Accelerating Throughput in Permissioned Blockchain Networks

Authors:
Samsung:
Kyusang Lee, Changsuk Yoon, Kiwoon Sung
IBM:
Nick Lincoln, Kangwuk Heo, Roman Vaculin, Robert Blessing-Hartley, Anthony O'Dowd, Kyungsoon Kelly Um

<Version 1>
<19.02.14>

Revision History

Sl No	Versions	Details of changes

Document Review and Approval

Sl No	Name	Date	Review / Approval
1	Samsung SDS, IBM	2019.02.13	Review and approval
2			
3			

SAMSUNG SDS

Table of Contents

SAMSUNG SDS

EXECUTIVE SUMMARY

Accelerator is a software component developed by Samsung SDS that is designed to improve the performance of a blockchain network in terms of transaction throughput. Inspired by multilevel queue scheduling [1], Accelerator enables the blockchain network to deal with a large volume of transaction requests from applications. Samsung SDS and IBM have engaged to validate the applicability of Accelerator to Hyperledger Fabric networks and define a roadmap for integration of Accelerator into the Hyperledger Fabric open source project. The team defined a test harness around Hyperledger Caliper and Fabric running on a bare metal system in the IBM Cloud.

As a result of our efforts, we are able to publish consumable and repeatable results to the Hyperledger Fabric community which demonstrate a significant increase in transaction throughput with a minimal impact to latency. Results demonstrate up to 10x performance improvement. This initial draft focuses on results attained prior to the publication of the source code for Accelerator and the Caliper benchmark. These artifacts will be made public in early 2019 in order to build support in the Hyperledger Fabric community for incorporating the Accelerator functions into the Hyperledger Fabric open source project.

THE CHALLENGE

A consistent challenge to implementers of blockchain solutions is the comparatively low transaction rates that blockchain technology currently delivers. It is estimated, for example, that the Bitcoin network achieves a rate of 7 transactions per second (TPS). With the increase in the variety of blockchain use cases, the need for enhanced performance through higher TPS has reached ever-heightening level. More specifically, there are blockchain use cases that require processing of over thousands of transactions per second in real time. Examples of the cases might be processing of data from numerous sensors or identity/authentication service provisioning.

Enterprise blockchain technology such as Hyperledger Fabric has proven capable of achieving much higher rates than public blockchain implementations, but the pressure to achieve even higher level of performance from the real-world use cases has been constant. This paper documents an approach through which the transaction throughput of a permissioned blockchain network can be increased significantly without changing the Hyperledger Fabric implementation. Also discussed are the applicable implementation patterns as well as contraindicated use cases.

ACCELERATOR APPROACH

Consensus is referred to the process by which a network of nodes provides a guaranteed ordering of transactions and validates the block of transactions [2]. It is an unconditional process in the blockchain system to provide trust to blockchain players, and it is generally composed of signing, exchanging, ordering, validation of transactions, etc. However, as parts of these tasks such as cryptographic operations are CPU-intensive [3] and some of processes such as validation of transactions and blocks should be performed in serial [4], there can be performance bottleneck in such systems. Therefore, it is generally considered that the transaction throughput of blockchain is lower than that of the legacy systems since they do not commonly use the consensus for a transaction settlement.

The key idea of Accelerator begins with collecting a few transactions which do not have correlation to each other. Uncorrelated, here, refers to the transactions that do not access the same key or address for ledger update, which means they are not likely to make ledger Multi-Version Concurrency Control (MVCC) collisions described in [5] during the consensus phase. The next step is having the grouped transactions consented upon together as opposed to individually. This is a workable solution because the transactions don't affect each other's commitment results. The simplest way to achieve this would be combining the uncorrelated transactions into a new transaction and submitting the new one to the blockchain network for the consensus. With this approach, more transactions can be processed under a single consensus phase and, as a result, it consequently leads to the increase of performance for both read and write transaction throughputs, i.e., RPS (Reads Per Seconds) and TPS (Transactions Per Seconds) [6], respectively.

