Distributed Resource Provisioning for Containers Using Machine Learning and Live Migration

Alejandro Gonzalez Araya

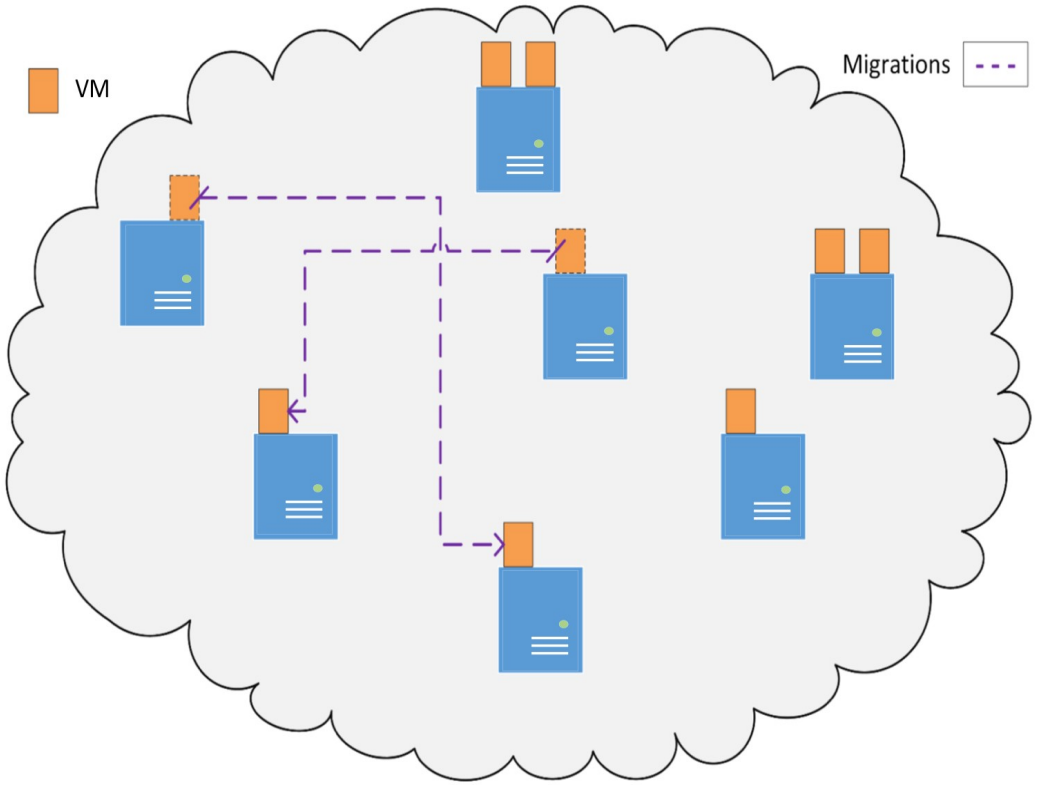
MsCpE Thesis Defense

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS
2020

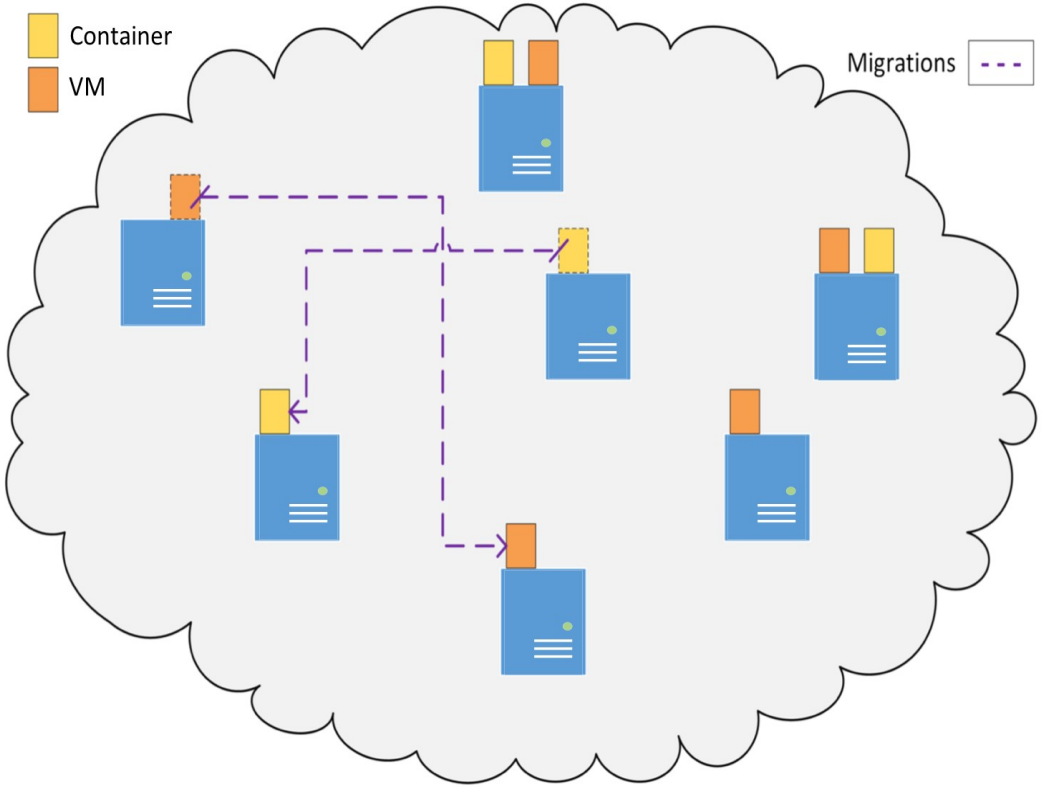
December 1, 2020

Introduction

- Cloud computing uses pools of virtual machines to provide shared computing resources.
- Provisioning and management of these resources are usually done using statistical algorithms to help decide how to better utilize available compute power;
- This is performed mostly by using live migration of virtual machines.



Typical Cloud Infrastructure Scenario. Virtual Machines are live migrated to improve overall resource efficiency



Modern Cloud Infrastructure scenario. Containers and Virtual Machines share physical infrastructure allowing for a more robust environment.

Introduction

- Efficient resource management is crucial for mission critical applications
- Adoption of virtualization in data centers has introduced abstraction to mission critical systems
- The resource usage policy for the virtualization platform being used is crucial in a modern datacenter environment
- In the container case, resource usage decisions are typically performed at container start
  - Limiting its execution to a single assigned node and the amount of resource efficiency that can be achieved at a particular time.

Motivation

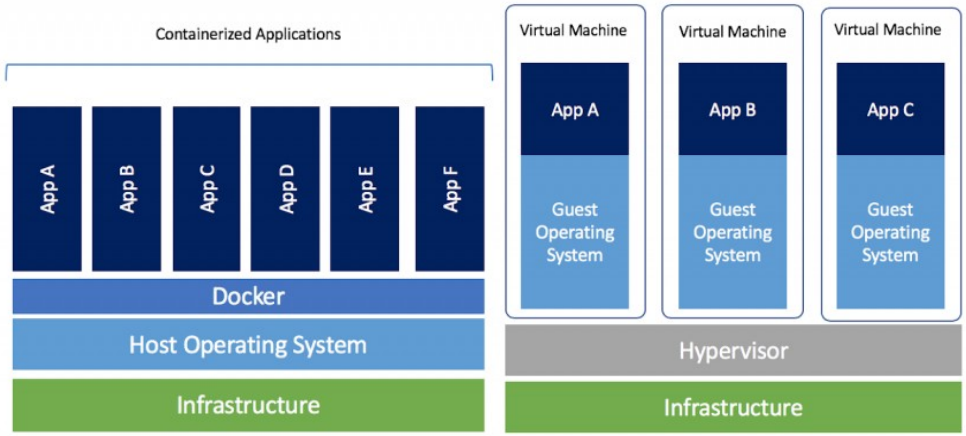
- Today's Cloud services are heterogeneous (PaaS, IaaS, Serveless)
- Thanks to the diversity of computational cloud resourceful the use of containers in Cloud environments have gained traction.
  - Containers provide a computationally cheaper solution for most task
- Most cloud systems are designed for virtual machines (VM)
- Mechanisms to handle containers and Vms on a heterogeneous cloud environment needs to be addressed.
  - No container live migration support for resource provisioning
- Using VM live migration techniques for containers is not something used for resource provisioning

Motivation

- Compute power is not always used to its potential.
- We want to use modern technologies to improve system resource utilization to improve the performance of running workloads.
- Current software stacks for container management do not use live migration for performance tuning.
- Our work consisted on developing a platform that improves system and workload performance using container live migration.
- Our works covers the research of container migration techniques, real-time system monitoring and resource provisioning policies.

Containers

- Operating system feature in which the kernel allows the existence of multiple isolated user-space instances
- Containers are a lighter-weight, more agile way of handling virtualization compared to virtual machines.
- Containers packages together everything needed to run a small piece of software as a system process by sharing the host's kernel.
- Contains provide a level of flexibility to handle many software environments and tasks



Live migration and Application Checkpointing

- Application checkpointing consists of saving a snapshot of the application's state, so that it can restart from that point in case of failure.
- CRIU is a software tool for the Linux operating system that allows to freeze a running application (or part of it) and checkpoint it as a collection of files on disk.
- These files can then be used to restore the application and run it exactly as it was during the time of the freeze.
- The use of a software tool like CRIU for application checkpointing together with a container manager as Docker, allows a developer to migrate a running container to another host

Resource Provisioning

- Resource provisioning describes the process of assigning appropriate resources to computing workloads.
- These tasks require the pairing of the best compute resource to a workload based on the application requirements.
- This is a complex job that needs to be optimized for the best use of the available resources.
- We monitor in real-time the system resource utilization metrics to determine system bottlenecks.
- We use container live-migration to improve system performance by evaluating the available resources on the infrastructure to better accommodate a container.

