

Improving Energy Efficiency of Permissioned Blockchains Using FPGAs

Nathania Santoso Haris Javaid
AMD, Singapore
{nathania.santoso, haris.javaid}@amd.com

Abstract—Permissioned blockchains like Hyperledger Fabric have become quite popular for implementation of enterprise applications. Recent research has mainly focused on improving performance of permissioned blockchains without any consideration of their power/energy consumption. In this paper, we conduct a comprehensive empirical study to understand energy efficiency (throughput/energy) of validator peer in Hyperledger Fabric (a major bottleneck node). We pick a number of optimizations for validator peer from literature (allocated CPUs, software block cache and FPGA based accelerator). First, we propose a methodology to measure power/energy consumption of the two resulting compute platforms (CPU-only and CPU+FPGA). Then, we use our methodology to evaluate energy efficiency of a diverse set of validator peer configurations, and present many useful insights. With careful selection of software optimizations and FPGA accelerator configuration, we improved energy efficiency of validator peer by $10\times$ compared to vanilla validator peer (i.e., energy-aware provisioning of validator peer can deliver $10\times$ more throughput while consuming the same amount of energy). In absolute terms, this means 23,000 tx/s with power consumption of 118W from a validator peer using software block cache running on a 4-core server with AMD/Xilinx Alveo U250 FPGA card.

Index Terms—Energy-efficient blockchains, Hyperledger Fabric, FPGA accelerators

I. INTRODUCTION

Blockchain technology is on the rise due to its capability of executing transactions (which contain business logic in the form of smart contracts) and storing them in a decentralized manner (same ledger distributed among multiple nodes). The distributed ledger contains blocks where each block has a hash value of itself and the previous block, assuring immutability of the ledger data. Hence, a network of nodes that implement a blockchain essentially provides an implementation of a distributed ledger for applications that will be developed and deployed on top of that blockchain. There are two types of blockchains: public (permissionless) and private (permissioned) blockchains. In a public blockchain, nodes do not require identity authorization to participate in the network. On the other hand, the identity of a node must be authenticated cryptographically to execute transactions in a permissioned blockchain. Bitcoin and Ethereum are examples of public blockchains, while Hyperledger Fabric [1] is an example of permissioned blockchain.

Hyperledger Fabric is one of the most popular permissioned blockchains for enterprise applications [2]. Most of the recent research on Fabric has only focused on improving its throughput (transactions per second, shortened as tx/s), overlooking its power and energy consumption. Sedlmeir et al. [3] estimated that although a Fabric network consumed 8 orders of magnitude less energy than a public blockchain, it still consumed

$10\times$ more energy than a centralized system. Furthermore, Fabric nodes are often deployed in datacenters where energy consumption has become an issue due to environmental and climate change concerns. Therefore, there is a need to explore energy efficiency of Fabric nodes. Previous works [4]–[6] have shown that validator peer node (which runs the validation of blocks and transactions before committing them to the ledger) is one of the major bottlenecks. It is a computationally intensive operation, and thus utilizes significant resources which will result in high power and energy consumption in the Fabric network.

In this paper, we propose a power/energy measurement methodology to conduct a comprehensive evaluation of Hyperledger Fabric validator peer in terms of its power/energy consumption and throughput. Typically, Fabric peer is deployed on a multicore server (i.e., CPU based system). Recently, Fabric peer has also been shown to run on a multi-core server with hardware accelerator on an FPGA card (i.e., CPU+FPGA based system) for high throughput [6]. In this context, **this paper has the following contributions:**

- We propose a power/energy measurement methodology for a CPU based system, where our approach handles varying number of vCPUs allocated from the physical cores of a multi-core server. We use this setup to evaluate energy efficiency of vanilla validator peer and a prominent software optimization (block cache from [4]).
- We integrate a power measurement module inside the hardware accelerator from [6] for power/energy measurement of a CPU+FPGA based system. We use this setup to evaluate energy efficiency of hardware accelerated validator peer with and without the software block cache.
- We finally present a comprehensive study of the interactions between power, energy and throughput of validator peer, and deduce many insights from comparison of CPU-only and CPU+FPGA systems for energy-aware provisioning of validator peers in a Fabric network.

Our experiments with Hyperledger Fabric v2.2 LTS running on a multi-core server with Alveo U250 card show that FPGA based hardware accelerator combined with software block cache can deliver $10\times$ more throughput than a CPU-only system with same energy consumption (153 vs. 15 tx/s/J).

II. BACKGROUND AND PRELIMINARIES

A. Hyperledger Fabric

Hyperledger Fabric is an open-source and enterprise-grade implementation of a permissioned blockchain. It is not associated with any cryptocurrency, and has applications in

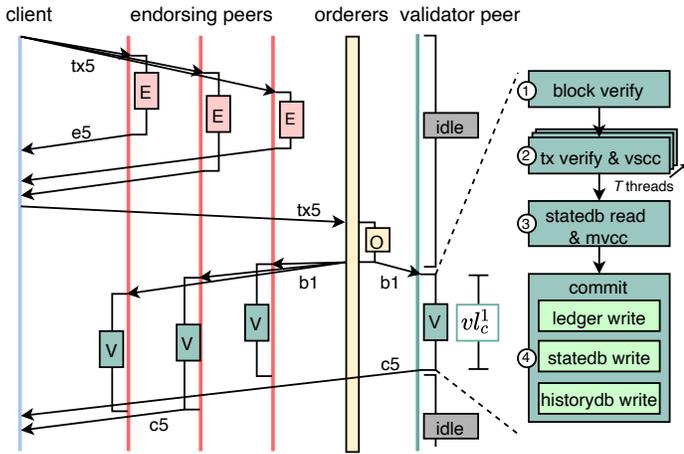


Fig. 1: Transaction flow in Hyperledger Fabric. txN = transaction N, eN = endorsement for txN, bN = block N, and cN = confirmation for txN. E = endorsement, O = ordering, and V = validation phases.

many diverse domains such as supply chain, banking/finance, healthcare, etc. Figure 1 depicts how a transaction flows through various nodes of a Fabric network (e.g., endorsing peers, validator peers, etc. shown on the top).