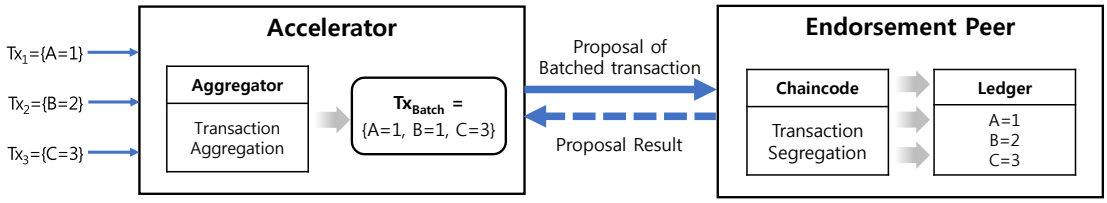

Figure 1. Acceleration Approach

Figure 1 shows the example scenario of transaction acceleration discussed above. There are three transactions, named Tx_1, Tx_2, and Tx_3, and each individually tries to update ledger state with different keys, A, B, and C. They are not correlated because they attempt to update values with the different keys, so no MVCC collision is expected during validation. When Accelerator receives those transactions during a specified period, for example 1 second, it creates a new transaction containing A, B, and C. It then submits the transaction proposal to blockchain peers for endorsement. In an endorsement peer, original transactions in the batched transaction are segregated for transaction simulation [5], and the proposed result is returned to the Accelerator.

The Accelerator feature can be implemented in either client applications or servers such as API gateway servers, which are used to provide convenient connection methods to blockchain networks for users. Acceleration in server, named server-side acceleration, is advantageous to throughput enhancement as it can utilize transactions from many clients for acceleration. Client-side acceleration, on the other hand, is beneficial to the case that there is no such a proxy or a gateway server in the blockchain network. Instead, additional implementation for acceleration feature is required in the client application.

IMPLEMENTATION

In its current incarnation, Accelerator exists as a stand-alone server for server-side acceleration. It is inserted between a blockchain application and Hyperledger Fabric networks as shown in Figure 2. Accelerator consists of three major components, which are Classifier, Aggregator, and Router, written in the Go programming language and utilizing Hyperledger Fabric Go SDK.

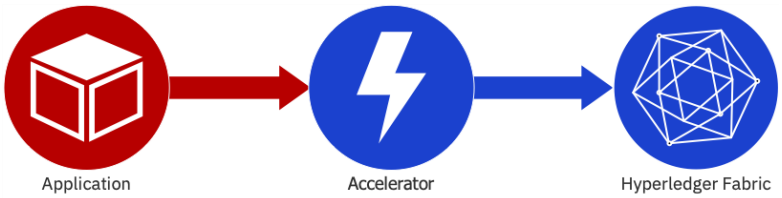

Figure 2. Accelerator as a stand-alone server between applications and Hyperleger Fabric

Components of Accelerator

The fundamental tasks of Accelerator are to:

- classify received transactions with their destination which can be determined by a combination of channel, chaincode, and function name,
- aggregate the classified transactions into a new batched transaction, and
- route the batched one to a blockchain network for consensus.

1. Classifier

All transactions requested by the applications are passed to Classifier and categorized according to the transaction type. The transaction type is currently defined by a combination of the following elements:

- Channel name
- Chaincode name
- Function name in the chaincode

2. Aggregator

A dedicated queue is assigned for each transaction type to collect the classified transactions. The classified transactions in a same batch should be in the same queue in Aggregator so that they can be processed altogether. Aggregator decides if it waits or submits a batched transaction to Router in accordance with the following conditions:

- Number of transactions
- Total size of transactions in bytes
- Wait time of the first transaction
- Occurrence of key duplication

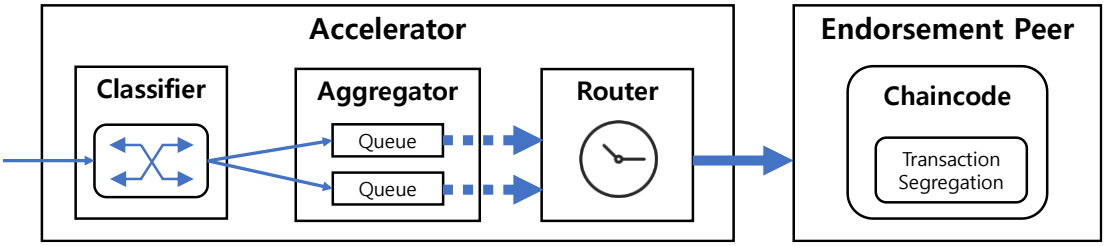

Figure 3. Accelerator Engine

Aggregator makes a batched transaction composed of all transactions in the queue as soon as when a combination of the conditions, which is generally given as a policy, is satisfied. For example, we can define a policy with a 10 as the number of transactions for collection, 1 MB for total size, 1 second for wait time. If any of one of these conditions is satisfied, for example 10 transactions arrive first, a batched transaction is generated and flushed out to Router.

