Exploring the Characteristics of Hyperledger Fabric in Resource Consumption

Jeongsu Kim, Kyungwoon Lee, Gyeongsik Yang, Kwanhoon Lee, Jaemin Im and Chuck Yoo
Department of Computer Science and Engineering
Korea University, Seoul, South Korea
{jskim, kwlee, ksyang, khlee}@os.korea.ac.kr, jaeminim95@korea.ac.kr, chuckyoo@os.korea.ac.kr

***Abstract*—As blockchain applications are available in clouds, the resource allocation of blockchain becomes an important issue. This paper investigates the resource consumption of Hyperledger Fabric that is a popular blockchain framework. As Fabric consists of several components, this paper characterizes the resource consumption of Fabric components. Then, we attempt to control the CPU allocation of the components and explore the performance implications. Our evaluation results show that the "proper" CPU allocation of Kafka component reduces the CPU usage of the overall Fabric by 14% but the performance of Fabric does not decrease. Our result suggests that clouds can benefit from such control of CPU allocation to Fabric.**

***Index Terms*—Blockchain, Hyperledger Fabric, Resource consumption, Performance, Resource allocation**

I. Introduction

Hyperledger Fabric [1] consists of multiple components (e.g., peers, orderers, and Kafka brokers) that perform different operations in the "execute-order-validate" architecture. It has various configurable parameters such as block size, endorsement policy, and the number of channels. Previous study [2] pointed out that such parameters affect the overall performance, and another study [3] presented the total CPU utilization of Fabric. These previous studies focus on achieving the maximum performance by allocating sufficient computing resources using individual VMs with a large number of virtual CPUs (e.g., 16 or 32).

However, when applications based on Fabric run on clouds, the VMs executing the components of Fabric share computing resources with other services. So, allocating excessive computing resources to the Fabric components can degrade the overall resource usage. Previous studies suggested a guideline for configuring the parameters to achieve high performance. However, their guidelines did not consider the resource consumption of Fabric components. So the characterization of resource consumption remains to be done for Fabric, which motivates this paper.

*This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation grant funded by the Korea government (MSIT) (No. 2015-0-00280, (SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA) and by National Research Foundation of Korea funded by the MSIT (No. NRF-2019H1D8A2105513).

TABLE I
Experiment Setup.

Parameters	Values	Parameters	Values
Processor	Intel E5-2650 v4 @2.2 GHz (12 cores*2)	# of channels	1
		StateDB	GoLevelDB
		Endorsement policy	OR(Org1, Org2)
Memory	128 GB	Block size/timeout	1MB/500ms
Disk	256 GB SSD	Caliper clients	5
Linux	v4.15.0	Total transactions	1000

II. Evaluation Methodology and Results

A. Evaluation Methodology

Table I shows our experiment setup which utilizes Fabric v1.4.1 with the Kafka consensus protocol. The configuration for Fabric is the same as previous research [2]: two organizations where each organization has two peers, two orderers, four Kafka brokers (one Kafka leader and three Kafka followers), and three ZooKeepers. All components run as containers on Docker v18.09.7. Experiments run the smallbank benchmark that is representative of banking application such as performing account opening, deposit, and money transfer. Note that we run the components in a single server in order to measure the genuine resource consumption of the components without the impact of the network topology between distributed servers. For performance monitoring, we use Caliper v0.1.0. Note that our experiments do not saturate the hardware capacity, specifically CPU by running the server equipped with 24 cores (i.e., 2400% of CPU in total).

B. Resource Consumption of Fabric Components

Fig. 1 is the results of experiments. It shows the average CPU usage of each component in Fabric measured by Caliper. The Kafka leader consumes the largest amount of CPU (98%) among all components, which means that the leader almost saturates a single CPU core. The reason for the high CPU usage of the Kafka leader is that the leader needs to order the transactions and sends the transactions to Kafka followers

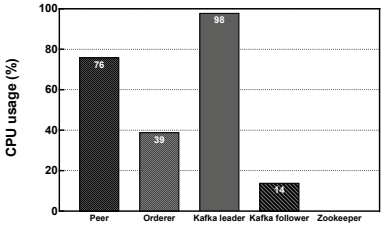

Fig. 1. CPU usage of the Fabric components.

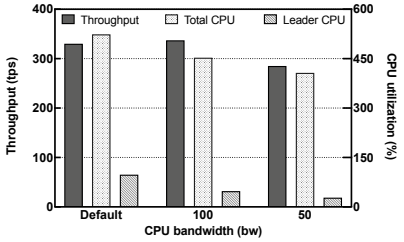

Fig. 2. Impact of controlling the CPU bandwidth of Kafka leader.

and orderers. Peers also consume a significant amount of CPU (76%) for the endorsement and validation of transactions. On the other hand, orderers, Kafka followers, and ZooKeepers consume a small amount of CPU less than 50%. This is because they only receive transactions or replicated transactions from the Kafka leader.

