E516 Midterm Report

Ahmad Faiz

March 30, 2024

Abstract

Virtual Machine (VM) consolidation is a crucial technique in cloud computing that aims to optimize resource utilization, reduce energy consumption, and ensure Quality of Service (QoS) for users. This midterm report presents the implementation and evaluation of a static threshold-based VM consolidation approach in a simulated cloud environment. The primary objective is to establish a baseline for comparison with more advanced algorithms, such as the Dynamic Load Mean and Multi-Objective Optimization (DLMM-VMC) algorithm proposed by Li and Cao (2022). The report outlines the methodology, experimental setup, and preliminary results obtained from the static threshold-based consolidation approach

1 Introduction

Cloud computing has revolutionized the way computing resources are provisioned and consumed, enabling on-demand access to scalable and virtualized resources. However, the rapid growth of cloud computing has led to significant energy consumption and resource wastage in data centers. According to Beloglazov et al. (2011) and Birke et al. (2012), the average CPU utilization of physical hosts in cloud data centers is only $15 - 20\%$, while an idle server consumes approximately 70% of its peak energy. This inefficiency not only contributes to increased operational costs but also has a detrimental impact on the environment.

Virtual Machine (VM) consolidation has emerged as a promising solution to address these challenges. By dynamically mapping VMs to physical hosts, VM consolidation aims to minimize energy consumption, reduce resource waste, and ensure Quality of Service (QoS) for users (Farahnakian et al., 2015). However, effective VM consolidation faces several challenges, including accurate host load detection, multi-dimensional resource consideration, balancing multiple objectives, and scalability (Li & Cao, 2022).

This midterm report focuses on the implementation and evaluation of a static threshold-based VM consolidation approach, which serves as a baseline for comparison with more advanced algorithms, such as the DLMM-VMC algorithm proposed by Li and Cao (2022). The static threshold-based approach relies solely on predefined resource utilization thresholds to categorize hosts as overloaded,

underloaded, or normal. VMs are "migrated" from overloaded to underutilized host to alleviate the average resource consumption across the universe of all hosts. Certain design may allow dedicated clusters to handle VM migration from checkpoints as well.

2 Experimental Setup

Li et. al. describe the data center resource representation model as:

Assume $H = h1, h2, ..., h_j, ..., h_n$ is the collection of a data center's hosts, where n is the number of hosts, each host has D-dimensional resources and $C_{h_j}^d, d \in 1, 2, ..., D$ represents the capacity of resources d of a host h_j.

$V = v1, v2, ..., vi, ..., v_m$ is the collection of the data center's VMs, where m is the number of VMs; similarly, each VM also has D-dimensional resources, and $C_{v_i}^d, d \in 1, 2, ..., D$ represents the capacity of resources d of a VM v_i.

Assume $VM(h_j)$ indicates the VM collection of a host h_j, $U_t^d(v_i)$ represents the actual utilization of resource d of a VM v_i at time t, and $U_t^d(h_j)$ is the actual utilization of resource d of a host h_j at time t and can be expressed as $U_t^d(h_j) = \frac{1}{C_{h_j}^d} \sum_{v_i \in VM(h_j)} U_t^d(v_i) * C_{v_i}^d$

This report attempts to simulate this setup on a jetstream-2 cloud instance using a multi-component architecture that leverages Docker containerization to simulate hosts and VMs for each hosts. We write a multi-threaded Python script to emulate virtual machine (VM) consolidation processes.

2.1 Cloud Simulation Using Docker

The cloud environment is simulated using three distinct hosts, each represented by a Docker container. These hosts are labeled as host1, host2, and host3. Each host is configured with a **docker-compose.yml file**, which specifies the setup of three VMs within the host. The VMs are created using Docker images built from a common **Dockerfile**. The configuration of each VM is designed to simulate a computing environment with resource constraints, defined as follows:

- **CPU Limit**: Each VM is allocated a maximum of 1.0 CPU
- **Memory Limit**: The memory capacity for each VM is set to 1 GB

To induce a load on the VMs and simulate a realistic cloud environment, the stress-ng tool is employed within the VMs. The tool generates a controlled load on the system by utilizing CPU, I/O, and memory resources. The stress-ng command is configured to run for a duration of 300 seconds, stressing one CPU, performing I/O operations, and allocating 128 MB of memory.