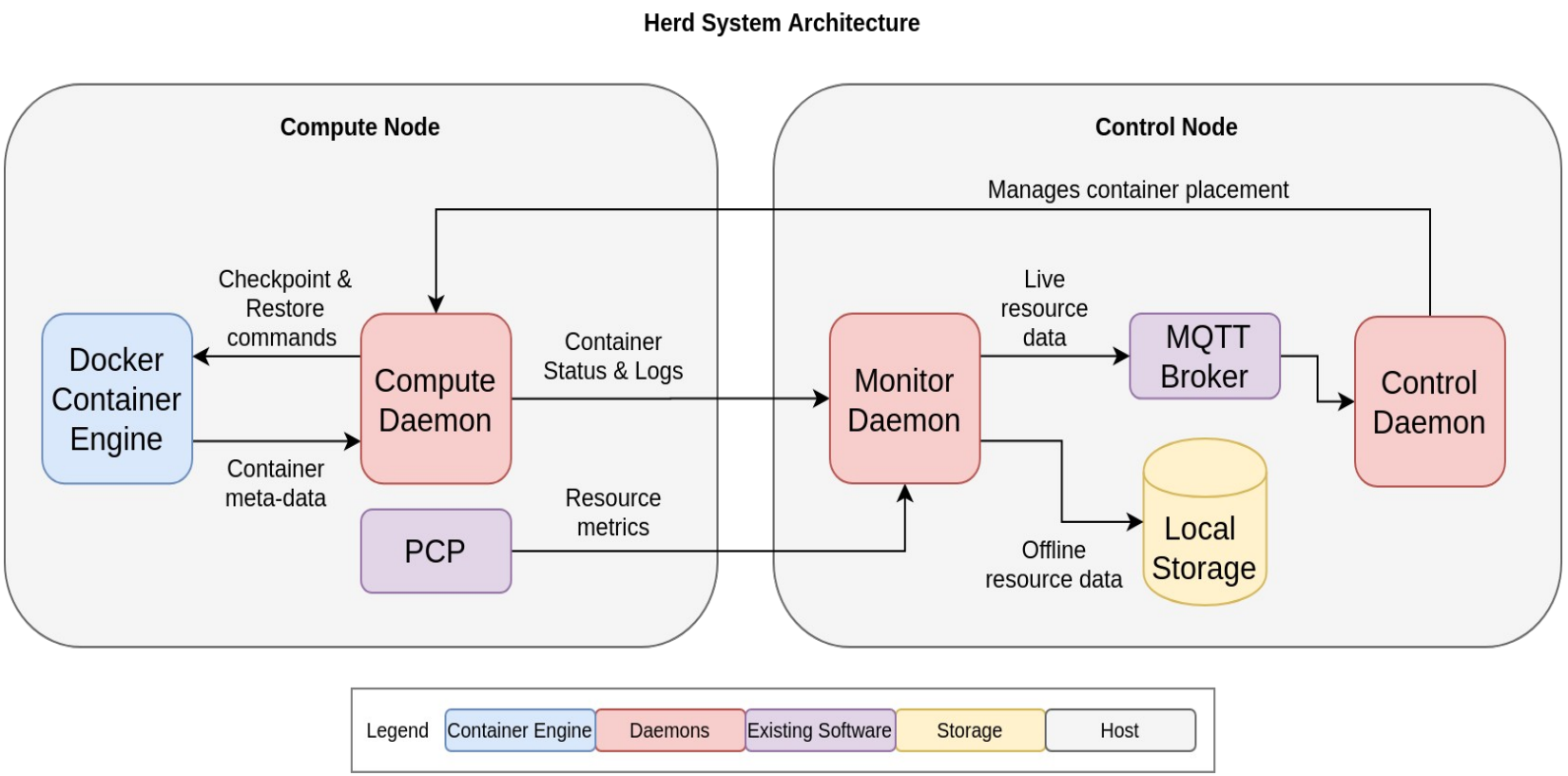
Machine Learning

- Machine learning algorithms build a mathematical model of sample data, in order to make predictions or decisions without being explicitly programmed to perform the task.
- We used machine learning to analyze system resource utilization, it allows us to select the most important features of a given workload to focus our resource provisioning policies on those system metrics.
- With machine learning we can forecast the performance of a container’s workload on another host to improve our container live migration policy.

Herd: Container Resource Provisioning System

- Our live migration toolset defines *compute nodes* and a *Control node* that act together to manage, run and migrate containers.
- The *Compute nodes* hosts *containers* and are responsible for orchestrating the container live migration tasks, well as receiving communication from the *Control node*.
  - The *Compute node* will send status and monitoring information to the *Control node*.
- The C*ontrol node* is responsible for monitoring the *containers* and *compute node's* performance.
  - Processes realtime monitoring information to determine the ideal container placement configuration, based on resource metrics and migration policy.
- The *control node* sends container migration commands to the *compute nodes*.

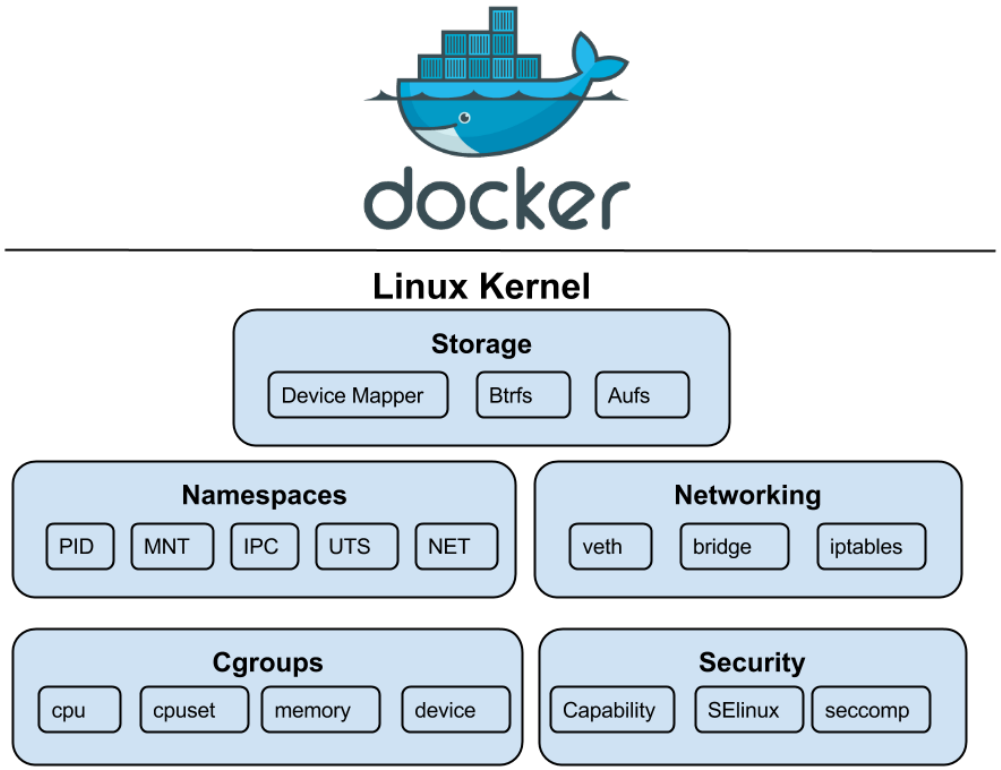
Herd System Architecture



Container Live Migration

Container Internals

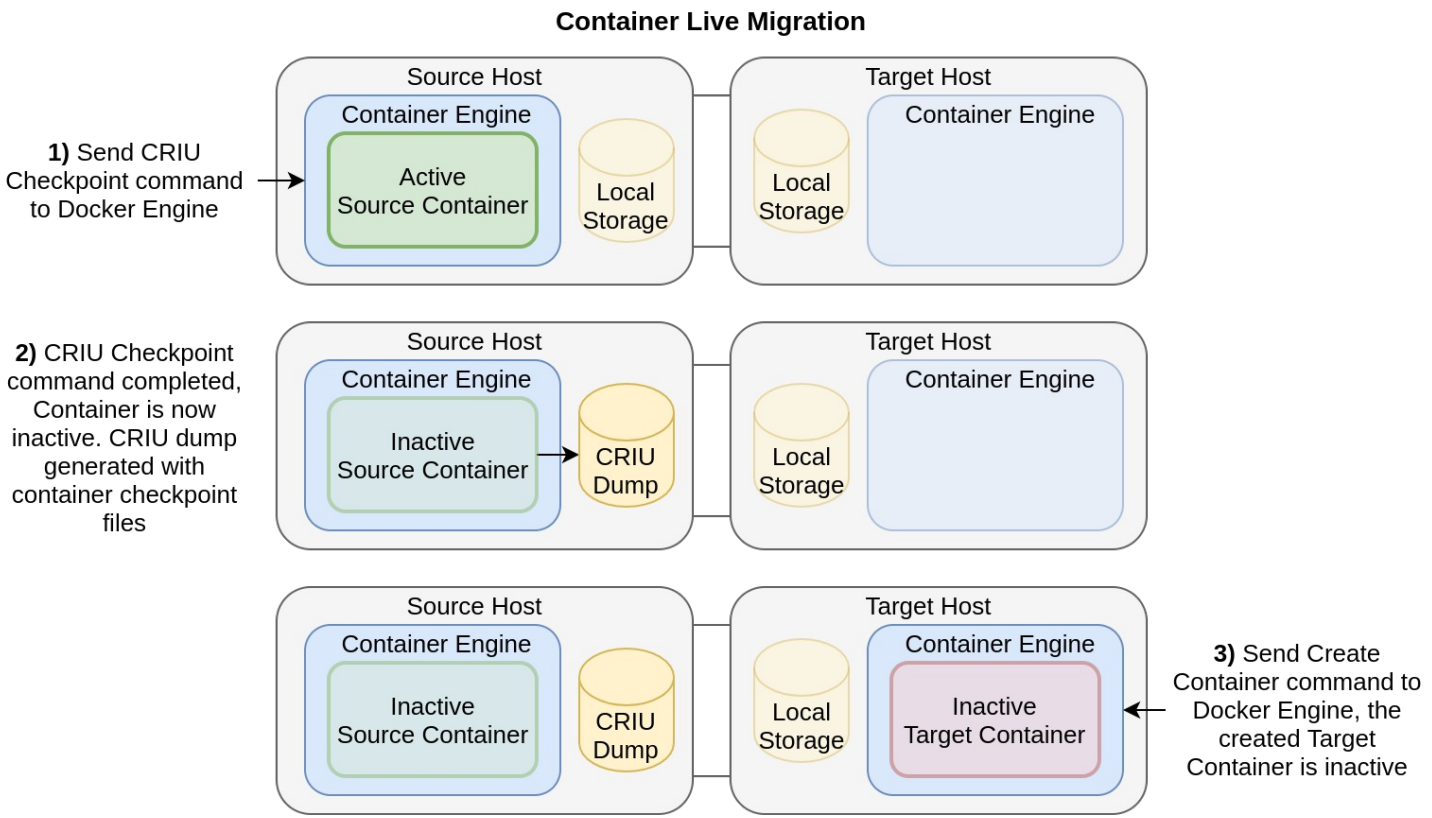
- Linux kernel feature to provide process isolation
- Namespaces for process isolation
- Control groups (CGROUP) for resource isolation
- Base image holds container’s filesystem



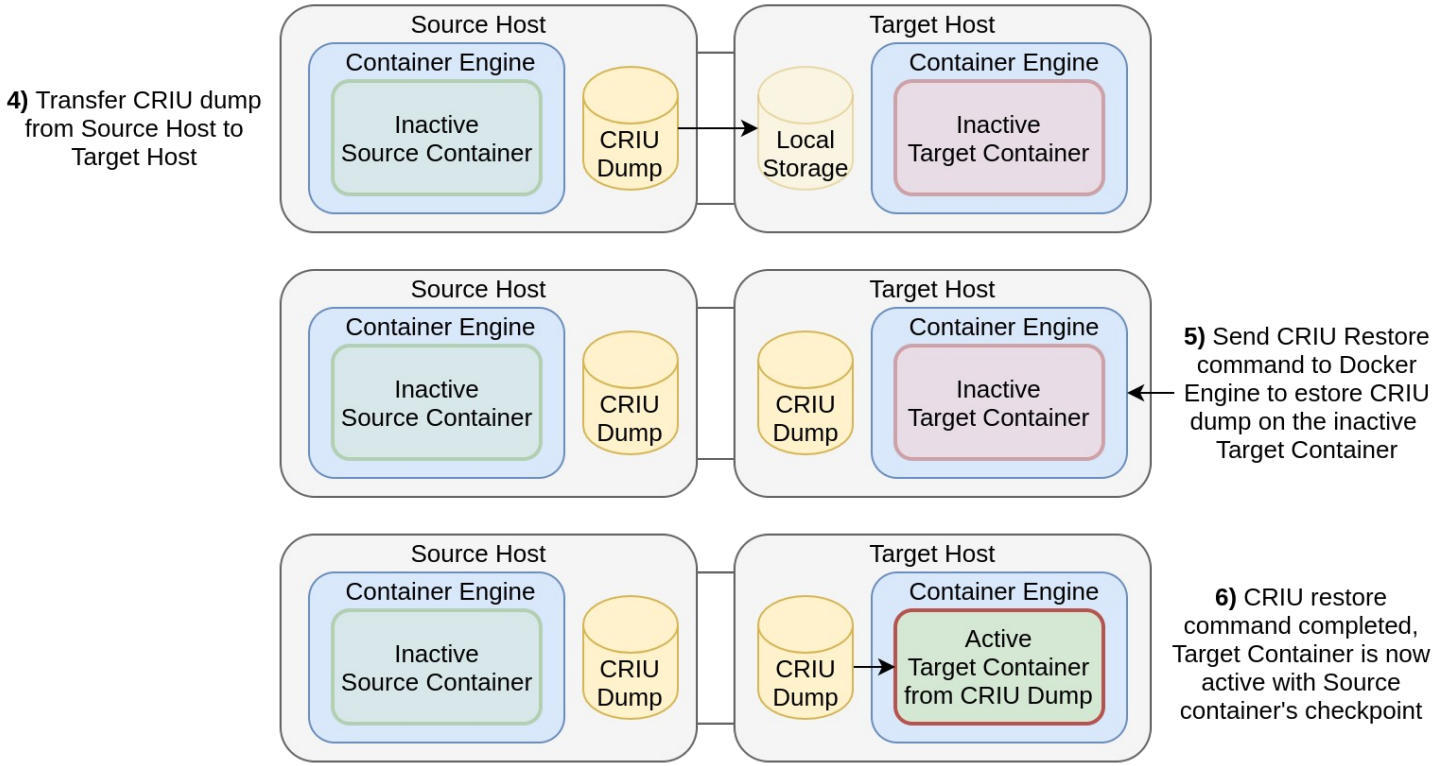
Application Checkpointing

- *Checkpoint and Restore in Userspace* (CRIU) is a software tool that handles the required task to checkpoint and restore an application.
- A checkpoint dump contains a process file descriptor information, memory maps and other stateful objects such as TCP sockets.
- File locks, devices, and task from other users are among the things that cannot be checkpointed by CRIU.
- By transferring a checkpoint to another machine, migration can be achieved.
- Docker provides, as an experimental feature, the ability to checkpoint and restore a container.
- Docker does not feature container live migration.

