# FMKe: a Real-World Benchmark for Key-Value Data Stores

Gonçalo Tomás
NOVA LINCS & DI, FCT,
Universidade NOVA de Lisboa,
Portugal
ga.tomas@campus.fct.unl.pt

Peter Zeller
University of Kaiserslautern,
Germany
p_zeller@cs.uni-kl.de

Valter Balegas
NOVA LINCS & DI, FCT,
Universidade NOVA de Lisboa,
Portugal
v.sousa@campus.fct.unl.pt

Deepthi Akkoorath
University of Kaiserslautern,
Germany
akkoorath@cs.uni-kl.de

Annette Bieniusa
University of Kaiserslautern,
Germany
bieniusa@cs.uni-kl.de

João Leitão
NOVA LINCS & DI, FCT,
Universidade NOVA de Lisboa,
Portugal
jc.leitao@fct.unl.pt

Nuno Preguiça
NOVA LINCS & DI, FCT,
Universidade NOVA de Lisboa,
Portugal
nuno.preguica@fct.unl.pt

## ABSTRACT

Standard benchmarks are essential tools to enable developers to validate and evaluate their systems' design in terms of both relevant properties and performance. Benchmarks provide the means to evaluate a system with workloads that mimics real use cases. Although a large number of benchmarks exist for database system, there is a lack of standard benchmarks for an increasingly relevant class of storage systems: geo-replicated key-value stores providing weak consistency guarantees. This has led developers and researchers to rely on ad-hoc tools, whose results are both hard to reproduce and compare.

In this paper, we propose the first standardized benchmark specially tailored for weakly consistent key-value stores. The benchmark, named FMKe, is modeled after a real application: the Danish National Joint Medicine Card. The benchmark is scalable, it can be parameterized to emulate a large number of access patterns, and it is also highly flexible, enabling its application on systems that offer different consistency guarantees and mechanisms.

## CCS CONCEPTS

• **General and reference** → *Evaluation*;

## KEYWORDS

Benchmark, Key-Value Store


*PaPoC'17, Belgrade, Serbia*

DOI: http://dx.doi.org/10.1145/3064889.3064897

**ACM Reference format:**
Gonçalo Tomás, Peter Zeller, Valter Balegas, Deepthi Akkoorath, Annette Bieniusa, João Leitão, and Nuno Preguiça. 2017. FMKe: a Real-World Benchmark for Key-Value Data Stores. In *Proceedings of PaPoC'17, Belgrade, Serbia, April 23, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3064889.3064897

## 1 INTRODUCTION

Standard benchmarks provide a uniform way for evaluating and comparing different systems. The most used benchmarks for databases (e.g. TPC-C [1], TPC-W [2], etc.) model realistic applications. As such, this type of benchmarks is expected to provide a more realistic performance evaluation than synthetic benchmarks, where individual operations are generated randomly according to some distribution defined in the workload.

While the TPC-* benchmarks work well for the evaluation of relational, strongly consistent database systems, they are a bad fit for evaluating eventually consistent key-value stores. The main issue is that they do not reflect the way key-value stores are typically used. For example, some aggregation queries in TPC-W are very expensive to implement on top of a key-value data model respecting the specification, which can render the value of experiments useless.

Given the need of evaluating their systems, many system developers opt for implementing their own version of popular applications, like Twitter, FusionTicket [3], or even TPC-C/TPC-W. Yet, there is no standardized way of comparing these ad-hoc implementations due to different codebases and the lack of a common specification. The Yahoo! Cloud System Benchmark (YCSB) [4] addresses this problem by providing a set of standard benchmarks that can be used to evaluate key-value stores. However, the operations of the benchmark consists of simple read/write operations, while real-life applications often use more complex access patterns.

In this paper, we present FMKe, a new application benchmark tailored to the evaluation of key-value stores providing weak consistency. It is based on a subsystem of the Danish National Healthcare

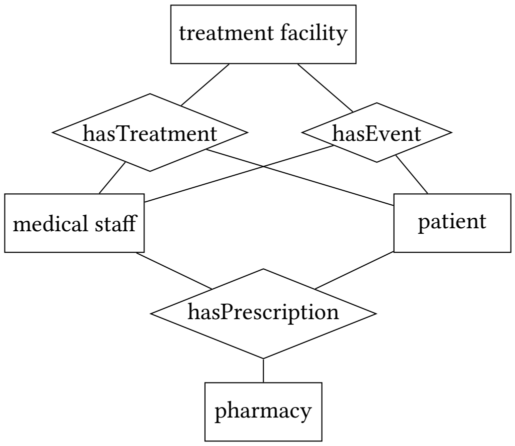


**Figure 1: Simplified ER diagram that models FMKe**

System (FMK, Fælles Medicinkort), and the workload is defined based on real-life statistics obtained from this production system. FMKe models the handling of prescriptions assigned to patients. This information is accessed concurrently by multiple entities, including medical facilities, such as hospitals, and pharmacies.

The benchmark can be used to evaluate any storage system providing weak consistency guarantees, but also includes variants for evaluating advanced database features, such as highly available transactions.

In the remainder of this paper we present the data model of the FMKe benchmark, describe the operations of the workload, and discuss its implementation and evaluation on top of Antidote [5], a key-value store that supports highly available transactions under geo-replication.

## 2 FMKE BENCHMARK

In this section we introduce the FMKe benchmark. This benchmark was designed based on a real system that operates at National level in Denmark, which is used to manage medical information, prescriptions, and treatment information for the population of the country. Our benchmark focus on the particular aspect of prescription management and does not include all operations available the the real FMK system.

