So sánh Thuật toán Đồng thuận RAFT và pBFT

Giới thiệu

Trong lĩnh vực hệ thống phân tán, việc đạt được sự đồng thuận giữa các nút là một yếu tố quan trọng để đảm bảo tính nhất quán và độ tin cậy của dữ liệu. Hai thuật toán đồng thuận phổ biến được sử dụng rộng rãi là RAFT và pBFT (Practical Byzantine Fault Tolerance). Nghiên cứu này sẽ đi sâu vào phân tích hai thuật toán này, so sánh ưu nhược điểm của chúng và đánh giá hiệu quả trong các trường hợp cụ thể.

Thuật toán RAFT

Tổng quan về RAFT

RAFT là một thuật toán đồng thuận được thiết kế để dễ hiểu và triển khai hơn so với các thuật toán thuộc họ Paxos [1]. Thuật toán này hoạt động dựa trên việc bầu chọn một nút leader chịu trách nhiệm nhận yêu cầu từ client và sao chép dữ liệu đến các nút follower khác trong hệ thống [2].

Để hiểu rõ hơn về RAFT, cần nắm được khái niệm "state machine". Mỗi nút trong hệ thống RAFT duy trì một state machine, nhận đầu vào là các lệnh từ log. Log này chứa các lệnh như "set x to 3" [4]. Thuật toán RAFT đảm bảo rằng nếu một state machine áp dụng lệnh "set x to 3" là lệnh thứ n, thì không có state machine nào khác áp dụng một lệnh thứ n khác. Điều này đảm bảo tính nhất quán mạnh mẽ giữa các nút trong hệ thống, ngay cả khi một số nút bị lỗi.

Cách thức hoạt động

RAFT hoạt động theo các nguyên tắc sau:

- **Bầu cử Leader:** Khi hệ thống khởi động hoặc leader hiện tại bị lỗi, các nút sẽ chuyển sang trạng thái candidate và bắt đầu quá trình bầu cử. Mỗi candidate sẽ gửi yêu cầu vote đến các nút khác. Nút nhận được đa số vote sẽ trở thành leader mới.
  - Để đảm bảo tính nhất quán trong quá trình chuyển đổi leader, RAFT sử dụng cơ chế so sánh term và index của các entry cuối cùng trong log. Log có term mới hơn sẽ được ưu tiên. Nếu term giống nhau, log dài hơn sẽ được coi là mới hơn [1].
  - Các "term number" trong RAFT đóng vai trò như đồng hồ logic, giúp quản lý việc bầu cử và duy trì tính nhất quán. Mỗi term bắt đầu bằng một cuộc bầu cử. Nếu một leader được bầu, nó sẽ duy trì vai trò cho đến khi term kết thúc hoặc nó gặp sự cố. Term number tăng dần theo thời gian và được sử dụng để xác định leader hiện tại và ngăn chặn các xung đột [5].
- **Sao chép Log:** Leader chịu trách nhiệm nhận các yêu cầu từ client và ghi chúng vào log của

mình. Sau đó, leader sẽ sao chép log này đến các follower.
- **Cam kết Log:** Khi một entry trong log được sao chép đến đa số các nút, entry đó sẽ được coi là committed.
    - "Commit index" được sử dụng để theo dõi tiến trình sao chép log và đảm bảo các cập nhật được áp dụng theo đúng thứ tự. Mỗi khi một entry trong log được committed, commit index sẽ tăng lên, và trạng thái chung của hệ thống được cập nhật để phản ánh các thay đổi do entry đó tạo ra [6]. Leader sẽ thông báo cho các follower về các entry đã được committed, và các follower sẽ áp dụng các entry này vào state machine của chúng.

Ưu điểm của RAFT

- **Dễ hiểu:** RAFT được thiết kế với mục tiêu dễ hiểu hơn so với Paxos [1], giúp việc triển khai và gỡ lỗi dễ dàng hơn.
- **Hiệu quả:** RAFT có hiệu suất cao và độ trễ thấp, phù hợp với nhiều ứng dụng thực tế.
- **An toàn:** RAFT đảm bảo tính nhất quán mạnh mẽ và khả năng chịu lỗi cao.
- **Hiệu suất đọc cao:** RAFT sử dụng "leader leases" để đạt được hiệu suất đọc cao với khả năng tuyến tính hóa đơn key. Leader lease cho phép leader phục vụ các yêu cầu đọc mà không cần phải tham khảo ý kiến của các follower, giúp giảm độ trễ và tăng thông lượng [3].

Hạn chế của RAFT

- **Khả năng chịu lỗi Byzantine:** RAFT được thiết kế để chịu lỗi crash, tức là các nút chỉ có thể dừng hoạt động. Thuật toán này ban đầu không xử lý được các hành vi độc hại (Byzantine failures) [7]. Tuy nhiên, các nghiên cứu đang được tiến hành để điều chỉnh RAFT cho các trường hợp lỗi Byzantine.

