Overload Control for μs-scale RPCs with Breakwater

Inho Cho, Ahmed Saeed, Joshua Fried, Seo Jin Park, Mohammad Alizadeh, Adam Belay
Trend: μs-scale RPCs

1. Fast Network: Network latency (~ 5 us)
2. Fast Storage: M.2 NVME SSD (~ 20 us)
3. In-memory operations: Memcached, Redis, Ignite
Trend: High Fan-out

Internet

Web server

https://

Remote memory

Encryption

Cache

Storage
Causes of Server Overload

- Load Imbalance
- Unexpected user traffic
- Packet bursts
- Redirected traffic due to failure
Performance Without Overload Control

Without overload control, server overload makes almost all requests violate its SLO.
Ideal Overload Control

should keep request queue **short**, but **not empty**
should inform clients about overload quickly
Strawman #1: Server-side AQM

Clients → Server

- Request
- Drop notification

Server → Clients

drop
Strawman #1: Server-side AQM

Request
Drop notification

drop
Strawman #1: Server-side AQM

The cost of packet processing is comparable to the service time

[Diagram showing the flow of requests from clients to the server with approximate equivalence between packet processing and request execution time.]
Strawman #2: Client Rate limiting
Strawman #2: Client Rate limiting

Probing server status incur high message overhead

Hey server, are you busy?
Hey server, are you busy?
Hey server, are you busy?
Hey server, are you busy?
Breakwater

Overload control scheme for μs-scale RPCs

<table>
<thead>
<tr>
<th>Components</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Credit-based admission control</td>
<td>Coordinates requests with minimum delay</td>
</tr>
<tr>
<td>2. Demand speculation</td>
<td>Minimizes message overhead</td>
</tr>
<tr>
<td>3. Delay-based AQM</td>
<td>Ensures low tail latency</td>
</tr>
</tbody>
</table>
Breakwater’s benefits

Handles server overload with μs-scale RPCs with

1. High throughput
2. Low and bounded tail latency
3. Scalability to a large number of clients

Handles server overload with μs-scale RPCs with

Breakwater

Ideal

Throughput (kreqs/s)

99%-ile Latency (μs)

Clients' Demand (Mreqs/s)

SLO

Clients' Demand (Mreqs/s)
Queueing delay as congestion signal
Comp. #1: Credit-based admission control

Breakwater controls amount of incoming requests with credits

For every RTT:
If delay < target:
    credit += A
Else:
    \[ B = \text{MAX}(1 - \beta \cdot \frac{\text{delay} - \text{target}}{\text{target}}, 0.5) \]
    credit ×= B
Comp. #1: Credit-based admission control

Breakwater controls amount of incoming requests with credits

Clients

Server

Client 1
Client 2
Client 3
Client 4

Credit Request Response

register
Comp. #1: Credit-based admission control

Breakwater controls amount of incoming requests with credits

[Diagram showing credit-based admission control]
Comp. #1: Credit-based admission control

Breakwater controls amount of incoming requests with credits

Client 1
Client 2
Client 3
Client 4
Server

deregister

Clients
Demand Message Overhead

Server needs to know which client has demand

Clients

Server

- Client 1
- Client 2
- Client 3
- Client 4

Credit  Request  Response
Demand Message Overhead

Server needs to know which client has demand

Clients

Server

- Credit
- Request
- Response
Demand Message Overhead

Server needs to know which client has demand

Clients

I have \( n \) requests!

Server needs to know which client has demand

\( \text{Credit} \)  \( \text{Request} \)  \( \text{Response} \)
Impact of Credit-based Admission Control

Credit-based admission control has lower and bounded tail latency but lower throughput.
Piggybacking Demand Information

Breakwater piggybacks clients’ demand information into requests.

Clients

I have $n$ requests!

Server

- Credit
- Request
- Response
Piggybacking Demand Information

Breakwater piggybacks clients’ demand information into requests.

Clients

Server

I have \( n \) requests!

(I have \( n \) more request)
Comp. #2: Demand Speculation

Breakwater speculate clients’ demand to minimize message overhead
Comp. #2: Demand Speculation

Breakwater speculate clients’ demand to minimize message overhead
Comp. #2: Demand Speculation

Breakwater speculate clients’ demand to minimize message overhead
Impact of Adding Demand Speculation

Demand speculation improves throughput with higher tail latency

Demand speculation improves throughput with higher tail latency.
Credit Overcommitment

Server issues more credit than the number of requests it can accommodate.
Incast Causing Long Queue

With credit overcommitment, multiple requests may arrive at the server at the same time.
Comp. #3: Delay-based AQM

To ensure low tail latency, the server drops requests if queueing delay exceeds threshold.
Comp. #3: Delay-based AQM

To ensure low tail latency, the server drops requests if queueing delay exceeds threshold.
Impact of Adding Delay-based AQM

Breakwater achieves high throughput and low and bounded tail latency at the same time.
Evaluation

Testbed Setup
- xl170 in Cloudlab
- 11 machines are connected to a single switch
- 10 client machines / 1 server machine
- Implementation on Shenango as a RPC layer

Synthetic Workload
- Clients generate request with open-loop Poisson process
- Requests spin-loops specified amount of time at server
- Exponential service time distribution with 10μs average
Evaluation

(1) Does Breakwater achieve high throughput and low tail latency even with demand spikes?
(2) Does Breakwater provide fast notification for the rejected requests?
(3) Is Breakwater scalable to many clients?

Baselines:
- DAGOR
  - priority-based overload control used in WeChat
- SEDA
  - adaptive overload control for staged event-driven architecture
High Goodput with Fast Convergence

Breakwater converges to higher goodput 25x faster than DAGOR and 79x faster than SEDA.
Breakwater maintains low tail latency even with load spikes.
Breakwater notifies rejected request to clients before violating its SLO.
Breakwater easily scales to 10,000 clients.
Conclusion

• Breakwater is a **server-driven credit-based** overload control system for **μs-scale RPCs**

• Breakwater’s key components include
  (1) Credit-based admission control
  (2) Demand speculation
  (3) Delay-based AQM

• Our evaluation shows that Breakwater achieves
  (1) **Low & bounded tail latency** with **high throughput**
  (2) **Fast notification** for a rejected request
  (3) **Scalability** to many clients
Thank you!

Breakwater is available at

inhocho89.github.io/breakwater/

Questions?
Inho Cho <inhocho@csail.mit.edu>