A client invokes a transaction by sending it to endorsing peers. Each endorsing peer will simulate the transaction by computing its input and output (read and write sets) against the locally stored state database¹. Afterwards, the endorsing peer will send the simulation results along with its digital signature back to the client. Once the client has enough endorsements, it will send the transaction along with the endorsements to the orderer for inclusion into a block. The orderer will create a new block and broadcast it to all the peers for validation and subsequent commit to the ledger. Note that each transaction contains the digital signature of the client that invoked it, while each block contains the digital signature of the orderer that created it. In Fabric, all digital signatures are based on Elliptic Curve Digital Signature Algorithm (ECDSA) scheme.

The validation and commit phase itself consists of several operations as shown on the right hand side of Figure 1. When a validator peer receives a block, it will first check the syntax of the block and verify the orderer’s signature on the block (step 1). Then, the validator peer will check the syntax of each transaction in the block and verify the client’s signature on each transaction (tx verify in step 2). Afterwards, for each transaction in the block, the signatures of endorsing peers from the transaction’s endorsements will be verified and evaluated against an endorsement policy² (validation system chaincode, shortened as tx vscc in step 2). If the endorsement policy is satisfied, then the transaction is marked as valid.

In step 3, multi-version concurrency control (shortened as mvcc) checks are applied to mark a transaction as valid/invalid. Once the entire block has been validated in steps 1–3, in step 4, the validator peer will commit the block to its ledger,

¹The state database contains the current snapshot of the ledger. For example, for a banking application, it would contain the current value of each account.

²An endorsement policy is associated with a smart contract/chaincode, and governs the business logic for approval of the transaction; e.g., policy of a money transfer chaincode between two banks may require valid endorsements from each bank, i.e., endorsements from Bank1 AND Bank2.

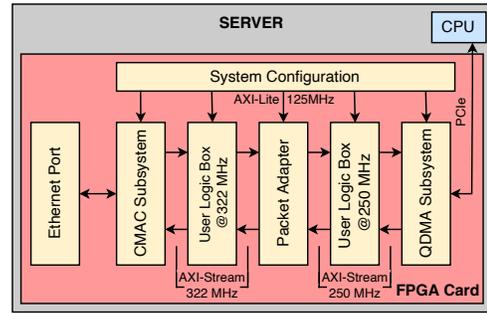


Fig. 2: Overview of OpenNIC shell. The Blockchain Machine hardware is part of User Logic Box @ 250MHz.

and update its state database (by applying write sets of valid transactions) and history database (for book keeping). Note that validator peers only validate and commit incoming blocks, while endorsing peers do the same in addition to endorsing incoming transactions.

The vanilla Fabric validator peer incorporates many software optimizations for improved throughput. For example, it uses multiple threads to verify and validate multiple transactions of a block in parallel (taking advantage of multi-core servers), which is depicted as T threads in Figure 1. Another software optimization is to use *block cache* which caches unmarshaled contents of a block for subsequent accesses, and has been reported to improve throughput significantly ($2.33\times$ in [4] and $1.67\times$ in [5]). However, block cache is not yet part of the official Fabric codebase, so we implemented it ourselves in Fabric v2.2 LTS to evaluate its energy efficiency.

B. Blockchain Machine

Javaid et al. [6] proposed a hardware accelerator called *Blockchain Machine (BMac)* for improving validator peer’s throughput, instead of relying on software optimizations. The BMac peer is designed for a multi-core server with a network-attached FPGA card (connected to the CPU via PCIe bus), where the CPU runs modified Fabric software while FPGA card is programmed with the hardware accelerator. Since this is the only hardware accelerator proposed for Hyperledger Fabric so far, we use it as the CPU+FPGA system in this paper.

Figure 2 provides a simplified overview of the open-sourced OpenNIC shell [7], which is the basis for BMac hardware. The OpenNIC shell is an FPGA based NIC shell for AMD/Xilinx FPGA cards and provides network connectivity through CMAC/Ethernet port interfacing and CPU connectivity through QDMA/PCIe interfacing. Consequently, a user-defined accelerator can be implemented inside the user logic box where it can access incoming data from the network through CMAC/Ethernet port while the CPU can access the output of hardware accelerator through PCIe/QDMA.

The BMac hardware is implemented as user logic box @ 250MHz, as shown in Figure 3. The blocks are received in FPGA card through the CMAC interface. The first module, protocol_processor, processes the incoming Ethernet packets and extracts relevant data, such as block id, transaction ids, ECDSA signatures, etc. The second module, block_processor, uses this data to validate the block and its transactions, commits all valid transactions, and then writes the validation results in a register map (for CPU access through QDMA).

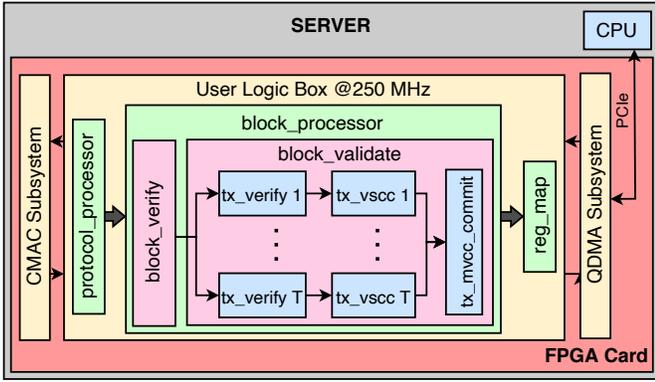


Fig. 3: Overview of Blockchain Machine hardware. Only the relevant modules of OpenNIC shell are shown here.

