Implémentation et expérimentation du protocole PBFT avec MPI

MOUAD BELKOURI

1. Introduction

Les systèmes distribués modernes sont de plus en plus utilisés dans des contextes critiques tels que les systèmes financiers, les bases de données distribuées, les réseaux blockchain et les infrastructures cloud. Dans ces environnements, la tolérance aux fautes est un enjeu majeur, en particulier face aux fautes dites byzantines, où certains nœuds peuvent se comporter de manière arbitraire ou malveillante.

Le protocole PBFT (Practical Byzantine Fault Tolerance), proposé par Castro et Liskov, constitue une solution efficace permettant d'assurer le consensus dans un système distribué même en présence de fautes byzantines. Contrairement aux approches basées sur la preuve de travail, PBFT offre de bonnes performances et une latence réduite dans des environnements à nombre de nœuds limité.

L'objectif de ce projet est d'implémenter et d'expérimenter le protocole PBFT en utilisant MPI (Message Passing Interface) afin de simuler la communication entre pairs dans un système distribué.

2. Problématique du consensus distribué

Dans un système distribué, le consensus consiste à faire en sorte que tous les nœuds corrects s'accordent sur une même valeur, malgré la présence de défaillances. Le problème devient plus complexe lorsque l'on considère les fautes byzantines, où un nœud peut envoyer des messages incohérents ou incorrects.

Le théorème fondamental de PBFT stipule que pour tolérer f fautes byzantines, le système doit contenir au minimum N = 3f + 1 nœuds. Cette condition garantit que les nœuds corrects sont majoritaires et peuvent imposer un consensus fiable.

3. Présentation du protocole PBFT

Le protocole PBFT repose sur un modèle client–serveur avec un ensemble de replicas, parmi lesquels un nœud est désigné comme primary. Le protocole se déroule en plusieurs phases bien définies :

1. PRE-PREPARE
2. PREPARE
3. COMMIT
4. REPLY

Chaque phase implique des échanges de messages permettant de vérifier la cohérence des requêtes et d'atteindre un consensus.

4. Architecture générale de l'implémentation

Dans cette implémentation, MPI est utilisé comme couche de communication entre les différents processus.

4.1 Répartition des rôles

- Processus 0 : Client
- Processus 1 : Primary
- Processus 2 à N : Replicas (backups)

4.2 Hypothèses

- Vue fixe (view = 0)
- Primary fixe
- Une seule requête client
- Communication fiable assurée par MPI

5. Description détaillée des messages

Plusieurs types de messages sont définis pour implémenter PBFT :

- request_t : requête envoyée par le client
- pre_prepare_t : message PRE-PREPARE envoyé par le primary
- prepare_t : message PREPARE échangé entre replicas
- commit_t : message COMMIT échangé entre replicas
- reply_t : réponse envoyée au client

Chaque message contient les champs nécessaires à la vérification de la cohérence (vue, numéro de séquence, type de requête, identifiant du processus).

6. Déroulement du protocole

6.1 Phase PRE-PREPARE

Le client envoie une requête au primary. Celui-ci attribue un numéro de séquence et diffuse un message PRE-PREPARE à tous les replicas.

6.2 Phase PREPARE

Chaque replica vérifie la validité du message PRE-PREPARE puis diffuse un message PREPARE à l'ensemble des autres replicas. La condition prepared() est satisfaite lorsque au moins 2f messages PREPARE valides sont reçus.

6.3 Phase COMMIT

Après la phase PREPARE, chaque replica envoie un message COMMIT. La condition committed_local() est satisfaite lorsqu'au moins 2f+1 messages COMMIT valides sont reçus.

6.4 Phase REPLY

Une fois la requête exécutée, chaque replica envoie une réponse au client. Le client valide le consensus lorsqu'il reçoit au moins f+1 réponses identiques.

7. Implémentation technique

L'implémentation est réalisée en langage C avec MPI. Des types MPI personnalisés sont définis afin de transmettre efficacement les structures de données.

Les fonctions principales incluent :

- execute() : exécution déterministe de la requête
- prepared() : vérification du quorum PREPARE
- committed_local() : vérification du quorum COMMIT

8. Expérimentations et résultats

8.1 Environnement de test

Les expérimentations ont été réalisées sur une machine Linux (Ubuntu) en utilisant OpenMPI. L'exécution du programme se fait via la commande suivante :

mpirun --oversubscribe -np X ./pbft_mpi

où X représente le nombre total de processus MPI (1 client + N replicas).

