Performance Characteristics of Two Blockchain Consensus Algorithms in a VMware Hypervisor

John M. Medellin*, Ph.D., C.P.A. and Mitchell A. Thornton, Ph.D., P.E.
Research Associate Professor and Cecil H. Green Chair in Engineering
Darwin Deason Institute for Cybersecurity, Lyle School of Engineering,
Southern Methodist University, Dallas, Texas 75275, USA
johnmedellin@verizon.net, mitch@lyle.smu.edu

Abstract— ***Two fundamental components of the Blockchain architecture are the encryption and the consensus algorithm. As blocks are appended to the chain they are encrypted with prior information to ensure all the chain's history is included when appending another block transaction. Once a transaction is added to the chain, other participants in the network must agree with the encryption and by default with the transaction. At some point in time, if enough participants have validated the transaction it becomes part of the permanently encrypted record of transactions in the chain. We selected the Byzantine Generals Problem and the RAFT consensus algorithms in order to test for scalability in VMware. These differ from the traditional "Nakamoto" proof of work consensus approach in that they assume that a majority of the participants are trusted sources as would be the case for a private blockchain application. These algorithms are suitable for use in processes that are focused on trust-based transaction validation for documents such as loan applications or cargo manifestos. Our experiments simulate the trusted network and the two algorithms mentioned above. The initial results indicate that both algorithms have better throughput performance when scaled and virtualized. We also scale the volume of blocks appended in each one and find that the RAFT algorithm significantly out-performs the Byzantine Generals' Problem both in raw metal and in private Cloud settings.***

Keywords— Cloud Scalability, Blockchain Consensus, Byzantine Generals Problem, RAFT.

I. INTRODUCTION

The well-known cryptocurrency publication authored by Satoshi Nakamoto gave much needed exposure to the Blockchain design pattern as a viable structure for ensuring transactions were verified and logged in a robust structure; the blockchain. Some key attributed of the blockchain include the avoidance of reliance upon a single "trusted" third party while also providing an anonymous and unregulated means of validation [11]. A common application of this structure is to support the exchange of value between peers; also known as a "crypto-currency" exchange. Since the Nakamoto paper was published, many hundreds of cryptocurrencies have been invented with the most well-known being Bitcoin. The objective of these public exchanges of value is to disintermediate traditional financial channels based upon controlling third parties in favor of peer-to-peer exchanges that can occur in an anonymous manner while also maintaining a high degree of security. The security of these currency exchanges is provided by a crowd-sourced validation task wherein the validation is centered upon solving a cryptographic hash [4]. The crowd-sourcing aspect is due to the fact that the first member of the public who successfully solves the hash problem is rewarded with some amount of cryptocurrency; a process commonly referred to as "mining." Members of the public are thus incentivized to be the first in solving the cryptographic hash problem and the side effect is that new transactions are quickly verified in an anonymous and secure manner.

The Blockchain design pattern has two significant requirements a) solid cryptography that includes aspects of prior hashing as a means of validating both the current transaction as well as the cumulative transaction chain and, b) use of a consensus algorithm for participants to agree and commit a particular transaction to the global tamper-resistant ledger [10]. The combination of these two attributes provides a degree of confidence that the transaction is authentic without relying upon some central authority. Furthermore, the cumulative nature of the cryptographic hash problem to be solved enhances the resiliency of the ledger because it becomes significantly more difficult to break as more cycles are added to the computed hashes. This is a computationally heavy process and one that would be described as a "resource-intensive" workload [7].

Private and public Blockchain implementations have different objectives. In a public Blockchain implementation, the identity of the participants are not generally known (as in the case of Bitcoin) and the only requirement for transaction participation is mathematical proof of execution. The objectives for private Blockchains respond to the needs

of providing a tamper-resistant method for safeguarding a ledger of transactions, documents, or events [14]. In this case, the participants know of each other's identity and trust one another by definition. However, as is the case with public blockchains, the majority of the trusted participants must agree on the authenticity of a particular transaction in order to add it to the chain. The most common reason this process is implemented within a private chain comprised of trusted parties is to ensure that imposters or a"man-in-the-middle" attack is not occurring. In this case, the proof of computation (Nakamoto style [11]) may not be the most efficient way to agree on the validity of a block.