Figure 4 shows how blocks and transactions are processed in BMac peer. The BMac hardware validates the block without any involvement of the CPU (right hand side). The same block is also received by the Fabric peer software running on the CPU (left hand side), which still executes some parts of the block and transaction verification that are not suitable for hardware accelerator. After that, the software skips validation operations and just reads validation results of the block from hardware, combines them with the original block, and then commits the updated block to ledger just like any other validator peer in the Fabric network. In other words, the validation phase is offloaded to the network-attached hardware accelerator on FPGA.

Note that the block_processor in BMac hardware has several pipeline stages and a configurable number of parallel validators, which is shown as T tx_validators in Figs. 3 & 4. Consequently, the block_processor processes multiple transactions in a parallel-pipelined fashion.

III. PROPOSED POWER MEASUREMENT METHODOLOGY

A. Motivation and Challenges

The Fabric validator peer with software optimizations has been shown to achieve a throughput of 14,000 tx/s [8], while the hardware accelerator has been shown to achieve a throughput of up to 69,000 tx/s [6]. Although these throughput numbers are impressive, all the previous works overlook

energy efficiency of the validator peer. Therefore, we aim to conduct a comprehensive study of not only the validator peer’s throughput but also its power/energy consumption. We list several challenges that exist when measuring power/energy in multi-core servers with an FPGA card, and then present our methodology to handle those challenges.

Challenge 1: The ideal method for measuring power of a server is to use a power meter to measure what is called the wall power, and then compute energy consumption. However, multi-core servers are typically housed in datacenter-like environments where physical access is limited. Therefore, we use hardware counters available in modern processors to measure energy consumption, just like many previous works do [9]. The issue with this approach is that such hardware counters are available at various granularities across different types of processor architectures. In some architectures, they are available for each core, while in other architectures, they are only available for each socket containing multiple cores which makes it impossible to directly get the energy consumption of each core. The availability of per-core energy consumption is pertinent because typical multi-core servers contain tens of physical cores where a subset of these cores is provisioned for a particular application as virtual CPUs (vCPUs) [10]. For example, a validator peer may be allocated only 8 vCPUs when running on a server with 32 physical cores. The challenge here is to measure energy consumption of a validator peer provisioned with varying number of vCPUs when per-core energy consumption is not available from hardware counters.

We propose a method in Section III-B2 where we stress different number of vCPUs in a server to empirically deduce power consumption of an idle core. Then, we use this idle core power to adjust the overall energy consumption of a validator peer. Note that our method can be skipped and replaced with per-core energy consumption values when they are directly available from hardware counters.

Challenge 2: The validator peer receives and processes a block, and then waits for the next block as shown in Figs. 1 & 4. The idle period between blocks depends on how fast an orderer can form a block, which further depends on the aggregate rate at which clients generate/send transactions. More clients sending transactions at higher rates will result in overall higher transaction send rate, which will translate to shorter idle periods between blocks.

Naively measuring energy consumption when idle periods are much longer than block validation time means that the energy consumption would be dominated by the idle power of cores which will mislead the analysis. Furthermore, any energy savings from speeding up the validation phase will not be apparent. Therefore, in an ideal setup, validator peers should be saturated, i.e., aggregate transaction send rate is high enough to result in minimal idle periods. However, saturating validator peers in a research/experimental environment may not be possible due to lack of high-end servers that can run tens of clients, meaning that there is not enough computational power to generate the required transaction workload. Hence, the challenge here is to automatically detect and exclude idle periods from the execution of a validator peer irrespective of the clients’ aggregated transaction send rate and orderer’s block creation rate.

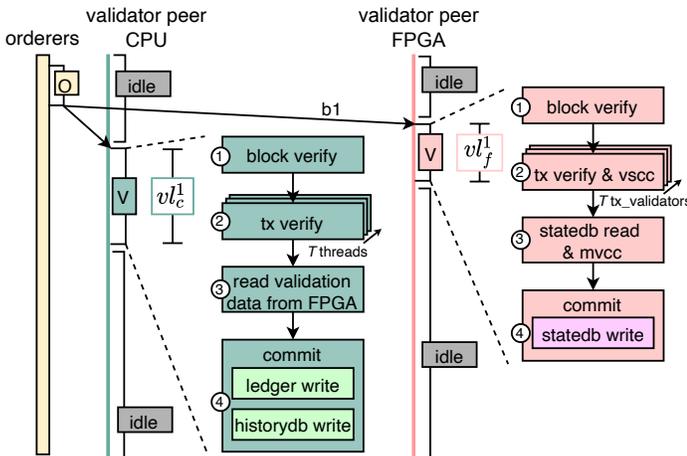


Fig. 4: Validator peer with Blockchain Machine hardware.

We propose a method in Section III-B3 which collects execution log of a validator peer and energy consumption log of the server while validator peer is running. We overlay these logs to detect and exclude idle periods for computation of total energy consumption, as if the validator peer was saturated.

Challenge 3: For the validator peer with hardware accelerator, we must measure energy consumption of both the CPU and FPGA because lightweight operations in Fabric software run on the CPU while computationally intensive operations run on the FPGA. For CPU energy consumption, we use the method proposed in Section III-B. However, for FPGA power/energy consumption, the OpenNIC shell does not provide any mechanism.

We use AMD/Xilinx Card Management Solution (CMS) IP to measure FPGA card power, and integrate it into OpenNIC shell as an additional subsystem. We describe the architecture of CMS IP, its integration, and its access from CPU in Section III-C. Finally, we propose our complete methodology in Section III-D which takes care of intricacies between CPU and FPGA validation phases to compute validator peer’s overall throughput and energy/power consumption for both CPU-only and CPU+FPGA systems.

