Predictive Placement of Geo-distributed Blockchain Nodes for Performance Guarantee

Junseok Lee*, Yeonho Yoo*, Chuck Yoo, Gyeongsik Yang
Department of Computer Science and Engineering, Korea University
{jslee, yhyoo, chuckyoo}@os.korea.ac.kr, g_yang@korea.ac.kr

***Abstract*—Blockchain-as-a-service (BaaS) in cloud datacenters is gaining widespread attention due to its high performance and privacy. However, existing BaaS solutions lack a method for deciding the proper placement of blockchain nodes across virtual machines in worldwide datacenters to achieve desired performance. Our motivating experiments show that transaction processing performance (TPS) varies ∼31.6% depending on the placements. To provide an automatic placement solution for BaaS, we propose *Cyan* that predicts the TPS for blockchain node placements. Our evaluations on Google Cloud Platform demonstrate that *Cyan* improves the TPS guarantee ∼2.39× compared to existing techniques.**

***Index Terms*—Blockchain-as-a-service, Hyperledger Fabric, Node Placement, Performance Guarantee**

I. Introduction

Recently, blockchain-as-a-service (BaaS) that operates in cloud datacenters has received widespread attention. It allows only pre-permitted and trusted participants to join the blockchain system, which provides higher security and privacy compared to public blockchains that permit anyone to join. Also, BaaS can scale to a larger amount of data (transactions) for large-scale enterprises [1] with the fluent computing infrastructure of datacenters. The most widely used platform for BaaS is Hyperledger Fabric [2] that is maintained by Linux Foundation. For example, BaaS providers, such as Google Cloud Platform and IBM Cloud, run Hyperledger Fabric for their BaaS offerings.

To initialize BaaS, system operators first create a virtual machine (VM) within cloud datacenters. The location of the VM is decided based on where the blockchain nodes are intended to be deployed. For example, if BaaS is to be deployed and run across the United States, Asia, and Europe, VMs to deploy the nodes are created in a datacenter in the United States, one in Asia, and another in Europe. On the created VMs, the system operators create blockchain nodes and operate the BaaS service to enter transactions, validate transactions, and maintain integrity on the validated transactions in the form of ledger across blockchain nodes [3].

*Junseok Lee and Yeonho Yoo are co-first authors. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A6A1A13044830), by the NRF grant funded by the Korea government (MSIT) (NRF-2023R1A2C3004145, RS-2024-00336564), by ICT Creative Consilience Program through the Institute of Information & Communications Technology Planning & Evaluation grant funded by the Korea government (MSIT) (IITP-2024-2020-0-01819), and by the Google Cloud Research Credits. Corresponding authors: Chuck Yoo and Gyeongsik Yang

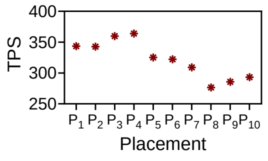
Considering the advantages of transaction consistency across blockchain nodes, significant efforts are dedicated to enhancing Hyperledger Fabric, such as analyzing transaction processing performance, ensuring high fairness, and increasing reliability [4]. Despite diverse efforts, the practical use of BaaS in real-world systems remains puzzling due to the black-box nature of blockchain node placement.

In our motivating experiments, the current black-box-based placement approach poses a significant problem. We run three different VMs on the Google Cloud Platform across datacenters in the United States, Europe, and Asia. Depending on the number of blockchain nodes per VM (placement strategy), we observe that the number of transactions processed by the BaaS per second (TPS) varied by ∼31.6% (details in §II-B). Several studies automated the creation of the nodes on VMs [5], but to our knowledge, the decision on proper placement to gain specific TPS remains unanswered. Due to a lack of options, operators of BaaS systems should rely on a time-consuming trial-and-error approach to find the placement for stable and reliable blockchain services in the cloud.

