Scheduling Microservices on Large-core machines using Placement and Coalescing

Vishal Rao, Vishnu Singh, Goutham Sekar,
Bharani U. K., Ruben J. M.

Subramaniam Kalambur, Dinkar Sitaram

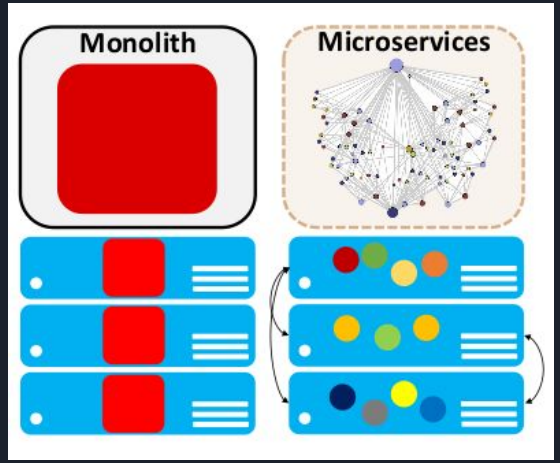
PES University
Cloud Computing and Big Data

Contributions

- Local Container Scheduler *(TRACPAD)* that uses historical runtime data to determine a container-to-core mapping that benefits performance.
- An understanding of how TRACPAD helps improve application performance
- An evaluation of TRACPAD and the factors that may affect the policies generated
- A technique to strategically coalesce containers to minimize the communication overhead induced by the decoupled nature of microservices

Introduction

- Cloud based applications are moving from a monolithic architecture to a microservices based architecture.
- Communication overhead incurred due to the decoupled nature of the microservices that impacts performance.
- Prior research focuses on studying scheduling and placement of microservices in a distributed computing environment.
- Our study focuses on scheduling microservices on high-core count servers.



Monolithic Arch vs Microservices Arch [1]

1. Y. Gan *et al* "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems"

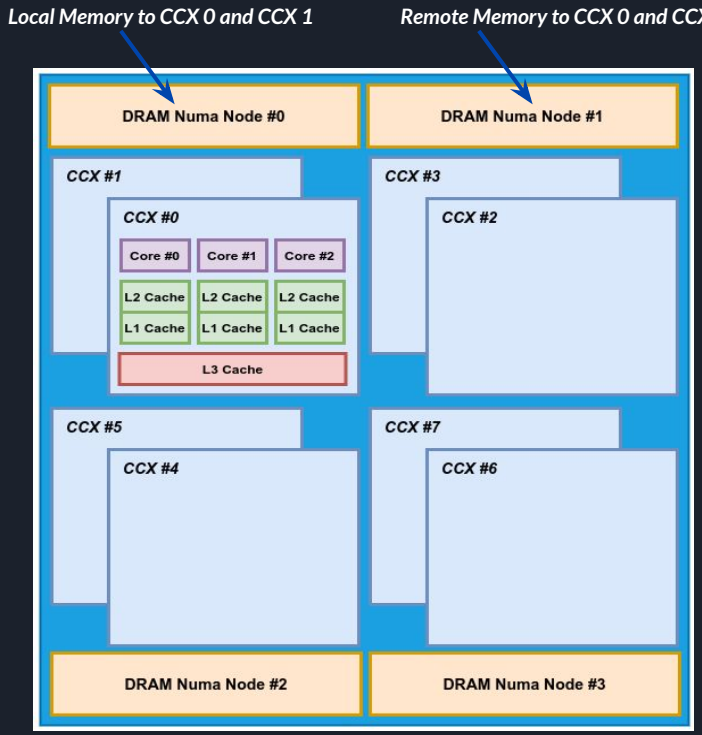
Introduction

Increase in per-socket core-count over the past few years.

(Present EPYC servers have as many as 64 cores/socket)

These server CPUs are based on chiplet designs and have multiple NUMA domains per socket.

