Trilobita: A Distributed Graph Computing System
Checkpoint 1

1. Group Members

Guo Yuchen	1004885
Guo Ziniu	1004890
Liang Junyi	1004891
Wang Yanbao	1004865
Xiang Siqi	1004875

2. Introduction

Trilobita is a distributed graph processing system designed to efficiently handle large-scale graph processing tasks in a distributed environment. This report is the checkpoint 1 report of the course 50.041 group project. It provides insights into the problem we aim to solve, the introduction to *Trilobita*, our implementation plan, and our validation approach.

2.1. Problem Statement

The problem that lies in this project is achieving both efficient and consistent processing of large graphs within a distributed, fault-tolerant, and adaptable framework. Large-scale graph computing is prevalent in many practical domains [1], necessitating distributed processing due to their size and complexity.
Inspire by and following the idea proposed by Pregel [1], *Trilobita* endeavors to incorporate several critical features, including:

1. **Fault Tolerance**: Ensuring the system can gracefully manage failures of both the master and worker servers, without imperiling data integrity and enabling a seamless continuity in server failure handling.
2. **Flexibility**: Enabling users to customize graph processing tasks through a range of user-accessible and user-definable interfaces.
3. **Efficiency**: Optimizing communication and processes to efficiently handle data transfer across distributed resources. Gracefully implement “Superstep” logic to minimize time spent on computing.
4. **Consistency**: Guarantee a consistent data model and processing logic across the entire distributed computation process.

3. System Design

Trilobita consists of several modules, including core, runtime, engine, common, config, and examples. These modules collectively form a cohesive system to facilitate graph processing in a distributed context.

1. **Core**: The core module is the central component of *Trilobita*, encompassing essential graph processing logic and basic system functionalities such as communications, etc. It manages tasks at the vertex level and houses the communication APIs designed for efficient message passing between vertices and information propagation within the distributed system.
2. **Runtime**: The runtime module establishes the necessary runtime environment for *Trilobita*. The launcher implemented in this module serves as the entrance point of a *Trilobita* running job. Additionally, It has a monitor module to track and record system performance during execution.
3. **Engine**: The engine module is responsible for managing the distributed servers and their workflows in the system. The master server coordinates workers and supersteps, managing worker failures and recoveries. Worker servers handle tasks at the vertex level. Some detachable functionalities such as Combiner [1] and Aggregator [1] are also provided by this module.
4. **Common**: The common module contains shared utilities, data structures, and components used across different modules of *Trilobita*.
5. **Config**: The config module houses system configuration and settings. It allows users to customize various parameters of *Trilobita*, such as the number of threads on each worker, *etc*.
6. **Examples**: The examples module plays the role of a showcase of *Trilobita*'s capabilities. It provides practical examples and use cases, demonstrating how users can define and execute their jobs using *Trilobita*.

3.1. Language Choice

We chose Java as the programming language of our project. It aligns with our vision of developing a robust system in a bottom-up manner. We trust implementing it in Java could enable us to more efficiently realize our vision.

While the group initially considered using Golang as the primary language, we eventually opted for Java due to Java's perfect object-oriented programming design pattern and exceptional scalability in handling complex workloads.

3.2. Development Approach

Our approach involves developing the distributed system from the ground up, ensuring that each API layer is optimized and finely tuned for distributed execution. From the perspective of module dependencies, the core module serves as the cornerstone of the system, forming the base layer. The engine module therefore builds upon the core, relying on its functionality to coordinate vertice and messaging. The runtime module, which defines and manages the runtime environment, is at the top of the system layers. We will thoughtfully integrate config and common modules throughout the architecture, to harness their shared utilities and data structures while maintaining system robustness.

3.3. Third-party Framework Usage - Kafka

For server-to-server communication within *Trilobita*, we choose to use Kafka. It provides us with a reliable communication method across machines and saves us more time on the implementation of the system and fault-tolerance mechanism. Other than that, all other parts of this system will be built from scratch. We will be leveraging some Java native packets such as *java.util.concurrent*, etc.

4. Validation

To validate the success of *Trilobita*, we will consider factors such as processing speed, resilience in the face of diverse failure scenarios, and scalability to larger graphs.

4.1. Use Cases

Based on the adaptability design principle of *Trilobita*, users will be able to define their computing jobs and entities within the system. We will exemplify this capability by two very fundamental use cases: Breadth-First Search and Shortest Path algorithm. If time is applicable, to make it more concrete, we might delve into more complicated scenarios, including neural network inference on edge devices [2] and PageRank [3], to demonstrate the system's applicability in real industrial contexts.

4.2. Distributed Node Simulation

For validation and testing purposes, we will simulate distributed nodes by employing a real computer cluster. This cluster may consist of 4 to 6 user laptops, replicating the distributed conditions. This allows us to evaluate *Trilobita*'s performance in a controlled environment.

5. Conclusion

Trilobita is a Pregel-like distributed graph processing system that prioritizes adaptability, efficiency, and robustness. In this report, we provide a concise overview of the problem scope and its definition. We introduce features of *Trilobita*, a system inspired by Pregel. Additionally, we present the structure of this project, the implementation plan, and the plan for use cases and validation.

6. Reference

[1] Malewicz, G., Austern, M. H., Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., & Czajkowski, G. (2010). Pregel: A System for Large-Scale Graph Processing. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 135-146. *https://doi.org/10.1145/1807167.1807184*

[2] Shuvo, M. M. H., Islam, S. K., Cheng, J., & Morshed, B. I. (2023). Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review. *Proceedings of the IEEE*, 111(1), 42-91. https://doi.org/10.1109/JPROC.2022.3226481

[3] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. In *The Web Conference*.