Thuật toán pBFT

Tổng quan về pBFT

pBFT (Practical Byzantine Fault Tolerance) là một thuật toán đồng thuận được thiết kế để hoạt động hiệu quả trong các hệ thống asynchronous và được tối ưu hóa cho độ trễ thấp [8]. Thuật toán này có khả năng chịu lỗi Byzantine, tức là có thể xử lý các nút có hành vi độc hại, chẳng hạn như giả mạo thông tin [10]. pBFT sử dụng các thuật toán mã hóa như chữ ký, xác minh chữ ký và hàm băm để đảm bảo tính bất biến, không thể giả mạo và không thể tranh cãi của thông tin [10].

pBFT giả định một mạng lưới gồm N nút validator, với ngưỡng chịu lỗi là F, có nghĩa là nó có thể chịu đựng tối đa F nút bị lỗi [11]. pBFT được sử dụng trong nhiều ứng dụng yêu cầu tính an toàn cao, chẳng hạn như blockchain [11].

Cách thức hoạt động

pBFT hoạt động theo ba giai đoạn chính:

- **Pre-prepare:** Nút primary (leader) được chọn thông qua quy trình "round-robin", đảm bảo phân phối quyền lực cân bằng và ngăn chặn điểm lỗi duy nhất [12]. Nút primary đề xuất một yêu cầu mới và gửi nó đến các nút khác.
- **Prepare:** Các nút nhận được yêu cầu sẽ kiểm tra tính hợp lệ của nó và gửi thông báo prepare đến các nút khác.
- **Commit:** Khi một nút nhận được đủ thông báo prepare, nó sẽ gửi thông báo commit đến các nút khác. Khi một nút nhận được đủ thông báo commit, nó sẽ thực thi yêu cầu.

Ưu điểm của pBFT

- **Chịu lỗi Byzantine:** pBFT có khả năng chịu lỗi Byzantine, giúp hệ thống hoạt động ổn định ngay cả khi có một số nút có hành vi độc hại.
- **Hiệu quả:** pBFT được tối ưu hóa để có độ trễ thấp [8], phù hợp với các ứng dụng yêu cầu thời gian đáp ứng nhanh.
- **Hiệu quả năng lượng:** pBFT có thể đạt được sự đồng thuận phân tán mà không cần thực hiện các phép tính toán học phức tạp (như trong PoW) [13].

Hạn chế của pBFT

- **Độ phức tạp:** pBFT có độ phức tạp cao hơn so với RAFT, khiến việc triển khai và gỡ lỗi khó khăn hơn.
- **Khả năng mở rộng:** pBFT không mở rộng tốt với số lượng nút lớn do chi phí truyền thông tăng cao.
- **Lỗ hổng bảo mật:** pBFT có thể gặp một số lỗ hổng bảo mật, bao gồm tấn công từ chối dịch vụ, tấn công Sybil và tấn công giả mạo [12].

So sánh RAFT và pBFT

Tiêu chí	RAFT	pBFT
Loại lỗi	Crash failures	Byzantine failures
Độ phức tạp	Thấp	Cao
Khả năng mở rộng	Tốt	Không tốt
Hiệu suất	Cao	Cao

Tiêu chí	RAFT	pBFT
Độ trễ	Thấp	Thấp
Độ phức tạp truyền thông	Thấp	Cao
Bảo mật	Chịu lỗi crash	Chịu lỗi Byzantine
Ứng dụng	Hệ thống phân tán yêu cầu tính nhất quán mạnh mẽ, dễ triển khai	Hệ thống yêu cầu tính an toàn cao, chịu lỗi Byzantine

Phân tích chi tiết:

- **Cách thức hoạt động:** RAFT dựa trên việc bầu cử leader và sao chép log, trong khi pBFT sử dụng cơ chế voting ba giai đoạn.
- **Ưu điểm:** RAFT dễ hiểu và triển khai hơn, trong khi pBFT có khả năng chịu lỗi Byzantine.
- **Nhược điểm:** RAFT không chịu lỗi Byzantine (tuy nhiên, các nghiên cứu đang được tiến hành để giải quyết vấn đề này), trong khi pBFT có độ phức tạp cao và khả năng mở rộng kém.
- **Số lượng nút:** RAFT hoạt động hiệu quả với số lượng nút lớn, trong khi pBFT phù hợp hơn với hệ thống có số lượng nút nhỏ.
- **Trường hợp lỗi:** RAFT phù hợp với các hệ thống mà lỗi crash là phổ biến, trong khi pBFT phù hợp với các hệ thống yêu cầu khả năng chịu lỗi Byzantine.
- **Tendermint (một biến thể của pBFT) đơn giản hóa quá trình thay đổi view so với pBFT truyền thống, giúp tăng hiệu quả hoạt động [14].**

Ví dụ:

- RAFT được sử dụng trong các hệ thống lưu trữ phân tán như etcd và Consul.
- pBFT được sử dụng trong các hệ thống blockchain như Hyperledger Fabric.

