Shimmy: Shared Memory Channels for High Performance Inter-Container Communication

Marcelo Abranches*, Sepideh Goodarzy*, Maziyar Nazari*, Shivakant Mishra, Eric Keller
University of Colorado, Boulder

*All three authors contributed equally to this paper
Crowd Monitoring

https://sequare.world/tag/computer-science/
Crowd Monitoring App in Edge Clouds

Using Berkeley socket with TCP/IP for containers communication

Not utilizing the network resources!

SLOW!
The Problem

If each link has an average of $X$ ms delay:

Making Resized, Objects detected, faces blurred images is going to have 5 time $X$ ms as average delay
What if?

we use shared memory channels for container communications?

+ Faster
  - Not going through network stack delays
+ Accessible

- What about remote communications?
- Do we have a central control over all the containers host?
- Modifying the Applications
What’s the solution?

- What about remote connections?
  - Remote communication is efficiently supported through synchronizing shared memory regions via **RDMA**

- Do we have a central control over all the containers host?
  - Modern infrastructures we can assume these applications are running within a container orchestration framework, which provides control over:
    - communication interface
    - communication medium
So We proposed ...

- Rethink the communication model
- Create shared memory channels between containers
- supporting both a pub/sub model and bi-directional streaming model
- Local communication is made more efficient
- Remote communication is efficiently supported through synchronizing shared memory regions via RDMA
- Not only applicable to the edge clouds but also beneficial in core cloud environments
The Architecture

Shm region

Container 1

Container 2
The Architecture

- Shimmy agent
- Shm region
- Container 1
- Container 2
The Architecture

Shimmy agent

Container 1

Shm region

RDMA Server

Client

Container 2

RDMA Server

Client

RDMA Client

Shm region

Shimmy agent

Container 3
The Architecture
Prototype
Evaluation

● Setup:
  ○ two Cloudlab Servers (1x Xeon E5-2450 processor (8 cores, 2.1Ghz), 16GB Memory (4 x 2GB RDIMMs, 1.6Ghz), 1 x Mellanox MX354A Dual port FDR CX3 adapter w/1 x QSA adapter) running Ubuntu 16.04. For our system we have built docker containers for the broker, publisher, and subscriber.

● We compared against Eclipse Mosquitto and Apache Kafka
  ○ Eclipse Mosquitto is based on pub/sub model which uses TCP/IP underneath
  ○ Apache Kafka is stream-processing software platform which uses TCP/IP
Local communication – 16 B messages
Remote communication - 16 B messages

![Graph 1: Throughput (Messages/second)]

![Graph 2: Latency (seconds)]
Local communication – 100 KB Messages

![Graph showing throughput and latency for different message sizes with bar charts for Shimmy, Mosquitto, and Kafka.]
Conclusion

- A new communication model based on shared memory channels
- Optimizes local communication, but supports remote communication through RDMA
- Developed an Initial prototype which demonstrated
  - 1.78x lower latency than mosquitto for 100KB messages local
  - 2.85x lower latency than Kafka for 100KB messages local
  - 27x lower latency than mosquitto for 16B messages local
  - 82x lower latency than kafka for 16B messages local
  - 21x lower latency than mosquitto for 16B messages remote
  - 66x lower latency than kafka for 16B messages remote
Future work

- Integration with Kubernetes
  - Create a shared memory channel
  - Colocation of containers
  - Security
- Load Balancing
Discussions

What kinds of feedback we are looking to receive?

- how can we improve our communication models and infrastructure to provide a complete low latency/high throughput platform for edge/cloud computing.
- What critical functionality in missing in our current proposal?
- What are other platforms that we should compare our platform with?
- Are there other technologies that we could leverage to improve our proposal?
- Are there other communication models or paradigms (other than pub/sub and streaming) that we should provide?
Discussions (2)

The open issues the paper does not addressed

- Applications would need to be modified to take advantage of Shimmy’s architecture, but we hope that the performance benefits will make it worth.
Thank you!