DOI: 10.5748/18CONTECSI/COM/ITM/6787

APPLICATION RESPONSE TIME COMPARISON BETWEEN ETHEREUM SMART CONTRACT AND SQLITE DATABASE

Renato Mota Ruiz - UNIFIEO - CENTRO UNIVERSITÁRIO FIEO - Orcid: https://orcid.org/0000-0001-5447-5376

Inacio Henrique Yano - EMBRAPA - Orcid: https://orcid.org/0000-0003-2698-6309

Alexandre De Castro - EMPRESA BRASILEIRA DE PESQUISA AGROPECUÁRIA - EMBRAPA - Orcid: https://orcid.org/0000-0002-2019-0142

Julio Cezar Souza Vasconcelos - FEDERAL UNIVERSITY OF SÃO CARLOS, UNIVERSITY OF SÃO PAULO - Orcid: https://orcid.org/0000-0001-6794-3175

This work aims to evaluate the storing and retrieving data response time using an Ethereum Smart Contract application to verify the feasibility of its utilization instead of using relational databases in web application development. To achieve the objectives of this work. There was a comparison between the Ethereum Smart Contract and the SQLite, considering response time as the user experience for database choice decisions in future application development. This study consisted of the development of two similar applications. The first one was the Ethereum Smart Contract Application, and the other was the SQLite Application. Using these applications to build graphs of response time behavior as the number of records processed grows. For storing data Blockchain application was much faster than the SQLite application. When retrieving data, the Blockchain application usually starts slower but finishes faster than the SQLite application. Blockchain is a recent technology for secure data storage in a distributed architecture. The hypothesis to be checked was if it also has a good response time compared with other databases. The contribution of this work is to provide information about the efficiency and possible user satisfaction of Blockchain applications.

Keywords: application development, blockchain, performance, security, user experience

1 Introduction

Blockchain is a recent technology that permits the transfer of assets and data in a secure way. The blockchain emerged from a study by Nakamoto (2008:21260) to create a decentralized ledger for the virtual cryptocurrency Bitcoin. However, as technology is in constant evolution, its use goes far beyond cryptocurrencies. Other applications have great potential for using blockchain, especially those that need transparency, security, traceability, and privacy (Bovério et al., 2018:109).

Due to its origin focused on transactions involving the transfer of values, blockchain has several security mechanisms. The distributed architecture in a point-to-point network avoids a single point of failure. Another valuable security aspect is the block construction process, which only allows the aggregation of a new block after the previous one has been recorded and distributed over the network. Finally, to avoid tampering, the hash of the precedent block is recorded in the current block. So that any change in an already written block would change the content of the following blocks, turning the storage data immutable (Miraz and Ali, 2018).

Currently, there are several studies to use blockchain usage beyond financial applications like health care (Agbo et al., 2019:56), voting systems (Fusco et al., 2018:221), agriculture (Yano et al., 2020:), among many other applications.

There are two types of blockchain networks:

1) Private or Permissioned blockchains. These networks belong to individuals or companies with the purpose to attend their commercial use. In some cases, companies join, deploy and share a proprietary network to exchange information between them, e.g., information systems like supply chain management (Gonczol et al., 2020:11856).

2) Public or Permissionedless blockchains, in these networks, anyone can have an account to use the services, e.g., cryptocurrencies blockchains networks like bitcoin (Michael et al., 2018).

The possibility of using private networks is one of the factors that most drives the spread of blockchain technology due to the restrictive access that facilitates the tests, development, and implementation of applications aimed at specific companies and market niches.

The Blockchain network is suitable for many applications via smart contracts. Smart contracts são programas de computador com funções, variáveis e orientada a eventos implementados em uma rede blockchain (Khatoon, 2020: 94). Each contract writes and reads data of its storage. The data definition on the smart contract defines if it is temporary (stored in memory) or it is permanent (stored in storage) (Gürsoy et al., 2020).

Since blockchain acts as a distributed database in a peer-to-peer network (Bragagnolo et al., 2018:9), it will be helpful to know its performance considering the user experience concerning other databases used for this purpose, the parameter used for comparison was the response time.