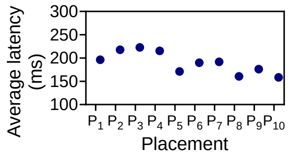
Therefore, this study proposes *Cyan,* a new blockchain node placement technique for guaranteeing desired BaaS performance. Our motivating experiments demonstrate that the appropriate blockchain node placement for achieving the specific desired TPS cannot be manually or statistically modeled by a single factor, such as networking bottlenecks between VMs. Thus, we apply a machine learning (ML) technique to find the proper placement that guarantees the desired TPS.

To design an ML model, we require a dataset of blockchain node placements and their performance to train it. However, to the best of our knowledge, such a dataset is lacking. Thus, we have developed a placement dataset generation tool for the target BaaS system. We carefully select input and output features to accurately detect the relationship between the placement and its TPS. Furthermore, we test various ML algorithms (e.g., random forest, gradient boosting, deep neural network, and support vector machine) and use the one that shows the best prediction accuracy. Using the prediction model, *Cyan* determines the proper blockchain node placement in advance of the real system running and creates the blockchain nodes according to the placement.

We fully implement *Cyan* that runs with real-world datacenters. Our evaluations are conducted using Google Cloud Platform and demonstrate ∼2.39× improvements in TPS guarantee compared to existing techniques.



(a) TPS per placement.



(b) Average latency.

Fig. 1: Motivation experiments in Google Cloud Platform.

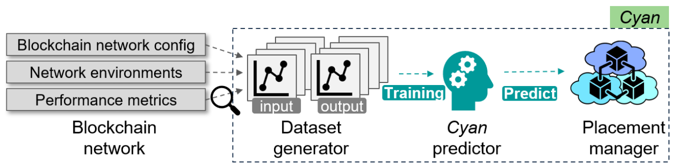


Fig. 2: *Cyan* architecture

TABLE I: BaaS configurations.

Feature	Example	Range
# of VMs	3	2–10
# of peer nodes	8	2–10
Latency between VMs	30ms, 100 ms, 250 ms	0.1–350
Type of chaincode operation	`put`	`put`, `get`
Average transaction size	350	20–19K
Transaction sending rate	500 transactions/s	300–800
Placement strategy	(3, 1, 4)	All possible

II. Background and Motivation

A. Background: Hyperledger Fabric

Hyperledger Fabric consists of two kinds of nodes: peers and orderers. A peer is the main node that receives data (transaction) in the blockchain network. The peer node also runs a program (chaincode), such as data retrieval, on the received transactions and validates them. An orderer is responsible for arranging the order of validated transactions in peer nodes using a consensus protocol, such as Kafka, PBFT, or Raft, and generates a block, which is stored in the ledger. The ledger is replicated across every peer; thus, every peer maintains validated transactions with integrity and consistency. All peer nodes receive, validate, and commit transactions through communications between each other. In general, orderer nodes are created in smaller numbers than peer nodes, which require less complexity in placement [6]. In this work-in-progress study, we focus on the placement of peer nodes that can significantly increase according to the number of users and data maintainers of BaaS.

B. Motivating Experiment

Experiment setup. We use Google Cloud Platform for experiments. We create and run three different VMs in the United States, Europe, and Asia datacenters, respectively. Specifically, each VM is provisioned as an e2-standard-8 instance at US-east, Europe-west, and Asia-northeast of GCP. The e2-standard-8 instance is a VM with eight vCPUs, 32 GB memory, and 100 GB disk. During experiments, the VMs do not suffer from a shortage of CPU or memory. For the motivating experiments, we place six peers on three VMs. Experiments with a higher number of peers are in §IV. We measure TPS by Hyperledger Caliper when changing all possible placements of peer nodes on the VMs. As for the workload, we use Fabcar with a 19 KB payload size and a sending rate of 400. Note that we also test different workloads, but due to the page limit, we present the representative results. All experiments are repeated at least 10 times.

TPS per placement. We measure the TPS for all possible placement strategies. For three VMs (VM1, VM2, and VM3) in our experiment setup, when we place one, two, and three different peers on three VMs, respectively, we denote the placement as (1, 2, 3). We ensure that at least one peer is placed on each server. Thus, for placing six peers across three different VMs, 10 different placement strategies are possible, as follows: (1,1,4), (1,2,3), (1,3,2), (1,4,1), (2,1,3), (2,2,2), (2,3,1), (3,1,2), (3,2,1), and (4,1,1). We denote each placement strategy from P_1 to P_{10}.