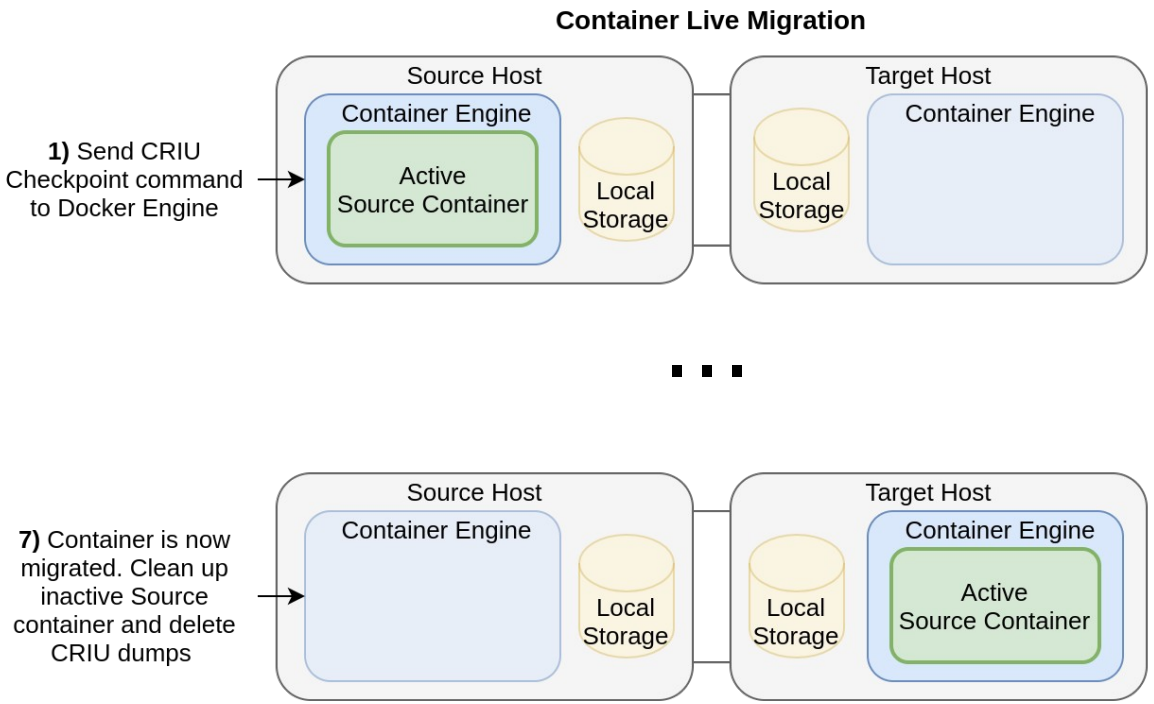
Container Live Migration



Container Live Migration



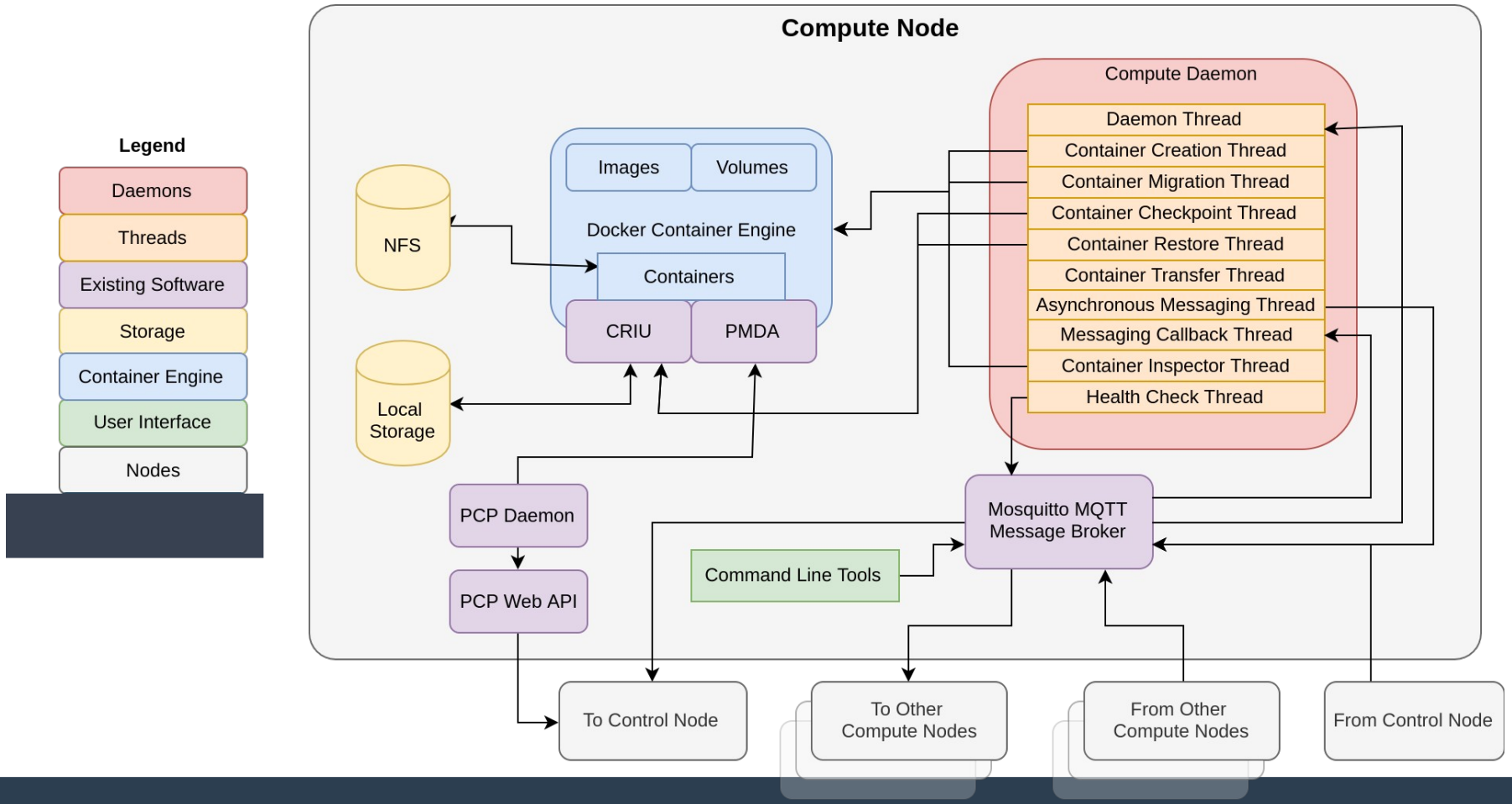
Container Live Migration



Container Live Migration sequence

- Checkpoint and restore (C/R) tasks for container migration
  - Launch Container A in Node 1
  - Checkpoint Container A
  - Create Container B in Node 2
  - Transfer Checkpoint A to Node 2
  - Restore Checkpoint A on Container B
  - Remove Container A in Node 1

Herd Compute

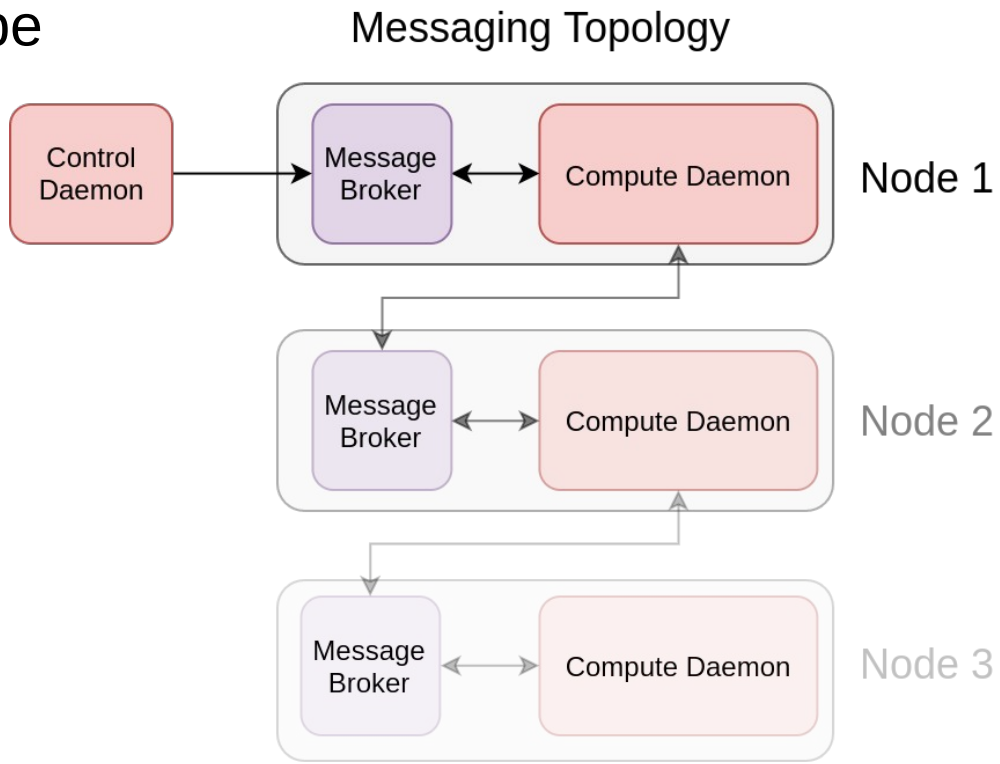


Compute Nodes

- Compute Daemon – Multi-threaded
  - Container Creation
  - Container Migration
  - Container Checkpoint
  - Container Restore
  - Container Transfer
  - Asynchronous Messaging
  - Messaging Callbacks
  - Remote Container Info
  - Extended Metric Streaming
- Message Broker
- Docker
  - CRIU
- PCP - Monitoring
  - PCP Daemon
  - PCP Web API
- Local Storage

System Messaging

- MQTT Bidirectional publish-subscribe
  - Also used for Inter process communication
  - Provides Quality of service (QoS) configuration
  - Acknowledgments and Wills
  - Message queuing
- Mosquitto broker
  - Automatic re-connection