B. CPU Power Measurement

1) *Intel Hardware Counters:* Modern Intel processors provide the Running Average Power Limit (RAPL) [9] interface for fine-grained measurement of energy consumption. The RAPL interface supports multiple power domains, e.g. package, PP0 and DRAM domains report energy consumption of an entire socket, only the cores in a socket, and only the connected DRAMs, respectively. The RAPL interface uses Model Specific Registers (MSRs) which are basically hardware counters for accumulated energy consumption of different modules, and hence not all domains are available on all processor architectures [11]. For example, Sandy Bridge supports PP0 domain while Haswell-EP does not. These MSRs are updated approximately every 1ms [9].

One can read energy consumption values directly from MSRs, or using *sysfs* interface or *perf* command on Linux. We use *perf* command in this paper because it abstracts away differences in processor architectures and provides a simple command-line interface instead. We conducted many experiments measuring energy consumption directly from MSRs and *perf* command, and found the difference to be always within 1.5%. For all our measurements, we include both the CPU and DRAM energy consumption.

2) *CPU Idle Power:* As explained in Section III-A, if per-core energy consumption is not available through a RAPL domain, then we need a mechanism to measure idle power of a core which can later be used to adjust socket energy consumption (RAPL package domain) for a more realistic measurement (Sec. III-D).

Our approach to measure idle core power is as follows. We measure socket energy consumption when there is no workload running on the server (all cores are idle). Then, we generate workload for only 1 core using *stress* command on Linux and measure socket energy consumption again. We repeat this process by generating workload for 1 more core every time until all the physical cores are used. Since *stress* command

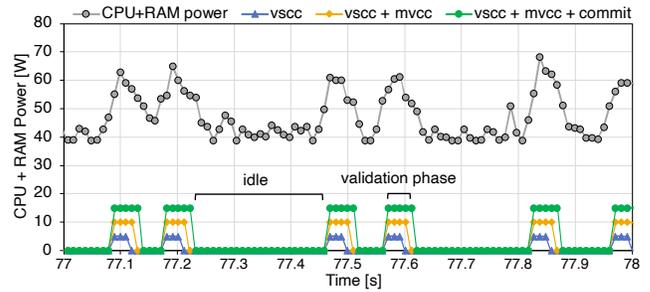


Fig. 5: Annotated time-series power measurements of validator peer.

fully utilizes a core (100% CPU utilization), in each step, we increase active cores by 1 while decreasing idle cores by 1. Hence, in this controlled setup, we expect a linear relationship $P_s = C_a \times P_c^a + C_i \times P_c^i$ where C_a and C_i are the number of active and idle cores respectively, while P_s , P_c^a and P_c^i are the power consumption of the entire socket, an active core and an idle core respectively. In our approach, P_s is measured as socket energy consumption divided by the measurement interval while C_a and C_i are known. Hence, we use linear regression to fit the measured data to deduce the values of P_c^a and P_c^i . We ran the stress commands natively on the server as well as within VMs which are provisioned with 1 vCPU, 2 vCPUs and so on. All our measurements are for an interval of 30 seconds. Based on this setup, the estimated idle core power $P_c^i = 1.44W$ in our servers. Note that the number of vCPUs we provision in our setup is always less than or equal to the total number of physical cores, thus we avoid the use of hyperthreading which is known to make power/energy measurements less accurate [12].

3) *CPU Idle Periods:* As explained in Section III-A, we need a mechanism to detect idle periods in a validator peer so they can be excluded when energy consumption is computed (in order to imitate a saturated validator peer regardless of the clients’ aggregated transaction send rate or orderer’s block creation rate).

Our approach is as follows. We collect execution log of a validator peer where we have modified the Fabric codebase to log timestamps for various operations, e.g. vscc, mvcc, etc. At the same time, we collect socket energy consumption log with timestamps of the server by running *perf* command with an interval of 10ms. We chose 10ms because it provided fine-grained granularity to measure energy consumption across all operations of the validation phase. Since both these logs have timestamps, we overlay the execution log on the energy consumption log to create an annotated time-series data, that is, we add markers in the energy consumption log to indicate start and end of each validation operation. Afterwards, we analyze the annotated time-series data to compute total energy consumption of the validator peer while excluding idle periods.

Figure 5 shows the annotated time-series power measurements (each energy consumption measurement divided by 10ms) for validator peer with markers added for vscc, mvcc and commit operations. When there are no markers, then there are no validation operations running and the validator peer is waiting for the next block. It is clear that power consumption increases significantly when blocks are being validated, while the power consumption is minimal during

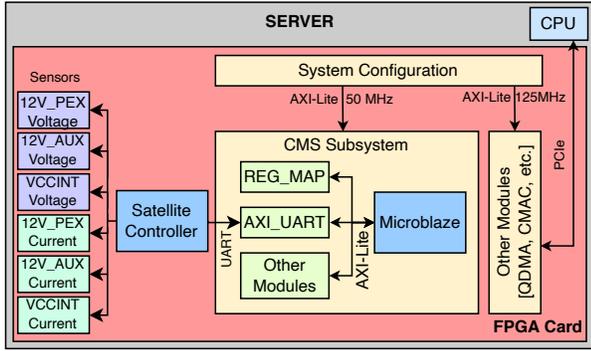


Fig. 6: Integration of CMS IP into OpenNIC shell for power measurements.

idle periods. Our peer runs in a VM (where execution log is captured) which is time-synchronized with the host server (where energy consumption log is captured), hence we use scripts to automatically overlay these logs, detect and exclude idle periods to compute total energy consumption. Note that even when a validator peer is saturated with enough transaction workload, our approach is still applicable since we only analyze idle periods from the execution log.

