Ship Compute or Data?
Why Not Both?

Jie You, Jingfeng Wu, Xin Jin, Mosharaf Chowdhury
Two Trends of System Design

Resource Disaggregation
- Disaggregated Storage
- Serverless Computing

Examples
- Ship Data

Storage-side Computation
- SQL User-Defined Function
- Redis Modules

Examples
- Ship Compute
Ship Data OR Ship Compute
Ship Data OR Ship Compute

Application Server
App Logic
Storage Server

Ship Data

Application Server
App Logic
Storage Server

Data Access
Ship Data OR Ship Compute

Ship Data

Ship Compute

Application Server

App Logic

Storage Server

GET/PUT

Application Server

App Logic

Data Access

Storage Server

Application Server

App Logic

Storage Server
Ship Data OR Ship Compute

- Application Server
  - App Logic
  - …
  - GET/PUT
  - Storage Server

- Application Server
  - App Logic
  - Data Access
  - Storage Server

- Application Server
  - RPC Endpoint
  - Storage Server

Ship Data
Ship Compute
Ship Data OR Ship Compute

Ship Data

Pros:
- Flexible provisioning
- Elastic Scaling

Cons:
- More network traffic
- Multiple network round-trips for a single request (latency inflation)

Ship Compute

Pros:
- Less network round-trips
- No marshalling overhead

Cons:
- Storage server can get overloaded

No One-Size-Fits-All Solution
Ship Data AND Ship Compute

• Measurement:
  • Workload: synthetic w/ configurable computation time and round-trips
    • Optimal: Best of sweeps across combinations of (ShipData%, ShipCompute%)

• Intuition: combining both → more CPUs utilized → higher throughput
1. Maximizes throughput under SLO constraints

2. Adaptively arbitrates between shipping compute and shipping data w/o workload knowledge

Latency-driven rate limitation and request arbitration

Dual-loop control with proved convergence
Kayak Overview

• **Combines** shipping data and shipping compute

• **Arbitrates** between shipping data and shipping compute

• **Applies** adaptive rate limitation to maximize throughput under SLO constraints
Algorithm Overview

• Kayak proactively tunes:
  • \( R \): The request rate limitation
  • \( X \): The fraction of requests to be executed using RPC/KV

• Challenges:
  • No closed-form expression of \((X, R) \rightarrow Latency\)
  • Tuning both \( X \) and \( R \) simultaneously is hard
Algorithm Design

• Solution:
  • 1. Optimize $X$ and $R$ separately via control loops
    • $R$: Rate limiting control loop
    • $X$: Request arbitration control loop
  • 2. Combine two control loops together

$R$: The request rate limitation
$X$: The fraction of request to be executed using RPC/KV
Dual Loop Control

- **Our Solution**: R-X dual loop control
  - Rate limiting loop before request arbitration loop
  - The other way around is unstable
- **Proved convergence of** $R$ and $X$
Evaluation

Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Comp. Time (μs/RTT)</th>
<th>Transactional</th>
<th>Time Varying</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>0.1~10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bimodal</td>
<td>0.1~1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>YCSB-T</td>
<td>&lt;0.1</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Baseline: ASFP [1]

[1] Adaptive Placement for In-memory Storage Functions (ATC '20)
Performance

Throughput (MOps) @ 200μs SLO

12.2%-63.4% improvement

Higher improvements w/ compute-intensive workloads
Convergence of Dual Loop Control

• **Workload:**
  - Bimodal: Switching between (1us, 100ns), w/ an interval of 5s
Fairness across Multiple Tenants

• Setup:
  • 4 tenants running on 4 separate applications servers
  • Single shared storage server

Fair sharing of Storage Server CPU
Ship Compute **AND** Ship Data

• Kayak
  • Proactively arbitrates between shipping data and shipping compute w/o workload knowledge
  • Maximizes throughput under latency SLO constraint
  • Ensures fairness across multiple tenants

• Thank you

https://github.com/SymbioticLab/Kayak