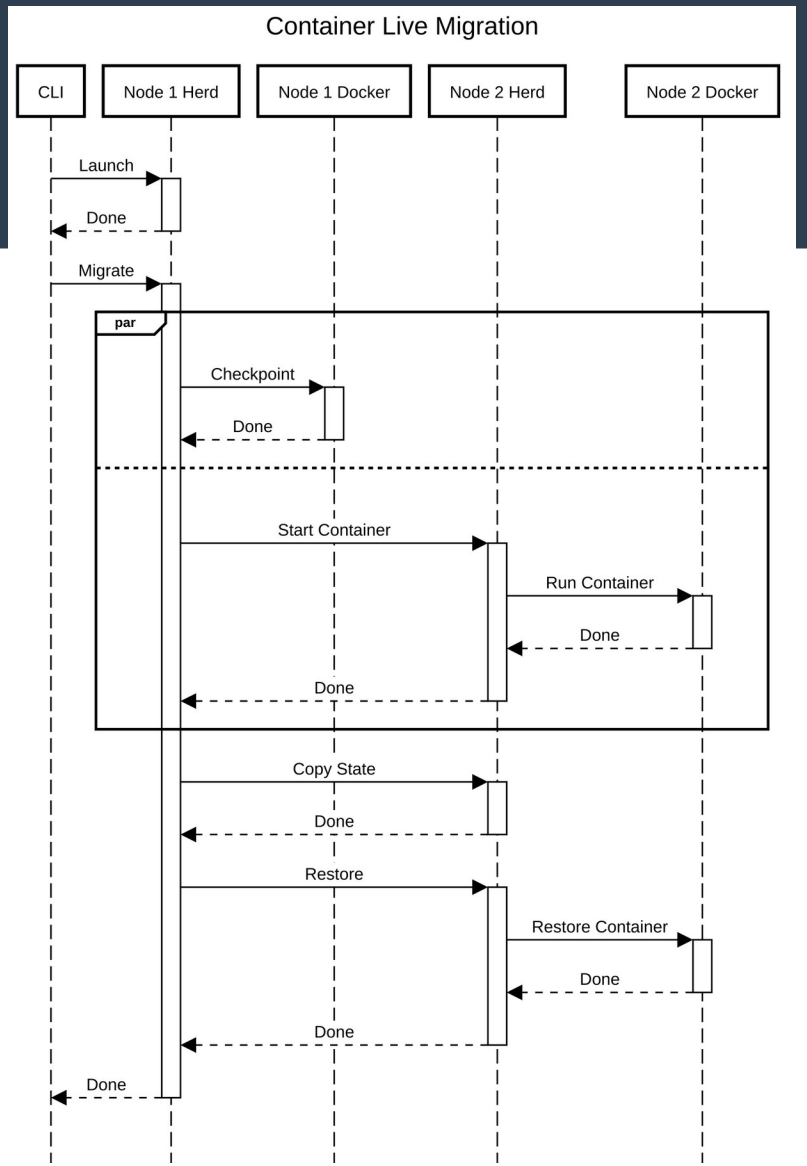


Live migration sequence

- Optimizing migration sequence
- Parallelize tasks by deploying target container while source container checkpoints
- CRIU operations are serial, our asynchronous sequence could interfere C/R operations
- We manage CRIU resources by synchronizing the the C/R tasks for parallel container migrations.

	Non Synchronized CRIU	Synchronized CRIU
Migration Time	1.82s	1.76s

Migration stress testing results after concurrency management

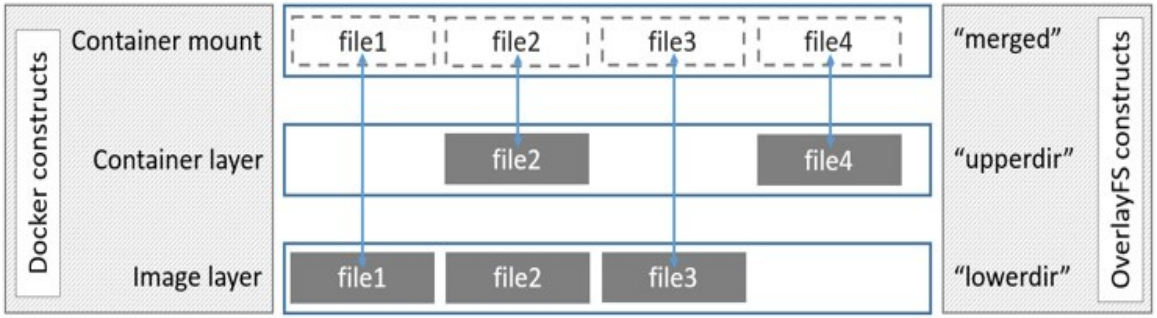


Container Networking

- MACVLAN network adapter
  - Driver available for Docker
  - Containers have own mac address, direct connection to network
- Implemented automatic network-bound container live migration sequence to Compute Daemon
  - Network communication reestablishment after live migration
  - Network communication between containers using host network
  - Ability to handle and live migrate multiple network-bound containers
    - Compared to other implementations that only handle 1 container per host

FileSystem Migration

- Live migration does not transfer container filesystem
  - No FS migration is quick but prone to failure as only RAM is transferred
- The Compute Daemon is capable of migration entire container FS
  - This operation is computationally heavy and takes time
- Containers work using an overlay filesystem (FS)
  - Only the changes to the base image are stored
  - All compute hosts have the base images



FileSystem Optimizations

- All compute hosts have the base OverlayFS image
- By only moving the difference layer quick FS live migration is possible
- Our implementation allows Herd to support more types of workload
- Improves C/R compatibility, achieves faster FS live migration
- No other container orchestration software uses this optimization

Migration mode	OLTP Client container	OLTP Server container
No FS transfer	3.06s	2.49s
Whole container FS	22.68s	26.89s
Diff layer	2.58s	2.69s

Filesystem migration duration comparison

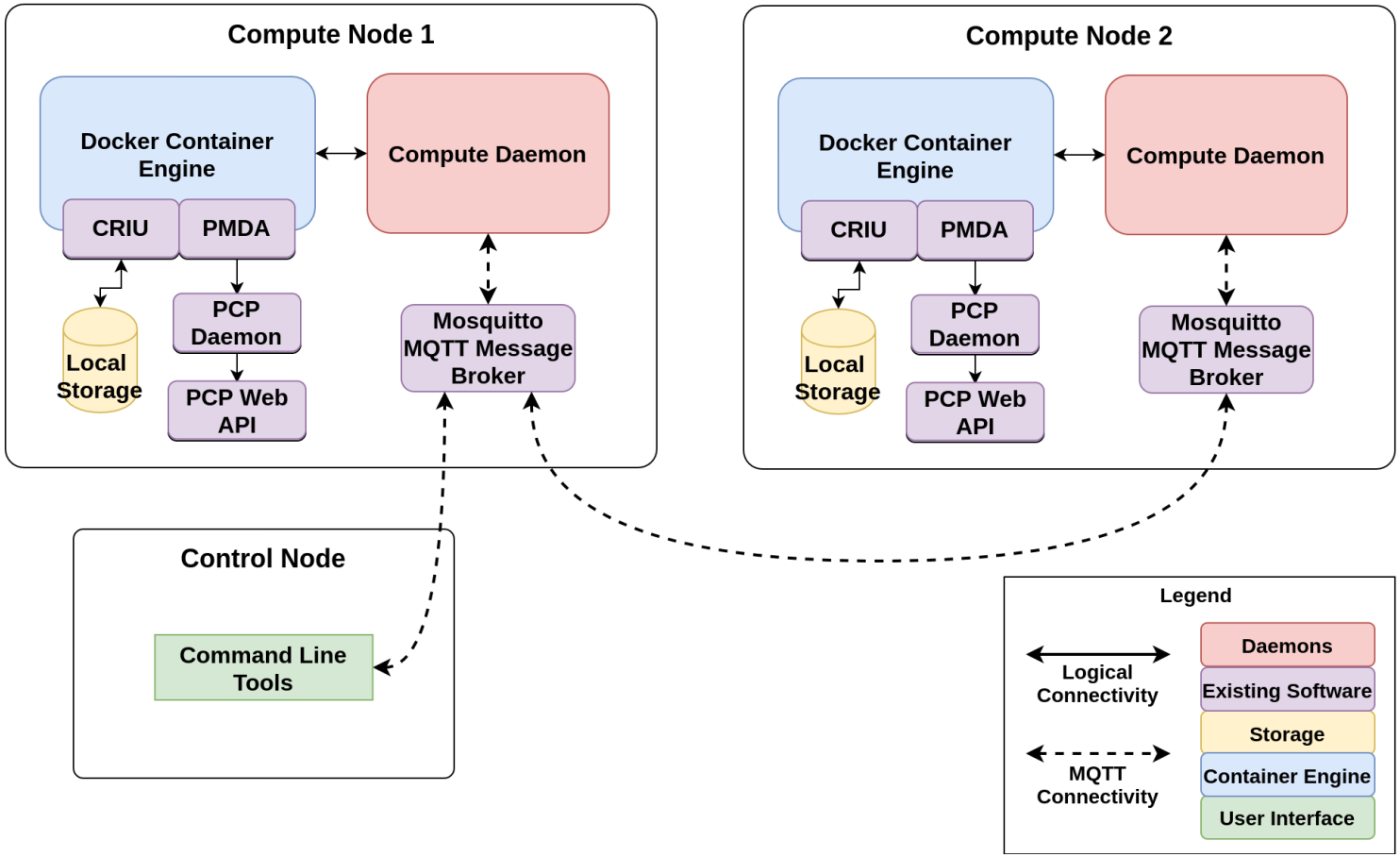
Migration Rollback

- Herd Compute can rollback a failed migration by using the same checkpoint/restore features used for migration

- Herd control handles errors in three main ways:
  - if checkpoint fails: rollback target container and keep source container running
  - if remote restore fails: rollback target container restore a source container from checkpoint
    - if local restore fails: rollback target container and restart source container

- In this case the container is restarted. A restarted container looses its context and can be consider a failed migration.

Compute Integration



System Monitoring

System Monitoring

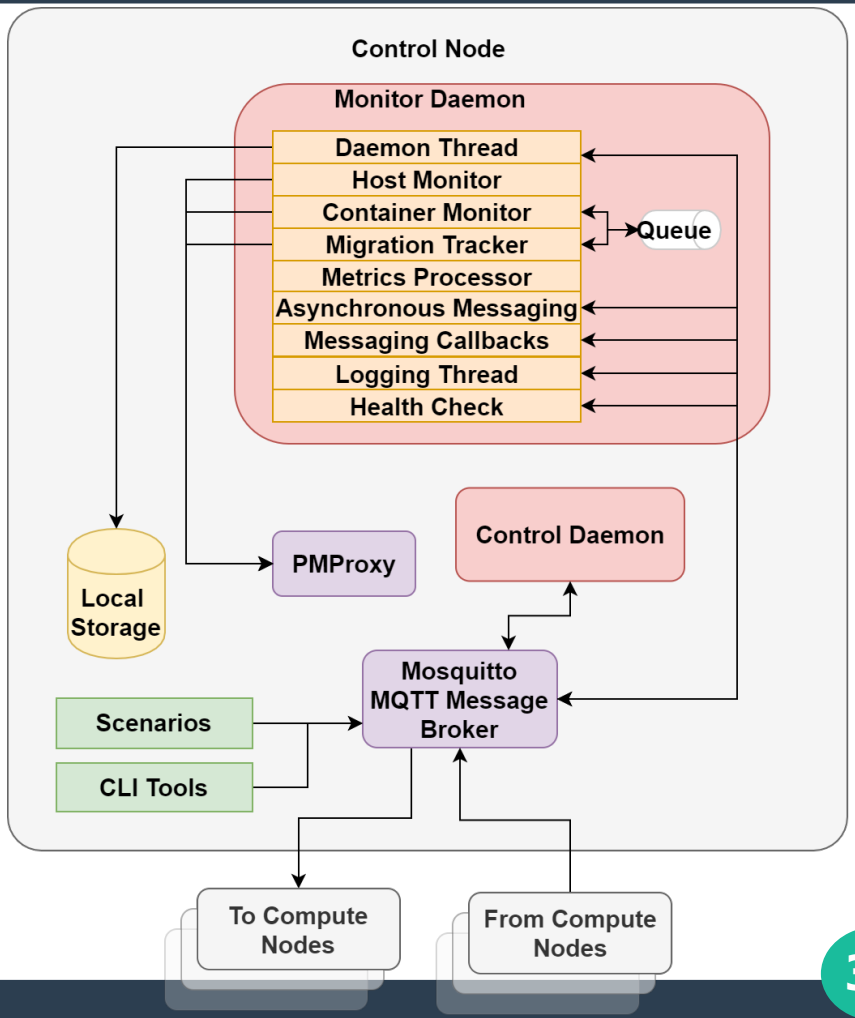
- Developed a customizable and lightweight resource monitoring software
- Provides resource utilization metrics of running containers.
- Provides resource utilization metrics of compute nodes.
- Track container migrations.
- Handle the task to gather and centralize live migrating container resource utilization metrics.
-

