Trilobita: A Distributed Graph Computing System
Checkpoint 3

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 3 report of the course *50.041* group. In this report, We introduce Fault Tolerance in the context of the *Trilobita* system, categorizing faults, outlining their impacts, and stating our final objective. We also present our progress since checkpoint 2. Alongside that, we demonstrate the fault tolerance capabilities of *Trilobita* using two concrete graph processing tasks: *Page Rank* and *Shortest Path Search*.

2.1. Problem Statement

Our problem statement remains the same compared to checkpoints 1 and 2. Overall, we are committed to developing a distributed graph processing system that aligns with the following features:

1. **Fault Tolerance**: the system can gracefully manage failures of both the master and worker servers, enabling a seamless continuity in server failure handling.
2. **Flexibility**: users should have the ability to customize graph processing tasks through user-accessible and user-definable interfaces provided by the system.
3. **Efficiency**: optimizing communication and processes to efficiently handle data transfer across distributed resources.
4. **Consistency**: guarantee a consistent data model and processing logic across the computation and fault-handling process.

3. Fault Tolerance

In this section, we define faults within the *Trilobita* framework and elucidate the consequential impacts they incur. Additionally, We articulate our final objective in fault tolerance for the final stage of the project.

3.1. Definition

In the context of *Trilobita*, a fault is defined as **a*ny unexpected failure of a server in the middle of executing a task.*** Faults can disrupt the normal flow of graph processing tasks and potentially lead to data inconsistencies or processing failures. To ensure robustness and continuity in the face of such faults, *Trilobita* distinguishes between two primary types of failures:

- **Master Failure**: A Master Failure occurs when the central coordinating node, known as the master server, becomes unexpectedly unavailable during the execution of graph processing tasks.
- **Worker Failure**: A Worker Failure refers to the unexpected cessation of failure of one or more worker servers responsible for executing specific parts of the graph distributed by the master.

3.2. Impact of Faults

The impact of faults in *Trilobita* is significant, influencing the overall robustness and continuity of the distributed graph processing system.

- **Master Failure**: The master is responsible for graph partition, superstep coordination, and overall system launching the terminating management. A failure in the master server could impact the system's ability to coordinate supersteps, causing stagnation in the graph processing workflow. Meanwhile, as the master handles faults that occur in workers, it plays a crucial role in the worker failure handling process by redistributing the graph and tasks to live worker servers. In such instances, the system's ability to recover from worker failures may be compromised without the presence of the master.
- **Worker Failure**: Worker servers handle the actual computation and processing of the graph. Failures in worker servers can lead to incomplete or incorrect task execution, and cessation in the ongoing superstep, which disrupts the overall flow of the distributed graph processing system.

3.3. Final Objective in Fault Tolerance

The ultimate objective of incorporating fault tolerance in *Trilobita* is to ensure the system's resilience in the face of faults, thereby promoting continuous and reliable graph processing in a distributed environment. Our final objective is two-fold:

1. **Master Failure Resilience**: *Trilobita* adopts a “backup” approach to handle master failures by introducing multiple replicas into the distributed system. In the event of a master failure, an election process utilizing the bully algorithm is initiated. This process ensures the transition to a designated replica, which assumes the role of the master and takes over the coordination of graph processing tasks.

2. **Worker Failure Resilience**: To address worker failures, *Trilobita* implements a continuous monitoring mechanism at the master server to assess the state of worker servers. Upon detecting a worker failure, the master initiates a graph repartition and task re-execution process.

4. Overall Progress

In the previous checkpoints, we have showcased *Trilobita*'s ability to run in a non-faulty environment. After checkpoint 2, we have done the following work.

1. **Fault Tolerance**:

   Given that our system's primary goal is to craft a system that is able to robust functioning in a fault-prone environment, we implemented multiple mechanisms to gracefully handle faults.

   - ***Heartbeat***: Servers in the system periodically emit their heartbeat signals. The master server actively listens to heartbeats from workers to ensure their liveness. In case any worker fails, the master will initiate a worker-faulty handling process, which repartitions the graph and restarts the superstep based on the most recent snapshot. Meanwhile, heartbeat exchanges occur between the master and its replicas. When the master encounters a failure, a master replica will promptly detect it, and [one replica will be elected to take over the master position]. In addition, when a failed worker rejoins the group, its heartbeat will be recaptured by the master. The master will therefore repartition the graph.