A noticeable condition is checking the occurrence of key duplication, and it is used to compose the batched transaction with uncorrelated transactions only. One of the implementation examples of this feature is 1) keeping the records of keys of the transactions in Aggregator and 2) confirming the key existence in the records whenever a new transition arrives in Aggregator. If any duplication is found, a new batched transaction will be created with the transactions in the queue. In this case, the newly arrived one is not contained in the batch to avoid the collisions and then occupies another queue. This condition check must be preceded by the three conditions mentioned above.

3. Router

Router sends a batched transaction generated by Aggregator to an endorsement peer node through Hyperledger Fabric SDK. Then, the peer operates a verification process for each transaction in the batched transaction. Thus, a chaincode which segregates a given batched transaction into the individual single transactions should be supplementary installed in the endorsement peer. Finally, Router delivers the results to each application who requests the individual transaction included in the batched transaction.

EVALUATION

Software Test Configuration

The following diagram shows the software configuration used to run use case tests. It details the main components in the test running on an IBM Cloud infrastructure as shown in Figure 4.

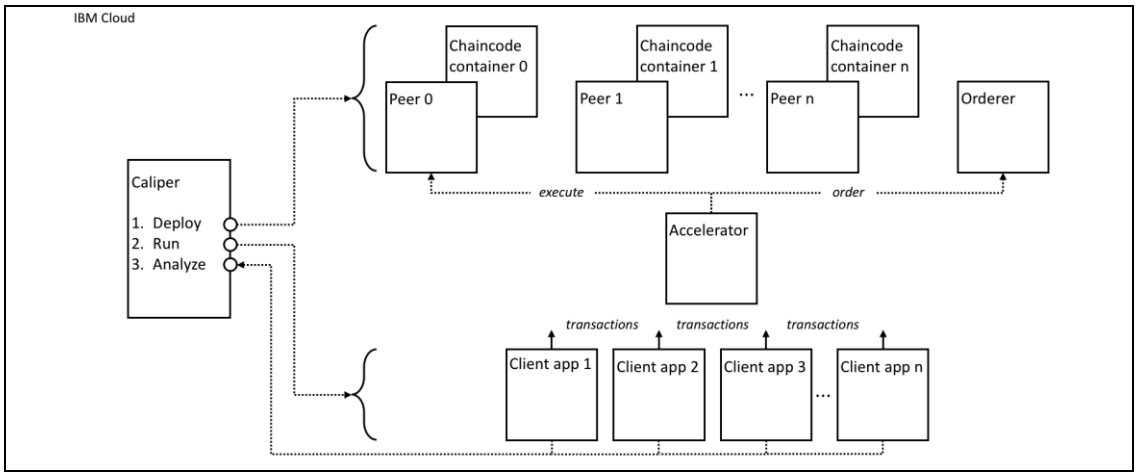

Figure 4. Configuration for performance evaluation

The software test configuration has the following components:

1. **Caliper**: This is the test harness technology used to deploy the test different components of the test, run the tests, and analyze the gathered results. Please refer to an official website [7] for further information.

2. **Peer nodes**: These are the Hyperledger Fabric nodes that host copies of the blockchain ledger. A variable number can be configured; the results in this paper are for a configuration of 3 peers. The peer and orderer nodes jointly form a blockchain network.

3. **Ordering node**: This is the Hyperledger Fabric component that creates blocks of ordered transactions and distributes them to the different peer nodes in the ledger. The peer and orderer nodes jointly form a blockchain network.

4. **Client applications**: These are the applications that submit transactions to the network via Accelerator. Client applications do not talk directly to peer and orderer nodes as they would do in a normal Hyperledger Fabric network; they connect to Accelerator.

5. **Accelerator**: This component mediates interactions between client applications and the peer and orderer nodes that form the blockchain network.

This software configuration is running on IBM Cloud with the following resources allocated:

- 32GB of RAM
- 3.8 GHz quad core CPU
- 10 Gigabit network
- 960GB SSD

These resources are available to all the components in the test configuration according to real-time demand; the chaincode containers, peer and orderer nodes, client applications, and Accelerator share the available network, storage and CPU resources.

Test Cases

The following use cases are run on this configuration.