Fig. 1a presents the TPS (y-axis) for the 10 different placements (x-axis). The highest TPS is 364 in P_4, and the lowest is 276.5 in P_8, which shows a difference of 31.6%. In other words, the TPS of the entire BaaS varies by 31.6% only due to the difference in node placement.

Because all transaction validations are achieved through communication between peer nodes, latency between these nodes is one of the important metrics affecting TPS. Fig. 1b shows the average latency between blockchain nodes on VMs per placement. We calculate the average latency values using the ping latency between geo-distributed VMs. In Fig. 1a, P_8 is the placement with the lowest TPS. However, P_8 does not have the highest (worst) latency; instead, it has the second-best (lowest) latency. For P_4, which shows the highest TPS, it does not exhibit the best (lowest) latency, but the eighth lowest. This means that considering networking bottlenecks alone is insufficient to determine proper placement to guarantee desired TPS. This observation leads us to design *Cyan* with ML to consider various BaaS factors.

III. *Cyan* Design

Fig. 2 shows the *Cyan* architecture that consists of three main components: 1) dataset generator, 2) *Cyan* predictor, and 3) placement manager. The dataset generator creates a dataset to train the *Cyan* predictor. The *Cyan* predictor is trained offline to predict TPS with a placement. The placement manager performs predictions on placement strategies and finds the proper one to achieve the desired TPS.

Dataset generator. The *Cyan* predictor requires a dataset for training. However, to our knowledge, there are no available datasets for training TPS in relation to peer placements. Thus, *Cyan* generates the dataset itself by 1) iteratively changing input features of the prediction models (to be explained in the next paragraph) and 2) measuring TPS (output feature).

We determine the input features to reflect three major factors that are known to be effective in analyzing TPS [1], [4]. Table I summarizes the input features, examples, and their ranges in the dataset. First, we include BaaS system factors. Specifically, we use the number of VMs and peer nodes to place. Also, we use the latencies between VMs to reflect the network communication overheads.

Second, we take features of the BaaS service complexity that users run. BaaS services are defined by entering and retrieving transactions into the ledger. We consider the more frequent operation type as an input feature (i.e., `put` and `get`) to consider the different operation complexity in our model. Also, we consider the average transaction size (payload size) and the number of transactions sent to BaaS (sending rate) that also affect TPS, as revealed in the previous study [4].

Third, we consider available placement strategies. When creating a dataset, we randomly set the above input features and then find all possible peer placements for the number of VMs and peers. We take all possible placements as data records. Note that, the placement manager (to be explained later) performs predictions on all possible placements and finds the most suitable placement to satisfy the desired TPS. We use 30K records that are enough to train the *Cyan* predictor.

***Cyan* predictor.** To discover ML algorithm with the highest accuracy for the *Cyan* predictor, we test and compare various algorithms, including linear regression, random forest, k-nearest neighbors, decision tree, gradient boosting, deep neural networks, and support vector machine. We train each algorithm using the dataset generated in the previous subsection. The collected dataset is divided into 80% for model training and 20% for testing (validation). For each model, we identify the hyperparameters that obtain the lowest prediction error through random search with at least 30 trials [7]. After thorough training, we select the random forest model because it provides the best results—$\sim$2.56$\times$ better for root mean squared errors and $\sim$4.36$\times$ better for mean absolute errors than others.

Placement manager. The placement manager selects a placement strategy that satisfies the desired TPS. With the *Cyan* predictor trained as mentioned above, the placement manager predicts the TPS values for all possible placement strategies for the given number of VMs and peers. Then, it selects the placement strategy that shows a predicted TPS most similar to the desired TPS. Once the strategy is determined, the placement manager deploys the peer nodes on the VMs and runs the BaaS services.

