Performance Comparison Framework for Consensus Algorithms

Radhika Dhawan, SBU ID# 112074050

Jatin Sood, SBU ID# 112079000

Rohit Aich, SBU ID# 112126618

I. Overview

We are writing our driver programs in DistAlgo. To start off, we will be using both Python [1],[2] and DistAlgo [3],[4] implementations of Paxos and Raft. We will also try to add other algorithms and implementations as and when we find those. Also, we will have a monitor program to control all the drivers. Here, we are using separate driver programs for each separate implementations, because each implementation has separate nomenclature and flow for different components. These driver programs would be responsible for capturing the different timing and memory usage details, and pass those on to the monitor program for comparison and visualization.

II. Project Design & Architecture

I. Driver Programs

- *Figure* 1 shows a schematic architecture diagram of our design. We are using separate drivers for each of our implementations. In our figure, Implementation A, B, C and D represents Paxos and Raft implementations in Python and DistAlgo respectively, as they are the examples we have taken to start with. The corresponding drivers (A..D), are used to control these implementations. The drivers are coded in DistAlgo, and are designed to capture CPU time and memory usage details of each implementation by varying the input parameters like number of runs, number of participating processes, message delay etc.
- The driver programs would send these details to a single controlling monitor program for performance comparison.
- We also plan to generate logs after certain specific steps inside the implementations. These logs, captured in specific formats will depict the inter-process communications. We plan to take these out and use them as an input for visualization tools (e.g.: Shiviz, Visdia etc).
- We are using the 'psutil' library of Python to capture the performance details.

II. Monitor Program

- We are designing a single monitor program that will be responsible for starting all the drivers in the system. Also, this monitor program will read all the logs that the drivers generate as outputs.
- From those captured logs, we plan to use the monitor program to create charts and graphs, to compare the time and space complexities of the different implementations.
- This decoupled architecture will help us to maintain an abstraction between the main monitor program and the end implementations. The modular nature would ensure that we could insert any other algorithms or implementations in our framework. We would just have to write a DistAlgo driver for the program, and our monitor would be able to compare performances their performances.