This work aims to evaluate the read and write response time of a small web application using a private Ethereum Blockchain network comparing with the same application using the SQLite relational database for a future resource utilization decision of application development.

2 Methodology

As described in the previous section, the objective of this work was to compare the response time between the Ethereum Smart Contract with the SQLite database. For this, there was the development of two very similar applications. One application for blockchain and another one for a relational database. Both with the same interface and recording the same content in the smart contract and the SQLite.

2.1 Environment

The operating system used to deploy the applications was Linux Ubuntu 20.04 on virtual machines, with one processor, 4 GB of RAM, 1 GB of swap area, and 25 GB of disk. The virtualizer used was VirtualBox 6.1, running below a Linux Operating System Ubuntu 20.04, 12-Core CPU, 32 GB of RAM, 2 GB of swap area, and 500 GB of disk. The choice for virtual machines was to create clean machines to avoid as much as possible the interference of other software when processing and collecting data.

The language used for the SQLite web application was JavaScript, and the blockchain web application was JavaScript and Solidity for the smart contract. Both applications run on the Node.js platform. Node.js choice was due to costs reduction with servers because it requires less memory and time for application development when compared to applications using, for example, PHP/Nginx (Shah, 2017). Google Chrome was the browser used as the users' interface (Figure 1). The interface passes one flag, number 1 to store data or number 3 to retrieve data to be processed.

Figure 1 – User Interface

Source: Authors

2.2 SQLite Application

The architecture used for SQLite Application development was MVC (Model, View Controller). This architecture divides the application into three main parts:

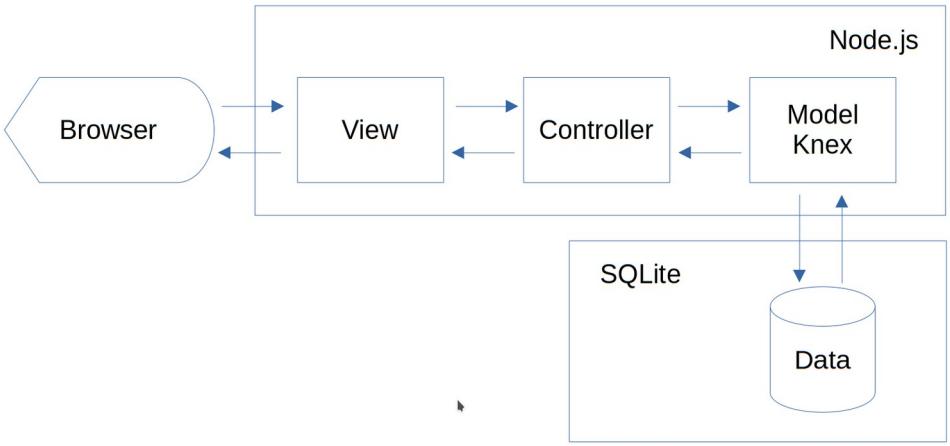
1) Model procedure defines the data structure and session control. In this application, the Model procedure used Knex (bin Uzayr et al., 2019: 377). Knex is a library for SQLite connection and standardized responses.

2) View procedure manages the user interface.

3) Controller is the event handling. It receives requests from View, retrieves data from Model to finally generate responses to View again (Pop and Altar, 2014:1172).

The user starts the application from the web interface that passes the flag to store or retrieve data. This event is captured from the View procedure. Handled by the Controller procedure and finished by the Model procedure with the database writing or reading data (Figure 2).

Figure 2 – SQLite Application Architecture



Source: Authors

For SQLite application response time measurement, there is a looping to store or retrieve the data a certain number of times. In this study, response time was measured for every 400 records processed, stoping when totaling 2000 records. Figure 3 shows the code for SQLite response time registration.