Important to understand how to schedule microservices on these machines



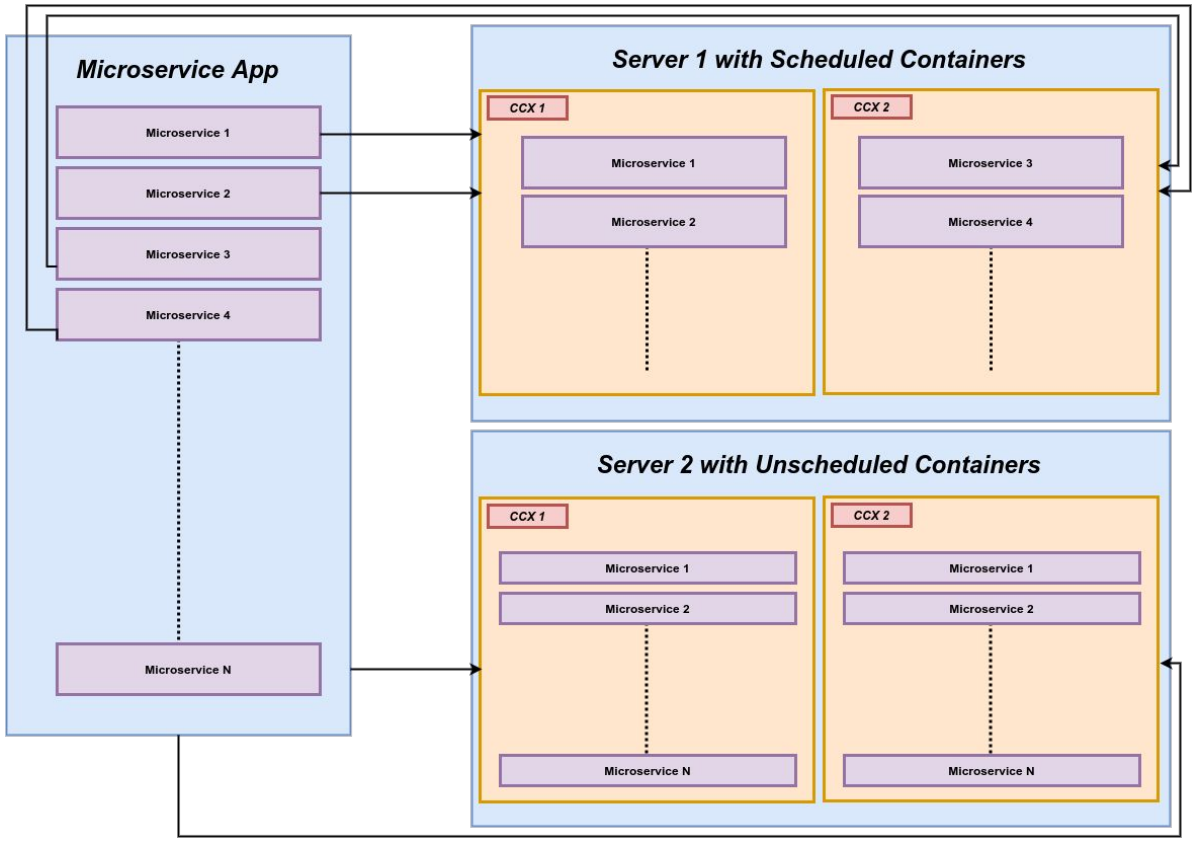
AMD EPYC 7401 - Single Socket Topology

How does our approach work?

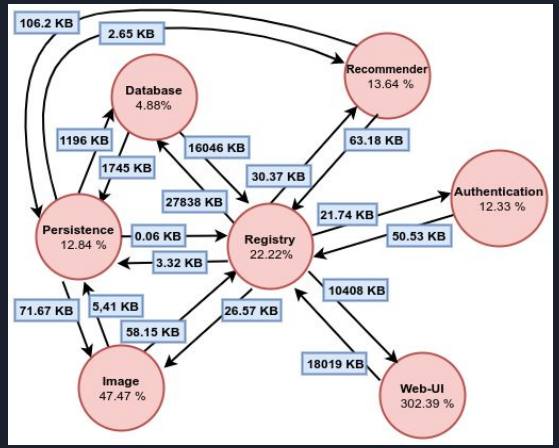
1. Analyze how the workload utilizes the system resources.

2. Schedule the Microservices by imposing resource constraints.

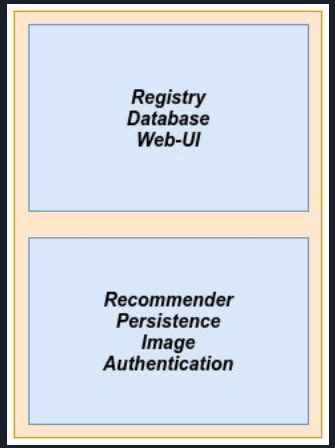
3. Allocate resources such that the application efficiently utilizes the resources of server.



