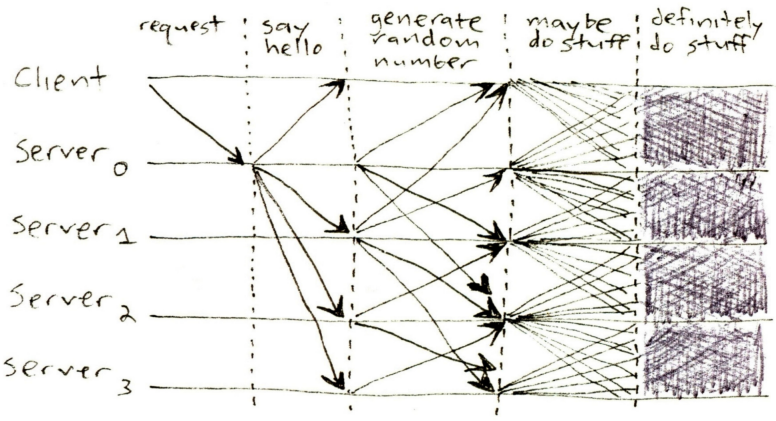
Attaque ou amélioration ou déploiement, de protocoles distribués de type “blockchain”



JAMES: I announce my desire to go to lunch.
BRYAN: I verify that I heard that you want to go to lunch.
RICH: I also verify that I heard that you want to go to lunch.
CHRIS: YOU DO NOT WANT TO GO TO LUNCH.
JAMES: OH NO. LET ME TELL YOU AGAIN THAT I WANT TO GO TO LUNCH.
CHRIS: YOU DO NOT WANT TO GO TO LUNCH.
BRYAN: CHRIS IS FAULTY.
CHRIS: CHRIS IS NOT FAULTY.
RICH: I VERIFY THAT BRYAN SAYS THAT CHRIS IS FAULTY.
BRYAN: I VERIFY MY VERIFICATION OF MY CLAIM THAT RICH CLAIMS THAT I KNOW CHRIS.
JAMES: I AM SO HUNGRY.

Credits: James Mickens “the saddest moment”

Encadrant: Matthieu Rambaud (dept INFRES)
Nombre d’étudiant·es minimum dans chaque instance de ce projet: 2
Nombre d’étudiant·es maximum dans chaque instance de ce projet: 6
Combien d’instances de ce projet proposez-vous ? 3
Tags : distributed algorithms, event-based, Byzantine consensus, digital signatures

1 Contexte

Une blockchain peut se voir comme une base de données :

- accessible à tous en lecture et en écriture, organisée sous la forme d’une chaîne ordonnée de “blocks” de taille fixe: $B_0 \leftarrow B_1 \leftarrow ...B_r$;
- telle que les opérations d’écriture sont seulement de type “append un nouveau block”: $B_n \leftarrow B_{r+1}$;
- et qui offre la garantie très importante suivante: si Alice et Bob font chacun une opération de lecture, alors l’output de l’un: $B_0 \leftarrow B_1 \leftarrow ... \leftarrow B_i$ est nécessairement un préfixe de l’output de l’autre: $B_0 \leftarrow B_1 \leftarrow ...B_i \leftarrow B_{i+1} \leftarrow ... \leftarrow B_j$. Autrement dit, il n’y a pas d’historiques conflictuels.

Pour l’implémenter malgré des participants corrompus, on utilise des protocoles distribués appelés “consensus Byzantins”. Ils sont exécutés par un certain nombre n de machines, appelées “participants”. Ils garantissent que la base de données fonctionne de façon correcte même si une fraction $< n/3$ des participants sont contrôlés par l’adversaire, et même si les délais de communication entre participants sont contrôlés par l’adversaire. La conception et l’analyse d’un protocole de consensus demande d’abondonner toute confiance en son prochain. Un Charlie corrompu pourrait très bien donner deux versions différentes de la vérité à Alice et Bob, qui ne se parlent pas entre eux à cause des délais de communication, et donc se soupçonnent l’un-l’autre d’être corrompus. C’est pourquoi, pour résumer grossièrement, chaque message de chaque Charlie doit être répété à tous, par $> 2n/3$ participants, pour être réputé avoir été “dit” (et prouver que Charlie n’a pas “dit” son contraire). Présenté comme cela, la complexité des communications semble en $\Omega(n^3)$, sans compter que tous les messages s’accompagnent de preuves cryptographiques de type “digital signatures”. J’ai (votre futur encadrant) publié un consensus en $O(n)$ [Ram20; AAR21] (vidéo plus longue [Ram21]). Je viens également de décrire plusieurs attaques qui invalident un protocole de consensus publié par des anciens de la blockchain Diem [Tea21] de Facebook. Les auteurs l’ont confirmée mi-octobre, je la rendrai publique dans les prochains jours. Cette attaque s’accompagne d’une réparation, dont j’ai l’orgueil d’estimer qu’elle est même plus rapide que l’état de l’art des consensus (dans des conditions d’utilisation défavorables).

2 Attendus du projet

Le sujet est découpable en plusieurs projets voire sous-projets, donc ce sera sur-mesure suivant les préférences.

Theoretical track (workload for 2 students) Les élèves souhaitant faire de la théorie pourront être évalués sur leur présentation de l'algorithme PBFT [CL99], et sur le travail d'archéologie suivant du consensus Byzantin. Il s'agit de comparer PBFT à l'optimisation [CL02] incomprise et oubliée, aux variantes sans signature de la thèse de Castro [Cas01] et de Cachin [Cac11], ainsi qu'au retour de Lamport par la fenêtre [Lam11].

