CS695: Assignment 4
Spring 2023-24
shine on you crazy diamond

Statutory note:

- This course is on a **no-plagiarism** diet.
- All parties involved in plagiarism harakiri will be penalized to the maximum extent.
- The Moss Detective Agency has agreed to conduct all investigations. https://theory.stanford.edu/~aiken/moss/ (https://theory.stanford.edu/~aiken/moss/)
- Byomkesh, Sherlock, Phryne, Marple, and Hercule are on standby.
- Hardcoding the runtime output in the code will be heavily penalized.
- Generative AI (ChatGPT, Gemini, etc.) is your friend, but it cannot generate outputs for you to submit.
- **Warning**: Submission Guidelines should be strictly followed; otherwise, your submission will not count. (0 Marks)

This assignment is yours to formulate, plan, execute and report.
The following items apply –

- Groups of 1 or 2 members are allowed.
  Group details for A4 to be update here (https://forms.gle/DRLffhsj8c3xyHhHA)
  (deadline: 10th April 2024, Wed. 9.30 am).
- Basic operations –
  design.build.experiment.repeat.report
  all work will need setup, design, implementation, measurements and reasoning.
- Scope of assignments is yours to determine and claim comprehesiveness, novelty, interestingness.
- **deadline – 20th April 2024**
- reporting –
  problem description, approach, findings, interesting aspects
- evaluation –
  value of A4 is a function of correctness, completeness and features.
  will be based on in-person demo+viva.

A list of a few high-level ideas that you can work with to develop and work with as as follows –

0. your own cool, cooler, coolest idea

1. sharing is caring (KSM and uksm)
Linux has a utility (kernel service) called KSM (Kernel Samepage Merging) which performs content based de-duplication (merging and sharing) of anonymous (private) pages in memory. Can be used processes address space regions that have been madivse'd as mergeable. This is similar to the apporoach in the [memmgmt] paper.

Your work can design a performance characterization study that formulates and empirically answer questions along the axes of workloads, setups, performance and cost metrics, and a set of configuration parameters (scan rate, etc.). Will also need to measure and report metrics/statistics to uncover/explain the behaviour and workings of the service.
A good starting point would be to replicate empirical questions of the [memmpmt] paper.

KSM (https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html) (you can use KSM with VMs, with qemu+kvm a VM is a process for the host).
madvise (https://man7.org/linux/man-pages/man2/madvise.2.html)

2. a study of migratory VMs
Migration of VMs is an essential primitive for management of VM-based cloud services. With Linux, qemu+kvm supports VM migration (with shared storage for VM images).

Your work can design a performance characterization study that formulates and empirically answer questions along the axes of workloads, setups, performance and cost metrics, and a set of configuration parameters (#iterations, n/w bandwidth limits etc). Will also need to measure and report metrics/statistics to uncover/explain the behaviour and workings of the service.
A good starting point would be to replicate empirical questions of the [livemigration] paper.

VM migration (https://www.linux-kvm.org/page/Migration)
migration with virsh (https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_administration_guide/sect-virtualization-kvm_live_migration-live_kvm_migration_with_virsh)

3. a study of migratory containers

Same as #2 but with containers.

criu (https://criu.org/Main_Page) (checkpoint and restore in userspace)
criu github (https://github.com/checkpoint-restore/criu)

4. what is your working set?

The [memmgmt] paper described a technique for periodic and controlled/sampling-based page invalidations to estimate working set size of applications.
Does this approach work?
How effective is the approach?
How efficient is the approach?
Is the approach agile?
… and more.

You work can answer these questions and more based on a set of workloads, setup configurations, changing parameters and set of application and system metrics.

The platform to use for this task can be the simple-kvm setup of Assignment #2 (https://docs.google.com/document/u/1/d/e/2PACX-1vTkWxzF23e3RykpBXpHcUBGcdEL-I2pNCsKF8rynKUdF3R8FVF_7rYzek0eVR38fw4Y292MXesqh7iX/pub).
The guest can used to execute applications with different types of memory access behaviours and the host/hypervisor can be used to setup a periodic page invalidation, fault handling and working set estimation.

5. FaaS + K8s = faast!

Kubernetes can be used to manage a cluster of nodes to instantiat container-based applications. As part of this work, you can build FaaS platform as a wrapper around K8s. Note that a FaaS platform needs to support function registration, trigger registration, trigger dispatch, metrics and more.
A default scope can be to build the FaaS platform, demonstrate its correctness and feature set and some performance benchmarking.

6. docker + request control = orchestra

Modify the conductor of Assignment #3 to work with docker containers, instead of the simpler setup that was part of A3.
Further, extend the conductor framework to provide a load balancer (and maybe other control mechanisms, e.g., scaling) feature that can steer requests across replicas of the same application.
Note that the load balancer implementation has to be yours and cannot reuse the docker command line feature for load balancing.
The replica setting can be part of container specification for the conductor. The load balancer itself can also be configure with different parameters based on which the balancing decisions can be made.

Scope of work would be to design and implementation of the specifications of containers (and replicas) and the load balancer, demonstration of correctness and features and performance benchmarking study.

7. containers + dags = applications

(a) A *serverless* way of application design is to setup a workflow/DAG of functions. The functions themselves would be executing as services within containers. The scope of this work would be to setup a DAG orchestration framework –- DAG specification to compose an application, function/trigger/DAG registration, and dispatch + orchestration to execute applcations (DAGs). The orchestrator can be built on top of a docker containers based setup and aim to demonstrate features, correctness and performance.

(b) Similar to 7(a), but slightly different. An alternate idea would be setup and empirically compare existing serverless workflow orchestrators from a feature and performance perspective.

some examples:
serverless workflow (https://serverlessworkflow.io)
kubeflow (https://www.kubeflow.org)
nextflow (https://www.nextflow.io)