Performance Co-Pilot (PCP)

- Lightweight, extensible monitoring software
- Gathers most of the resource data from Linux's *proc* filesystem
- Provides real time resource utilization data
- Distributed REST API for remote monitoring
- Can monitor independent CGROUPs

Herd Monitor

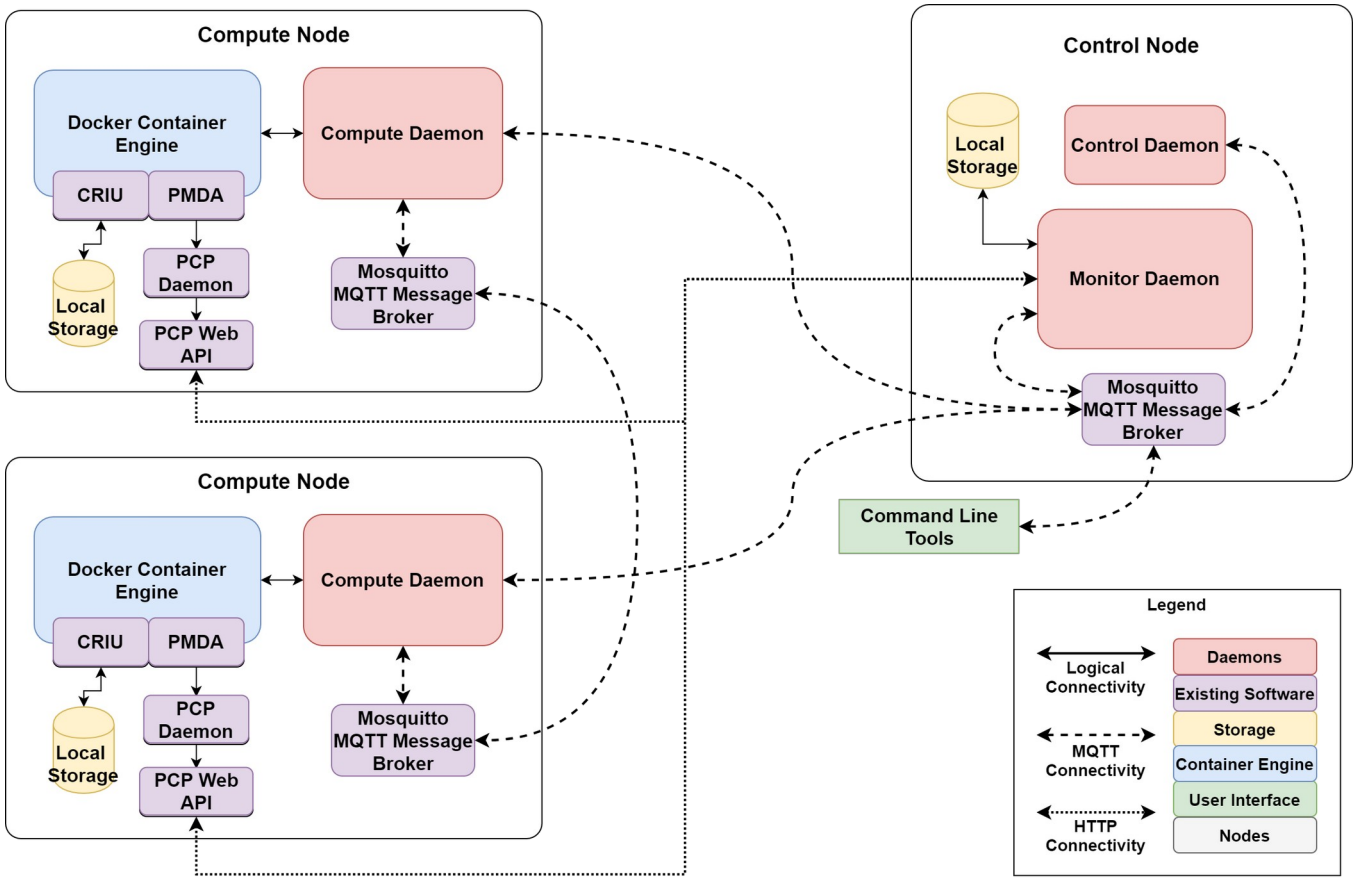
- Monitoring Daemon – Multi-threaded
  - Host Monitoring
  - Container Monitoring
  - Container Migration Tracker
  - Metrics processor
  - Asynchronous Messaging
  - Messaging Callbacks
  - Container Health check
  - Host-independent Container log



Monitor Daemon

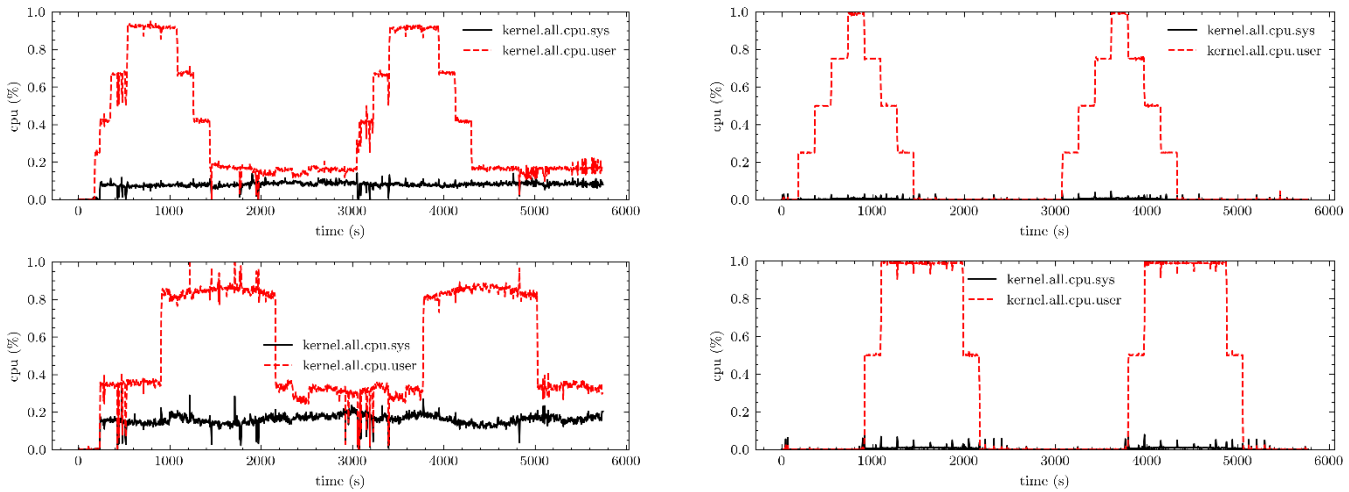
- Real-time data streaming to Control Daemon
  - Extended metric processing from Compute Daemon
- PCP error handling
- Host and Container monitoring
- Container live migration tracking
- Container standard output logging
- Lambda architecture for online and offline data processing
- Fully Concurrent and customizable

Compute and Monitor Integration



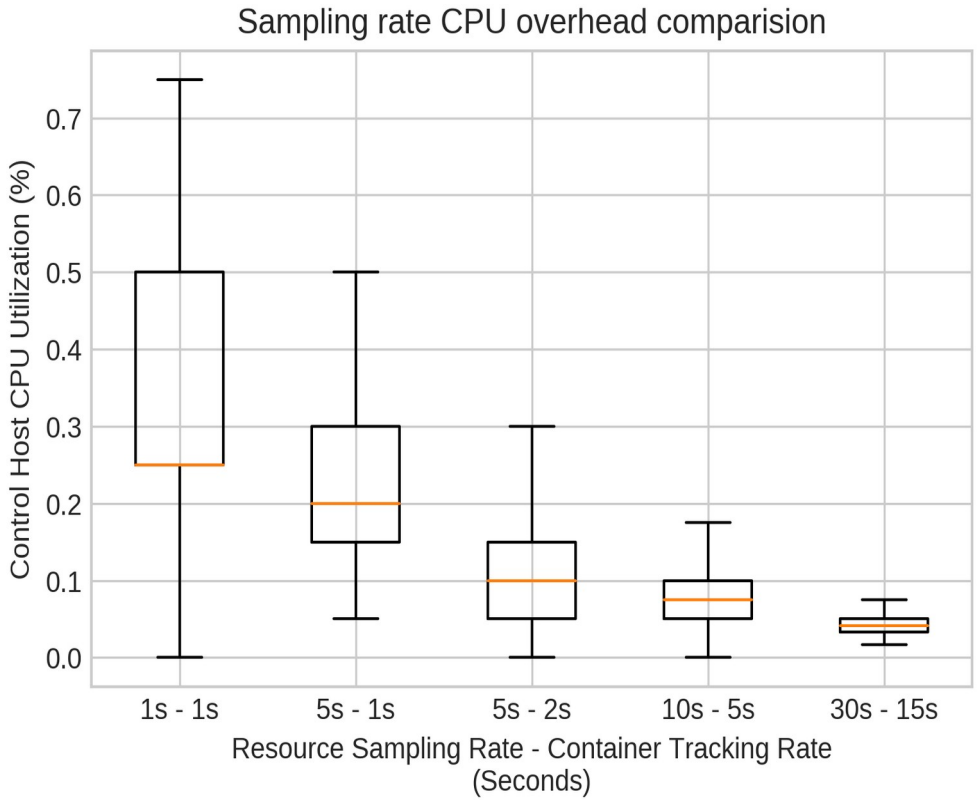
Herd Monitor Overhead Analysis

- Evaluated Herd Monitor overhead over Control Node
- Improved performance by parallelizing threads
- Resource reuse and concurrency management
- Different sampling rates for each type of monitoring