   - ***Master Replica***: Some servers will serve as master replicas in the system. They will not participate in the computation process. In the system, their duty is to detect the master's failure and take over the master's position in case the master is down.

   - ***Snapshot***: Periodically, the master server initiates a request for all workers to transmit their respective local vertex values to the master. After receiving, the master proceeds to update the graph stored within its local memory based on the received vertex values from workers. The updated graph is then encapsulated into a snapshot, which is stored in the master's persistent memory and synchronized across master replicas.

2. **Architecture**:

   We enhanced the system's APIs to improve the usability of the system. Such improvement covers from message transmissions to server initializations.

   - ***TrilobitaEnvironment***: We added an environment component to the system, which integrates the initiation and launch processes of workers and the

master. This enhancement enables users to configure their jobs and start servers by utilizing the APIs provided by *TrilobitaEnvironment*.

3. **Task Examples**:
   To demonstrate the performance, fault-tolerance capability, and adaptivity of our system, we incorporated additional tasks into the implementation.

   - ***Shortest Path Task***: In addition to the *Page Rank Task*, we implemented the *Shortest Path Search* algorithm to further exemplify the functionality of our system.

5. Demonstration

We will demonstrate our system's fault-handling capability at the current stage through two concrete graph-processing tasks, the *Page Rank Task* and the *Shortest Path Task*. See Appendix Fig [1], [2] for more information about these two tasks. We summarize the failure types and their corresponding handling mechanisms at the current stage as follows.

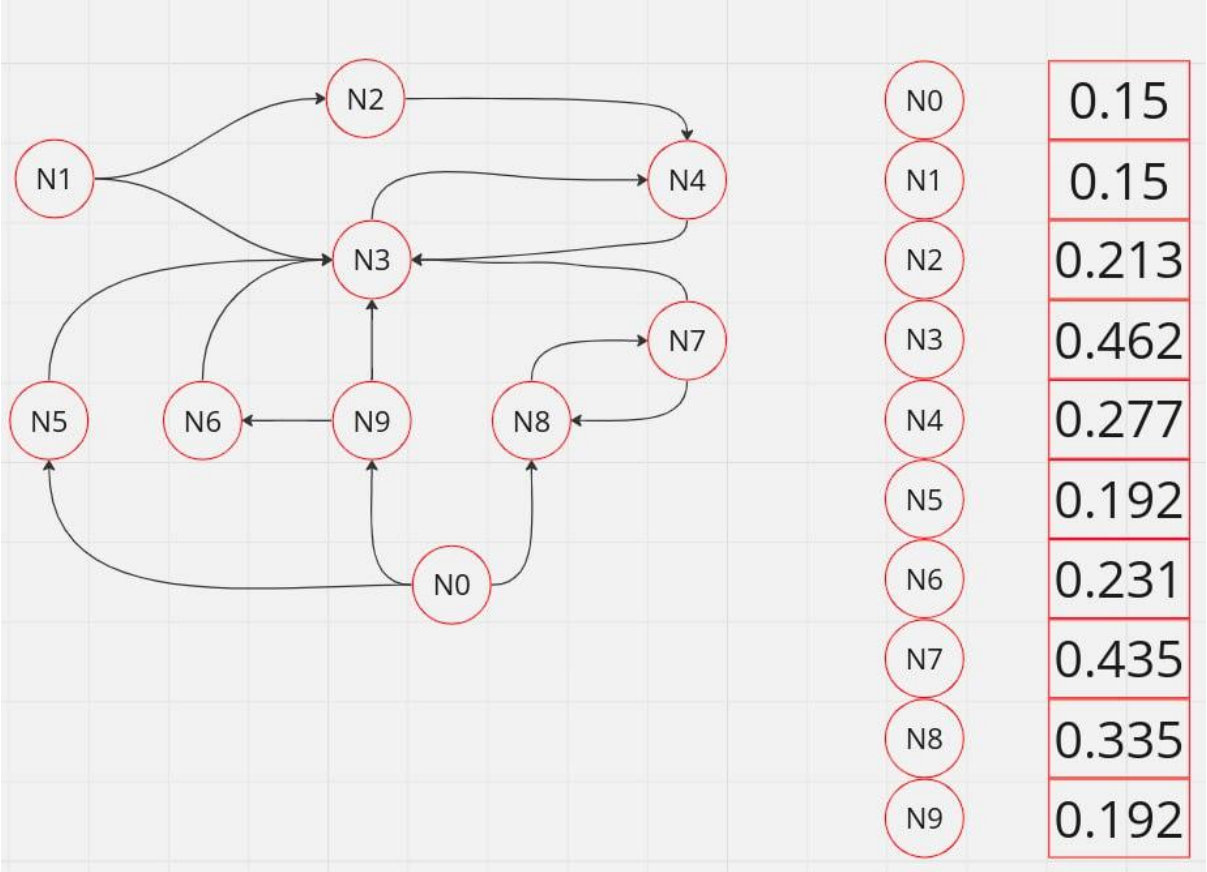
Type of Failure	**Handling Mechanism**
Master Failure	The master will reparation the graph, and restart computing from the most recent snapshot.
Worker Failure	A master replica will become the master, and restart computing from the most recent snapshot.