Attack track (workload for 3 students) Il s'agit d'utiliser l'implémentation (en Rust) du protocole attaqué (une variante se trouve ici [Xia21]), de la faire exécuter par 4 ou 7 machines, et de reproduire une ou deux attaques en jouant le rôle de l'adversaire. La difficulté consistera à orchestrer l'ordre dans lequel les messages arrivent (quitte à tous les ralentir énormément), pour que les participants honnêtes aient le comportement malheureux prédit.

Large scale deployment track (workload for 4 students) Il s'agit de reproduire les tests de performance: au choix ou bien du consensus [GKS+22] en utilisant leur code source [Xia21; Son21]; ou bien du consensus [ZDZZ23] en utilisant leur code source [DZ23]. Le premier est en Rust et le deuxième en Go, cependant le travail se consentre sur les scripts pour lancer les tests, qui sont dans tous les cas en Python. Il est attendu un déploiement sur trois types de réseaux. Tout d'abord sur quelques serveurs AWS ou Amazon EC2, en utilisant les scripts des auteurs. Puis, sur le réseau de tests de blockchains d'EDF (situé à 400m de l'école), ou comme solution de secours sur des VPS. Enfin, sur le logiciel de simulation de consensus [BTR23]. Il sera intéressant de voir jusqu'à quelle échelle ce dernier se comporte comme les expérimentations réelles.

Implementations of Improvements tracks (workload for 3, 4 and 5 students) Il y a trois sous-projets d'implémentation possible, le premier étant le plus simple. Chacun peut donner lieu à la co-publication d'un article de recherche, s'il s'accompagne de tests de performance (donc en collaborant avec le large-scale deployment track). Les trois sous-projets se feraient naturellement sur la base d'implémentations en Rust existantes de consensus proches. Mais il est également possible, au contraire, de les implémenter sur la nouvelle plateforme en Go [DZ23], vendue comme plus flexible. Ce serait par contre en construisant sur la base d'un consensus plus éloigné [ZDZZ23].

Le 1er sous-projet (charge de travail pour 3 élèves) serait d'implémenter deux algorithmes de consensus simples publiés dans [Ram20; AAR21], qui s'obtiennent comme variantes de l'implémentation en Rust [Son21].

Le 2e sous-projet (charge de travail pour 4 élèves) serait d'implémenter deux algorithmes. Tout d'abord ma réparation, puis l'état de l'art [GZT+22], tous les deux s'obtenant comme simples variantes de l'implémentation en Rust [Xia21].

Le 3e sous-projet (charge de travail pour 5 élèves) consisterait à implémenter une réparation et amélioration du consensus [BKL+23], qui vient d'être présenté il y a quelques semaines à la meilleure conférence en sécurité. Ce sous-projet chevauche une parties des deux précédents.

References

[AAR21] M. Abspoel, T. Attema, and M. Rambaud. "Brief Announcement: Malicious Security Comes for Free in Consensus with Leaders". In: *PODC*. 7/2021.

[BKL+23] E. Blum, J. Katz, J. Loss, K. Nayak, and S. Ochsenreither. "Abraxas: Throughput-Efficient Hybrid Asynchronous Consensus". In: *CCS*. 2023.

[BTR23] C. Berger, S. B. Toumia, and H. P. Reiser. "Scalable Performance Evaluation of Byzantine Fault-Tolerant Systems Using Network Simulation". In: 2023.

[Cac11] C. Cachin. *Yet Another Visit to Paxos*. https://cachin.com/cc/papers/pax.pdf. 2011.

[Cas01] M. Castro. "Practical Byzantine Fault Tolerance". PhD thesis. MIT, 2001. URL: %5Curl%7Bhttps://www.microsoft.com/en-us/research/wp-content/uploads/2017/01/thesis-mcastro.pdf%7D.

[CL02] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Trans. Comput. Syst.* (2002).

[CL99] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance". In: *OSDI*. 1999.

[DZ23] S. Duan and B. Zhao. *Waterbear BFT platform*. https://github.com/fififish/waterbear. 2023.

[GKS+22] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang. "Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback". In: *FC*. we refer to the 18 June 2021 version on arxiv. 2022.

[GZT+22] B. Guo, Y. L. and Zhenliang Lu, Q. Tang, J. Xu, and Z. Zhang. "Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice". In: *NDSS*. 2022.

[Lam11] L. Lamport. "Byzantizing Paxos by Refinement". In: *DISC*. 2011.

[Ram20] M. Rambaud. "(v1 2020-11-29) Malicious Security Comes for Free in Consensus with Leaders". In: *eprint 2020/1480* (2020).

[Ram21] M. Rambaud. *Maliciously secure consensus from succinct arguments, and short transparent threshold signatures*. https://www.youtube.com/watch?v=y3o34Mf-BhQ. 2021.

[Son21] A. Sonnino. *Implementation of Jolteon*. https://github.com/asonnino/hotstuff/. 2021.

[Tea21] T. D. Team. *DiemBFT v4: State Machine Replication in the Diem Blockchain*. https://developers.diem.com/pa
diem-consensus-state-machine-replication-in-the-diem-blockchain/2021-08-17.pdf. 2021.

[Xia21] Z. Xiang. *Implementation of the fallback of Ditto*. github link to the file fallback.rs. 2021.

[ZDZZ23] H. Zhang, S. Duan, B. Zhao, and L. Zhu. "WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT". In: *USENIX Security*. 2023.