C. FPGA Power Measurement

Modern FPGA cards have on-board voltage and current sensors to enable power measurements. Several power domains are supported (e.g., total card power, only FPGA power, only BRAM power, etc.) but not all of them are available in every FPGA card. AMD/Xilinx provides Card Management Solution (CMS) IP [13] that interacts with these sensors to report power consumption of the corresponding domains.

Figure 6 shows how we integrate the CMS IP into the OpenNIC shell as an additional subsystem. Internally, the CMS subsystem consists of a Microblaze processor that runs a firmware that interacts with an on-board satellite controller through UART. The satellite controller accesses on-board sensors to read voltage and current values for different power domains. The Microblaze firmware polls the satellite controller approximately every 120ms to read the new values, and writes them into a shared memory (REG_MAP) for access by host CPU (server). The CMS subsystem is connected to OpenNIC shell's QDMA subsystem through the system configuration module using AXI-Lite interface. As a result, the host CPU can access REG_MAP of CMS subsystem over PCIe to read power measurements reported by the satellite controller.

We use AMD/Xilinx Alveo U250 FPGA card in our experiments. The U250 card supports either the total card power domain (12V_PEX and 12V_AUX voltages/currents) or only FPGA power domain (VCC_INT voltage/current) [13]. We use the total card power as power consumption of the hardware accelerator even when we use only CPU and DRAM energy consumption for CPU based system (instead of total motherboard power). In other words, total card power is an upper-bound on the power consumed by the hardware accelerator.

To compute energy consumption of BMac hardware, we collect FPGA power consumption log by running a script on the host that reads and logs total card power with timestamps. The Fabric peer software running on the CPU reads and logs latency to validate each block from BMac hardware registers

in its execution log (in addition to the logging described in Sec. III-B3). Since both these logs have timestamps, we use scripts to automatically merge them to compute energy consumption of the validation phase on FPGA. Note that the smallest interval for FPGA power measurements is 120ms (due to CMS IP limitation), so to be consistent with 10ms CPU energy measurement interval in Sec. III-B3, we assume the same power consumption for all the occurrences of validation phase in the 120ms interval.

D. Bringing It All Together: Validator Peer Throughput and Energy Consumption

In this section, we describe how we put together all the proposed approaches to compute validator peer's throughput and total energy consumption. We use the following terms (which are all measured in seconds):

- vl_c^i and vl_f^i : validation latency of i -th block on CPU and FPGA respectively (Fig. 1 and Fig. 4 respectively).
- tt_c and tt_f : total time to validate all blocks on CPU and FPGA respectively (computed as sum of vl_c^i and vl_f^i respectively for all blocks).
- tt : total time to validate all blocks in a validator peer, computed as $\max(tt_c, tt_f)$ because validation phase executes on both CPU and FPGA in parallel.

The throughput of a validator peer is computed as total transactions in all blocks divided by tt , which is essentially the rate at which transactions are committed by the peer. The total energy consumption and average power consumption of a validator peer are computed as:

$$E = E_c + E_f = E_{vp} + E_{ip} - E_{cores}^u + E_f$$

$$P = E/tt$$

where E_c and E_f denote energy consumption of CPU and FPGA respectively, and the remaining terms are defined as:

- E_{vp} : Energy consumption of the validation phase on CPU, excluding idle periods. It is computed as described in Sec. III-B3, and is essentially the energy consumption of CPU and DRAM over the tt_c period.
- E_{ip} : Energy consumption of CPU and DRAM when the CPU is waiting for the FPGA (i.e., $tt_f > tt_c$ and $tt = tt_f$). It is computed as $P_{ip} \times (tt - tt_c)$ where P_{ip} is computed as the average CPU + DRAM power from all the CPU idle periods (e.g., about 40W in Fig. 5). This term ensures that CPU and DRAM energy consumption is included even when CPU is idle and waiting for the next block due to slower validation phase on FPGA.
- E_{cores}^u : Energy consumption of unused physical cores of a socket, that is, cores that are not provisioned as vCPUs for the validator peer. It is computed as $P_c^i \times C_i \times tt$ where P_c^i and C_i are computed as described in Sec. III-B2. As an example, if a socket has 10 physical cores and validator peer is run in a VM with 8 vCPUs (pinned to this particular socket), then the idle power of 2 cores is subtracted to compute a more realistic energy consumption of the validator peer.
- E_f : Energy consumption of the validation phase on FPGA, computed as described in Sec. III-C over the tt period. Note the use of tt period instead of tt_f to ensure

that FPGA energy consumption is included even when its idle and waiting for the next block due to slower validation phase on CPU.

Note that when validator peer is run on a CPU-only system, all the FPGA related terms (tt_f , E_f , etc.) are zero, and hence all the above equations degenerate into simpler CPU-only equations. For direct comparison between CPU-only and CPU+FPGA systems, we exclude ledger write latency from vl_c^i because ledger write operation is always executed on CPU, and can either be executed asynchronously in the background or on a separate storage server [4].

Our overall approach is algorithmically described below, and is implemented using fully automated scripts:

- Gather validator peer’s execution log, socket energy consumption log and fpga power consumption log.
- Process the above logs to compute tt_c , tt_f and tt .
- Process execution and socket energy consumption logs to compute E_{vp} and P_{ip} , and thus E_{ip} .
- Process execution and fpga power consumption logs to compute E_f .
- Compute validator peer’s throughput, total energy consumption and average power consumption.

