Pennsylvania State University
Department of Computer Science and Math
COMP512 - Advanced Operating Systems

AVD Pipeline

AV Distributed Pipeline

Aishwarye Omer & Ved Patel
ado5146 vmp5265

Supervisor:
Dr. Linda Null

Version 0.0.1

Harrisburg, February 2020

Table of Contents

1. Introduction

"***Nothing is particularly hard if you divide it into small jobs***"
– Henry Ford

This one sentence was the motivation behind the first pipeline architecture ever to be used in a commercial business process. Since then, not only car manufacturing companies, but the computer industry also has made full use of this sophisticated process of pipelining.

2. Problem Statement

A pipeline architecture is a system in which multiple tasks are divided into subtasks. These subtasks are processed in different stages in parallel where the output of one stage serves as the input of the next stage. While one stage is working on some input, the consecutive stages are working on the output produced by the previous stages thereby increasing the throughput of the system without affecting latency. The most desirable goal of a pipeline architecture is that every stage is balanced i.e. every stage takes roughly the same amount of time to compute on the input given to it.

In traditional distributed systems, a server communicates with one or more clients whenever they are involved in processing some part of a task. When one client finishes its work, it sends the information to server and the server then gives that information to the other client. We reduce this interaction by introducing a pipeline such that the clients mimic the stages of the pipeline and the output produced by one client/stage is directly passed to the next client/stage. The previous client can now receive the next set of input and the process goes on.

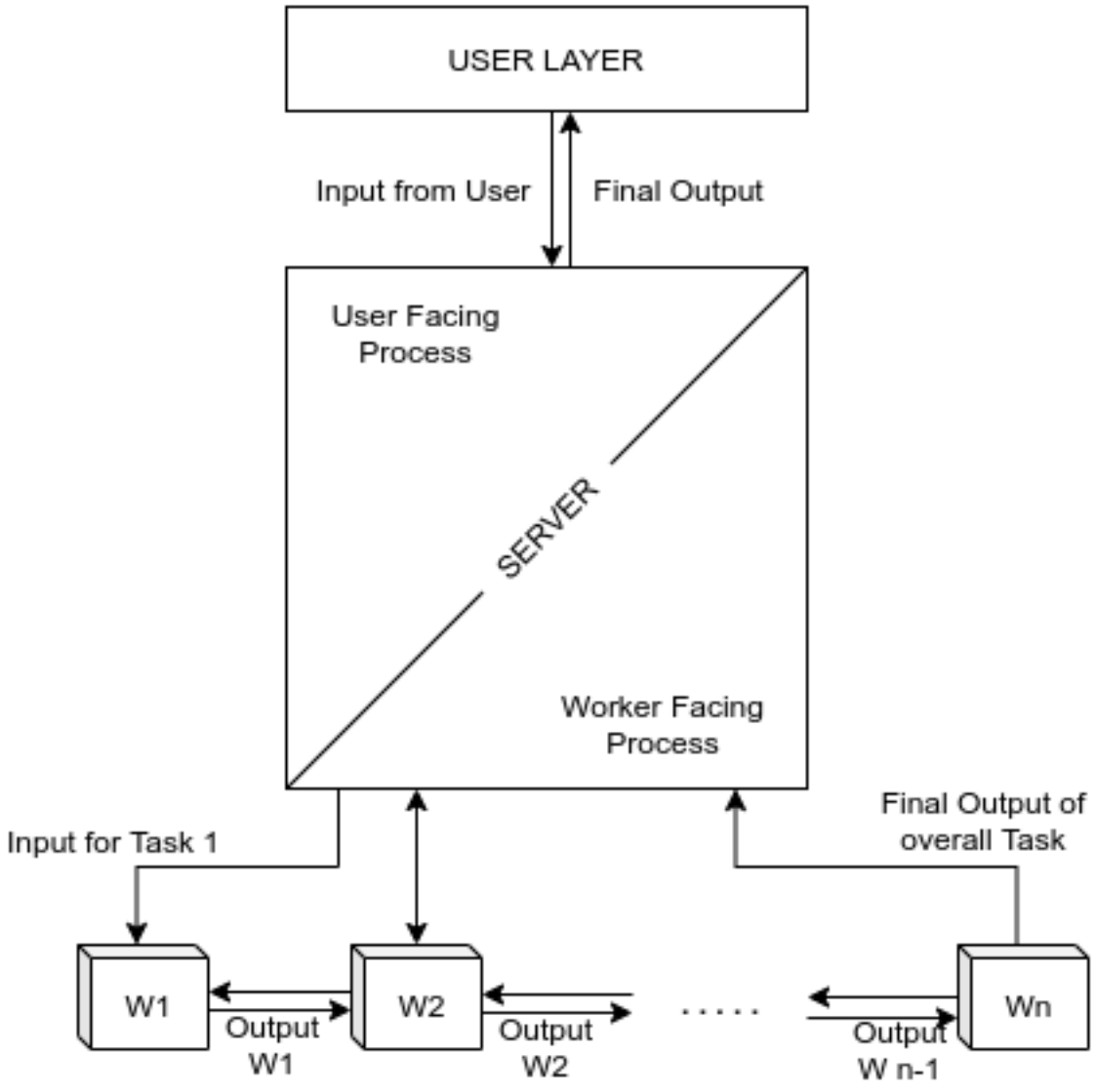
Through this project, we aim to improve the throughput of a distributed system by integrating a pipeline architecture with a distributed system.