The initialization of the cloud environment is managed by a shell script named **start-app.sh**. This script executes the docker-compose up command for each host, resulting in the instantiation of the VMs with the predefined load conditions.

2.2 Python Script for Monitoring and Migration

The core of the VM consolidation process is a Python script named **VM-consolidation.py**. This script is responsible for monitoring the resource utilization of VMs and orchestrating the migration of VMs from overloaded hosts to underutilized ones. The script is composed of two primary threads: the MonitorThread and the MigrationThread.

- **MonitorThread**: This thread continuously monitors the resource utilization of each VM on all hosts. It calculates the average CPU and memory utilization across the VMs within a host and determines the load status of the host based on static thresholds. The thresholds are defined as follows:
  - **Upper CPU Threshold (THR-max-cpu)**: 55%
  - **Lower CPU Threshold (THR-min-cpu)**: 20%
  - **Upper Memory Threshold (THR-max-mem)**: 55%
  - **Lower Memory Threshold (THR-min-mem**: 20%

  The host status is categorized as **overloaded, underloaded, or normal**. If a host is identified as overloaded, the VM with the highest resource utilization is selected and enqueued for migration.

- **MigrationThread**: This thread is responsible for handling the migration of VMs. It checks the migration queue every 2 seconds for VMs that need to be migrated. The migration process involves stopping the VM on the current host, committing its state to a new Docker image, and starting a new VM from that image. The migration is simulated on the local Docker host for simplicity.

2.3 Simplifications and Assumptions

For the purpose of this experimental setup, several simplifications have been made to focus on the process flow and the functionality of the VM consolidation algorithm:

- Same-Host Migration: Instead of migrating VMs to an actual underutilized host, the committed container image is restarted on the same host. The final report will be using algorithms to detect underutilized host and use TCP sockets to send the commited image to the target host.
- Static Thresholds: The thresholds for determining the load status of hosts are statically defined
- Load Inducement: The stress-ng tool is used to create a predictable and consistent load on the VMs
- Thread Synchronization: A simple locking mechanism is used to synchronize access to shared resources between threads.

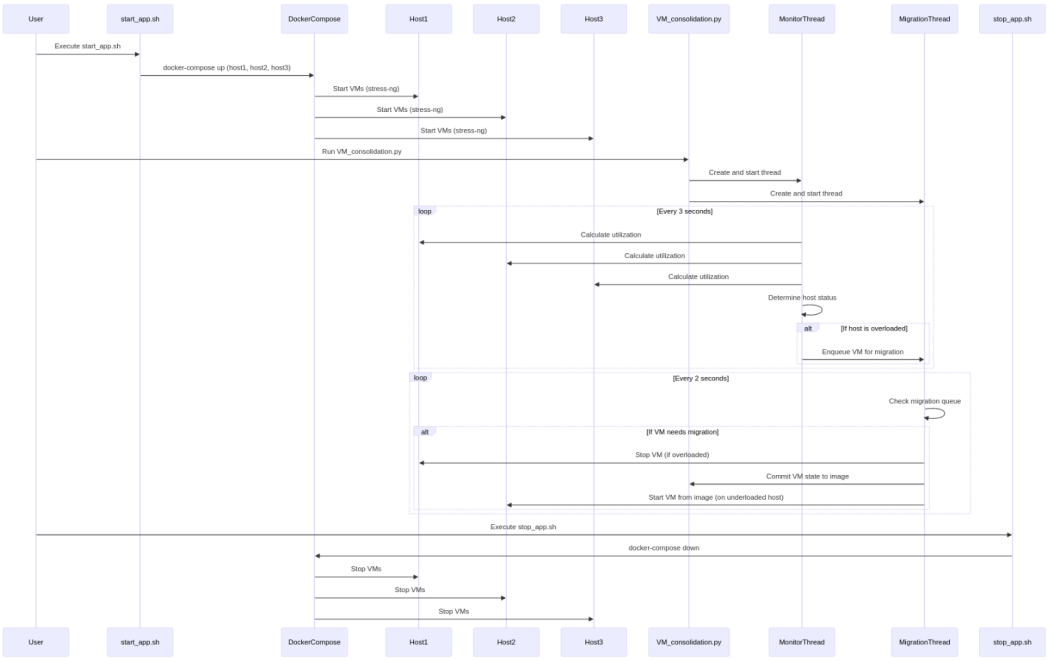

Figure 1: State-Space diagram for the implemented VM consolidation process

The algorithm does not consider dynamic workload fluctuations or other factors such as energy consumption, resource waste, or migration overhead. It serves as a baseline for comparison with more advanced algorithms like DLMM-VMC to be implemented in the final report.