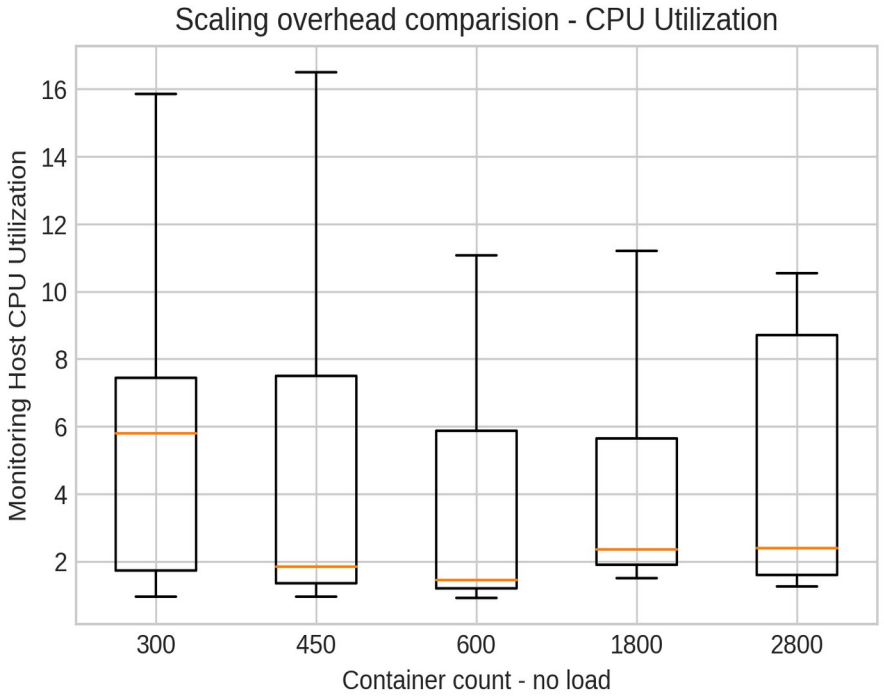
Herd Monitor Overhead Analysis

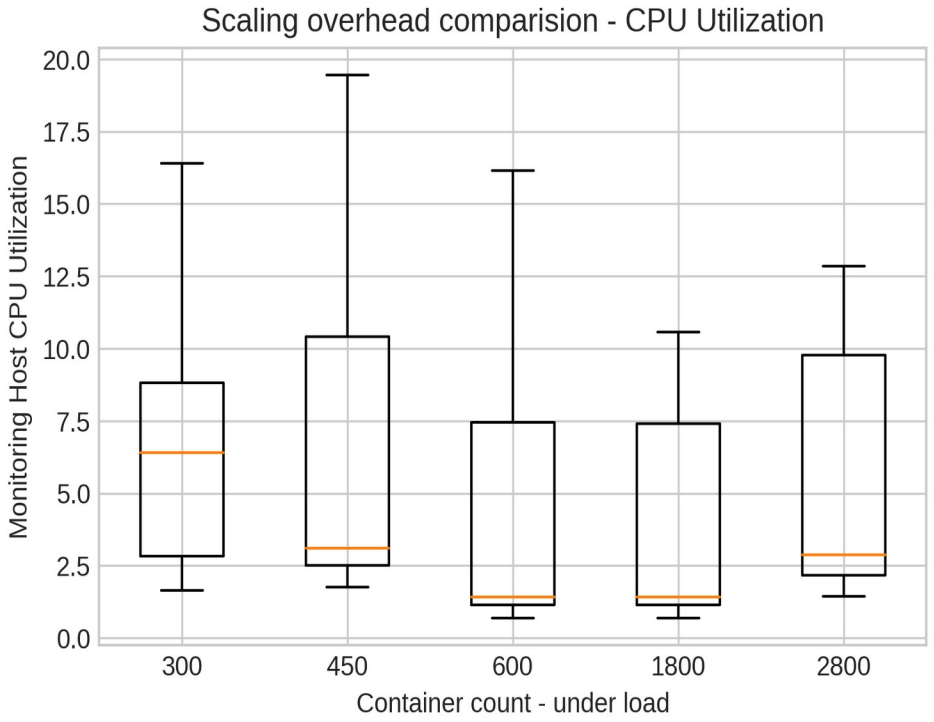
- Control Node CPU utilization under different Herd Monitor Sampling Configurations



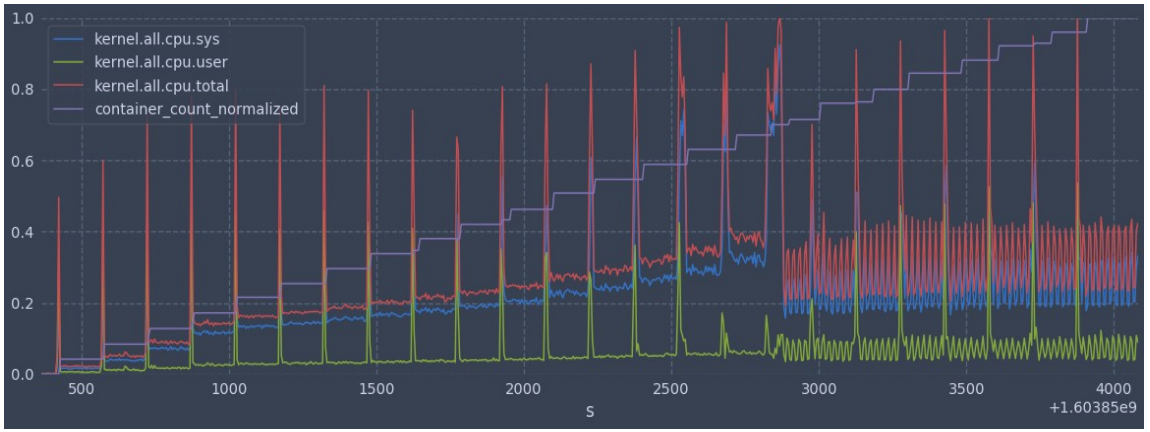
Herd Monitor Overhead Analysis

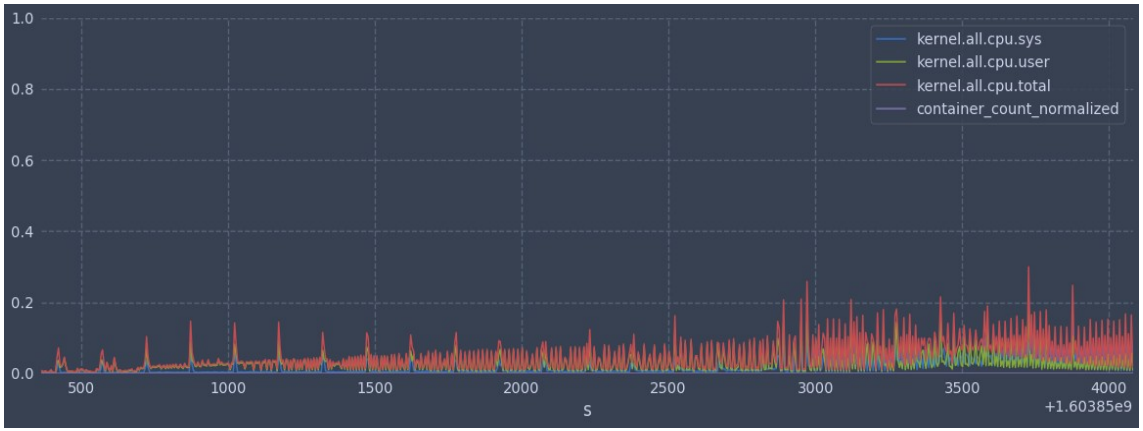
- Control Node CPU utilization while monitoring many containers
  - 5 second sampling rate, 2 second container tracking rate





Herd Monitor Overhead Analysis





Compute Node (top), Control Node (bottom) CPU utilization under 0-3600 monitored containers

Resource Provisioning

Workload characterization

- Workload characterization by small controlled microbenchmarking
- Policy testing by large controlled workloads
  - Randomly generated workload scenarios
- On-the-fly system reconfiguration
  - Deploy multiple VM compute hosts with different hardware configurations
- Gather data and analyze each run
- Overlap “staircase” scenarios

Performance Validation

- Sysbench synthetic CPU benchmark
  - Prime number verification
  - Total operations per runtime
  - Reports by internal the instant operations during runtime
  - Limit execution time to benchmark the total operations and instant operations

```
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 2

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 20000


Test execution summary:
    total time:                          18.0683s
    total number of events:              10000
    total time taken by event execution: 36.1322
    per-request statistics:
         min:                                  3.44ms
         avg:                                  3.61ms
         max:                                  6.77ms
         approx.  95 percentile:               5.05ms

Threads fairness:
    events (avg/stddev):           5000.0000/7.00
    execution time (avg/stddev):   18.0661/0.00
```

Performance Validation Experiment

- Synthetic benchmarks
  - Dependent variable for our experiment
  - Used to correlate (if any) the monitored resource utilization metrics vs the operations per seconds from the benchmark
  - Used to have a final total operations score to validate overall performance gains from migration
  - Instant report of op/s used to validate performance changes

CPU Workload Characterization

- 2-core and 4-core hosts
- All containers CPU utlil vs time

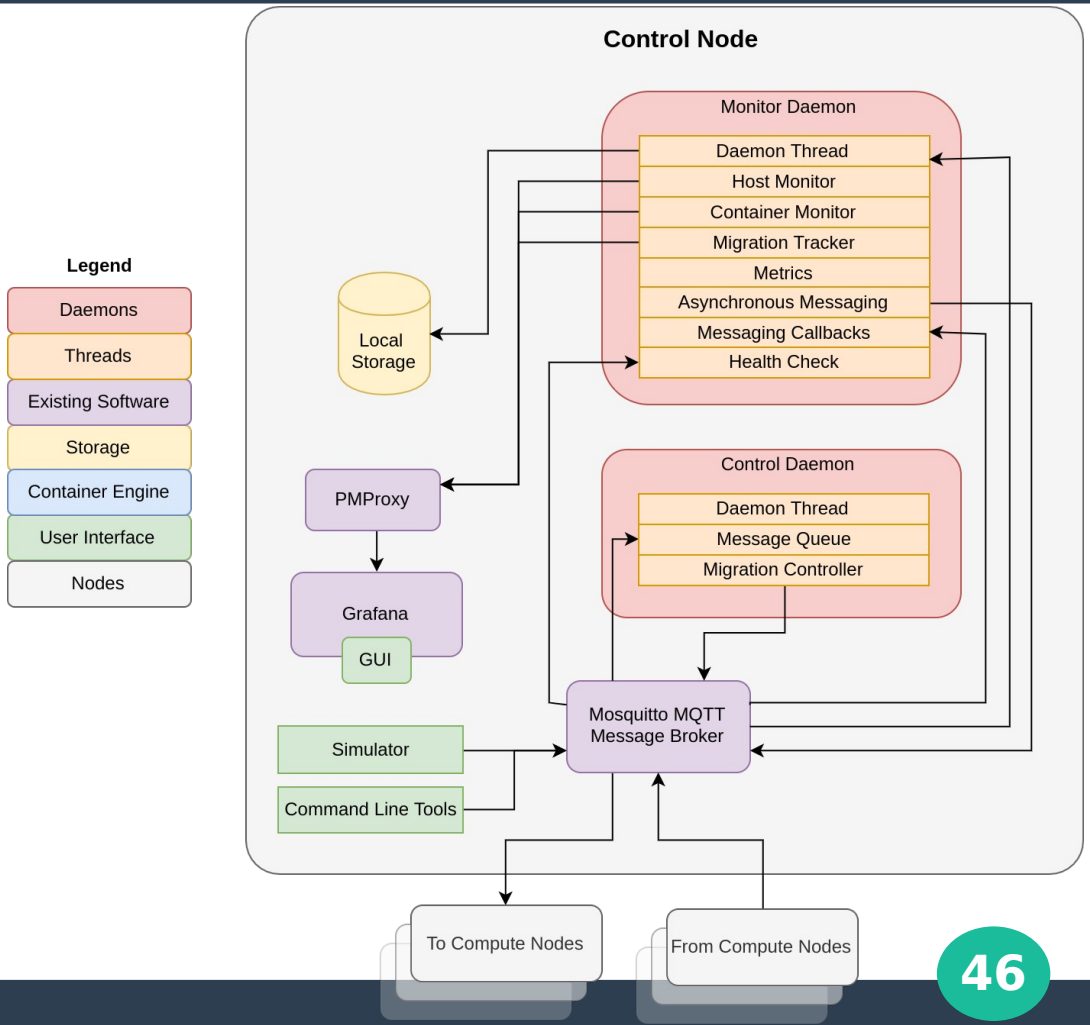
Performance Validation

Single Container CPU Util (top) Perforamnce Metrics OP/s (bottom) for a vs Time

- 2-core hosts
- 1-thread containers
- Single Container metric for all hosts vs time
- Colors: hosts

Herd Control

- Control Daemon – Multi-threaded
  - Migration Controller
  - Message Queue
  - DataFrame processor
- Real-time data stream processing from Monitor Daemon
- Sends migrations commands to Compute Daemons
- Live migration error handling



CPU workload container placement policy

Container live migration policy based on CPU workload characterization

- Host and Container CPU Utilization
- Host container count
- Reward threshold values

	Control	CPU Greedy Policy	Gain vs Control
Average Hosts CPU utilization	0.68	0.80	17.94%
Average Containers CPU utilization	0.73	0.80	10.69%
Average Workloads Ops	579.85	617.03	6.41%

