Comparing Blockchain and Cloud Services for Business Process Execution

Paul Rimba*, An Binh Tran*, Ingo Weber*†, Mark Staples*†, Alexander Ponomarev*, Xiwei Xu*†
* Data61, CSIRO, Sydney, Australia
† School of Computer Science and Engineering, UNSW, Sydney, Australia
Email: {firstname.lastname}@data61.csiro.au

***Abstract*—Blockchain is of rising importance as a technology for engineering applications in cross-organizational settings, avoiding reliance on central trusted third-parties. The use of blockchain, instead of traditional databases or services, is an architectural choice in the development of a software system. The costs of execution and storage are important non-functional qualities, but as yet very little has been done to study them for blockchain-based systems. We investigate the cost of using blockchain using business process execution as a lens. Specifically, we compare the cost for computation and storage of business process execution on blockchain vs. a popular cloud service. First, we capture the cost models for both alternatives. Second, we implemented and measured the cost of business process execution on blockchain and cloud services for an example business process model from the literature. We observe two orders of magnitude difference in this cost.**

***Index Terms*—blockchain; cloud; business process; design; cost**

I. Introduction

Blockchain is an emerging technology for software applications in cross-organizational settings. Blockchain provides decentralized record-keeping and computation, and an alternative to conventional use of central trusted parties. The use of a blockchain, instead of a traditional database or service, is an architectural choice. Previously, we explored using blockchain as a connector in a software architecture [10]. We later demonstrated how a business process model can be executed on the blockchain [8]. This allows the execution of a cross-organizational business process without the need for a trusted third-party, supports automatic payments and escrow, and provides an immutable transparent log of past interactions.

Software architecture is concerned with trade-offs between non-functional qualities in the design of software systems. Bass *et al.* [1] describe many non-functional qualities for architecture, including cost and scalability, and discuss the importance of trade-off analyses. The (monetary) costs of execution and storage are important non-functional qualities for systems, but there is very little analysis of them for blockchain. Blockchains enable decentralized trust in storage and execution, but may bring trade-offs for execution cost and latency.

We investigate these issues in the business process context to provide a uniform perspective. We previously [8] examined latency when using blockchain for sending transactions during business process execution and observed overall cost from actual use. In this paper, we compare the (monetary) cost implications of executing business processes on a blockchain vs. a popular cloud service. Specifically, we capture the cost models for business process execution on the public Ethereum blockchain as well as Amazon Simple Workflow Service (SWF). These cost models (Section III) can be used to estimate costs under different workload and settings. To ensure they capture actual cost, we implemented a process model for both platforms and ran corresponding experiments (Section IV) to measure actual cost. Analyzing these results, we also compare the cost of business process execution on blockchain vs. cloud in our experimental context, in which we observed that Blockchain costs two orders of magnitude higher than SWF.

II. Background

Blockchain and Smart contracts. Blockchain is a replicated distributed ledger that verifies and stores transactions occurring in a peer-to-peer network [6], [7]. A blockchain system does not rely on the business operations of any central trusted authority. Instead, its trustworthiness is derived from the blockchain software and incentive mechanisms for processing nodes in the network. The blockchain data structure is a timestamped list of blocks. *Blocks* are containers aggregating transactions. Every block is identifiable and linked to the previous block in the chain through cryptographic hashes.

Smart contracts [5] are semi-autonomous programs running on the blockchain. They can store and update variables, and instantiate and invoke other smart contracts. Ethereum [9] is the most widely used blockchain that supports a Turing-complete scripting language (Solidity) for smart contracts. Trust in the valid execution of the code arises from trust in the integrity of the blockchain.

Blockchain for Business Process. In previous work [8], we proposed a technique to translate models of collaborative business processes – such as the one shown in Fig. 1 – to smart contracts on Ethereum. This allows organizations to collaborate without coordination from a mutually-trusted third-party: their collaboration can be controlled on a neutral platform, the blockchain.

We use blockchain to execute processes in two ways [8]: (i) As a *choreography monitor*, the blockchain records the process execution status for participants, while processing message exchanges. Smart contracts check that interactions conform with the model; (ii) As an *active mediator* among participants, the blockchain coordinates collaborative process execution. Smart contracts drive the process and perform data transformations.