IV. Evaluation

In this section, we implement and evaluate *Cyan*. *Cyan* is implemented with about 7000 lines of code and can operate in real-world datacenters. We evaluate *Cyan* on the Google Cloud Platform with a similar setup in §II-B.

To our knowledge, none of the existing studies provide peer node placement decisions. So, we implement and compare *Cyan* to two techniques as follows: random placement and latency-aware placement. Random placement chooses one strategy randomly among possible placements, which is similar to the existing black-box-based approach. We also implement latency-aware placement that considers the networking delays between peers. Given the number of VMs and peers specified by users, this technique calculates the average latency between peers across all available placement strategies (similar to Fig. 1b). Also, we observe that our evaluation setup shows the minimum and maximum TPS of 0 and 800, respectively.

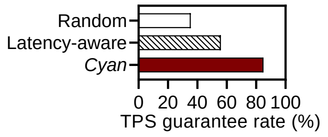


Fig. 3: TPS guarantee rate.

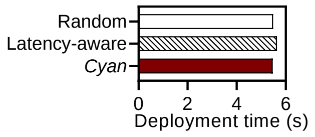


Fig. 4: Deployment time.

If the desired TPS from the user is 200, this matches the lower quartile (25%) at the minimum and maximum TPS range. Latency-aware placement selects the placement that shows the average latency close to the lowest 25%.

We evaluate *Cyan* for TPS guarantee and deployment time. We repeat the experiment trials 1000 times by selecting a random number of peers, VMs, chaincode type, latencies of VMs, transaction size, and sending rate for each trial. Each value is chosen from the range shown in Table I (e.g., $\sim$10 VMs and $\sim$10 peers). We select values that are not used in the model training as much as possible.

TPS guarantee. Fig. 3 presents the TPS guarantee rates of three techniques. For each of the 1000 experiments, a trial is considered to have a guaranteed TPS when the actual TPS differs from the desired TPS by within 10%. We calculate the ratio of TPS guaranteed trials out of the 1000 trials. In Fig. 3, *Cyan* improves the performance guarantee by 2.39$\times$ and 1.52$\times$ compared to random placement and latency-aware placement, respectively.

Deployment time. Fig. 4 shows the deployment time that is the sum of the durations in deciding the proper placement strategy and creating the peer nodes on the VMs of Google Cloud Platform. In Fig. 4, the three techniques show similar deployment times with only a 3.1% difference. The results demonstrate that the overhead of *Cyan* in predicting proper placement strategy is not significant in the deployment process.

V. Conclusion and Future Work

This study introduces *Cyan*, a predictive peer node placement technique for TPS guarantees. *Cyan* enhances TPS guarantees by $\sim$2.39$\times$ without significant deployment delays. In future work, we will extend *Cyan* to consider orderer node placements. Also, we will cover heterogeneous VM capacities, such as CPU, memory, and storage.

References

[1] J. Dreyer, M. Fischer, and R. Tönjes, "Performance analysis of Hyperledger Fabric 2.0 blockchain platform," in *Workshop on Cloud Continuum Services for Smart IoT Systems*, 2020, pp. 32–38.

[2] E. Androulaki *et al.*, "Hyperledger Fabric: a distributed operating system for permissioned blockchains," in *Thirteenth EuroSys*, 2018, pp. 1–15.

[3] S. Mitrevska *et al.*, "Blockchain as a service, an overview on AWS and its BaaS," in *30th Telecommunications Forum*. IEEE, 2022, pp. 1–4.

[4] G. Yang *et al.*, "Resource analysis of blockchain consensus algorithms in Hyperledger Fabric," *IEEE Access*, vol. 10, pp. 74 902–74 920, 2022.

[5] W. M. Shbair, M. Steichen, J. François, and R. State, "BlockZoom: Large-scale blockchain testbed," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 5–6.

[6] A. Song *et al.*, "Fast, dynamic and robust byzantine fault tolerance protocol for consortium blockchain," in *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking*, pp. 419–426.

[7] Y. Yoo *et al.*, "Machine learning-based prediction models for control traffic in SDN systems," *IEEE Transactions on Services Computing*, 2023.