1. **Simple Query**: A technical use case to measure simple ledger query on a single peer.
2. **Simple Open:** A technical use case to measure a simple update to a ledger.
3. **Smallbank Query**: A more "real world" query of a bank account in a ledger.
4. **Smallbank Operations**: A more "real world" update to a bank account held in a ledger
5. **Smallbank Clash**: A more realistic update to a bank account, with a variable degree of account collisions requiring transaction resubmission.

In **Simple Query**, an application invokes a chaincode via Accelerator, which collects a set of concurrent requests into a single batch (called a ***job***). When the batch is full (according to an ***Aggregator***), Accelerator invokes a modified chaincode with the job. The chaincode modification deblocks the job into individual invocations, which are then executed as normal. In **Simple Query**, each invocation queries the ledger by key for a single state using the *getState()* API. The aggregated response is returned to Accelerator, which dissembles it and notifies the applications with the result of their query.

In **Simple Open,** processing proceeds as **Simple Query**, but the chaincode invocations involve state changes rather than queries. Specifically, chaincode invocation updates the ledger by creating a new state using the *putState()* API. The aggregated response is returned to Accelerator which creates a transaction, orders it and waits to be notified that it has been committed. In then notifies the individual applications that their transaction has been committed (or failed).

The **Smallbank** use cases are more realistic than the **Simple** use cases; they attempt to model a ledger of bank accounts which are being queried or having funds transferred between them. Most importantly, transaction collisions can occur where overlapping transactions are processed at the nearly same time. Processing proceeds as **Simple Open,** but chaincode invocations are subject to world state collisions. Specifically, each chaincode invocation updates the ledger with the *getState()* and *putState()* APIs, but there is an *opportunity* for different transactions to update the same state. Such a collision will result in an invalid **job** transaction being detected after the transaction has been ordered, resulting in the failure of all transactions in a job.

Detailed Result

The graphs in Figure 5 correspond to the two basic forms of Accelerator use case – **Simple Query** and **Simple Open**. For both use cases, we can see how the driven transaction rate on the x-axis affects the achieved transaction rate on the y-axis; we vary the driven transaction rate, and the achieved transaction rate is the dependent variable. In the base case, without Accelerator, we can see linear growth up to 1,500 transactions per second (TPS). Above this rate, there is no gain in the achieved rate no matter how much the driven rate is increased. The system is effectively saturated at this point.

In the meantime, the result with Accelerator is significantly improved. We can see that for driven transaction rates up to 1500 TPS, the achieved transaction rate grows in the same manner as the base rate. However, with Accelerator, we are able to linearly drive query transactions up to an achieved rate of 11,000. Above this rate, transactions start to queue, as with the base case. For **Simple Query**, using Accelerator represents an improvement of approximately 600%. For **Simple Open,** we can see that the achieved rates are less than **Simple Query** because this use case has more work to do – transactions need to be executed, ordered then validated before they appear on the ledger. In the base case, we can see a linear growth up to 450 TPS, but after this point, the achieved rate again starts to plateau as the system becomes saturated.

Again, the result with Accelerator is significantly improved. We can see that there is near linear growth up to an achieved transaction rate of 5,000 TPS, which represents an improvement of 1,000%. This is even better than query; the use of Accelerator is more advantageous for update style transactions than query style transactions.