Triggers connect the process executing on blockchain with the external world. Smart contracts cannot directly interact with the external world, so triggers instead interact with smart contracts. For the purposes of this paper, each participant operates their own trigger, within their own organizational context, using a local blockchain node. Triggers monitor execution of process instances. During execution, a trigger translates API calls from its owner to smart contract invocations,

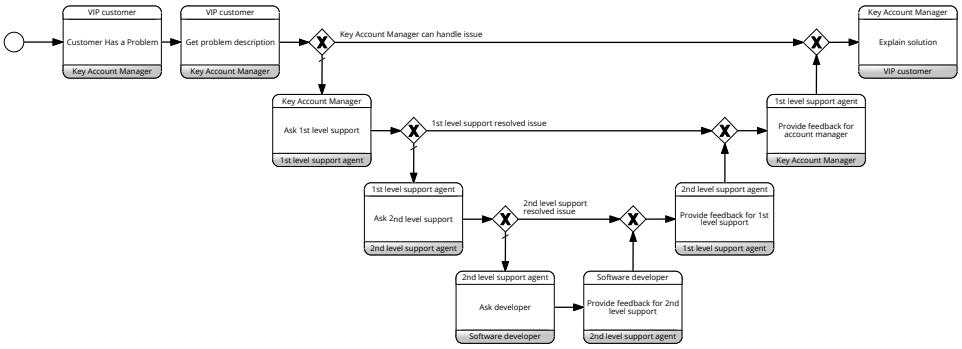

Figure 1. Incident Management Case Study Workflow, adapted from [4, p.18]

and smart contract events to API calls. As such, they connect enterprise systems with smart contracts executing processes.

Cost Analysis. Prior research on cost models and estimation in software engineering research has mainly focused on the cost of software development [2]. In contrast, we focus on operational costs, *i.e.,* monetary cost of execution under different architecture choices.

III. Cost Models

We describe models to estimate the cost of running business processes on two different types of infrastructure. We use Ethereum as a blockchain infrastructure and Amazon Simple Workflow Service (SWF) as cloud infrastructure.

A. Blockchain (Ethereum) Cost Model

There are three types of transactions in Ethereum: financial transfers, message calls, and contract creation. Each has the following basic elements: `from`, `to`, `gasLimit`, `value` and `data`. The `from` and `to` fields signify the sender and the recipient of the transaction respectively. For a *financial transfer* transaction, the amount transferred is given in the `value` field. The `data` field is optional but can contain data in arbitrary other forms. The fee for a transaction with attached data covers the cost for storing the data permanently in the blockchain and is proportional to the size of the data. A *message call* transaction invokes a function of a contract, where the `data` field carries the method to be invoked and the parameters. The `gasLimit` is used to specify the maximum gas that can be used in this transaction. *Gas* is paid for each bytecode instruction that is executed. Finally, a *contract creation* transaction is indicated by a `to` value of NULL, and `data` contains the contract bytecode. For both *message call* and *contract creation* transactions, the `value` field is optional.

We divide our cost model into two parts, one for deploying smart contracts and one for executing process coordination. A contract creation transaction includes compiled bytecode in the `data` field, and the permanent storage of this data incurs cost.

When a contract is created, a particular Ethereum address is subsequently used to interact with that contract. The costs of contract creation are outlined by Wood [9]. We refer to this cost as transactional cost, $\mathcal{C}_{create}$. The contract creation transaction costs 21,000 gas for the transaction itself ($\mathcal{C}_{tx}$), plus 32,000 for allocating a new address ($\mathcal{C}_{addr}$), plus the cost of data payload ($\mathcal{C}_{pload}$, the size of contract bytecode multiplied by gas per byte) and plus any additional gas that is consumed by the opcodes in the function definition ($\mathcal{C}_{fn_{def}}$). We represent this as a formula as shown in Equation 2. At the time of writing, the cost of payload for contract bytecode is 200 gas per byte while the cost of payload for data in a financial transaction and message call is 68 per non-zero byte and 4 per zero byte.

$$\mathcal{C}_{pload} = payload\ (in\ bytes) \times \mathcal{C}_{gas/byte} \tag{1}$$

$$\mathcal{C}_{create} = \mathcal{C}_{tx} + \mathcal{C}_{addr} + \mathcal{C}_{pload} + \mathcal{C}_{fn_{def}} \tag{2}$$

In Ethereum, a contract can create another contract. This is an internal transaction and is cheaper because internal transactions do not incur $\mathcal{C}_{tx}$. The cost of creating a new contract by an existing contract, $\mathcal{C}_{create_{internal}}$, is shown in Equation 3.

$$\mathcal{C}_{create_{internal}} = \mathcal{C}_{addr} + \mathcal{C}_{pload} + \mathcal{C}_{fn_{def}} \tag{3}$$