3. Design

1. Approach

To implement a pipeline architecture with a distributed system, we first break the application task into sub tasks. These sub tasks will in fact be major macro tasks of the application and according to number of sub tasks, the workers would be chosen and assigned those tasks. These workers become the stages of the pipeline.

2. System Architecture



The above architecture has two major components:

(a) **Server:** It will be responsible for maintaining the network and controlling the workers along with deploying the application over the network. The server is divided into two logical halves based on what type of process it will handle. It shows the two major processes running on the server:

   i. **User Facing Process**: This logical side of the server which will take input from the user to run an application over the pipelined workers. It will provide an interface in the form of a terminal window or a custom graphical user interface to the user to run their application.

   ii. **Worker Facing Process**: This side of the server will accept input from the user facing side of the server and will be responsible for dividing up the task into stages for the pipeline. Once it divides the tasks, worker nodes are chosen for the pipeline depending upon the number of stages needed for the application. Each stage is given to a worker node. This side also maintains the status of the workers along with communicating with neighboring workers for input, output or running status update.

(b) **Workers:**

   i. They will be responsible for computation of the task or the application and return successful output back to the server.

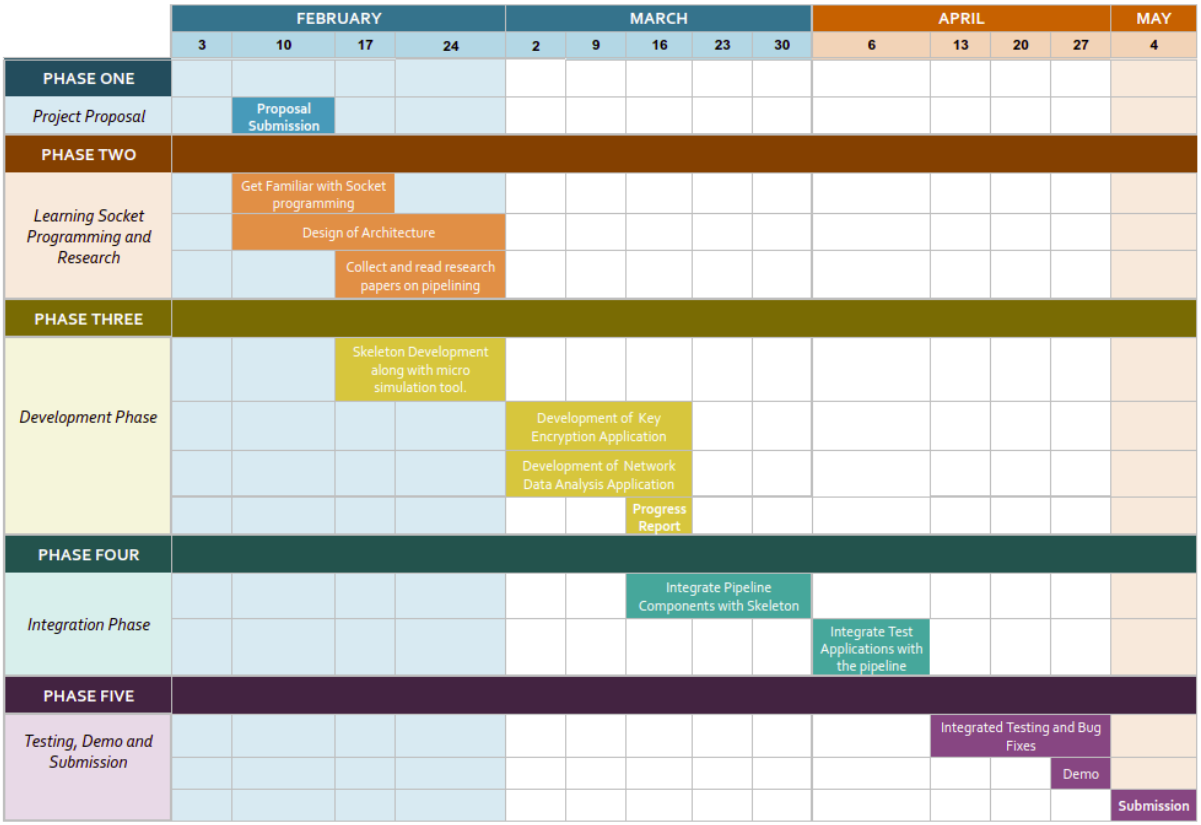
   ii. The computation will be divided between **N** workers where the output of one stage will be input for the second and so on.

   iii. Each stage or worker is dependent upon the output coming from the previous stage

   iv. The workers in the pipeline will be chosen according to the need of the application. For example, If the application needs 3 interconnected stages in the pipeline, 3 workers would be chosen where the first worker will receive input from the server and the last worker will return the output to the server.