Recently the application of the blockchain philosophy within private distributed systems has enjoyed widespread and diverse applications. Thus, our focus in the research presented here is with regard to private block chain performance. To this end, we have created custom software to test the scalability and performance of two consensus algorithms. We have selected the Byzantine Generals Problem described by Lamport, Shostak, and Pease [6] as one that has been used in the community for a long time as a benchmark for a variety of different studies. The second algorithm we consider is a variation of the method proposed by Ongaro and Ousterhout [12] that preserves many of the cyber-resilient attributes of the Byzantine Generals approach while significantly simplifying the computation due to the trust-election and usage of randomness for the valid time period of election. Our encryption algorithm is based on a SHA-256 workload according to the model in Stallings [13]. The specific architecture of the two consensus algorithms is discussed later in this document.

Our analysis presents the initial work done on both the consensus algorithms and the aspects of scalability of heavy computational workloads. We next describe the software architectures of the two custom simulators built and the virtualized environment in the form of a private cloud that we have provisioned for this research. Finally, we conduct scalability experiments both in a single- and multi-core environment to detect and determine the benefits of moving from a bare metal implementation to a virtualized cloud.

We believe the contribution of this investigation to the Cloud domain is to encourage private blockchain implementations for documents of trust to move into a virtualized environment for the greater efficiencies it can deliver.

Currently we report on the results of our initial exploratory study into the nature of resource consumption of these consensus algorithms under very controlled conditions. However, our results are very encouraging and motivate us to pursue ongoing and more detailed research. To this end, further details on the limitations of this initial research are provided along with our plans for extending this work.

II. RELATED WORK

This research combines the domains of consensus algorithms and scalability in private cloud environments. We begin by describing the functional aspects of the Byzantine Generals Problem and RAFT that assist in arriving at a decision to append a block. Next, we describe the seminal work accomplished in the area of workload simulations and computational partitioning of algorithms on private hypervisors in preparation for measuring the two fundamental differences in the two models: ongoing voting versus using a trust election.

A. *The consensus algorithms.*

The Byzantine Generals problem was popularized due to its representation of a generic scheduling problem in the area of distributed systems processing. The premise of the problem is that the Byzantine empire's army is led by a plurality of Generals and that the army is tasked with the objective of conducting a siege upon a particular city. For the siege to be successful, each division of the army, led by by different particular Generals, must all attack the city simultaneously. Communication among the army divisions occurs via human messengers among the respective Generals. It is assumed that the only way the Generals can communicate with one another is through the use of verbal messages from the messengers. The interesting aspect of the Byzantine Generals problem is that some unknown minority of the Generals are traitors and they thus attempt to undermine the siege through sending false information concerning the planned time of attack to the loyal and honest Generals that represent the majority of the army divisions. At some point in time in the future, all of the loyal Generals will eventually agree upon a time for the siege to occur after a sufficient number of messengers have been dispatched and their messages delivered. The "solution" of the Byzantine Generals problem is then the determination of when the correct time for the siege to initiate has been received and agreed upon by a majority of the Generals leading the army. This problem then characterizes a block chain consensus task wherein it is assumed that imposters or other bad actors are present within the private system.

The Byzantine Generals Problem can be described in an enumerated fashion as follows:

1. There are a number of Generals that are leading the Byzantine Empire's army in a siege of a particular city.
2. They must all attack together at a given time in order to conquer the city.
3. They communicate through messengers that deliver information through verbal channels only.
4. A minority of the Generals are traitors that will try to undermine the remaining majority of honest and loyal Generals by providing false information.
5. At some point in time, all the loyal Generals will agree on a time and when the majority of them agree the time

for attack is set. Determining when this time occurs is the "solution" to the problem.

The RAFT approach is based upon the premise that the private network utilizing a blockchain for trusted maintenance of a shared transaction log is comprised of some number of computational nodes where each node is an independent participant in validating and updating the overall log. Within this model time is considered in discretized intervals or epochs. Furthermore, the epoch lengths are non-uniform and modeled as a sequence of random variables. At the beginning of each discrete epoch, the individual computation nodes each casts a vote for one of their peers and the votes are all tallied to determine which of the collection of computational nodes receives the majority of votes. The RAFT approach also contains provisions for handling tie votes. For the purpose of simplicity, we can assume that the voting and tallying process is repeated with some element of randomness added until a majority vote is achieved. The eventual winner of the vote is then the computational node that is trusted by all others and whom has the responsibility of issuing the duration of the next time interval or epoch. This process is referred to as a "trust-election". In the context of a provate blockchain, the winner of the trust-election is the computational node whose copy of the global blockchain is assumed to be the correct and validated version that is then shared and copied by all other nodes in the network. This process of holding trust-elections and updating blockchain copies the repeats for the processing lifetime of the private network.

