CS 4513 - Distributed Systems
Project final (29 points)

Project final: Bring Your Own Project!

This is a group project.

Our last project is designed to provide you with more flexibility to explore. After all, this is likely to be one of your last projects done at WPI given that you are graduating! Even though this course is coming to an end, there are still so much more to learn about distributed systems. One of the key goals of this project is to convey this sentiment and to provide a high-level roadmap for your own adventure.

To help you get started, we provide the following three potential options for you to choose from. Each option includes recommended workflow and examples. Of course, you are welcome and encouraged to propose and come up with your own ultimate project—provided it is related to distributed systems.

Option 1: Implementation

If you are interested in gaining more experiences with Go beyond the *Project MapReduce*, you probably will like this option. At the high level, this option is about adding more features to the simple MapReduce framework you have implemented previously.

For this option, the recommended steps are following:

1. Read the original MapReduce paper from Google to gain deeper understanding about the design.
2. Pick one feature that interests you the most and sketch out what you need to do to incorporate the said feature into your existing MapReduce framework.
3. Design a few corresponding test cases.
4. Implement the feature based on the test cases.

Option 2: Case Study

If you are interested in seeing how classical distributed systems techniques are being used in modern distributed systems, then you might like this option. At the high level, you will pick an open-source distributed system and do a deep dive of its internal with the key goal of identifying which techniques are used and for what.

Here are a list of modern distributed systems that you might find interesting to explore:

1. Apache Spark is an in-memory data processing framework. Github link: `https://github.com/apache/spark`
2. Alluxio which is a virtual distributed storage system. Github link: `https://github.com/Alluxio/alluxio`
3. Pytorch Distributed which allows training deep learning models on different workers. Github link: `https://github.com/pytorch/pytorch/tree/master/torch/distributed`

Option 3: Measurement

At the high level, you will pick a distributed systems related technology, learn how to use it, and measure the performance. For example, if you want to know more about `grpc` which is Google's RPC system, the recommended steps are following:

1. spend some time reading the docs for `grpc`: `https://www.grpc.io/docs/`
2. make sure you understand how to measure event times with precision with functions `clock_gettime()` and `gettimeofday()`.
3. think about performance metrics that are important for RPC
4. come up with a list of factors such as data types and network conditions that might lead to different RPC performance.
5. vary each factor while keeping the other factors constant to perform the corresponding measurements.
6. (Optional) If you are really interested in RPC and are curious about how `grpc` compares to other systems such as `Apache Thrift`, then you can repeat the above steps and do a shoulder-to-shoulder comparison.[1]

Here are a list of distributed systems technology that you might find interesting to measure:

1. Any cloud-based services, such as Virtual Machines, Containers, and serverless: as a first-time user, you can get one-year free access to most of AWS services.
2. Any open-source distributed systems: including the ones mentioned in the Option 2: Case Study.

Checkpoint Contributions

Checkpoint is not required for this project.

Deliverables and Grading

When submitting the project, please make sure to satisfy the following requirements:

- Relevant deliverables to this project such as code and PDF writeup.
- A document called README.txt explaining the project, any defects, and anything that you feel the teaching staff should know when grading the project. Only plaintext write-ups are accepted; Markdown is allowed.
- A document called self_assessment.txt that describes your group's self-assessment of your performance (see below).

Please compress all the files together as a single .zip, named `YOUR_WPI_USERNAME_project_final.zip`, archive for submission. As with all projects, please **only use standard zip files** for compression; **.rar, .7z, and other custom file formats will not be accepted**.

[1] This is an useful skill to have as business decisions are often made by critically analyzing comparable products.

Grading

This is probably the last project you will complete at WPI. We will be experimenting with a self-assessment based grading.

At a high-level, each group should reflect this final project process and use the following guideline to perform self-assessment. The self-assessment should include two parts: a category selected from the following five, and a brief paragraph that explains the selection rationale.

- **Exceptional (29 points)**: The project is well done and represents the high-quality expected of a senior-year computer science student. Depending on the exact deliverables, e.g., design, implementation, or case study, the specifics can differ. For example, if the deliverable is code, then exceptional can mean that the code implements all the features specified in the design document and runs reasonably efficiently. No functional errors are present. *The code is organized, easy to read, and well documented.* As an another example, if the deliverable is a research paper, then exceptional can mean that the case study is detailed and clear. It contains all the required components and presents a convincing case for the proposed idea and design. The use of visual aids such as diagrams and figures are appropriate and professional. **The remaining categories assume the deliverable is code in explaining the specifics. Requirements for other deliverable types can be deducted.**

- **Exceeds expectations (29 points)**: The implementation generally works, but has one or two mild weaknesses. Weaknesses can include occasionally unclear code or mild inefficiencies. The code may not fully address the design proposed in the previous milestones, but includes reasonable justifications and alternative implementations. A minor functional error or two may be present. *The code is organized and well documented.*

- **Meets expectations (25 points)**: The implementation has obvious flaws, with a significant functional error or multiple minor ones. The code may be unclear in multiple places or has some significant instances of inefficiency. The code does not fully address a couple key design proposed in the previous milestones, and does not include reasonable justifications. However, some alternative implementation is included. *The code is organized and contains sufficient documentation for teaching staff to understand the key ideas.*

- **Improvement needed (21 points)**: The implementation mostly works, but has several missing or poorly implemented components. The code does not fully address a number of the key design proposed in the previous milestones, and does not include reasonable justifications nor alternative implementations. *The code may be challenging to understand in multiple places or has sparse documentation.*

- **Unsatisfactory (17 points)**: The implementation has severe flaws that undermine its proposed features. The implementation does not address any of the previous design aspects. The implementation does not support any relevant features of the proposed application. *The code is missing critical components as well as documentations, hindering the teaching staff's ability to assess the project quality.*