Figure 3 – SQLite Response Time Registration Code

```
var t0 = performance.now()
console.log("T0: " + t0  + " ms.")

for (let i = 0; i < 2000; i++) {
    var userx = "User " + i;
    var emailx = "user" + i + "@test.com";
    await connection('user').insert({
        id: i,
        name: userx,
        email: emailx
    });
    if (i == 399) {
        var t1 = performance.now()
      }
      if (i == 799) {
        var t2 = performance.now()
      }
      if (i == 1199) {
        var t3 = performance.now()
      }
      if (i == 1599) {
        var t4 = performance.now()
      }
}

var t5 = performance.now()
console.log("t0 - t1: " + (t1 - t0)  + " ms.")
console.log("t0 - t2: " + (t2 - t0)  + " ms.")
console.log("t0 - t3: " + (t3 - t0)  + " ms.")
console.log("t0 - t4: " + (t4 - t0)  + " ms.")
console.log("t0 - t5: " + (t5 - t0)  + " ms.")
```

Source: Authors

2.3 Ethereum Smart Contract Application

The smarts contracts are codes written in high-level languages. Usually, the language used for the Ethereum Blockchain network is Solidity (Khatoon, 2020: 94). There is an example of the development, compilation, and deployment process of a Solidity smart contract in detail at Yano et al. (2020). In this work, the Solidity version was 5.0. Figure 4 lists the smart contract code used in this work.

Figure 4 – Smart Contract Code

```
pragma solidity ^0.5.0;

contract UserStore {
  struct usStore {
  uint id;
  string name;
  string email;
}

mapping(uint => usStore) public usermap;

  event usData(address _from, uint _id, string _name, string _email);

  function set(uint _id1, string memory _name1, string memory _email1) public {

    usermap[_id1].id = _id1;
    usermap[_id1].name = _name1;
    usermap[_id1].email = _email1;

    emit usData(msg.sender, _id1, _name1, _email1);
  }

  function get(uint _id2) view public returns(uint, string memory, string memory) {

    return (usermap[_id2].id, usermap[_id2].name, usermap[_id2].email);
  }
}
```

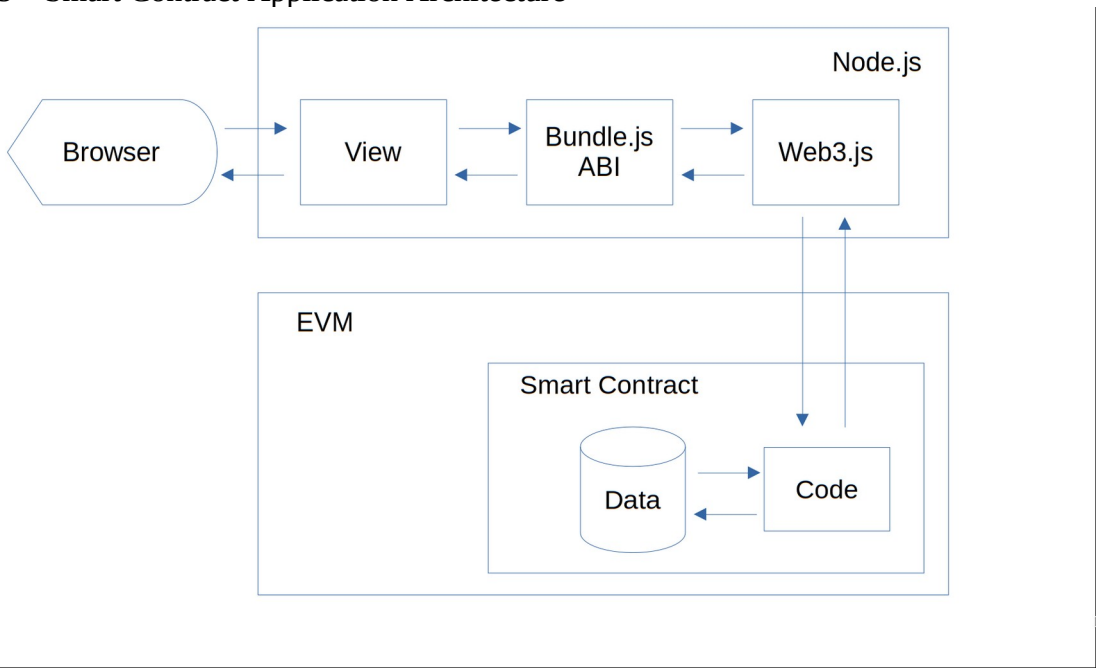
Source: Authors

The UserStore contract stores the data at the usStore Struct. The user id maps the usStore Struct to retrieve the user name and email using the functions set and get repetitively. The emit function stores the variables declared by the event command in transaction logs.