4. Timeline and Test Plan

Timeline



Test Plan

Testing will be done in two phases.

1. **Simulation Testing**: In this testing, we will develop a small simulation tool which will run on a single machine and emulate the user, worker and server processes running on separate threads. These threads will communicate with each other using sockets or pipes.

   Simulator will be used to test small independent features for unit testing. This will help in increasing the efficiency of the testing.

   (a) It makes experimenting easy and safe. Development and testing of modules can be done on simulator and once module has been thoroughly tested, it can be directly integrated with the mainline code.

   (b) Developer will not be required to bring up the whole setup every time they want to test the code.

2. Once all the modules are integrated with the mainline. We will bring up a test bed setup, either in-house or on the cloud. The project will be tested as a singular system for

   (a) All the feature independently.

   (b) Robustness of the system.

   (c) Check for security flaws

   (d) Check for system resource usage such as memory, CPU etc.

5. Challenges

Some of the major challenges would be

1. **Minimal messages between clients**

   To decrease latency and improve overall throughput and utilize more bandwidth, the number of messages sent and received between clients should be as low as possible.

2. **Balanced Stages**

   Keeping the stages balanced in terms of computation time so that no stage is too fast or too slow for the other nodes.

3. **Debugging**

   Debugging the pipeline in case of any breakdown or inconsistency

4. **Fault Tolerance**

   Fault tolerance in case the server goes down or one of the stages goes down.

6. Expected Scenarios

There are three expected scenarios

1. **Minimal Case:** Build the pipeline and run one application successfully.
2. **Expected Case:** Along with the minimal case, run the second application successfully
3. **Best Case:** Along with the expected case report the network status of the system.