[Table 1] Fault Types and the Current Handling Mechanism

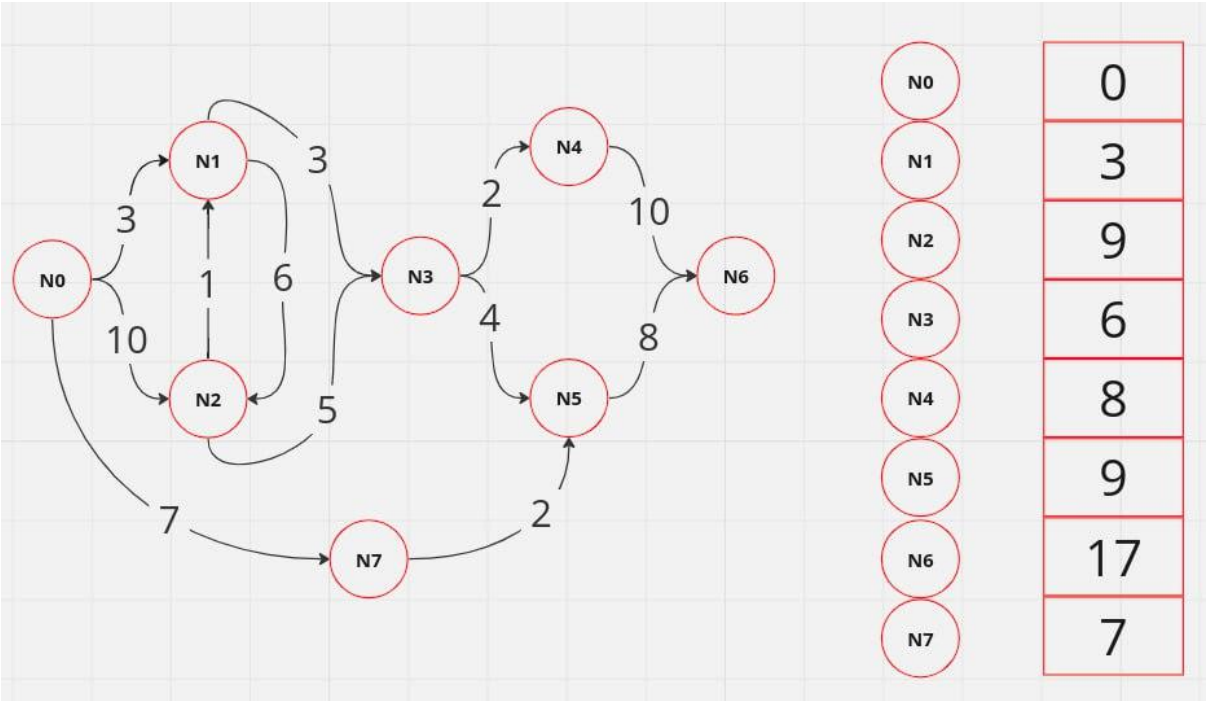
6. Conclusion

In conclusion, this report encapsulates the ongoing progress of the Trilobita distributed graph processing system. Delving into *Fault Tolerance*, we've defined and categorized faults in our system, outlining their impacts and detailing our final objectives to improve the system's resilience. Furthermore, We underscore the *Overall Progress* commitment to fault tolerance, architectural enhancements, and task example implementations. We use a live *Demonstration* to prove the system’s fault-handling capability at the current stage.

Appendix



[Fig 1] PageRank Graph: *The graph shows the structure of the Page Rank Task graph*



[Fig 2] Shortest Path Graph: T*he graph shows the structure of the Shortest Path Task graph*