So sánh đơn giản và khả năng chịu lỗi Byzantine:

Một điểm quan trọng cần lưu ý là RAFT ưu tiên tính dễ hiểu và triển khai, trong khi pBFT tập trung vào khả năng chịu lỗi Byzantine. Sự đánh đổi này ảnh hưởng đến việc lựa chọn thuật toán cho các ứng dụng cụ thể[7].

Kết luận

Cả RAFT và pBFT đều là những thuật toán đồng thuận hiệu quả, mỗi thuật toán có những ưu điểm và nhược điểm riêng. Việc lựa chọn thuật toán phù hợp phụ thuộc vào yêu cầu cụ thể của hệ thống, bao gồm loại lỗi cần xử lý, độ phức tạp, khả năng mở rộng và hiệu suất.

Đối với các ứng dụng ưu tiên khả năng mở rộng và dễ dàng triển khai khi có lỗi crash, RAFT là

lựa chọn được khuyến nghị. Tuy nhiên, khi khả năng chịu lỗi Byzantine là quan trọng nhất, pBFT, mặc dù phức tạp, nên được xem xét, đặc biệt là trong các hệ thống có số lượng nút hạn chế[7].

Nguồn trích dẫn

1. Raft (algorithm) - Wikipedia, truy cập vào tháng 1 15, 2025, https://en.wikipedia.org/wiki/Raft_(algorithm)
2. www.yugabyte.com, truy cập vào tháng 1 15, 2025, https://www.yugabyte.com/tech/raft-consensus-algorithm/#:~:text=Raft%20is%20a%20consensus%20algorithm,%2C%20strongly%2Dconsistent%20distributed%20systems.
3. What is the Raft Consensus Algorithm? - Yugabyte, truy cập vào tháng 1 15, 2025, https://www.yugabyte.com/tech/raft-consensus-algorithm/
4. Raft Consensus Algorithm, truy cập vào tháng 1 15, 2025, https://raft.github.io/
5. Raft Consensus Algorithm: Mastering Distributed Systems - Mindbowser, truy cập vào tháng 1 15, 2025, https://www.mindbowser.com/raft-consensus-algorithm-explained/
6. The Raft Algorithm: Achieving Distributed Systems Consensus | by Radovan Stevanovic | Coinmonks | Medium, truy cập vào tháng 1 15, 2025, https://medium.com/coinmonks/the-raft-algorithm-achieving-distributed-systems-consensus-e8c17542699b
7. Performance Analysis and Comparison of Non-ideal Wireless PBFT and RAFT Consensus Networks in 6G Communications - arXiv, truy cập vào tháng 1 15, 2025, https://arxiv.org/pdf/2304.08697
8. www.geeksforgeeks.org, truy cập vào tháng 1 15, 2025, https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/#:~:text=Practical%20Byzantine%20Fault%20Tolerance%20is,optimized%20for%20low%20overhead%20time.
9. Practical Byzantine Fault Tolerance (PBFT) - Crypto.com, truy cập vào tháng 1 15, 2025, https://www.crypto.com/glossary/practical-byzantine-fault-tolerance-pbft
10. An Introduction to PBFT Consensus Algorithm | by TRON Core Devs - Medium, truy cập vào tháng 1 15, 2025, https://medium.com/tronnetwork/an-introduction-to-pbft-consensus-algorithm-11cbd90aaec
11. PBFT, IBFT and QBFT Consensus Algorithms Explained - The Web3 Labs Blog, truy cập vào tháng 1 15, 2025, https://blog.web3labs.com/web3development/comparing-byzantine-fault-tolerance-consensus-algorithms
12. Practical Byzantine Fault Tolerance: Full Details - Tectum.io, truy cập vào tháng 1 15, 2025, https://tectum.io/blog/practical-byzantine-fault-tolerance/
13. practical Byzantine Fault Tolerance(pBFT) - GeeksforGeeks, truy cập vào tháng 1 15, 2025, https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/
14. Compared with traditional PBFT, what advantage does Tendermint algorithm has? Technical guys, here here! : r/cosmosnetwork - Reddit, truy cập vào tháng 1 15, 2025, https://www.reddit.com/r/cosmosnetwork/comments/8i42qa/compared_with_traditional_pbft_what_advantage/
15. Performance comparison of Raft, Pbft, and PoW. - ResearchGate, truy cập vào tháng 1 15, 2025, https://www.researchgate.net/figure/Performance-comparison-of-Raft-Pbft-and-PoW_tbl1_364567131