Greedy Policy performance

	Control	Ideal	CPU Greedy Policy	Gain vs Control
Average Hosts CPU utilization	0.46	0.66	0.55	19.49%
Average Containers CPU utilization	1.23	1.38	1.33	7.74%
Average Workloads Ops	2,025.04	2,328.41	2,173.40	7.33%

Greedy Policy performance compared to ideal case

CPU multi threaded workload container placement policy

Container live migration policy based on CPU multi threaded workload characterization

- Host and Container CPU Utilization
- Host container count
- Container PID Count
- Reward threshold values

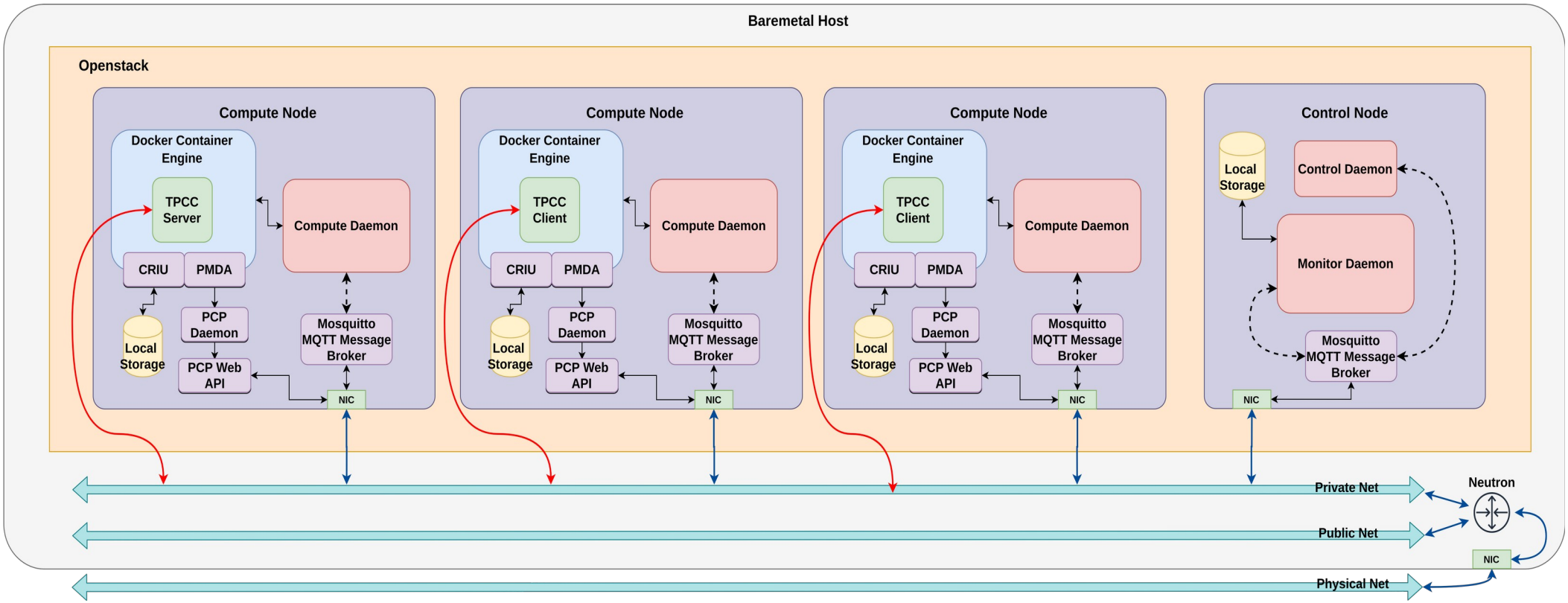
	Control	Ideal	CPU Greedy Policy for Multi-thread Containers	Gain vs Control
Average Hosts CPU utilization	0.46	0.66	0.59	28.67%
Average Containers CPU utilization	1.23	1.38	1.43	16.56%
Average Workloads Ops	2,025.04	2,328.41	2,336.46	15.38%

Improved Greedy Policy performance

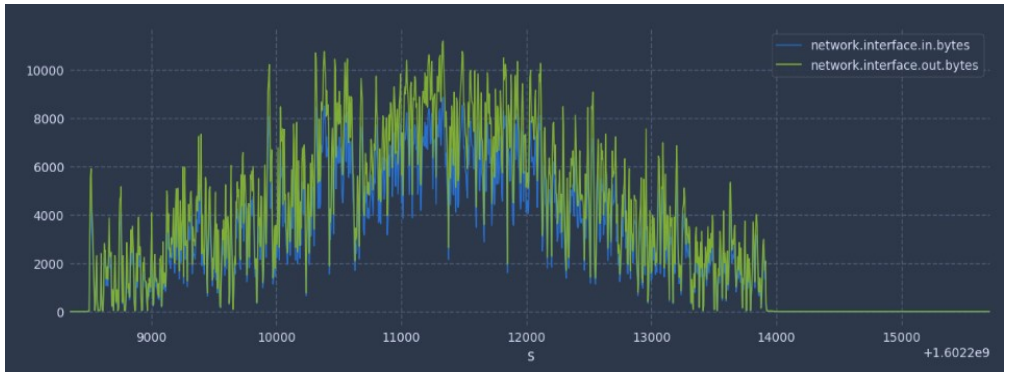
TPC-C Benchmark

- TPC Benchmark C is an on-line transaction processing (OLTP) benchmark.
  - Multiple transaction types, more complex database and overall execution structure.
  - Mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution.
  - Measured in transactions per minute (tpmC).
  - Database is comprised of nine types of tables with a wide range of record and population sizes.

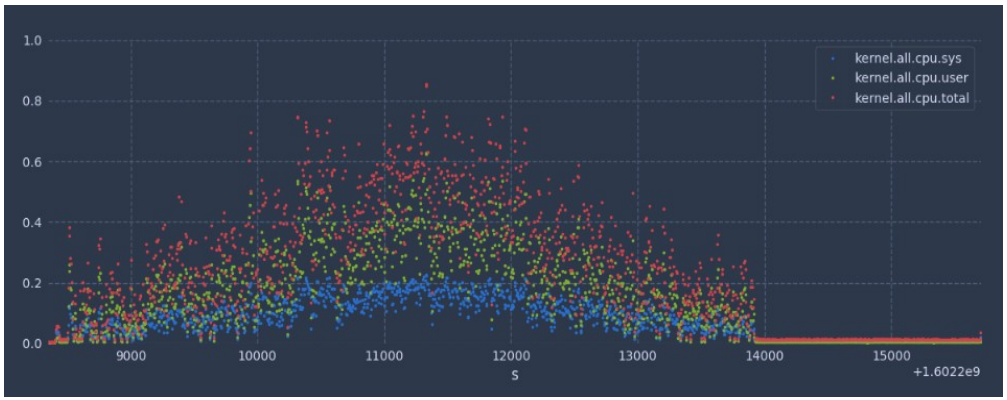
Full System Integration



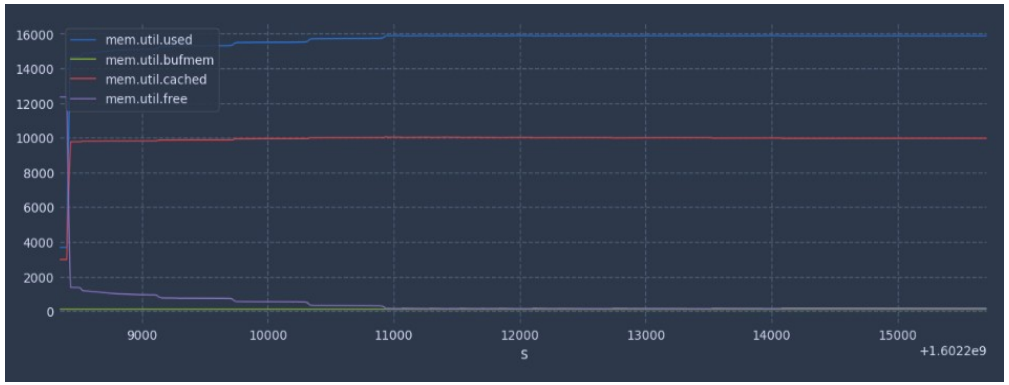
TPC-C Workload Characterization



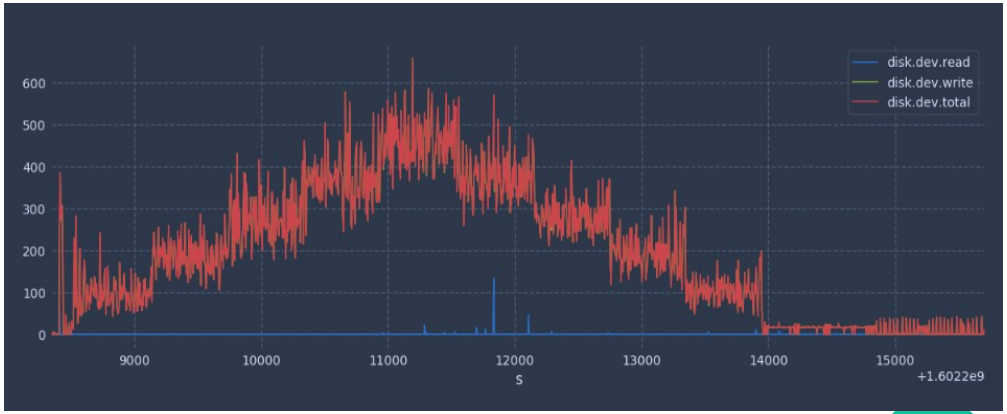
Network throughput on compute node on staircase scenario



CPU time on compute node on staircase scenario



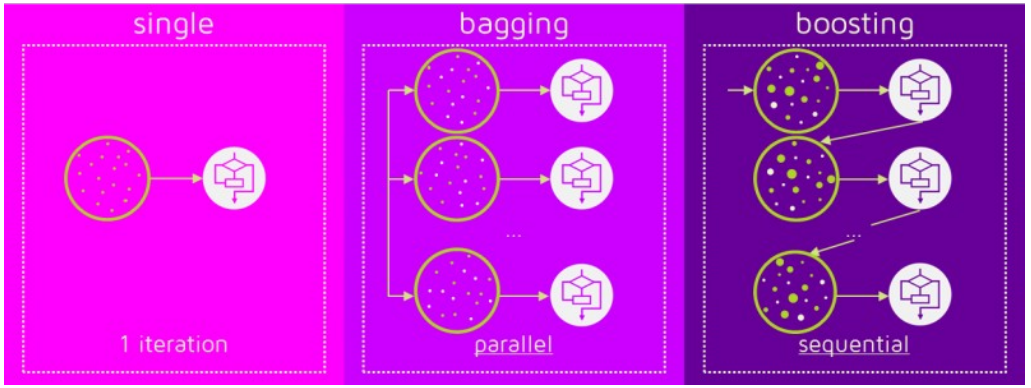
Memory usage on compute node on staircase scenario



Disk ops on compute node on staircase scenario

Machine Learning

- Gradient boosting is a machine learning technique for regression problems.
- Used for estimating relationships between an outcome variable and features.
- Gradient boosting works by producing a group of weak prediction models that can be used together to form a strong prediction model.