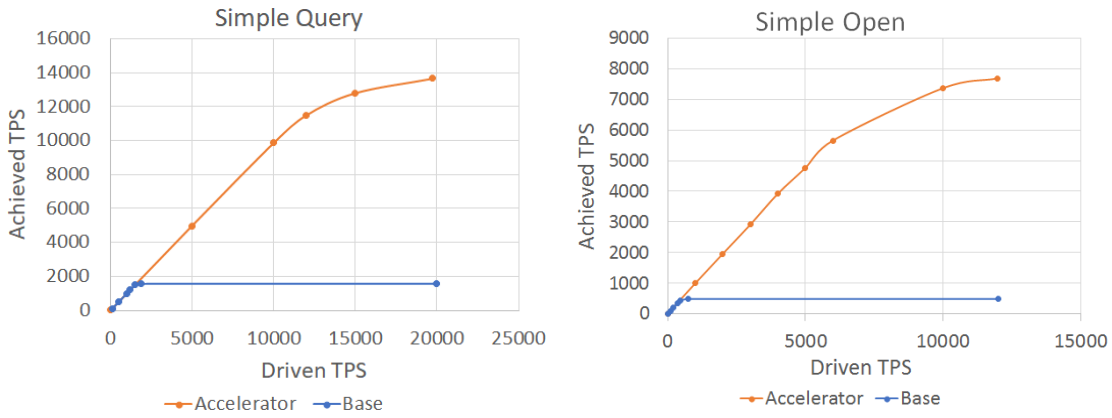

Figure 5. Performance benchmark of Simple scenario

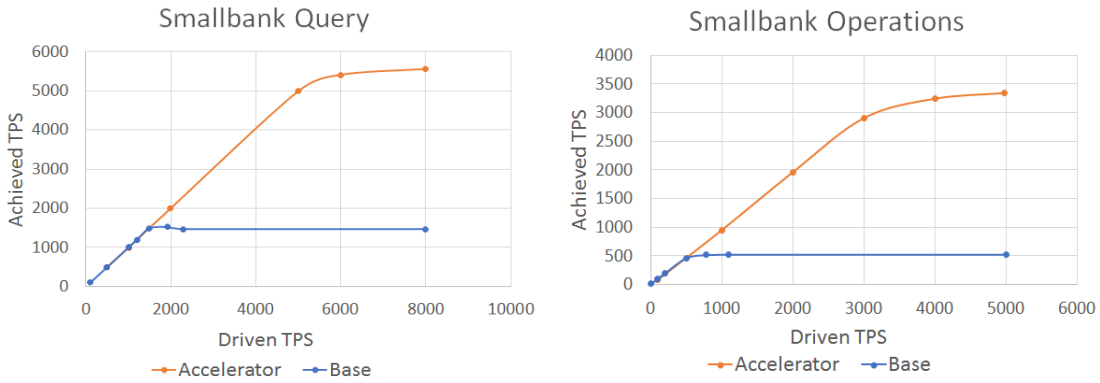

Figure 6. Performance benchmark of Smallbank Operations without collision

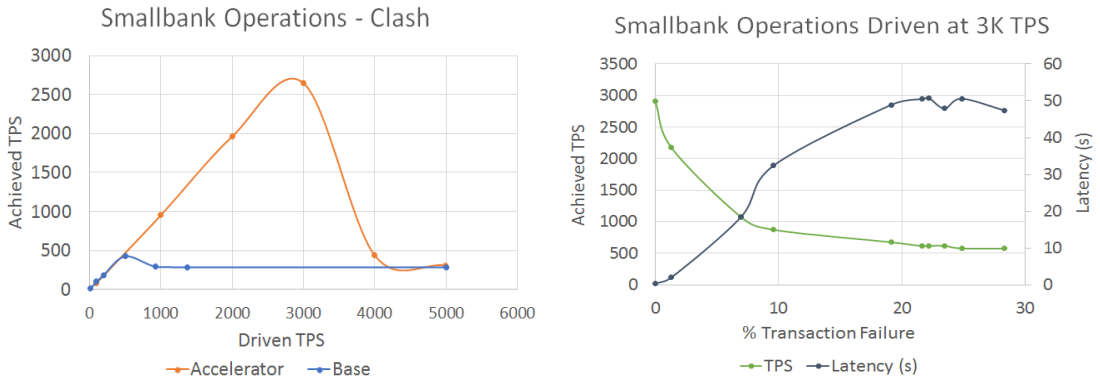

Figure 7. Performance benchmark of Smallbank Operations with clash

The graphs in Figure 7 correspond to the more real-world versions of the use cases – **Smallbank query** and **Smallbank operations**. For the **Smallbank** use cases, we can see very similar behavior to the **Simple** use cases. Accelerator makes a significant difference in throughput for these more real-world use cases. The improvement ratios are of the order of 300% and 600% respectively. We continue to see that the use of Accelerator is more advantageous for update style transactions than query style transactions, although both are significantly improved.

The biggest difference with **Smallbank** use cases are clear however, when collisions are introduced, and the graphs below correspond to the more real-world versions of the ledger update use case with collisions included. The first graph is the same as **Smallbank Operations**, but with collisions occurring naturally. Up to about 3,000 TPS, collisions will not occur at any significant rate, and so the achieved transaction rate follows the same trajectory as **Smallbank Operations**. However, above this rate, collisions start to occur and increase, and thus affects the throughput rate. We see as the driven rate is increased, achieved rate becomes the same as that without Accelerator.

In the second graph, a fixed driven rate of 3,000 TPS is used and the collision rate varied. The graph shows the effect of increasing the collision rate on achieved TPS and latency on the y-axis. We can see that as the collision rate is increased, the transaction rate declines, and the latency increases.