8.2 Test avec p = 5 processus (f = 1)

Dans ce scénario, le système contient 1 client et 4 replicas. Selon la formule PBFT N = 3f + 1, le système peut tolérer jusqu'à 1 faute byzantine.

L'exécution montre que :

- Tous les replicas reçoivent correctement le message PRE-PREPARE.
- Les messages PREPARE sont échangés entre les replicas.
- La condition prepared() est satisfaite (au moins 2f messages PREPARE).
- La condition committed_local() est satisfaite (au moins 2f+1 messages COMMIT).
- Chaque replica exécute la requête et envoie un REPLY au client.

Le client reçoit au moins f+1 réponses identiques et valide le consensus avec succès.

8.3 Test avec p = 7 processus (f = 1)

Dans ce test, le nombre de replicas est porté à 6, ce qui augmente la redondance du système.

Les résultats observés confirment que :

- Le protocole reste correct même avec un plus grand nombre de nœuds.
- Les quorums PBFT sont respectés.
- Le consensus est atteint sans ambiguïté.

Ces tests démontrent la scalabilité relative du protocole PBFT pour un nombre modéré de nœuds.

9. Analyse détaillée du code source

9.1 Organisation générale

Le code est structuré autour de trois rôles distincts : client, primary et replicas. Chaque rôle est implémenté par une fonction dédiée.

9.2 Gestion des messages

Les structures C représentent fidèlement les messages PBFT. Des types MPI personnalisés sont créés pour assurer une communication correcte entre processus.

9.3 Vérification des quorums

Les fonctions prepared() et committed_local() implémentent les conditions théoriques du protocole PBFT. Elles garantissent que le consensus ne peut être atteint que si une majorité suffisante de replicas est d'accord.

9.4 Exécution déterministe

La fonction execute() garantit que tous les replicas corrects appliquent la même opération dans le même ordre, assurant ainsi la cohérence de l'état global.

10. Comparaison avec Raft et Paxos

10.1 PBFT vs Paxos

Paxos tolère uniquement les fautes par arrêt (crash faults) et ne prend pas en compte les comportements byzantins. PBFT est donc plus robuste mais aussi plus coûteux en communication.

10.2 PBFT vs Raft

Raft simplifie la compréhension du consensus mais reste limité aux fautes non byzantines. PBFT est mieux adapté aux environnements hostiles.

11. Figures et schémas conceptuels

11.1 Schéma de déroulement du protocole PBFT

Afin de mieux comprendre le fonctionnement du protocole PBFT, il est utile de représenter graphiquement les échanges de messages entre les différents acteurs du système distribué, à savoir le client, le primary et les replicas (backups).

Le schéma de déroulement du protocole PBFT met en évidence les quatre phases principales :

1. PRE-PREPARE
   Le client envoie une requête au primary. Celui-ci attribue un numéro de séquence à la requête et diffuse un message PRE-PREPARE à l'ensemble des replicas. Ce message permet d'annoncer l'ordre d'exécution de la requête.

2. PREPARE
   Après réception et vérification du message PRE-PREPARE, chaque replica diffuse un message PREPARE aux autres replicas. Cette phase permet de vérifier que tous les nœuds corrects sont d'accord sur la requête et son ordre d'exécution.