Bagging (independent models) & Boosting (sequential models)

XGBoost

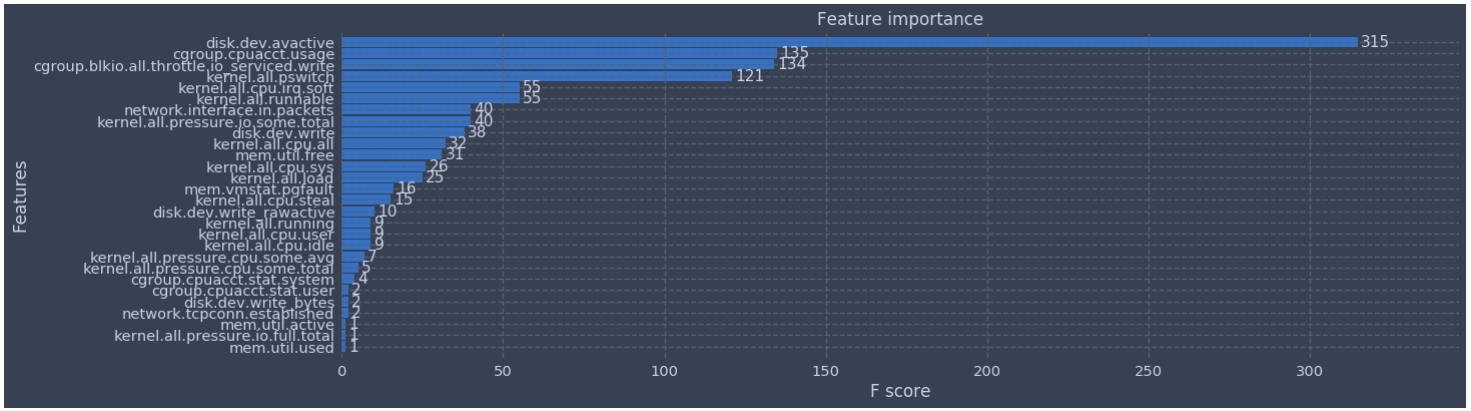
- Feature selection
- Performance forecasting
- Provides relatively accurate forecasting for its short training time.
- XGboost provides multiple approximation algorithms with various degrees of accuracy and training speed.
- We found that its GPU and CPU performance is similar for the hists tree construction algorithm that is an optimized approximate greedy algorithm.

Forecasting

- Use instance performance metrics (ops) to train model to forecast container performance
- By utilizing resource metrics, we can forecast the performance of a container when running on another host.
- Our dependent variable is the container ops, our independent variable are the resource metrics of the container itself and its host.
- We forecast a container’s performance by using the resource metrics of a container and those of a target host.
- We predict the operations per second of a container on said target host.
- Re-sample data to handle the asynchronic nature of Herd Monitor.

Forecasting

Forecast of a Memory Workload’s performance over time for a single container vs actual operations per second

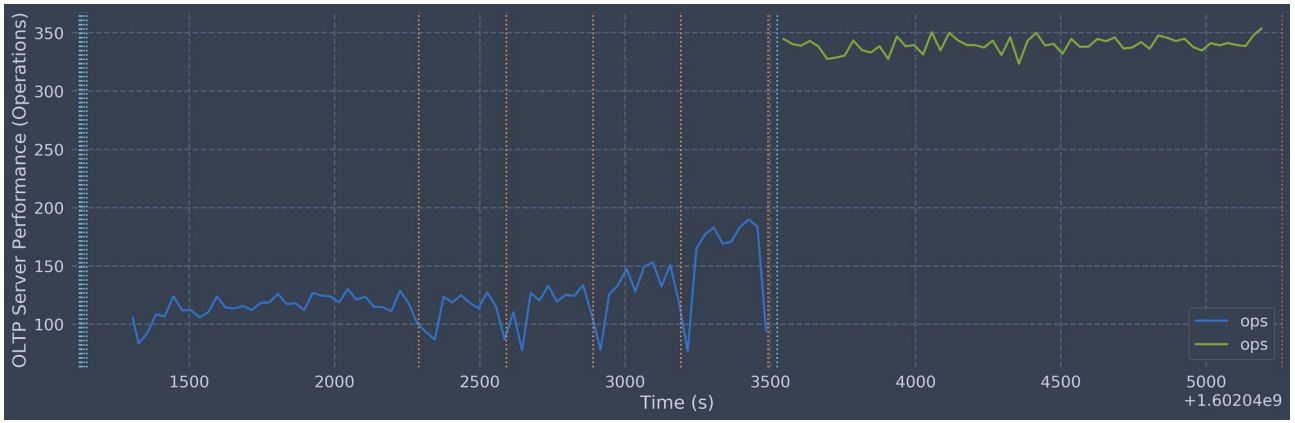


Feature importance plot for a OLTP workload

OLTP Policy

Container live migration policy based on TPC-C workload characterization

- Host disk ops, Container disk ops
- Host Disk activity %
- Host container count
- Reward threshold values



OLTP Server container performance gains due to live migration

OLTP Policy

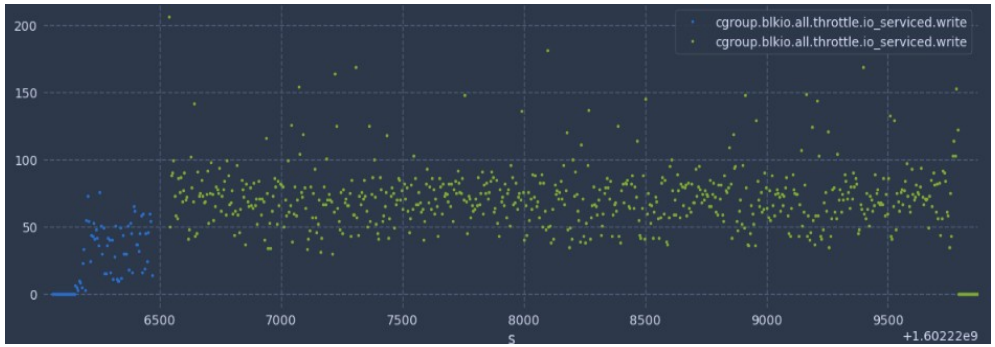
Find container with the highest disk ops and set as candidate, migrate the candidate container to the compute host with minimum disk ops if:

- Candidate container has enough resource metric samples.
- Compute hosts with the minimum disk ops compared to the compute host with maximum disk ops have reached the disk ops reward threshold.
- Compute host with the highest disk activity has reached the disk active threshold.
- Compute hosts with the minimum disk activity compared to the compute host with maximum disk activity have reached the disk activity reward threshold.
- Candidate container has reached the disk ops baseline threshold.
- Compute host with the minimum disk ops is not the same as the candidate container's current compute host.

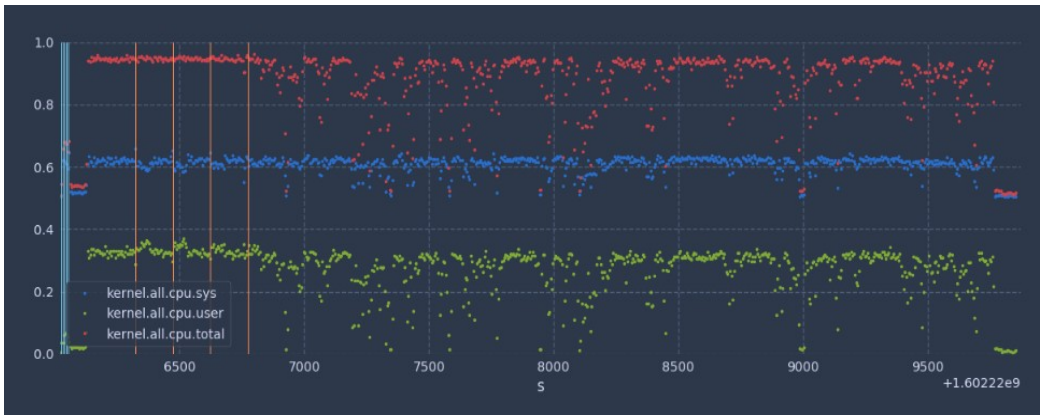
If those conditions are not met, re-evaluate the system and search for new candidate containers, if no new candidates are found evaluate the system with the *CPU Greedy Policy for Multi-thread Containers*.

Disk Ops Baseline = 8000
Disk Ops Baseline Threshold = 0.005
Disk Ops Reward Threshold = 0.35
Disk Active Threshold = 0.15
Disk Active Reward Threshold = 0.15
Missing Samples Threshold = 0.02
Herd Monitor Sampling Rate = 5s
Herd Monitor Container Tracking Rate = 2s
Herd Control Sample Window= 30s
Herd Control Evaluation Rate= 150s
Herd Control Initial Evaluation Delay = 150s

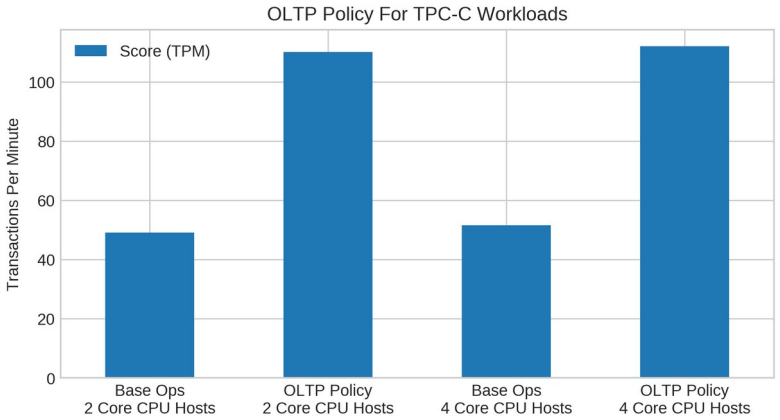
OLTP Policy for TPC-C



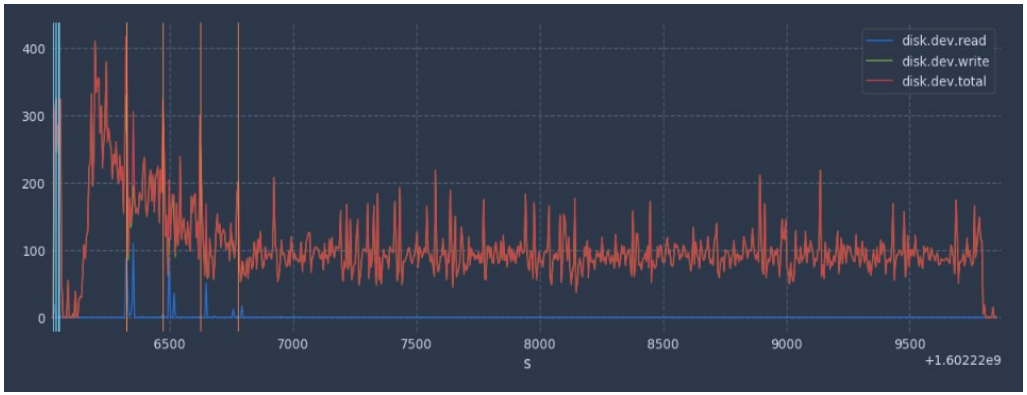
Disk ops on TPCC server container with OLTP policy



CPU time on compute node on TPCC workload with OLTP policy



49.05 vs 110.17 tpM, 51.51 vs 112.01 tpM



Disk ops on compute node on TPCC workload with OLTP policy

Conclusion

- Developed an optimized container live migration toolset that can quickly migrate network-bound containers with filesystem support.
- Designed and developed a lightweight resource monitoring tool that supports live-migrating containers.
- Compared the performance of different live migration techniques and developed our own optimized techniques.
- Developed a resource provision toolset that implements different container placement policies.
- Developed a migration rollback scheme for and fault tolerance and error handling.

Conclusion

Developed a methodology to process system and container resource information for machine learning purposes.

- Designed a machine learning model for container resource provisioning and forecasting.
- Designed and compared different container provisioning algorithms.

A. Gonzalez, E. Arzuaga “HerdMonitor: Monitoring Live Migrating Containers in Cloud Environments” *2020 IEEE International Conference on Big Data (Big Data),* 2020

Future Work

- Continue evaluation different migration policies.
- Perform more system optimizations and handle support for other real-time streaming platforms and container orchestration software.
- Continue testing our system for other uses cases and platforms, such as IoT systems.
- Finishing porting our toolset to handle ARM computer architectures and testing for other computer architectures such as RISC-V.
- Filesystem optimization can potentially open another area of research to handle better support for container live migration.
- We plan on publishing our toolset components and large monitoring dataset (1.2GB) as an open source project.

Q/A