This evaluation finds that each component consumes different amounts of computing resources depending on the operations in the "execute-order-validate" architecture of Fabric. Note that the entire resource consumption of all components is 1,134% of CPU (including the clients and other system daemons that consume 610%). This means that the components run with sufficient computing resources in the experiments.

C. Controlling CPU Bandwidth to the Kafka Component

For CPU allocation, we utilize the CPU bandwidth control mechanism of the Linux CPU scheduler (called CFS). The CPU bandwidth control allows processes to run for the allocated time slice in a period. For example, when the period for the bandwidth control is 5 ms and if 50bw[1] of CPU bandwidth is allocated to a process, the process can use 2.5 ms of time slice every 5 ms. When the runtime exceeds 2.5 ms, the process is de-queued from the CPU runqueue. Note that there can be a difference between the CPU bandwidth and the CPU usage of a process when the process does not fully utilize the allocated time slice. By default, there is no CPU bandwidth control, and so the time slice is not fixed and is determined based on previous execution time and priority [4].

Based on the results in Fig. 1, we attempt to control the CPU bandwidth of the Kafka leader because it has the largest CPU consumption. We experiment with three different CPU allocations of the Kafka leader. One is without CPU control (Linux default) so it is denoted as Default. The second is with the allocation of 100bw, and the third is with 50bw. For each experiment, we measure the throughput of Fabric, the total CPU usage of the Fabric, and Kafka leader CPU usage. Fig. 2 is the results of the experiments – x-axis is the experiments, y-axis in the left is the throughput of Fabric in tps, and the y-axis in the right is CPU usage. In Fig. 2, when the Kafka leader is allocated to 100bw, in comparison with Default, the total CPU usage of Fabric decreases from 524% to 453%, but the throughput increases slightly from 330 tps to 337 tps. And CPU usage of the Kafka leader decreases from 98% to 48%. When the CPU bandwidth is reduced to 50bw, the CPU usage of the leader decreases to 28%. The total CPU usage and the performance also decrease to 407% and 285 tps, respectively.

These results show that controlling the CPU bandwidth of Kafka has an impact on the performance and resource consumption. Importantly, allocating 100bw of CPU bandwidth makes the total CPU usage of Fabric reduced by 14%, which can benefit the overall CPU utilization of clouds. This result is meaningful because the Fabric performance does not decrease with the reduced CPU usage of Fabric. We investigate further to find the cause in the following subsection.

[1]To distinguish between CPU bandwidth and CPU usage, we use the unit of bw to the CPU bandwidth.

TABLE II
SYSTEM LEVEL PROFILING DEPENDING ON THE CPU BANDWIDTH.

	Time slice	Total runtime	# of context switches/s
Default	0.25 ms	1,114 ms	3,830
100bw	0.35 ms	659 ms	2,445

D. System-level Profiling

This subsection investigates the detailed impact of CPU bandwidth control in terms of time slice, the total runtime, and the number of context switches per second using a system-level profiling tool, perf sched. Table II shows the profiling results comparing the default setting (Default) and 100bw CPU bandwidth in Fig. 2. When the CPU bandwidth is controlled to 100bw, the time slice of Kafka is 0.35 ms while it is 0.25 ms in Default. This indicates that controlling the CPU bandwidth to 100bw allows the Kafka to run longer time slice than without CPU bandwidth control (i.e., Default). Note that the total runtime and the number of context switches of Kafka decrease significantly by 41% and 36%, respectively, when compared to Default. This demonstrates that we can reduce the CPU usage of Fabric without sacrificing the throughput. Considering that CPU is a critical resource in clouds, proper control of CPU bandwidth can improve the resource efficiency for Fabric.

III. CONCLUSION

The components of Hyperledger Fabric execute different operations on the "execute-order-validate" architecture. This paper explores the characteristics of the components in CPU consumption, which leads to a finding that properly controlling CPU bandwidth of Fabric does not sacrifice the performance of Fabric but reduces the overall CPU consumption. We plan to extend our research to other Fabric workloads.

REFERENCES

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

[2] H. Javaid, C. Hu, and G. Brebner, "Optimizing validation phase of Hyperledger Fabric," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2019, pp. 269–275.

[3] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing Hyperledger Fabric blockchain platform," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 264–276.

[4] J. Bouron, S. Chevalley *et al.*, "The battle of the schedulers: Freebsd ULE vs. Linux CFS," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 85–96.