IV. EVALUATION SETUP & EXPERIMENTAL RESULTS

A. Hyperledger Fabric Network and Application Setup

We create a Fabric network with two organizations where each organization has 1 endorsing peer and 1 validator peer, and a solo RAFT orderer. These organizations interact with each other through the smallbank smart contract [14] where AND endorsement policy is configured, meaning that all transactions must be approved by both organizations. This is a typical setup used in many previous works [4], [5], [15], and is representative of two banks processing banking transactions. We use Hyperledger Caliper [16], the standard blockchain benchmarking tool, with in-house scripts to automatically bring up the Fabric network, generate transaction workload (30,000 random transactions), collect logs, and report throughput and energy/power consumption of the validator peers.

B. Hardware/Software Setup

We use dual-socket servers where each socket has 10 Intel Xeon 4114 @ 2.2GHz cores. Each peer is run in its own VM which is provisioned with a certain number of vCPUs and 1GB RAM per vCPU, where these vCPUs are pinned to physical cores. The pinning process allows us to measure energy consumption more accurately. For example, we can create two VMs each with 8 vCPUs pinned to physical cores from two separate sockets. As such, these VMs will not interfere with each other and their energy consumptions can be measured by collecting energy consumption of the sockets separately. Likewise, if vCPUs of a VM are pinned to physical cores in both sockets, then we only create a single VM in that server. Both the orderer and Caliper are run in their own VMs with 8 vCPUs. We used Fabric v2.2 with LevelDB for each peer, and the number of vscc threads is the same as vCPUs.

All our experiments use the Fabric network from above where organization 1’s validator peer is run on a VM (CPU-only system) and then run again on a VM with FPGA card

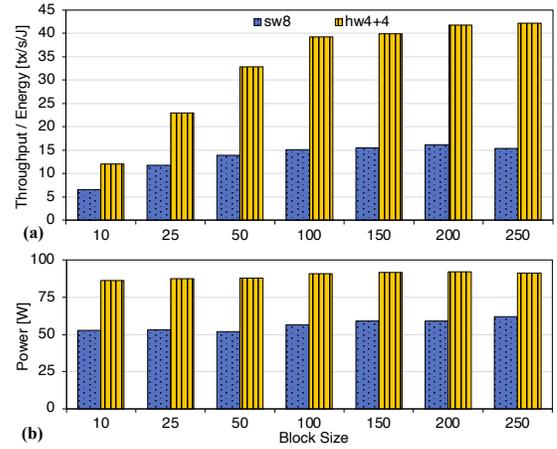


Fig. 7: (a) Energy efficiency, and (b) power consumption of a sw and hw peer. (CPU+FPGA system). The BMac hardware on FPGA card is configured with varying number of tx_validators where 2 ECDSA engines are used in each tx_vscc instance for AND endorsement policy [6].

C. Evaluation Metrics

The primary performance and energy efficiency metrics we use are throughput (the rate at which peer commits transactions, measured as tx/s) and throughput/energy (transactions committed per second while consuming 1 Joule of energy, measured as tx/s/J) respectively. In some cases, we also report the average power consumption and highlight why energy consumption is more relevant than just the power consumption. All these metrics are computed as described in Section III-D.

D. Validator Peer Energy Efficiency

We first create two configurations to understand the energy efficiency of validator peer:

- **swN**: vanilla validator peer using N vCPUs (CPU-only system)
- **hwN+M**: BMac validator peer using N vCPUs and M tx_validators in hardware (CPU+FPGA system)

1) *Varying Block Sizes*: Figure 7a reports the energy efficiency of sw8 and hw4+4 validator peers. We consider 8 vCPUs as the common multi-core configuration for sw peer, while 4 vCPUs with 4 tx_validators is the minimal hw peer configuration in our setup. Typically, 4 vCPUs are enough for hw peer because its software only commits blocks to disk-based ledger. It is evident that hardware accelerator significantly improves energy efficiency, up to 2.8× for block size 250 (15 vs. 42 tx/s/J). From a deeper analysis, we derive the following insights.

Insight 1: The improvement in energy efficiency is quite notable for smaller block sizes (e.g., 10, 25 and 50). Small block sizes do not have enough transactions to fully utilize underlying compute resources and amortize the overhead of block processing. This is even more pertinent for hardware where internal pipelines do not fill up completely when block size is small. Therefore, one should choose a large enough block size keeping in mind the available compute resources for higher energy efficiency (e.g. 50 in this case).

Insight 2: The energy efficiency saturates after a particular block size (e.g., after 50 and 100 for sw8 and hw4+4 peers

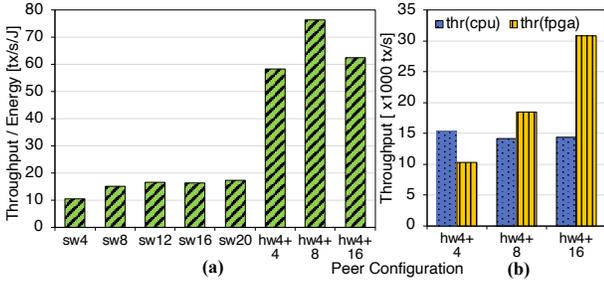


Fig. 8: (a) Energy efficiency, and (b) Throughput of various sw and hw peer configurations.

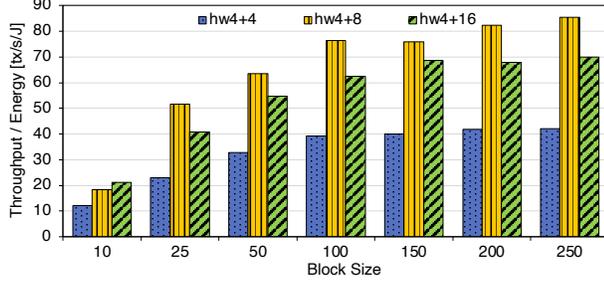


Fig. 9: Energy efficiency of various hw peer configurations.

respectively). Once the block size is large enough to fully utilize the underlying compute resources and amortize the overhead of block processing, the validator peer achieves a steady state. Thus, increasing the block size further does not bring any improvements in throughput and/or energy consumption, however it does increase the memory footprint (storing larger blocks requires larger buffers in hardware). Therefore, one should choose a block size that just saturates energy efficiency of the validator peer.