A sequence diagram with specifics to the design is given in Figure 1.

2.4 Running the application

The application consists of a set of Docker containers that simulate virtual machines (VMs) on multiple hosts, and a Python script that manages the monitoring and migration of these VMs based on their resource utilization.

2.4.1 Granting Execution Permissions

```
chmod +x start_app.sh
chmod +x stop_app.sh
```

2.4.2 Start simulation

```
./start_app.sh
```

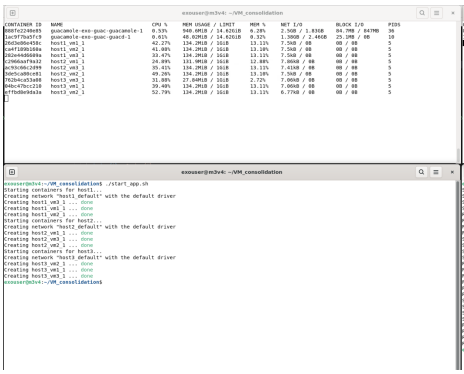

Figure 2: On Application start

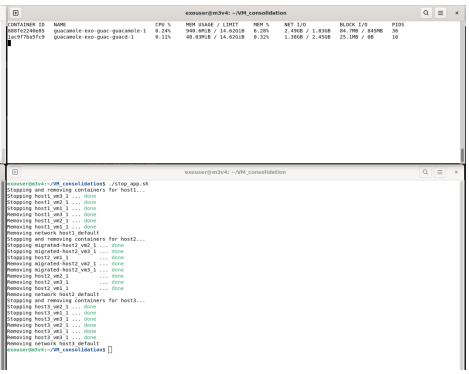

Figure 3: On Application stop

2.4.3 Running the Monitoring and Migration Script

```
python3 VM_consolidation.py
```

2.4.4 Stop simulation

```
./stop_app.sh
```

2.4.5 Logging

A file named VM-migration.log is created which logs the INFO messages and can be used for evaluation

The final code repository can also be found at the following link:
Github

3 Screenshots

Figure 2,3,4 are screenshots taken during the simulation run:

4 Next Steps

- Implement host detection for underloaded host and use TCP sockets to start the committed overloaded VM to the target host
- Implement a framework to report on resource usage metrics encompassing energy and nwteork bandwidth
- Implement the multi objective optimization problem algorithm using Ant Colony Optimzation as described by Li et. al.
- Compare the results of dynamic consolidation with results fro this static threshold implementation

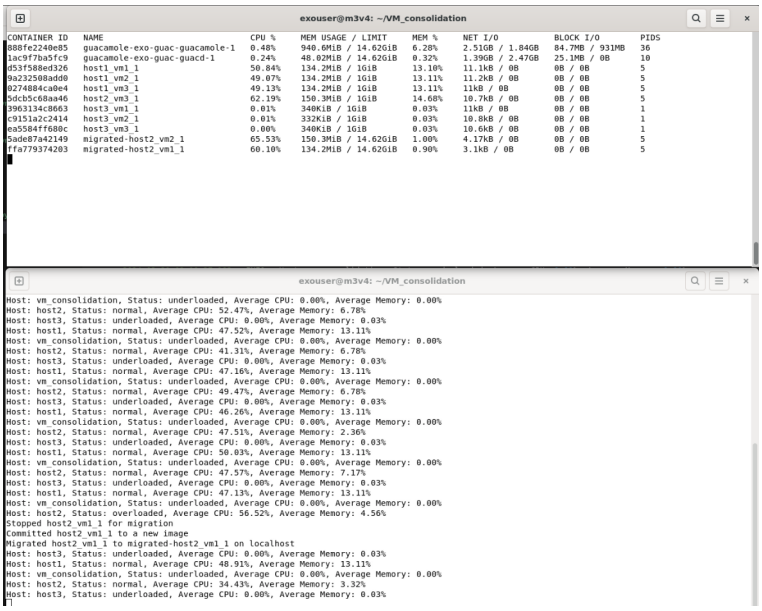

Figure 4: Application Run