How is the Placement Scheme generated?



2 Partitions



We model the application workload as a weighted graph -

- Node weights represent individual container resource utilization metrics
- Edge weights represent communication payload size

TRACPAD partitions this *Container Communication Model* to generate the placement scheme

Topology and Resource Aware Container Placement and Deployment - TRACPAD

- Algorithm used to partition the Container Communication Model
- Each Partition represents the set of containers collocated on the same CCX.
- Partitioning method is as follows -
  a. Minimize the EdgeCut (Sum of the Cut Edges in the graph partition). This is equivalent to minimizing inter-CCX communication.
  b. Balance the node weights in every partition. This is equivalent to minimizing resource contentions.

Reduce communication overheads and Increase the efficiency with which resources are utilized.

Coalescing Containers

- Eliminates all communication overhead between the coalesced containers.
- Greedy method used to select which containers to coalesce.

- Containers need to be checked for compatibility -
    - Containers used a storage-backends not used for coalescing.
    - Containers that have dependency conflicts are not compatible.

    (If two containers require different versions of GCC, they cannot be coalesced)

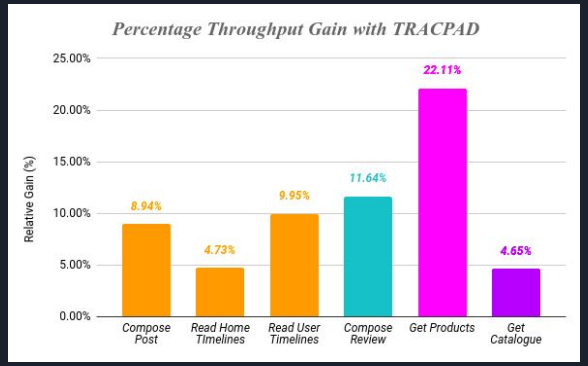
Summary of Applications and Workloads

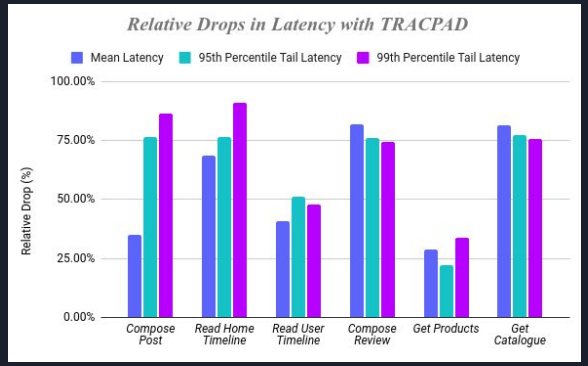
Application	Workloads	No. of Containers	Languages/Frameworks	Storage Backends
Social Network[1]	Compose Post Read Home Timelines Read User Timelines	30	C/C++, Java, GoLang, JavaScript, Scala, RabbitMQ, Apache Thrift	MongoDB, Redis, Memcached
Media Microservices[1]	Compose Review	33	C/C++, Java, GoLang, JavaScript, Scala, RabbitMQ, Apache Thrift	MongoDB, Redis, Memcached
TeaStore[2]	Get Products Add to Cart	7	Java, Apache Tomcat	MariaDB
Sock Shop[3]	Get Catalogue	15	GoLang, NodeJS, Java Springboot	MongoDB, MySQL

1. Y. Gan *et al* "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems"
2. J. von Kistowski *et al* "Teastore: A microservice reference application for benchmarking, modeling and resource management research"
3. SockShop Microservice Application - https://microservices-demo.github.io

Evaluation of TRACPAD
Performance Comparison with Baseline

Application performance is evaluated using Throughput, Mean Latency and Tail Latency.