Insight 3: Figure 7b shows the average power consumption of the two validator peers. The increase in power consumption across different block sizes is not significant which is expected because compute resources are unchanged. Interestingly, hw4+4 peer consumes $1.5\times$ the power of sw8 for block size 250. However, it also improves throughput by $2\times$ (due to faster block validation, though not shown in the figure), which more than compensates for the increased power consumption resulting in higher energy efficiency. Therefore, looking at just the power consumption can be misleading and throughput/energy provides better insights.

2) *Varying Compute Resources:* Figure 8a shows the energy efficiency of the sw and hw peers when the number of vCPUs and tx_validators is changed with fixed block size of 100. In general, hw peer delivers much higher energy efficiency than sw peer when more compute resources are added, from 58 tx/s/J to 76 tx/s/J vs. a maximum of 17 tx/s/J from sw, resulting in $4.5\times$ improvement. We deduce the following insights.

Insight 1: The energy efficiency of sw peer saturates after a particular number of vCPUs (e.g., 12). The validation phase only uses parallel threads for vsc operation, and mvcc and commit operations are executed sequentially without any pipelining, limiting the maximum throughput achievable. The extra power consumption from additional vCPUs is not compensated by notably higher throughput, and hence energy efficiency saturates. For example, sw12, sw16 and sw20 peers consume 66W, 76W and 80W respectively, but deliver about

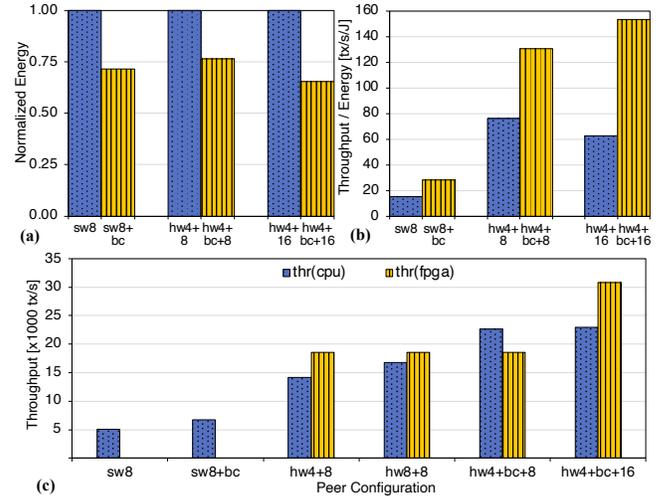


Fig. 10: (a) Normalized energy consumption, (b) Energy efficiency, and (c) Throughput of various sw and hw peer configurations.

the same 17 tx/s/J. Therefore, one should provision vCPUs that will just saturate energy efficiency of the validator peer but with minimal power consumption.

Insight 2: The sudden drop in energy efficiency of hw4+16 vs. hw4+8 is quite unexpected. Since validation phase occurs in parallel on both the CPU and FPGA, we plot their individual throughputs in Figure 8b. The CPU throughput remains the same as the number of vCPUs does not change, while the FPGA throughput increases significantly from 4 to 16 tx_validators. FPGA is the bottleneck in hw4+4 peer while CPU is the bottleneck in hw4+8 and hw4+16 peers. Thus, in hw4+16, power consumption from additional tx_validators is not compensated at the validator peer level because of the much lower CPU throughput. Figure 9 shows that the trend is same between hw4+8 and hw4+16 peers across all block sizes except 10, where CPU and FPGA throughputs are similar (not shown in figure). Therefore, one should match the compute power between the CPU and FPGA in order to achieve the highest energy efficiency.

3) *Software Block Cache:* We create two more variants of the validator peer to understand how software optimizations can affect its energy efficiency. We chose to implement the block cache from [4] in peer software because it is not yet part of Fabric v2.2 and has been shown to improve performance significantly [4], [5].

- **swN+bc:** vanilla validator peer using N vCPUs and software block cache
- **hwN+bc+M:** BMac validator peer using N vCPUs and software block cache, and M tx_validators in hardware

Figures 10a & 10b report the normalized energy consumption and energy efficiency of various sw and hw peers for a block size of 100. We do not include hw4+4 peer because FPGA is the bottleneck (Fig. 8b), thus software optimizations will not result in any noticeable improvement in its throughput or energy efficiency.

Insight 1: The software block cache can bring in significant energy savings across both sw and hw peers, up to 35% in Fig. 10a. The peer software avoids many redundant operations because of the block cache, which results in faster block validation and hence lower energy consumption. The energy

savings are more notable when CPU is the bottleneck in hw peer. Depending on the difference between CPU and FPGA throughput (Fig. 8b), FPGA can be idle for long periods because of the CPU. The use of block cache results in shorter FPGA idle periods, which means higher energy reduction due to FPGA's high power consumption. Figure 10b shows that the use of block cache results in the expected energy efficiency trend between hw4+8 and hw4+16 peers, in contrast to the unexpected trend in Fig. 8a. This is because CPU throughput is now closer to FPGA throughput which results in improved energy efficiency (compare hw4+16 in Figs. 8b & 10c).

Insight 2: Figure 10c reports how the CPU and FPGA throughputs are affected when resources/optimizations are changed. We show only the most interesting validator peer configurations here. Comparing hw4+bc+8 and hw8+8 with hw4+8 reveals that the use of block cache has much larger improvement in CPU throughput than adding more vCPUs (57% vs. 18%). Furthermore, CPU is still the bottleneck in hw8+8 peer. Therefore, one should carefully consider the interactions between hardware resources and software optimizations when provisioning a validator peer.