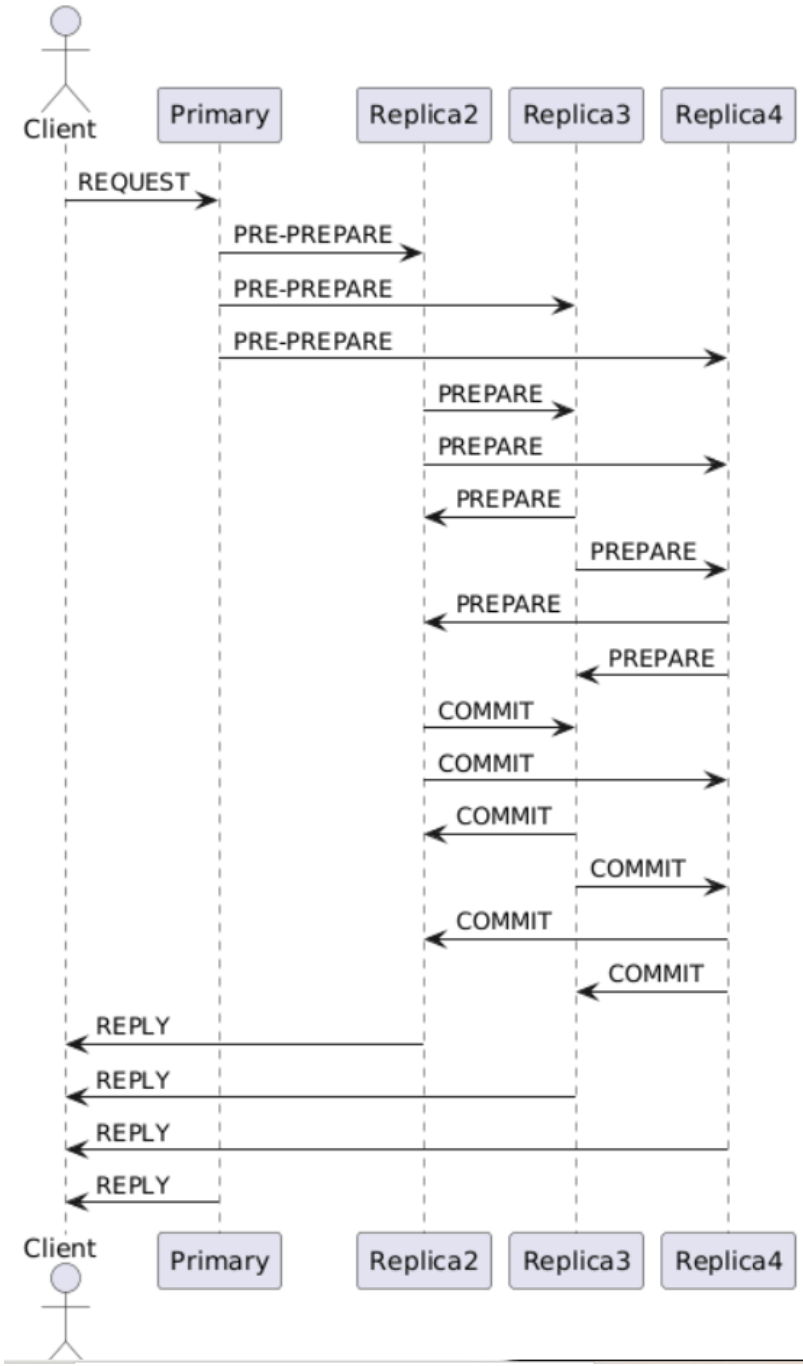
3. COMMIT
   Lorsque la condition de quorum est satisfaite (réception d'au moins 2f messages PREPARE valides), chaque replica envoie un message COMMIT à l'ensemble des autres replicas. Cette phase confirme définitivement la décision.

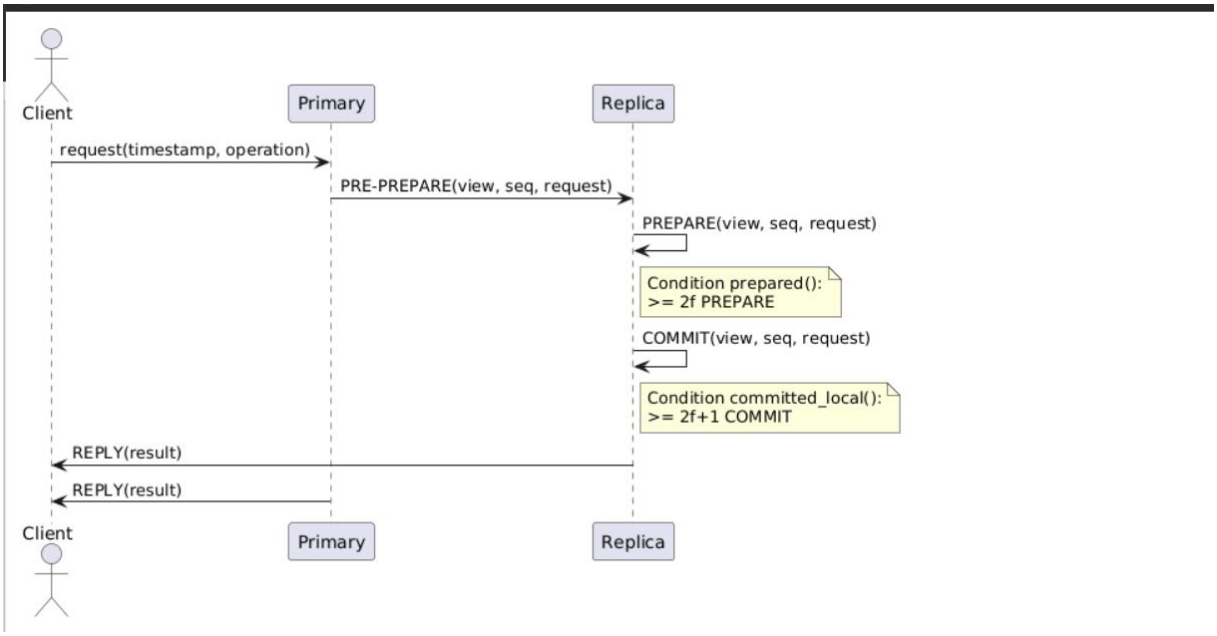
4. REPLY
   Une fois la condition committed_local() satisfaite (au moins 2f+1 messages COMMIT), chaque replica exécute la requête et envoie une réponse au client. Le client valide le consensus lorsqu'il reçoit au moins f+1 réponses identiques.

