INSTITUT
POLYTECHNIQUE
DE PARIS

A Single-Writer Multiple-Readers Register in Message-passing

CSC5004

Cloud Computing Infrastructures

Authors:

Nevena Vasilevska
Teodor Cvijovic

Introduction

In this project, we took on the challenge of implementing the ABD algorithm, a nifty solution for enabling different computers to share a Single-Writer Multiple-Readers (SWMR) register within a messaging system. Guided by the principles laid out by Attiya et al., the algorithm provides a smart way for distributed processes to agree on things while allowing one process to write and many to read. This report outlines the step-by-step process, from setting up crucial variables to managing quorum systems and handling the details of sending, receiving, and resolving requests. Thorough validation ensures that the resulting code is robust, well-prepared to handle the intricate aspects of working in distributed systems.

Source code of the implementation can be found in the following address: https://github.com/teodorcvijovic/abd-registry .

01 Initialization

In the initial phase of our implementation, we focused on the vital process of initializing the ABD algorithm within the RegisterImpl class. Essential fields, namely 'value,' 'label,' and 'max,' were seamlessly integrated into the codebase. The open method was meticulously configured to set up these variables, establishing a solid foundation for subsequent operations. To facilitate seamless communication between nodes in our distributed system, we utilized the JGroups library. This enabled effective connectivity and interaction, with the RegisterImpl instance aptly registered as a listener, ensuring responsiveness to the distributed environment's dynamics.

02 Quorum System

In the implementation of our distributed system, the establishment of a robust quorum system played a pivotal role in ensuring the reliability and consistency of operations. The Majority class, a key component of this system, was meticulously completed, incorporating methods to ascertain the size of the majority and select a random majority. Within the RegisterImpl class, we introduced a quorumSystem field, acting as the linchpin for managing quorum-related functionalities. Upon the occurrence of a new view change, a crucial event in our distributed environment, this field was aptly initialized. A new instance of the Majority class was created, leveraging the information from the updated view to dynamically adapt the quorum system. This strategic approach not only enhances the system's resilience but also ensures that it remains well-aligned with the evolving network topology, promoting the consistent and reliable execution of distributed operations.

03 Sending and Replying to Requests

In this stage of the implementation, we focused on refining the mechanisms for handling requests within the RegisterImpl class. The read, write, and execute methods were meticulously coded to ensure smooth execution. A safeguard against improper operations was established by throwing a new IllegalStateException if a client attempted to execute a write operation on a non-writable register. The execute method was further enhanced to facilitate the sending of commands to a quorum of replicas, subsequently awaiting the completion of the future. This meticulous process ensures that the system operates seamlessly in response to client requests while maintaining the integrity of the shared register.

04 ABD Algorithm Logic

A comprehensive understanding of the ABD algorithm's intricacies was crucial in this phase. We delved into the provided pseudo-code, deciphering the underlying logic. Subsequently, the receive method was implemented to handle the algorithm's logic when a request is received. To streamline the process and store responses effectively, a new field named 'replies' was introduced. This field proved instrumental in updating the CompletableFuture when a quorum of responses was received, aligning the algorithm with the distributed nature of our system.

05 Repairing Incomplete Writes

Foreseeing potential challenges, we conducted a thoughtful analysis to identify possible inconsistencies that could arise in scenarios where the writer fails during a write operation. To address these challenges, we implemented the read-repair mechanism of ABD. During the second phase of the read, these stored values played a critical role in repairing incomplete writes, ensuring data consistency in the face of potential failures.

Conclusion

The implemented ABD algorithm allows sharing a register among distributed processes in a message-passing system, ensuring linearizability and handling scenarios such as incomplete writes through a read-repair mechanism. The quorum system employed uses the majority quorum strategy for robustness. The code base provides a foundation for further refinement and extension.