The RAFT consensus model can be described as an enumerated set of steps within a method as follows:

1. There are a number of nodes in a network that can carry out computations and will participate in placing their blocks on the blockchain.
2. Time is partitioned into epochs (our term; meaning intervals of time) and each epoch's duration is chosen at random.
3. Before an epoch begins, each of the nodes votes for a node (presumably through a mathematically sound way of selecting who to vote for), the one node with majority of votes wins.
4. If there is a tie then another election begins immediately (please note that the original document has 5 nodes participating to minimize chances of a tied-vote).
5. The winner issues the epoch's duration and a secret that is used by all nodes in encryption of transactions during the epoch.
6. As an epoch ends, another trust-election begins and so the process continues.

Finally, for sake of baseline comparison, the original Nakamoto proof-of-work algorithm as originally intended to support public blockchains for cryptocurrency transactions or exchanges is described as follows in an enumerated form:

1. Participants purchase an entitlement that allows them to transact with others in the network for some predetermined amount of cryptocurrency.
2. When a new currency exchange or transaction occurs, there is a need to record it in a ledger that is accomplished by appending it to the other blocks in the chain.
3. The transacting node recomputes the cryptographic hash result of previous transactions that are not yet committed and derives an encryption hash value of its own.
4. If the assumptions of the blockchain encryption are still valid (e.g. another node has not appended their transaction to the ledger while the encryption computations were being executed) then the node reports the transaction to the rest of the nodes as a valid addition and reports the computation power it has spent doing work. This number accounts for a percentage of the total computation power (which is known by all).
5. More nodes add their encrypted transaction blocks and when a transaction block yields an accumulated load of 51% or more since the original transaction block was appended this will cause the original one to be deemed finalized and it will be committed to permanency. In this manner consensus regarding the ledger to be accepted as valid is achieved by a majority of the participating nodes in the public system.

B. Workload Partitioning and Simulation in Cloud Environments.

Additional challenges take place when a program is re-deployed to run in a Cloud environment. In these situations, the system is translated from a physical (raw metal) implementation to a virtualized (logical) environment and additional performance penalties are incurred to support virtualization [8]. An additional penalty or "virtualization overhead" that can range from 40 to 80% initially is incurred simply for translation from the logical to the physical implementation. As the algorithm scales, the initial penalty reduces while the "true" work, the "workload," reflects the behavior of the application objectives being accomplished [9]. Most workloads will exhibit additional consumption characteristics and some will be markedly more inefficient in the key items such as time and CPU usage.

Generally speaking, any interrupts associated with the code being executed in the Cloud environment will be much more expensive than on raw metal due to the significant interrupt serving penalties that occur in a virtualized system [7]. Specifically interrupts such as multithread discipline through mutex critical regions and condition codes tend to be more expensive than single-threaded algorithms. In addition, the synchronization of multiple cores within the

threads cause for the threaded resources to continue consuming resources although in an idle mode (in a critical region). The combination of the above additional penalties are the reasons why we have executed the experiments described below.

III. Experiment Design

We designed our experiments to simulate the scaled execution of the previously described algorithms in a private cloud. Custom software was generated in *C* and is described below by using diagrams similar to those proposed by Booch in RUP [1] and Larman in UML [5]. The posix paradigm coding was similar to that referenced in Butenhof [2] both for mutual exclusion through mutexes and also using thread-specific data partitioning. We note that the software implemented for these experiements is available through Github [3].

Byzantine Generals Problem Architecture

The Byzantine pseudocode is as follows (please refer to the Activity diagram immediatelly after):

<begin>

<receive transaction and encrypt>

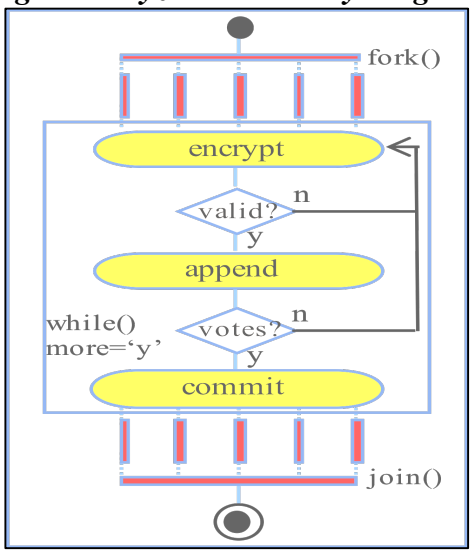
<if transaction ok for next, append & increment vote count>

<else recompute>

<if votes=majority, commit transaction>

<loop to begin>

Figure 1: Byzantine Activity Diagram



RAFT Architecture

The RAFT pseudocode follows and an activity diagram for the code is given.