Ce schéma permet de visualiser clairement le flot des messages et de comprendre comment PBFT garantit la tolérance aux fautes byzantines grâce aux quorums.

11.2 Diagramme UML de séquence

Un diagramme UML de séquence permet de représenter clairement l'ordre des messages échangés lors d'une requête.


Client
Primary
Replica2
Replica3
Replica4
REQUEST
PRE-PREPARE
PRE-PREPARE
PRE-PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
REPLY
REPLY
REPLY
REPLY
Client
Primary
Replica2
Replica3
Replica4



12. Limites et perspectives

Cette implémentation reste volontairement simplifiée. Les perspectives d'amélioration incluent l'implémentation du view-change, l'ajout de signatures cryptographiques et la simulation de nœuds byzantins réels.

Analyse critique et limites

Cette implémentation est volontairement simplifiée à des fins pédagogiques. Elle ne prend pas en compte :

- le mécanisme de view-change,
- les signatures cryptographiques,
- les nœuds réellement byzantins,
- la gestion de multiples requêtes concurrentes.

Malgré ces limites, elle permet de comprendre et d'expérimenter les mécanismes fondamentaux de PBFT.

Conclusion :

Ce projet a permis d'implémenter avec succès le protocole **PBFT (Practical Byzantine Fault Tolerance)** en s'appuyant sur **MPI comme infrastructure de communication distribuée**, permettant ainsi de simuler un environnement de communication entre pairs. L'architecture mise en place, fondée sur un modèle client–primary–replicas, respecte fidèlement les principes théoriques du protocole PBFT et met en œuvre l'ensemble de ses phases fondamentales : **PRE-PREPARE, PREPARE, COMMIT et REPLY**.

Les expérimentations réalisées avec différents nombres de processus ont démontré que le système atteint correctement le consensus, même en présence de fautes tolérées, conformément au modèle **N = 3f + 1**. Les résultats observés confirment que les mécanismes de quorum assurent la cohérence globale de l'état du système et garantissent que seuls les résultats validés par une majorité suffisante de replicas corrects sont acceptés par le client. Cette validation expérimentale met en évidence la robustesse et la fiabilité du protocole face aux défaillances potentielles.

Au-delà de l'implémentation, ce projet a permis d'approfondir la compréhension des problématiques liées au consensus distribué et à la tolérance aux fautes byzantines, en mettant en lumière les compromis entre performance, complexité et sécurité. Bien que volontairement simplifiée à des fins pédagogiques, cette implémentation constitue une **base solide et extensible** pour des travaux futurs.

Des perspectives d'amélioration naturelles incluent l'implémentation du **view-change** afin de gérer la défaillance du primary, l'intégration de **mécanismes cryptographiques** pour l'authentification des messages, ainsi que la simulation de **nœuds byzantins réels** et de requêtes concurrentes. Ces extensions permettraient de se rapprocher d'un système PBFT pleinement opérationnel et applicable à des environnements distribués réels et critiques.