These contracts run in the Ethereum Virtual Machine (EVM). In this work, Truffle Develop provided the EVM (Robertson, 2018). Each smart contract has its area of code and storage (Figure 5). Firstly, the user interacts with the HTML interface (Viewer process in Figure 5). The action captured by the Viewer interface starts Bundle.js functions which call the corresponding procedures at the smart contract using Web3.js libraries by ABI (Application Binary Interface). The process finishes with the set or the get operation in the smart contract.

For Smart Contract response time measurement also there is a looping to store or retrieve the data a certain number of times. The code difference is only for the methods for data handling. In this work, response time was measured for every 400 records processed, stoping when totaling 2000 records. Figure 6 shows the code for Smart Contract response time registration.

Figure 5 – Smart Contract Application Architecture



Source: Authors

Figure 6 – Smart Contract Response Time Registration Code

```
161     var t0 = performance.now()
162     console.log("T0: " + t0  + " ms.")
163
164     for (let i = 0; i < 2000; i++) {
165     var userx = "User " + i;
166     var emailx = "user" + i + "@test.com";
167     userStore.methods
168       .set(i,userx,emailx)
169       .send({from: accounts[0]});
170       if (i == 399) {
171          var t1 = performance.now()
172           }
173       if (i == 799) {
174          var t2 = performance.now()
175           }
176       if (i == 1199) {
177          var t3 = performance.now()
178           }
179       if (i == 1599) {
180          var t4 = performance.now()
181           }
182     }
183         var t5 = performance.now()
184         console.log("t0 - t1: " + (t1 - t0)  + " ms.")
185         console.log("t0 - t2: " + (t2 - t0)  + " ms.")
186         console.log("t0 - t3: " + (t3 - t0)  + " ms.")
187         console.log("t0 - t4: " + (t4 - t0)  + " ms.")
188         console.log("t0 - t5: " + (t5 - t0)  + " ms.")
```

Source: Authors

3 Results

There were five response time measurements according to Tables 1 for Storing Data and 2 for Retrieving Data. For storing data Blockchain application was much faster than the SQLite application. When retrieving data, the Blockchain application usually starts slower but finishes faster than the SQLite application. Figures 7 and 8 show the graphs of the storing and retrieving data behavior as the number of records processed grows.

Table 1 – Storing Data Response Time in Milliseconds

	1		2		3		4		5	
Records	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite
400	220	4855	234	4978	228	5400	227	5270	212	4991
800	309	9764	324	10254	320	10865	316	10519	302	10203
1200	419	14784	443	15585	430	16244	430	15840	422	15374
1600	517	19954	539	20705	525	21457	526	21189	523	20498
2000	630	25035	681	26102	652	26793	650	26479	644	25566