The second part of our model concerns the cost for executing the business process, and is summarized in Equation 4. A coordination message is treated as a function call in Ethereum. This costs 21,000 gas for the call itself, plus any additional gas that is consumed by the opcodes present during the function execution ($\mathcal{C}_{fn_{exec}}$) and the cost for the data payload.

$$\mathcal{C}_{coord} = \mathcal{C}_{tx} + \mathcal{C}_{pload} + \mathcal{C}_{fn_{exec}} \tag{4}$$

The costs calculated with Equation 2 and 4 are in gas. In order to convert these costs into ether, which is the digital currency in Ethereum, the total gas consumed must be multiplied by the gas price in wei (one wei is 10^{-18} ether). Finally, the cost in ether can be converted into another currency through an exchange service at some exchange rate, $EXC_{ETH2CUR}$. We specify this in Equation 5.

$$\mathcal{C}_{in\$} = \mathcal{C}_{inGas} \times gasPrice \times 10^{-18} \times EXC_{ETH2CUR} \tag{5}$$

Equations 2 and 4 are concerned with the setup and coordination cost for blockchain. Note that, in the blockchain setup, the interface VM operates a full node in the blockchain network. As such, if the VM is not constantly online, the required duration for this VM needs to include the time to synchronize the blockchain with the network.

B. Amazon SWF Cost Model

For the Amazon Simple Workflow Service (SWF)[1], more usage will result in cheaper cost per unit. The main elements are: workflow, actor, task, and signal. Workflows organise activities performed by actors in a sequence. A workflow in SWF represents an instance of a business process, while actors play roles in the process. There are two different types of tasks:

[1] https://aws.amazon.com/swf/ and https://aws.amazon.com/swf/pricing/

Table I
BUSINESS PROCESS MAPPING TO SWF AND BLOCKCHAIN ELEMENTS

Business Process	Blockchain	Amazon SWF
Process Instance	Instance Smart Contract	Workflow
Conformance Checking	Contract execution (partial)	Decision task
Activity	Contract execution (partial)	Activity task
Incoming message	Transaction	Signal
Outgoing message	Entry in contract event log	Notification

activity and decision. Activities schedule a notification for actors to proceed with the next activity. Decisions determine whether the current state of execution conforms with the workflow and which activity is next. A signal is an externally triggered event. Table I shows the mapping of a business process to elements of Blockchain and SWF.

Every actor in the business process implements its own trigger program, which interacts with Amazon SWF through API calls. Decision and Activity tasks require the actor to have a running Amazon SWF worker module, operating on either AWS EC2 or the actor's infrastructure.

The total cost for SWF-based execution has several components. The cost for workflow instances $\mathcal{C}_{wf}$ can be calculated by multiplying the number of instances with the SWF cost of starting a workflow execution ($\mathcal{SWF}_{wf}$) as in Equation 6.

$$\mathcal{C}_{wf} = \#wf \times \mathcal{SWF}_{wf} \tag{6}$$

The *execution* of activity tasks is done by the SWF worker, discussed below. The cost for *scheduling* tasks, $\mathcal{C}_{task}$, is the price per task ($\mathcal{SWF}_{task}$) multiplied by the sum of executed activity tasks and decision tasks. See Eq. 7. Note, the number of activities in a process instance is the number of SWF activity tasks, whereas the number of decision tasks is that number *plus one* additional decision task (immediately after the start of the workflow instance).

$$\mathcal{C}_{task} = (\#actTask + \#decTask) \times \mathcal{SWF}_{task} \tag{7}$$

The number of signals is the number of activities in a business process instance. The cost of signals, $\mathcal{C}_{sig}$, is the number of signals by the price per signal, as in Equation 8.

$$\mathcal{C}_{sig} = \#signals \times \mathcal{SWF}_{signal} \tag{8}$$

Data generated during workflow execution is stored by SWF for a user-specified duration after completion (*retT*), charged per 24 hours. The workflow execution time (*execT*) is also charged per 24 hours at the same rate ($\mathcal{SWF}_{ret}$). See Equation 9. The cost of data transferred, $\mathcal{C}_{dat}$, in and out during workflow execution, is the total payload data size (*payload*) by cost per data unit ($\mathcal{SWF}_{data}$). See Equation 10.

$$\mathcal{C}_{ret} = (execT + retT) \times \mathcal{SWF}_{ret} \tag{9}$$

$$\mathcal{C}_{dat} = payload \times \mathcal{SWF}_{data} \tag{10}$$

The total cost of business process execution on Amazon SWF is in Equation 11, and is the sum of Equations 6 to 10.

$$\mathcal{C}_{swf} = \mathcal{C}_{wf} + \mathcal{C}_{task} + \mathcal{C}_{sig} + \mathcal{C}_{ret} + \mathcal{C}_{dat} \tag{11}$$

Equation 11 is the coordination cost for SWF services and does not include costs of VMs for triggers and SWF workers.