<begin>

<vote>

<if majority; designate node, issue secret & epoch duration>

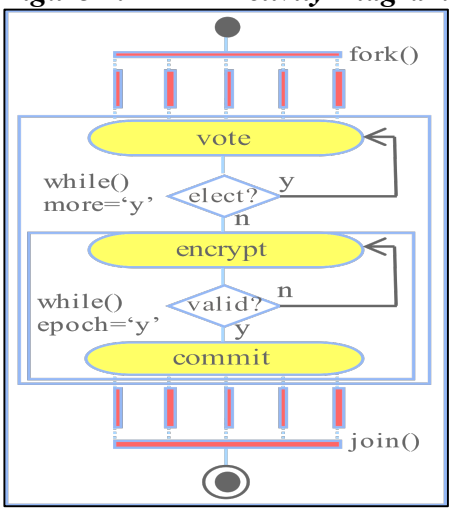
<encrypt>

<if transaction ok for next, append >

<else recompute>

<if epoch over, loop to begin>

Figure 2: RAFT Activity Diagram



Infrastructure Architecture

The infrastructure used to execute these trials was as follows:

Hardware Infrastructure

- Apple MacIntosh MacBook Pro
- Intel Itanium 5 Quad Core CPU; 3.1Ghz
- 8GB RAM/1TB SSD

Software Infrastructure – Bare Metal

- UBUNTU Linux v 16.03
- GCC/pthreads package

Software Infrastructure – Hypervisor

- VMware Workstation Pro 14
- UBUNTU Linux v 16.03 guest Operating System
- GCC/pthreads package

The baseline trials were conducted on bare metal and scaled at 1, 2 and 3 Million blocks to be generated. The private cloud trials were executed under the same volumes and varying the number of cores assigned to the virtual machine. The results are included in section IV and the discussion of results in section V of this document.

IV. Experiment Results

We began by running the algorithms coded in C on "bare metal;" directly on the machine without using VMware to obtain a baseline for the purpose of performance comparisons. The bare metal machine has a quad-core processor that was used to support four independent threads. Furthermore, the machine is also equipped with a hyper-threading feature that allows for eight independently-addressable threads. Our baseline results are reported in Table 1 where CPU utilization and actual measured and corresponding runtimes are given.

Table 1: Resource Consumption, Bare Metal, 4 Cores

Volume	CPU%	Time
1M Blocks		(min:sec)
Byzantine	94%	2:37
RAFT	97%	1:06
2M Blocks		
Byzantine	93%	6:33
RAFT	91%	2:11
3M Blocks		
Byzantine	93%	9:52
RAFT	92%	3:18

Next, we ported the software to a virtual machine implemented via the use of VMware with the following characteristics:

- 1 GB Memory
- 1 Core
- 50 GB Secondary Storage

The results for the virtualized implementations are included in Table 2.

Table 2: Resource Consumption, VMW, 1 Core Virtual Machine

Volume	CPU%	Time
1M Blocks		(min:sec)
Byzantine	91%	0:07
RAFT	80%	0:02
2M Blocks		
Byzantine	91%	0:14
RAFT	80%	0:04
3M Blocks		
Byzantine	91%	0:21
RAFT	80%	0:07

Next, we reran the algorithms in a new virtual configuration by reconfiguring the virtual machine to include four cores. It is noted that we tested both VM configurations for increased RAM and found no impact in scaling it to 2GB. The results are reported in Table 3.

Table 3: Resource Consumption, VMW, 4 Core Virtual Machine

Volume	CPU%	Time
1M Blocks		(min:sec)
Byzantine	95%	2:31
RAFT	87%	0:48
2M Blocks		
Byzantine	95%	5:05
RAFT	94%	1:41
3M Blocks		
Byzantine	95%	7:39
RAFT	96%	2:34

V. Results Discussion and Future Plans

This is an initial experiment in comparing the scaling of private blockchain algorithms. Additional research is needed in varying the algorithms at the application level as well as tuning the other parts of the virtualized stack; host operating system, virtual machine allocations, guest operating systems and other such components. All of the stack components, both in the real and virtualized environments, were installed

with the initial configuration or "out of the box" as suggested by the vendors.