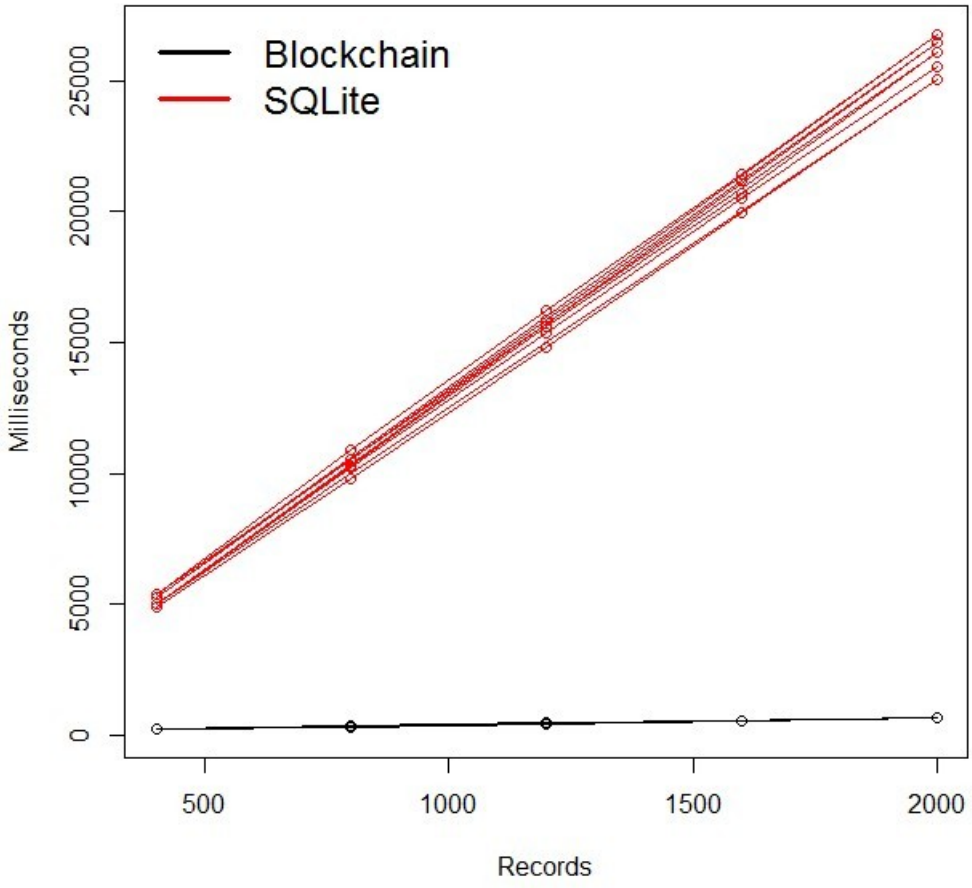
Source: Authors

Table 2 – Retrieve Data Response Time in Milliseconds

	1		2		3		4		5	
Records	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite	Blockchain	SQLite
400	111	134	134	132	150	141	144	141	151	135
800	168	234	186	244	202	254	201	252	204	257
1200	245	334	262	344	273	365	282	365	275	391
1600	318	450	334	460	347	491	353	487	343	507
2000	387	568	417	579	419	596	423	621	410	630