In summary, the **Smallbank clash** use case and the two graphs above show that the benefits of Accelerator are lost once the rate of collision gets to about 10%. This reinforces the fact that for Accelerator to work best, transactions should be independent of each other.

USE CASE RECOMMENDATION - SCENARIOS

Simple Example for IoT

IoT devices and/or sensors applications represent a class of applications that exhibits characteristics particularly suitable for Accelerator. Namely, IoT applications (1) require high throughput due to potentially extremely high volumes of transactions and (2) are at the same time particularly suitable for acceleration due to transaction independence. IoT applications typically collect data and transactions from a high volume of independent devices with each device typically having its own unique identifier and each device possibly periodically generating either time series data or independent sensor readings which are not likely to correlate with other devices' data because of the data characteristics. Due to that, it is typically possible to record each transaction in a separate, unique key which in turn leads to a very low possibility of the correlated data occurrence (e.g., transactions trying to update data with the same address or key at the nearly same time). In such cases, Accelerator can aggregate many transactions without expecting collisions between transactions. The client-side acceleration scheme may be used in an IoT Access Point.

Simple Example for Financial Services

As the second class of applications, we have selected an example application scenario from the Financial Services space in which transactions correlations and possible update conflicts are more likely. This case has individual bank account holders sending transactions for money transfer that may have transactions with many correlations. Because it is possible and, in some cases, even likely that potentially many transactions can be associated with a particular account, it is more likely to encounter transaction update conflicts that are trying to update the same key concurrently at about the same time. Such conflict situation can make application of Accelerator less efficient as we are going to illustrate. However, unless the transactions with higher correlation are generated at the nearly same time, i.e., within the same block generation period (e.g. 1 sec), one can still expect the Accelerator benefits. Server-side Acceleration may be suitable. In most practical scenarios in this category while the conflicts and concurrent updates are possible, they will be typically spread over longer period of time and therefore make the use of Accelerator still very suitable.

Not applicable/Contraindicated use cases, in summary

Accelerator may be less suitable for the cases where many simultaneous, concurrent transactions are trying to update a very small number of addresses with the same keys. In such cases, the many interdependent concurrent updates of the same key can lead to potentially many update conflicts which in turn create a situation where Accelerator performance decreases substantially and therefore makes its use less applicable. Such scenarios may include applications in which many users or blockchain clients want to update data on a very small number of addresses or with the same key values. For example, the scenario may include financial data exchanges (e.g., some stock exchange scenarios) on a few public accounts to deal with many concurrent requests from a large number of clients' accounts (e.g., high volume of stock exchange transactions need to be recorded or netted or settled with respect to a small number of accounts). In this case, the number of aggregated transactions in a time (i.e., transactions that are "non-conflicting") would be very small so as to keep avoiding the collisions, which means there would be small or no acceleration gains in such a case.

CHALLENGES WITH THE CURRENT ACCELERATOR APPROACH

There are some challenges with the current Accelerator approach. They can be overcome by closer integration with Hyperledger Fabric, but, as currently implemented, they represent limitations to its adoption. These sub-sections explain the challenges and discuss how they might be overcome.

Transaction "blurring"

The transaction submitted by an application to Accelerator is not the transaction that appears on the ledger. Instead, the ledger contains multiple transactions that have been "blurred" together. This occurs because Accelerator blocks transactions into jobs, and the actual submitted transaction is the sum of all the individual transactions within a job. This creates challenges for auditability, as users cannot look through the ledger and identify the transactions they have submitted. It makes it difficult for applications, administrators and auditors to link the actual submitted transaction to the one that appears on the ledger. This is a significant problem given purpose and value of a blockchain as an immutable transaction log.

Transaction proposal "mis-signing"

Without Accelerator, the transaction that is submitted by an application is signed by it and the peer that executes it, and these signatures are retained with the transaction as it is recorded in the ledger. When Accelerator is used, the transaction that appears on the ledger is the result of a collection of transactions – a job - and it is this ***aggregated*** transaction that is signed by Accelerator and the peer. It means that the transactions that appear on the ledger are not signed by the submitting applications, and again this presents a significant challenge to the current implementation given the purpose and value of a blockchain as a true record of agreed transactions.

User Chaincode Wrapper

Although it is a fairly mechanical process, chaincode developers have to write a chaincode wrapper to de-block a job into its individual transactions that are then individually executed to generate an aggregate transaction response. This is an extra development task and requires chaincode to be re-installed and re-instantiated. As discussed below, this would be a relatively straightforward feature to be built-into Fabric.