A. *Results Discussion*

In the baseline case, CPU utilization appears to have little variation between all of the results; the Table 1 average was 94% rounded. However, we observe that the algorithms have radically different execution times with most executions of the RAFT method tending to finish in approximately one-third of the time as compared to that required by the Byzantine Generals problem solution approach. These results are congruent to those that have been obtained by Ongaro and Ousterhout [12] and confirm the generally accepted position regarding the efficiency of RAFT over Byzantine Generals. These conclusions then form the basis for our comparison to the virtualized environments that is the primary research result reported here.

In the second experiment with results summarized in Table 2, we observe that the required execution time is radically improved by utilizing a single core. This case is similar to what is expected that the use of one validation machine would incur if used in the algorithms above. These results however can be extrapolated by multiplying them by four for comparison with the multiple cores instantiation. In the case of RAFT's usage of CPU (80% consistent), this is because RAFT holds a trust election at a 500-block frequency and not all the threads are validating results all the time and the trust election is held using thread-specific data rather than mutex variables which is slightly more efficient in a single core machine [2].

In the final case, the results obtained are very similar to the results on bare metal. At the higher levels of scaling (1M and above) the typical behavior of virtual machine workloads will begin to approach the behavior under bare metal (the virtualization overhead is much smaller than the application workload itself) and is consistent with other results we have published in this area [9].

Therefore, we conclude that, based on these initial results, the algorithms consume nearly the same resources that they consume in the bare metal case even when scaled to significant volumes.

B. *Future Plans*

As mentioned above, these results have not been optimized in all aspects of the application and virtualization stack. For example, CPU speeds and allocations need to be varied (virtualization stack) as well as the length of blocks before a new trust election occurs in RAFT. For these intial results, trust-election repetitivity is fixed at 500; however, future efforts will determine what happens when the repetitivity is varied from lower or higher values. These additional efforts will be investigated and reported in successive experiments.

REFERENCES

[1] G. Booch; "Object-Oriented Analysis and Design with Applications" c. Addison Wesley Longman, Inc. 1994, Reading, Massachusets.

[2] D. Butenhof; "Programming with POSIX® Threads, The Addison-Wesley Professional Computing Series" c. Addison-Wesley 1997, Boston, Massachusets.

[3] www.github.com

[4] R. Johnsonbaugh; "Discrete Mathematics 8th edition" c. 2018 Pearson Education, Inc., New York, New York.

[5] C. Larman; "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)" c. Pearson Education, Inc. 2005, Upper River, New Jersey.

[6] L. Lamport, R. Shostak, M. Pease; "The Byzantine Generals Problem" ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, p. 382-401.

[7] M. Liebowitz, C. Kusek, R. Spies; "VMware vSphere Performance – Designing CPU, Memory, Storage, and Networking for Performance-Intensive Workloads" c. John Wiley & Sons, Inc. 2014, Indianapolis, Indiana.

[8] J. Medellin, O. Barack, Q. Zhang, L. Huang; "Enabling Migration Decisions through Policy Services in SOA Mobile Cloud Computing Systems" 2015 International Conference on Grid, Cloud Computing & Applications "GCA'15" p. 23-30.

[9] J. Medellin, F. Lee, L. Budhi, S. Gennepally; "Measuring SPECjbb2015 Virtualization Overhead and Application Workloads in a Mircrosoft Hyper-V Cloud" 2017 International Conference on Grid, Cloud & Cluster Computing "GCC'17" p. 3-10.

[10] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, C. Qinjun; "A Review on Consensus Algorithm of Blockchain" 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC) p. 2567 – 2572.

[11] S. Nakamoto; "Bitcoin: A Peer-to-Peer Electronic Cash System" www.bitcoin.org.

[12] D. Ongaro and J. Ousterhout; "In Search of an Understandable Consensus Algorithm" Proceedings ATC'14 USENIX Annual Technical Conference (2014), USENIX.

[13] W. Stallings; "Cryptography and Network Security, Principles and Practice 7th edition"c. Pearson Education Limited 2018, London, United Kingdom.

[14] X. Xu, I. Weber, M. Staples, L. Zhu et. al.; "A Taxonomy of Blockchain-Based Systems for Architecture Design" Proceedings 2017 IEEE International Conference on Software Architecture, p. 243-252.