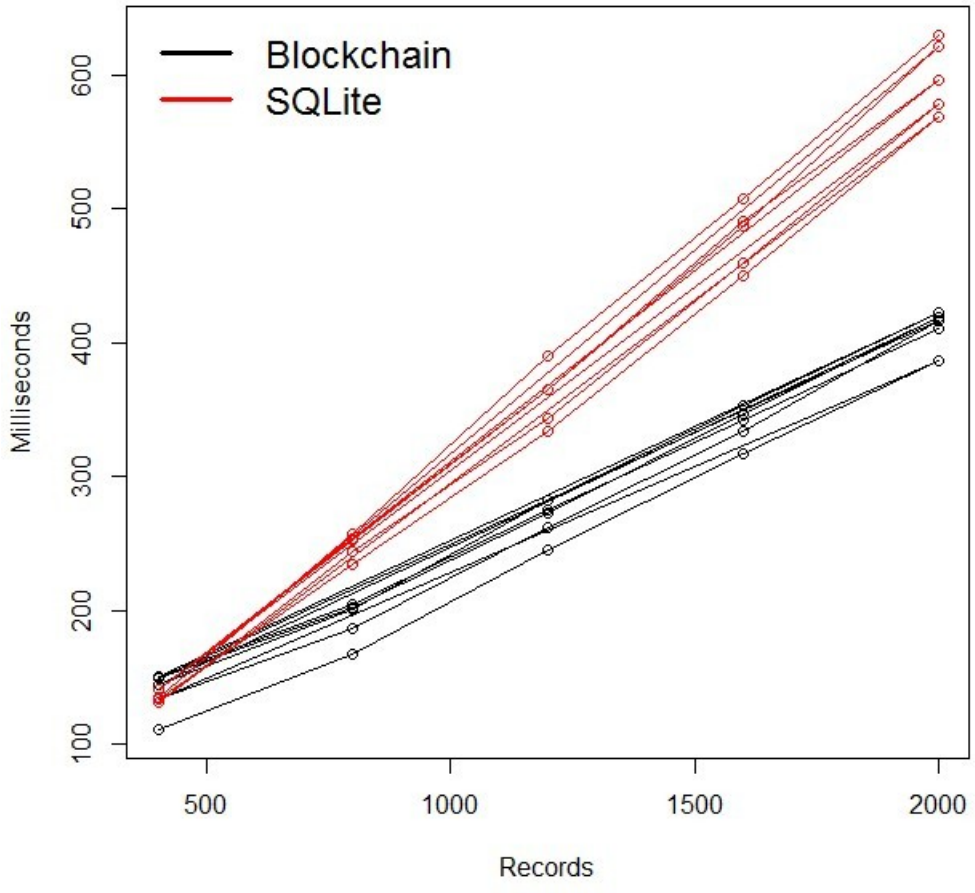
Source: Authors

Figure 7 – Comparison Graph for Storing Data: Blockchain x SQLite



Source: Authors

Figure 8 – Comparison Graph for Retrieving Data: Blockchain x SQLite



Source: Authors

4 Conclusion

Given the observed results, the analysis of the data presented in the graphs, it is concluded that the performance of BlockChain is better than SQLite. Blockchain also has security attributes like data immutability, distributed architecture that avoids a single point of failure. In stored data readings, Blockchain had a little smaller response time than SQLite. In the data recording process, the response time of Blockchain was very much lower when compared to SQLite. These results demonstrate that Blockchain surpasses SQLite in a stand-alone server environment.

In continuation of this study, the response time measurements will be focused on a multi-point network to test a Blockchain network against a cluster of databases.

5 References

Agbo, C. C., Mahmoud, Q. H., & Eklund, J. M. (2019, June). Blockchain technology in healthcare: a systematic review. In Healthcare (Vol. 7, No. 2, p. 56). Multidisciplinary Digital Publishing Institute.

bin Uzayr, S., Cloud, N., & Ambler, T. (2019). Knex and Bookshelf. In JavaScript Frameworks for Modern Web Development (pp. 377-426). Apress, Berkeley, CA.

Bovério, M. A., & da Silva, V. A. F. (2018). Blockchain: uma tecnologia além da criptomoeda virtual. *Revista Interface Tecnológica*, *15*(1), 109-121.

Fusco, F., Lunesu, M. I., Pani, F. E., & Pinna, A. (2018, September). Crypto-voting, a Blockchain based e-Voting System. In *KMIS* (pp. 221-225).

Gonczol, P., Katsikouli, P., Herskind, L., & Dragoni, N. (2020). Blockchain implementations and use cases for supply chains-a survey. *Ieee Access*, *8*, 11856-11871.

Gürsoy, G., Brannon, C. M., & Gerstein, M. (2020). Using Ethereum blockchain to store and query pharmacogenomics data via smart contracts. *BMC medical genomics*, *13*, 1-11.

Khatoon, A. (2020). A blockchain-based smart contract system for healthcare management. Electronics, 9(1), 94.

Michael, J., Cohn, A. L. A. N., & Butcher, J. R. (2018). Blockchain technology. *The Journal*, *1*(7).

Miraz, M. H., & Ali, M. (2018). Applications of blockchain technology beyond cryptocurrency. arXiv preprint arXiv:1801.03528.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.

Pop, D. P., & Altar, A. (2014). Designing an MVC model for rapid web application development. Procedia Engineering, 69, 1172-1179.

Robertson, J. E. (2018). Developing a Decentralised Game Using Blockchain Technology.

Shah, H. (2017). Node. js challenges in implementation. Global Journal of Computer Science and Technology.

Yano, I. H., CASTRO, A. D., CANÇADO, G. D. A., & da SILVA, F. C. (2020). Storing data of sugarcane industry processes using blockchain technology. In *Embrapa Informática Agropecuária-Artigo em anais de congresso (ALICE)*. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 40., 2020, Foz do Iguaçu. Contribuições da engenharia de produção para a gestão de operações energéticas sustentáveis: anais. Rio de Janeiro: ABEPRO, 2020.

6 Acknowledgments

The authors thank Embrapa/Coplacana/CNPq and Embrapa/Coplacana/FAPED to grant scholarships for Renato Mota Ruiz and Julio Cezar Souza Vasconcelos respectively.