Non-standard APIs

Finally, the APIs provided by Accelerator to applications are only gRPC-based; they are significantly different to the regular SDK programming languages. It means that applications which currently use the JavaScript, Java or GOLANG SDKs have to change to exploit Accelerator. Again, this would be a relatively straightforward feature to be built-into Fabric.

Transaction blurring, transaction mis-signing, user chaincode wrappers and non-standard APIs could all be overcome by making Hyperledger Fabric aware of the concept of a job. The peers and orderers that form a network could treat a job as a collection of transactions and treat them accordingly at execution, ordering and validation time, making sure that transactions proposals, and response signatures were preserved. In this way the benefits of aggregation would be gained without these downsides. Transaction collisions cannot be overcome; other techniques not discussed in this paper are required to alleviate this issue.

DISCUSSION

Applicability

Accelerator provides the improved performance in terms of transactions per second when data are less correlated with each other; e.g., data from a comparatively large number IoT sensor/edge devices as they mostly generate sporadic data. When it comes to the entire volume of such transactions, the larger volume of data would lead to better performance since Accelerator works in a way of aggregating up to enough number of transactions and committing them in a batch. That is, according to what specific business use cases or scenarios Accelerator supports, its expected performance benefits will vary. Therefore, it is recommended to consider applicability based on such performance characteristics and experimental test results to get more precise TPS expected.

Adaptability

Accelerator can adapt to the entire blockchain network condition according to, for example, the number of transactions that it is supposed to process in a given time. For example, if the number of transactions that Accelerator receives as inputs is getting smaller, it would be better to decrease Aggregator's window size, i.e., the number of transactions for the aggregation, to reduce the additional transaction latency [6], which can be caused by a wait time condition in the Aggregator. In the meantime, if Accelerator is processing transactions more and more in time, it needs to increase the window size to maximize the TPS. To enhance this capability, it is necessary to consider developing additional accelerator monitoring and analytics components described in the following sections.

Monitoring

Accelerator's status data by monitoring can be utilized to maximize the adaptability. It collects Accelerator-related system information such as:

- how many transactions are submitted during a specific time
- how many transactions are committed correctly
- how many batched transactions fail
- how long it takes to reach to the consensus and get finality
- the entire computing and network resource consumption, etc.

This information would be beneficial not only when system operators check the system health but also when Accelerator Analytics, described in the following paragraph, decides when it should increase or decrease the window size and even when it should turn on or off the Accelerator.

Analytics

Based on the information gathered by Accelerator Monitoring, Analytics can provide more insightful adaptation capability to Accelerator. By analyzing transaction processing-related status such as the number of inbound and outbound transactions, and their occurrence distribution in time, Accelerator Analytics can suggest optimal parameters such as dynamic window size for greater efficiency. Accelerator Analytics is a component where users and developers build the appropriate models for specific use cases and they can load such models on. Accelerator, in the meantime, will have handler mechanisms to interact with Accelerator Monitoring and Accelerator Analytics. In this sense, an Accelerator that is supplemented by both monitoring and analytics will have dynamic adaptation capability, which can reduce the transaction latency while increasing transaction throughput performance.

ROADMAP

The purpose of this collaboration between IBM and SDS on the Accelerator is to, by running efficiency and performance tests, propose additional functions for the Hyperledger projects such as Hyperledger Fabric SDK, improve performance efficiency and functionality, and, ultimately, integrate the additions with the IBM Blockchain Platform (IBP).

Socialize

The first stage, Socialize, took its shape through the first workshop between IBM and SDS. In this stage, based on the results from the assessment of Accelerator that Samsung SDS and IBM worked together, we hold the purpose to form this whitepaper collaboratively. This stage includes preparation for us to share Accelerator in the next step, "INNOVATE". Accordingly, the design concept and principles, the problem background, the assumption of use cases and scenarios, main functions of Accelerator, and the framework of testing and evaluation are explained in this whitepaper. The underlying system is built by using as-is Hyperledger Fabric docker images that no modification from the public github are implicated. We present the performance comparison results on the system with and without Accelerator as well.