4) *Summary of Results:* We conclude that hw4+bc+16 peer synergistically benefits from its hardware resources and software optimizations, resulting in 153 tx/s/J which is the highest in all our experiments. This is a 10× improvement over 15 tx/s/J of sw8 (representative of publicly available validator peer running on a common 8-core server). This means that hw4+bc+16 peer can deliver 10× more throughput than sw8 while consuming the same amount of energy. In absolute terms, this translates to 23,000 tx/s with a power consumption of 118W.

V. RELATED WORK

Many recent works have proposed software and hardware optimizations to improve validator peer performance, such as parallel validation of transactions and/or pipelined execution of validation operations [4], [5], [17]–[19], caching unmarshalled blocks [4] and offloading compute-intensive operations to specialized hardware [6]. All these works only focus on performance improvements and overlook power/energy consumption of the validator peer, which is the focus of this paper. Many of these optimizations have already been incorporated into official Fabric v2.2 codebase (e.g., [5], [17], [18]), so our evaluation already includes them. The software block cache [4] is not yet part of the official codebase, so we implemented it ourselves for evaluation.

There is very little research on power/energy consumption of blockchains especially permissioned blockchains like Hyperledger Fabric. The work in [3] estimated energy consumption of different blockchains, and emphasized that further detailed energy efficiency studies are needed especially for permissioned blockchains. The study in [20] proposed an analytical approach to estimate energy consumption of PoW-based blockchains and used Bitcoin as an example. Both these works are analytical in nature and do not use actual power/energy measurements as we have done in this paper. We are the first to conduct a comprehensive study of energy efficiency of Hyperledger Fabric's validator peer with actual power/energy measurements, and present insights for energy-aware provisioning of validator peers in a Fabric network.

VI. CONCLUSION

In this paper, we proposed a power/energy measurement methodology for CPU and CPU+FPGA based systems in order to evaluate energy efficiency (throughput/energy) of Hyperledger Fabric's validator peer. We presented many useful insights from our comprehensive evaluation of a diverse set of validator peer configurations. We concluded that the right combination of hardware resources and software optimizations is essential for achieving highest energy efficiency. We achieved up to 153 tx/s/J compared to 15 tx/s/J of vanilla validator peer.

VII. ACKNOWLEDGMENTS

The authors thank Rajesh Panicker from NUS and Sundararajao Mohan from AMD for their valuable support.

REFERENCES

- [1] Hyperledger, "Hyperledger Fabric," 2019. [Online]. Available: <https://www.hyperledger.org/projects/fabric>
- [2] M. del Castillo, "Forbes Blockchain 50 2021," 2021. [Online]. Available: <https://www.forbes.com/sites/michaeldelcastillo/2021/02/02/blockchain-50/?sh=58a32cb8231c>
- [3] J. Sedlmeir, H. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: Beyond myth," *Business & Information Systems Engineering*, 2020.
- [4] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019.
- [5] P. Thakkar and S. Natarajan, "Scaling blockchains using pipelined execution and sparse peers," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2021.
- [6] H. Javaid, J. Yang, N. Santoso, M. Upadhyay, S. Mohan, C. Hu, and G. Brebner, "Blockchain machine: A network-attached hardware accelerator for hyperledger fabric," in *International Conference on Distributed Computing Systems (ICDCS)*, 2022.
- [7] AMD, "OpenNIC Project," 2021. [Online]. Available: <https://github.com/Xilinx/open-nic>
- [8] C. Gorenflo, "FastFabric v1.4 Implementation," 2020. [Online]. Available: <https://github.com/cgorenflo/fabric/tree/fastfabric-1.4>
- [9] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2018.
- [10] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P.-O. Östberg, and E. Elmroth, "Power-performance tradeoffs in data center servers: Dvfs, cpu pinning, horizontal, and vertical scaling," *Future Generation Computer Systems*, 2018.
- [11] Intel, "Intel 64 and ia-32 architectures software developer's manual volume 4: Model-specific registers," Tech. Rep., 2021.
- [12] Y. Zhai, X. Zhang, S. Eranian, L. Tang, and J. Mars, "HaPPy: Hyperthread-aware power profiling dynamically," in *USENIX Annual Technical Conference (ATC)*, 2014.
- [13] Xilinx, "Alveo Card Management Solution Subsystem Product Guide PG348 v4.0," Tech. Rep., 2022.
- [14] Hyperledger, "Hyperledger Caliper Benchmarks," 2020. [Online]. Available: <https://github.com/hyperledger/caliper-benchmarks>
- [15] L. Zhu, C. Chen, Z. Su, W. Chen, T. Li, and Z. Yu, "BBS: Micro-architecture benchmarking blockchain systems through machine learning and fuzzy set," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.
- [16] Hyperledger, "Hyperledger Caliper," 2019. [Online]. Available: <https://www.hyperledger.org/projects/caliper>
- [17] P. Thakkar, S. Nathan, and B. Vishwanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," in *MASCOTS*, 2018.
- [18] H. Javaid, C. Hu, and G. Brebner, "Optimizing validation phase of hyperledger fabric," in *MASCOTS*, 2019.
- [19] L. Kuhring, Z. István, A. Sorniotti, and M. Vukolić, "StreamChain: Rethinking Blockchain for Datacenters," 2020. [Online]. Available: <http://arxiv.org/abs/1808.08406>
- [20] V. Coroama, "Blockchain energy consumption: An exploratory study," Tech. Rep., 2021. [Online]. Available: <https://www.aramis.admin.ch/Default?DocumentID=68053>