (2x or more)
RANS Transport: Priority Assignment

- Dynamic replica assignment
- Fine grained monitoring of congestion window
- Dynamically reprioritize flows
- Feedback to Application (e.g. Improved replica assignment)
Overview

• Duplicate-Aware Scheduling Framework
• Redundancy-Aware Network Stack
• Preliminary Results
Preliminary Evaluation: ns-2 setup details

Storage scenario

Client

10 servers

1Gbps

get(10MB)

Replica 1

Replica 2
Preliminary Evaluation: ns-2 setup details

- Storage scenario

Diagram:
- Client
- 10 servers
- Replica 1
- Replica 2
- Bottlenecks
- Bandwidth: 1Gbps
- Get(10MB)
Preliminary Evaluation: ns-2 setup details

Storage scenario

Traffic Details

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total requests</td>
<td>20K</td>
</tr>
<tr>
<td>Arrival process</td>
<td>Poisson</td>
</tr>
<tr>
<td>Server &amp; replica selection</td>
<td>Uniformly random</td>
</tr>
</tbody>
</table>
Preliminary Evaluation: ns-2 setup details

- **Storage scenario**

  - **Client**
  - **10 servers**
  - **Replica 1**
  - **Replica 2**
  - **get(10MB)**
  - **1Gbps**

  **bottlenecks**

- **Traffic Details**

<table>
<thead>
<tr>
<th>Traffic Details</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total requests</td>
<td>20K</td>
</tr>
<tr>
<td>Arrival process</td>
<td>Poisson</td>
</tr>
<tr>
<td>Server &amp; replica selection</td>
<td>Uniformly random</td>
</tr>
</tbody>
</table>

- The only source of stragglers is load imbalance.
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)

![Graph showing request completion time vs. load percentage]
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
  + PQs
  + Purging
  + Byte Aggregation (RANS)
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
  + PQs
  + Purging
  + Byte Aggregation (RANS)

![Graph showing request completion time vs. load percentage]

Load (%)

Request completion time (s)

~2X
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
- + PQs
- + Purging
- + Byte Aggregation (RANS)

Expecting more gains even at lower loads with additional straggler sources.
Average request completion time of:

- No duplicates (baseline)
- 2-copies (proactive w/o PQ)
  + PQs
  + Purging
  + Byte Aggregation (RANS)

Expecting more gains even at lower loads with additional straggler sources.

50-80% improvement over the baseline across all loads.
Summary & Future work

- The Issue of Stragglers

- Duplicate-Aware Scheduling Framework
  Simple yet challenging solution

- RANS
  A first step towards a duplicate-aware network

- Implementing in HDFS and Cassandra
RANS: Feedback and Discussion

- Ali Musa Iftikhar (musa@cs.tufts.edu)
- Fahad R. Dogar (fahad@cs.tufts.edu)
- Ihsan A. Qazi (ihsan.qazi@lums.edu.pk)

Transport Link
- Network
- Physical

Layer
- Application
- Transport
- Network
- Link
- Physical

New Role
- Duplicate-Awareness
- Expressive Interface
- Point to multipoint Byte aggregation Priority assignment + purging
- Priority Queues + purging
- Same as before
- Same as before
Possible questions – backup slide

- Preemption overhead
  - Not really an issue in the network because packets are small.

- Packet purging
  - PFC (back pressure, build queues at the end hosts and purge them)
  - Drop the entire duplicate queue (easier than per-packet drops)
  - Recent trend towards programmable switches

- Gains with PQ
  - More gains with failures as stragglers (primary undergoes a failure)
  - Also more benefits with different resources

- Duplication overhead at client
  - Client is usually not the bottleneck

- Non-Idempotent requests
  - We are targeting the class of apps which have flexible end points and require at least once semantics

- Replicating only small packets and prioritizing them
  - Only beneficial with bursty small flows
  - HDFS have a typical chunk size b/w 64MB-128MB

- Quorum systems
  - RANS complements such systems, they can use this technique and send K out of N requests at high prio while N-K as backups

- Can’t just implement at the app and get the same benefits?
  - Network could be a bottleneck
  - Fine grained control, much more control

- Root causes of performance improvement
  - PQ avoids overheads
  - Now we can easily get the benefits of duplications like aggregation etc.
  - Purging will also at times purge primary making the system more efficient.
Food for thought

e.g. Google’s Geo-Distributed Database “Spanner” (OSDI’12)

DC Primary

DC Failover

Inter DC Duplicate-Aware Scheduling
Food for thought

- Inter DC Duplicate-Aware Scheduling

Spell check
- Search suggestions
  - Search engines drop spell check, suggestions, etc. at high loads.
  - Can benefit from duplicate-aware scheduling.

Pre fetch

DC Primary

DC Failover

e.g. Google’s Geo-Distributed Database “Spanner” (OSDI’12)
When RANS works best?

• Application fanout is high and stragglers are frequent.
• End-points are flexible and “at least once” semantics are sufficient.
• Client is not the bottleneck.
• Request sizes are small (or preemption overhead is minimal).