Innovate

The purpose of this stage is to develop an "Innovation Sandbox" available in a public github repository [8] where anyone can download, test, and reproduce the expected performance results using Accelerator on Hyperledger Fabric. This allows us to clearly demonstrate the level of performance of improvement through Accelerator as well as to provide deeper understanding of the technology described on the whitepaper. The Innovation Sandbox will be provided with a docker image of Accelerator and a performance benchmark tool, which additionally has an adaptor module based on Hyperledger Caliper open source [7] to connect Accelerator for performance evaluation. Technical details will be presented to the audience who are working on/with Hyperledger-based projects at technology conferences such as a technical session at IBM THINK. This stage will provide a conduit for valuable feedbacks from the Fabric community with regard to the appropriateness of including Accelerator into Fabric open source projects. Further, additional use cases/refinements may also be identified.

Develop@Scale

In "DEVELOP@SCALE" stage, we target to release Accelerator as an open source component after getting agreements from blockchain community, so that even more engineers can jointly develop and improve Accelerator together. Specifically, the current version of Accelerator can be refactored and incorporated into one of Hyperledger opensource projects. For example, the client-side acceleration approach can be embedded into Hyperledger Fabric SDK in this stage first. Other Fabric enhancements made at this point will be under consideration to integrate alongside the Accelerator capability into the open source. Also, along with opening the source code to the public, Samsung or IBM may design the additional tools (not subject to open source) in need for Accelerator, for example, deployment and monitoring tools, and the systems from such design will be useful when Accelerator is applied to the commercially operating services such as the IBM Blockchain Platform. Additional components, such as Accelerator Analytics described in the Discussion may also be designed during this stage to provide additional business opportunities derived from the incorporation of Accelerator into Hyperledger opensource projects.

Optimize

"OPTIMIZE" stage is to further optimize and advance Accelerator. More specifically, Accelerator will be in a form of server-side one, adding a service discovery function to consider multi-region operation where a number of peers are geographically dispersed and aligning with other enhancements that will be made at this point in Hyperledger Fabric. If the server-side Accelerator is added to the client-side one from the previous stage, its performance benefits will be even improved because the traffic from more endpoints and client software can be intensively collected and processed together at the server-side Accelerator. In addition, Accelerator can reach to comparatively better performance if the client (with Accelerator) is geographically close to peer node in Fabric network while it would suffer from performance degradation if the clients are geographically dispersed.

CONCLUSION

Samsung SDS and IBM have developed and evaluated Accelerator intensively and formulated roadmaps to work with open source communities. Accelerator consists of three major and one supplementary component: Classifier, Aggregator, Router, and the chaincode installed in endorsement peers. All components work to increase TPS in a way of bundling uncorrelated transactions in a batched transaction and submitting it to the Hyperledger Fabric network which can lead to reduce the total required number of endorsement procedures. By deploying Accelerator in front of the Hyperledger Fabric network and connecting to peer nodes, we have successfully developed Accelerator without any modification on Hyperledger Fabric at this phase. According to our evaluation and assessment, Accelerator reaches up to 10x TPS performance on IBM Cloud testing environment. By its design, Accelerator can be easily plugged into an existing Hyperledger Fabric-based blockchain network, and we recommend that current users of Hyperledger Fabric try it out during the "*Innovate*" phase (https://github.com/nexledger/accelerator).

Accelerator's performance is affected by the characteristics of data and transactions. If they are strongly correlated and the correlations occur at the nearly same time, the throughput gain in terms of TPS will not be as high as ten times. For example, a high-frequency trading system would not easily expect the best performance benefits. A more adaptable use case is the one where transactions are uncorrelated or correlated but sporadic. Examples may include not only an IoT scenario, but financial services such as Smallbank operations shown in Evaluation section. In such use cases, Accelerator is able to show the best performance improvement.

REFERENCES

[1] Chopra, R. (2016) Operating System - A Practical Approach, S Chand & Co Ltd
[2] Hyperledger Architecture Working Group, Hyperledger Architecture, Volume 1 - Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus
[3] Elli Androulaki, et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the Thirteenth European conference on Computer systems, EuroSys '18, New York, NY, USA, 2018. ACM.
[4] Thakkar P, et al. Performance benchmarking and optimizing Hyperledger Fabric Blockchain Platform. arXiv preprint arXiv:1805.11390. 2018.
[5] https://hyperledger-fabric.readthedocs.io/en/release-1.4/readwrite.html
[6] Hyperledger Performance and Scale Working Group, Hyperledger Blockchain Performance Metrics
[7] Hyperledger Caliper, https://www.hyperledger.org/projects/caliper
[8] Innovation Sandbox, https://github.com/nexledger/accelerator