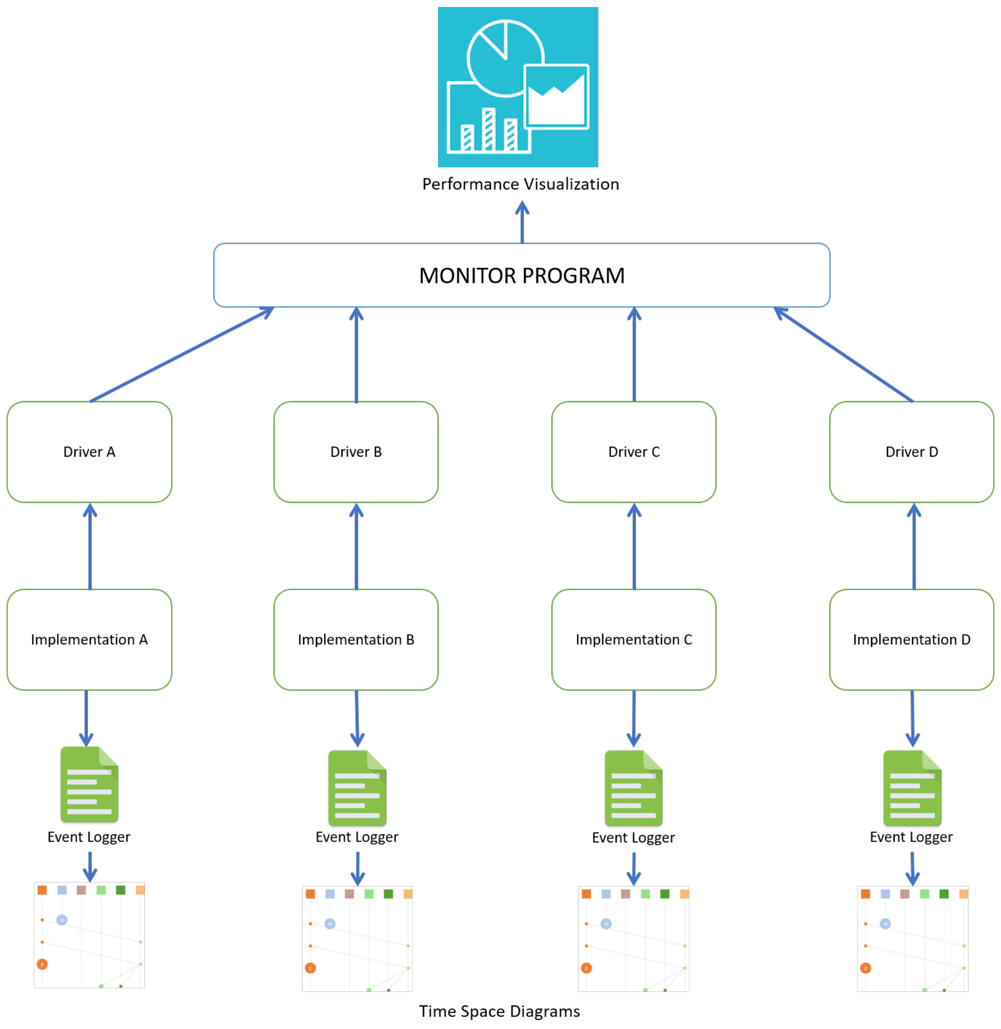

Figure 1: Architecture Diagram

III. Implementation & Discussion

The preliminary implementation has three primary parts. We are using the VRA Paxos implementation in DistAlgo as our first example. We have created a driver around it to run the code and log the results in a file. Further, we are attempting to log the movement of messages between the Acceptors, Learners and Proposers, and log them in specific formats, so they could be parsed by visualization tools like Shiviz. Further, We are using the python library 'matplotlib' to create graphs and charts based on the logged performance data by our driver.

I. Metrics Used For Performance Measurement

- One of the primary metrics to understand better implementations is to check the lines of code in them. We plan to use **lines of code** as one of our metrics.
- As we are comparing distributed consensus algorithms, we also plan to count the number of messages needed for a set of processes to reach consensus. We will call this count **message complexity**, and propose to use this as a metric.
- We would measure the memory utilized for each implementations of these consensus algorithms, and measure the **space complexity**.
- We would also log the **elapsed time** and the actual **CPU time** for each implementations, for different runs and input parameters.
- We would use all the aforementioned metrics to create visual charts for better comparison and presentation.

II. Logging Formats

- For each of the implementation, we are going to record the events between the interacting processes using logs and feed it into the respective Event Logger. An event in a participating process can be defined as initializing of a process, sending a message to another process or receiving a message from another process.
- When a process is initialized, it logs its name, clock value along with the execution information.

```
<Replica:28005> {"<Replica:28005>":1} Initialization complete
```

- When a process sends a message to another process, it logs its name, its own clock value along with clock values of other processes it has received so far from other processes and event information. It then sends its clock value to the receiving process along with the message.

```
<Client:28008> {"<Client:28008>":2} Sending request to all replicas.
```

- When a process receives a message from another, it receives that process clock value together with the message. Similarly to the sending process, it logs its name, its own clock value along with clock values of other processes it has received so far from other processes along with the event information.

```
<Replica:28006> {"<Replica:28006>":2, "<Client:28008>":2} Received request from client
```

- We use following regex in shiviz to parse event logs of an implementation from Event Logger.

```
(?<host>\S*) (?<clock>{.*})\ (?<event>.*)
```

IV. Project Plan forward

- We plan to integrate visualization tools like 'Shiviz', 'Visdia' etc with our implementation. We are currently working on generating logs specific to these tools. We plan to do a short feasibility study to understand which tool would be the best to integrate.
- As a stub preliminary implementation, we have created a simple bar chart depicting the elapsed time in different runs from the performance log of vRA Paxos in DistAlgo. We plan to extend this to other implementations as well, and also include other charts for improved and better understanding.
- Also, we have plans to use these data to draw inferences about the efficiency of the algorithms.
- This project can be extended to make the drivers more decoupled, so that a single driver can be used for multiple algorithms and implementations. This can be done by creating a reusable library, and extending that in the implementations, and generating logs there itself. We will try to attempt this if we get time.

References

[1] Cocagne, T 2018, 'multi-paxos-example', GitHub Repository: 'https://github.com/cocagne/multi-paxos-example.git'.

[2] Ozinov, F 2018, 'PySyncObj', GitHub Repository: 'https://github.com/bakwc/PySyncObj.git'.

[3] Lin, B 2017, 'vrpaxos', GitHub Repository: 'https://github.com/DistAlgo/distalgo/tree/master/da/examples/vrpaxos'.

[4] Lin, B 2017, 'raft', Github Repository: 'https://github.com/DistAlgo/distalgo/tree/master/da/examples/raft'