Experiment - Ownership using Persmissioned Blockchains

Robert Diebels

As part of a Master graduate project at the Universiteit van Amsterdam

May, 2017

Abstract

Index terms— thesis, permissioned blockchains, experiment, design, setup

1 Introduction

In 2005 [5, p. 736] a survey on the usage of controlled experiments in Software Engineering (SE) was conducted. It was found that 103 articles out of 5453 articles examined contained some form of controlled experiment.

2 Experimental design

This experiment evaluates the validity of the graduate projects' hypothesis. This section specifies the graduate projects experimental design used for evaluation and which studies used in determining the design.

Hypothesis Blockchains outperform current ownership-validation systems in terms of verification and usage-permission granting.

2.1 Experimentation

Evaluating hypothesis validity is achieved by measuring aspects of blockchain implementations determining their performance. The experiment defined in [1, p.64] is used as the basis for the graduate projects' experimental design.

As in [1] the experiment encompasses the evaluation of (1) an implementation of the use case without faults, (2) a fault-tolerance implementation where every 3 seconds a randomly selected validating node is crashed, (3) and a network-latency tolerance implementation, where time-outs are introduced before every read and write.

2.2 Variables

Independent The following independent variables are part of the experiment. A number of variables depend on external factors. I_{min_val} is dependent on the minimum number of validating nodes demanded by the protocol of a blockchain implementation and transaction-size is dependent on the Minimum Viable Data (MVD) of the use-case.

1. CPU speed (GHz),
2. RAM (GB),
3. Number of validating nodes (N_{val}) (I_{min_val}, 64),
4. Number of faulty nodes (N_{fault}),
5. Block-size (MB) (128, 32768),
6. Transaction-size (KB) (MVD),
7. Run-time (200 blocks)

Dependent The following dependent variables are to be measured for performance evaluation.

1. Throughput, or transactions per second (Tx/s).
2. Latency, block-formation per second (Bf/s).

Possible confounding factors Network latency? ...

2.3 Implementations

In [3] recommendations are made for software research to ensure their algorithms are reproducible. These recommendations and the suggestion made in [2] to use docker containers serve as the base criteria for an implementations inclusion into the experiment. A description of all recommendations from [3] can be found in Table 1.

The implementations found in [4] are evaluated and included in the experiment if found they fulfill the criteria.

2.4 Use-case

Entities ...

MVD ...

Transaction-types ...

2.5 Deployment

Hardware ...

Containerization All blockchain implementations are containerized using docker and placed in the Docker Hub. Container images will be made available at `https://hub.docker.com/u/robertdiebels/`.

3 Analysis

This section explains where, how and what data will be collected and how that data will be processed and analyzed.

3.1 Data collection

Location Data related to both independent and dependent variables will be gathered from each node participating in the block-chain network.

Procedure

Types

3.2 Data processing

Output Gathered data-dumps are processed on each node and transformed to a JSON output format. A full verion of the output format can be found in Appendix B.

3.3 Approach

Samples Samples are collected from 3 or more permissioned blockchains.

Pairing Samples are unpaired as the samples are taken from different permissioned blockchain implementations which are unrelated an of which sample sizes may vary.

Gaussian To determine if the samples have a normal-distribution (Gaussian) the D'Agostino-Pearson normality test will be used.

Test Depending on the results of the normality test either a Kruskal-Wallis or a one-way ANOVA test will be applied to determine the difference in means to compare one independent variable. Comparing two independent variables will be done with either a two-way ANOVA or a Scheirer-Ray-Hare test.

References

[1] Ethan Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. PhD thesis, 2016.

[2] Jürgen Cito, Vincenzo Ferme, and Harald C Gall. Using docker containers to improve reproducibility in software and web engineering research. In *International Conference on Web Engineering*, pages 609–612. Springer, 2016.

[3] Tom Crick, Benjamin A Hall, and Samin Ishtiaq. " can i implement your algorithm?": A model for reproducible research software. *arXiv preprint arXiv:1407.5981*, 2014.

[4] Robert Diebels. Literature survey - ownership using persmissioned blockchains. apr 2017.

[5] Dag IK Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, N-K Liborg, and Anette C Rekdal. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering*, 31(9):733–753, 2005.

Appendices

A Recommendations

Recommendation	Description	Used as criteria
I	"[a paper] must describe the algorithm in such a way that it is implementable by any reader of that algorithm.". This recommendation is interpreted as the state of the documentation of an implementation.	Yes
II	"We recommend that code be published under an appropriate open source license...". This recommendation is interpreted as an implementation being open-sourced and open to modification for personal use.	Yes
III	"[we] recommend that basic programming and computational skills are taught as core at undergraduate and postgraduate level.". This recommendation was not used as an inclusion criteria. The aim the experiment is to compare implementations. Educational backgrounds are not considered.	No
IV	"The use of a principled, high-level programming language in which to write your software helps hugely with the maintainability, robustness and openness of the software produced.". Interpreted as an implementation being written in a commonly used high-level language in the industry. [Java, Go, C#, C++]	Yes
V	"Testing new complex scientific software is difficult – until the software is complete, unit tests may not be available. You should thus aim to link to/from publicly-shared code: shared code is inherently more test-able.". Interpreted as an implementations test infrastructure being clearly defined.	Yes

VI	"Code should always include links to papers publishing key algorithms and the code should include explicit relationships to other projects on the repository (i.e. Project B was branched from Project A).". Not used. Evaluating proper referencing in implementation code is not within the experiments' scope.	No
VII	"Providing the source code of the tool helps, of course. But you must also provide details of precisely how you built and wrote the software.". Interpreted as the presence of docker containers and build-documentation.	Yes
VIII	"Avoid creating new representations when common formats already exist. Use existing extensible internationally standardised representations and formats to facilitate sharing and re-use." Not used. Evaluation of representations within code is not within the scope of this experiment.	No
IX	"Benchmarks should be public. They should allow anyone to contribute, implying that the tests are in a standard format." Benchmarks are optional as the experiment sets out the evaluate performance in a self-defined use-case.	Optionally
X	"The Web and the cloud really do open up a whole new way of working...". Interpreted as deploy-able to cloud infrastructure.	Yes

Table 1: Definitions of recommendations by [3] and their usage as inclusion criteria.

B Output format

```
{
    "runtime": {
                    "start": "<time-stamp in milliseconds>",
                    "stop": "<time-stamp in milliseconds>"
                },
    "blocks":  [
                    {
                        "hash":       "<Hash>",
                        "creation":   "<time-stamp in milliseconds>",
                        "transactions": [
                                            {
                                                "type": "<String>"
                                            }
                                        ]
                    }
                ]
}
```

Listing 1: JSON Output