IV. EVALUATION

A. Experiment Setup, Datasets, and Methodology

The goal of this experiment is to compare the cost of business process execution on Ethereum blockchain vs. Amazon SWF, and to assess the accuracy and limitations of the Amazon SWF and Blockchain cost models. We used a process for Incident Management from the literature [4, p.18]. Fig. 1 shows the process model, which has nine tasks, and six gateways. The model has four conforming traces, all of which we use. Such a process would be cross-organizational if, e.g., first-level support was outsourced. The test instances for the process are read from a message trace log file. For each log line, we send a message to the respective actor's trigger, which sends a transaction to the blockchain or a signal to Amazon SWF.

Blockchain. For Incident Management, we reuse the results from our previous work [8], with experiments on the public Ethereum blockchain. Each actor maintains a local Ethereum node, running the go-ethereum (geth) version 1.3.5. Each node is connected to the public Ethereum blockchain and compiled our smart contracts using Solidity compiler version 0.2.0 with optimization enabled. We implemented the triggers in Node.js using the Ethereum library web3 version 0.15.1.

Amazon SWF. For Amazon SWF, each actor was implemented with a business process trigger in Java, using the AWS SDK for Java version 1.11.13. This trigger calls the Amazon SWF API to send signals to the Amazon SWF. We deployed the trigger and SWF worker on an EC2 t2.micro VM.

B. Blockchain Results

For the Incident Management process, we reuse results reported in [8]. In these experiments on public Ethereum, we ran 32 process instances with a total of 256 transactions. The factory contract deployment cost 0.032 Ether (approx. US$ 0.36), and Incident Management instances, with data transformations, cost on average 0.0347 Ether, or approx. US$ 0.39. The exchange rate[2] applied above was recorded on 26/8/2016, where 1 ether was worth US$ 11.32. The experimental data (transactions) for the experiment on the public blockchain are publicly viewable at the factory contract's address, e.g. via Etherscan[3].

C. Amazon SWF Results

In the SWF experiment, we created a new process instance, *i.e.,* SWF Workflow instance, for each run. On receiving a signal, Amazon SWF schedules a *decision task* for the worker. This checks the received signal for conformance with the business process implemented in the workflow instance, to progress the workflow state accordingly.

We deployed an EC2 t2.micro VM for the trigger and the Amazon SWF task worker and executed process instances in sequence. For each process instance, the initialization creates a new workflow (instance) and a decision task to instruct the workflow to wait for the first signal. For each additional message, the trigger sends one signal which results in one activity task and two decision tasks: the workflow schedules a decision task each time it receives a signal or a completion message from an activity task. Thus, for X process instances

[2] https://poloniex.com/exchange#usdt_eth
[3] https://etherscan.io/address/0x09890f52cdd5d0743c7d13abe481e705a2706384

with a total of Y events, there are X workflows, $Y-X$ signals and activity tasks, and $2Y-X$ decision tasks.

In our experiment, we set both the data retention rate and workflow execution to one day. The total cost for the experiment with 1,000 process instances was US$0.925, resulting in an average cost of US$0.000925 per process instance. If we increased the data retention to 365 days, the cost per process instance would be US$0.002745. Table II shows the cost breakdown. The data transfer volume for Incident Management was 358 MB, which is rounded up to 1GB.

Table II
AMAZON SWF COST BREAKDOWN — INCIDENT MANAGEMENT

Element	elements in experiment	Unit cost (US$)	Total cost (US$)
Decision Task	15,000	0.000025	0.375
Activity Task	7,000	0.000025	0.175
Signal	7,000	0.000025	0.175
Workflow	1,000	0.0001	0.1
Retention (24h)	1,000	0.000005	0.005
Exec time (24h)	1,000	0.000005	0.005
Data Transfer	1	0.09	0.09

D. Comparative Analysis

In the Incident Management process, executing one process instance costs US$0.000925 on average on Amazon SWF. In comparison, executing the same process instance on Ethereum costs on average 0.0347 Ether, or approx. US$ 0.36, plus 0.032 Ether (US$0.36) as a one-time cost for deploying the factory contract. Excluding the one-time factory contract deployment, the cost per process instance on blockchain is currently two orders of magnitude higher than on Amazon SWF. Blockchain stores the result in perpetuity (as long as the blockchain is in existence), while SWF has a 90-day limit on data retention. To put the higher *one-time cost for executing* a process instance on Ethereum into perspective with the *ongoing cost for data storage* on Amazon SWF: the data needs to be retained for 71,814 days or approx. 197 years to reach break-even.