Throughput uplifts range from ~4% to ~22%

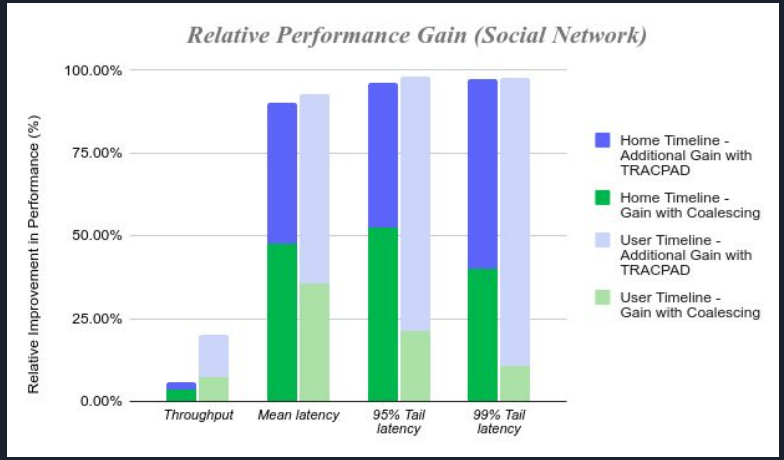
Mean latency improvements range from ~28% to ~81%

95th Percentile Tail Latency improvements range from ~22% to ~76%

99th Percentile Tail Latency improvements range from ~30% to ~90%

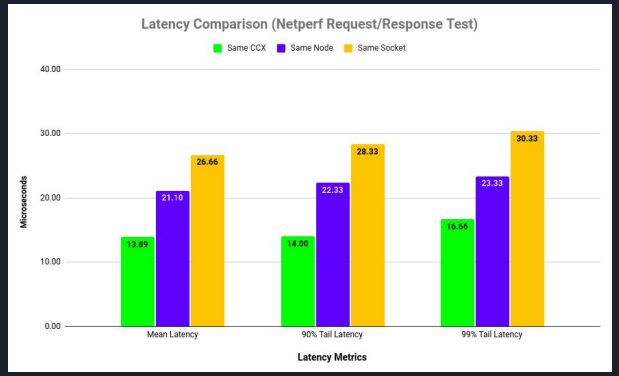
Evaluation of TRACPAD
Additional Perf Improvement with Coalescing

- TRACPAD helps reduce communication overhead.
- Coalescing of containers further minimizes the communication overhead.
- Amount of data transmitted decreases by ~10%



Performance uplift with TRACPAD and Coalescing

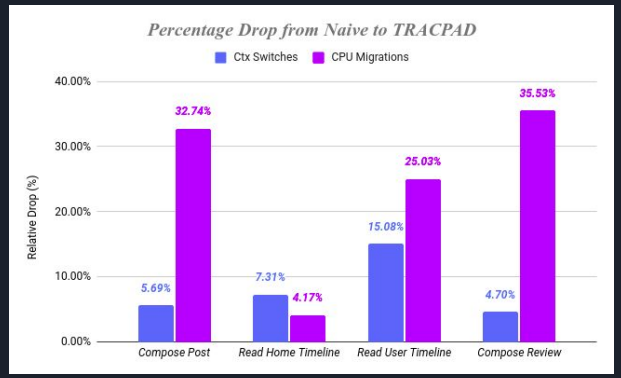
Why does TRACPAD work?



Co-locating containers on the same CCX helps reduce the communication latencies between them.

Containerized NetPerf server and client used to in the experiment to measure latencies in 3 cases -

1. Server and client on same CCX
2. Server and client on different CCXs but same NUMA node
3. Server and client on different NUMA nodes but same socket

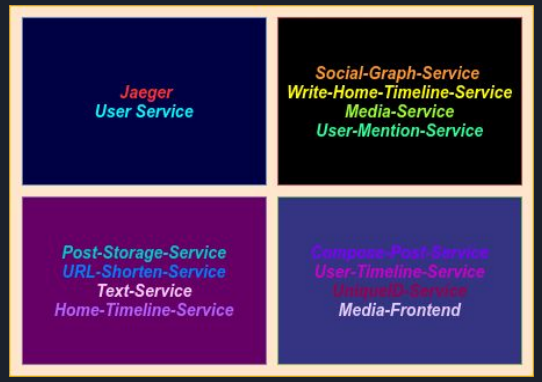


Affinitizing containers to cores reduces Context Switches and CPU Migrations.

CPU Migrations can be very expensive on systems with multiple NUMA nodes.

Factors affecting Scheduling Policies

Request Packet Size - 2kB, Small Database



Request Packet Size - 256B, Small Database

Request Packet Size - 256B, Large Database

Database sizes and Request Packet sizes can impact the scheduling policies

Further Work

- Factor in the impact of NUMA into the Container Communication Model.
- Investigate security implications of Coalescing
- Automate Container Coalescing and integrate it with the TRACPAD scheduler

Thank You