Ethereum cost estimates from the online tool[4] have a difference of up to $\pm$ 2.4% for factory contract and process instance deployment. For the cost of coordination ($\mathcal{C}_{coord}$), the tool estimates this as transactional execution cost ($\mathcal{C}_{fn_{exec}}$) + 21,000 gas ($\mathcal{C}_{tx}$) + cost of payload ($\mathcal{C}_{pload}$). The payload cost is (4 bytes of function signature + parameters in bytes) x $\mathcal{C}_{gas/byte}$. For most activities in Incident Management, our cost model estimates gas usage accurately, with the exception of the *customer_has_problem* activity which has unusual gas refund behavior, affecting cost by 15,000 gas. Achieving accurate gas usage and cost estimation for function execution is best achieved by deploying a private Ethereum blockchain.

The Amazon SWF cost model is accurate in estimating the costs for the SWF elements, with a possible variation for workflow execution time and data transfer. Estimating the cost of VM based on the maximum throughput of the VM type and the workload may vary due to performance variation in AWS EC2 and complexity of the activity task implementation.

One benefit of a cost model is the ability to predict cost for different workloads. Having previously validated the cost models for Ethereum blockchain and Amazon SWF, this gives us all the components needed for us to predict the cost of business process execution for different workloads.

[4]https://ethereum.github.io/browser-solidity

E. Discussion

In this paper, we only studied one business process model, Incident Management. Whether this process model is representative carries a threat to the generalizability of our results. This is because differences in structure and data payloads of different process models might impact costs. While our cost model should handle such differences, we have not confirmed its accuracy for other models.

Another question concerns the applicability of this work, given the fast pace of change in the blockchain world. As an example, in a different thread of work [3], a subset of us investigates options to reduce cost for process execution on blockchain. In particular, we minimize the storage space required to capture the process execution state, and minimize the write operations into the persistent variables. These improvements lead to changes in cost, i.e., reductions of up to 25%. From our preliminary analysis, the cost models and methodology outlined in this paper applies – the only changes are in the values for some of the variables in the blockchain cost model. As such, there is an indication that the approach is applicable under such changes.

V. CONCLUSION

Blockchain is of rising importance as a technology for engineering software applications in cross-organizational settings. The use of blockchain is an architectural choice, which affects non-functional qualities of a system such as cost. In this paper, we compared the cost of business process execution on blockchain against cloud services using an example process from literature. Through our calculations and experiments, we have shown that the cost for business process execution on Ethereum blockchain can be two orders of magnitude higher than on Amazon SWF: for the processes, the average cost per process instance was US$ 0.36 vs. 0.0010 at current prices and exchange rates. Furthermore, the experiments confirmed that the cost models capture actual cost with relatively high accuracy. In future work, we plan to devise a method to estimate process execution cost based on a model and historical execution data.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, Boston, MA, USA, 3rd edition, 2012.

[2] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches - a survey. *Annals Softw. Eng.*, 10(1-4):177–205, 2000.

[3] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber. Optimized execution of business processes on blockchain. *arXiv preprint*, Dec. 2016. http://arxiv.org/abs/1612.03152.

[4] Object Management Group. BPMN 2.0 by Example. www.omg.org/spec/BPMN/20100601/10-06-02.pdf, June 2010. v1.0. Accessed 10/3/2016.

[5] S. Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. *AI Matters*, 1(2):19–21, Dec. 2014.

[6] M. Swan. *Blockchain: Blueprint for a New Economy*. O'Reilly, US, 2015.

[7] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IACR Cryptology ePrint Archive*, 2015:464, 2015.

[8] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. Untrusted business process monitoring and execution using blockchain. In *Intl. Conf. Business Process Management (BPM)*, Sept. 2016.

[9] G. Wood. Ethereum: A secure decentralized generalised transaction ledger — homestead draft. Technical report, 2016.

[10] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen. The blockchain as a software connector. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2016.