When designing FMKe, we did not have access to the exact data model used by the original FMK application. Thus, we have designed the data model (presented below) based on the operations provided by FMK to manage prescriptions.

### 2.1 Data Model

FMKe[6] is a system that manages medical information about patients. In this domain there is a need to keep records for pharmacies, treatment facilities, patients, prescriptions, patient treatments, and medical events (such as taking medicine or medical prognosis updates). The benchmark includes a set of application-level operations, each one of them leading to the execution of a sequence of read and update operations on these entities. The set of hospitals, pharmacies, patients and medical staff act as static entities in the benchmark, so records of these entities can be populated in data stores prior to the benchmark execution (and these can be scaled in number to fit the needs of the system under evaluation). Figure 1 presents a simplified view of the main entities.

### 2.2 Workload Operations

Table 1 shows the operations performed by the benchmark together with their relative frequency. We have developed two variants of the benchmark with different data layouts. The non-normalized variant follows the strategy of storing data in a denormalized form, which allows to serve most reads without joining data from different records. The other variant stores data in a normalized form, which leads to smaller object sizes, but requires to join data from multiple records when reading. Table 1 includes the respective number of reads and writes for the operations in the two implementations. We now describe the individual operations.

**Create prescription** registers a new prescription record that is associated with a patient, medical staff and pharmacy. After creation the prescription is considered to be *open* (i.e. it was not yet handled by a pharmacy in order to deliver medicine to the patient).
**Process prescription** changes the state of a prescription record to signal that it has been handled, so it transitions to the *closed* state.
**Get staff prescriptions** returns all prescription records that are associated with a specific medical staff member.
**Get pharmacy prescriptions** returns all prescription records associated with a pharmacy.
**Get processed prescriptions** returns only prescriptions that have been handled (closed).
**Get prescription medication** returns the medication for a specific prescription record.
**Update prescription medication** changes the medication for a prescription that has not been processed.

### 2.3 Benchmark Characterization

FMKe models as closely as possible the real production system FMK. Benchmark operations and their frequency have default values based in statistics from the real-world system. The benchmark is naturally affected by the number of entities in the data store on which these operations are performed.

Table 2 presents the values for the parameters used in the results presented in the next section – the numbers of hospitals and pharmacies is close to the real numbers, while for patients and doctors the number is between $\frac{1}{3}$ and $\frac{1}{5}$ of the real value. In those experiment, we present results in three different settings where we vary the number of data centers of the deployment.

The benchmarks can be parameterized to use value that match the needs of the system being evaluates, either by changing the number of entities used as well as changing the frequency of each operation.

## 3 PRELIMINARY EXPERIMENTAL RESULTS

To show the feasibility of the benchmark, we present some preliminary performance results. To this end we have implemented an initial prototype of the benchmark, composed by three components:

**Clients** The clients issue HTTP/REST requests to the application server, encoding the application operations (section

| Operation | Frequency | Non-normalized # reads | Non-normalized # writes | Normalized # reads | Normalized # writes |
|---|---|---|---|---|---|
| Get pharmacy prescriptions | 27% | 1 | 0 | N | 0 |
| Get prescription medication | 27% | 1 | 0 | 1 | 0 |
| Get staff prescriptions | 14% | 1 | 0 | N | 0 |
| Create prescription | 8% | 5 | 4 | 5 | 4 |
| Get processed pharmacy prescriptions | 7% | 1 | 0 | N | 0 |
| Process prescription | 4% | 4 | 4 | 1 | 1 |
| Update prescription medication | 4% | 4 | 4 | 1 | 1 |

**Table 1: Number of read and write operations per FMKe operation. For some operations in the normalized variant the number of reads depends on the current number of prescriptions associated to pharmacy, staff, etc. (denoted by $N$); this number varies over time.**

| Entity | Number |
|---|---|
| Patients | 1,000,000 |
| Hospitals | 50 |
| Pharmacies | 300 |
| Doctors | 5,000 |

**Table 2: Number of entities for a workload targeted at performance evaluation**

2.2). This module is implemented using Basho Bench [7], an open source benchmarking framework.

**FMKe application server** The FMKe application server receives client requests, and for each application operation issues a number of operation to modify the state of the database.

**Database** The data of the benchmark is stored in the database. In our current prototype, we only support Antidote [5].

We ran our experiments in the Amazon Web Service (AWS) infrastructure. Each data center instance consists of four m3.xlarge machines running the Antidote database servers, four m3.xlarge machines running the FMKe application server and four m3.xlarge running the Basho Bench workload generator. A m3.xlarge machine has 4 vCPUs, 15GB of memory and 80 GB of SSD disk. We used the Ireland, Frankfurt and N. Virginia AWS data centers, with the following mean round-trip-time between machines in those data centers: Ireland-Frankfurt: 22.4$ms$; Ireland-N.Virginia: 84.9$ms$; Frankfurt-N.Virginia: 89.7$ms$. The mean round-trip-time between two machines inside a DC was 0.55$ms$.

Figure 2 shows a throughput-latency plot for the FMKe benchmark on the Antidote system [5]. The plots shows measurements under three different deployments where we vary the number of data centers in each deployment. We based the measurements on a version of FMKe with normalized data layout. The results show that Antidote scales linearly with the number of DCs. The reason for this is that the majority of operations in the workload are read-only. As read-only operations involve only a single DC in Antidote, they do not generate any additional load on the other DCs. Operations that update the database generate additional load when forwarding updates, but the efficient mechanism for update propagation used

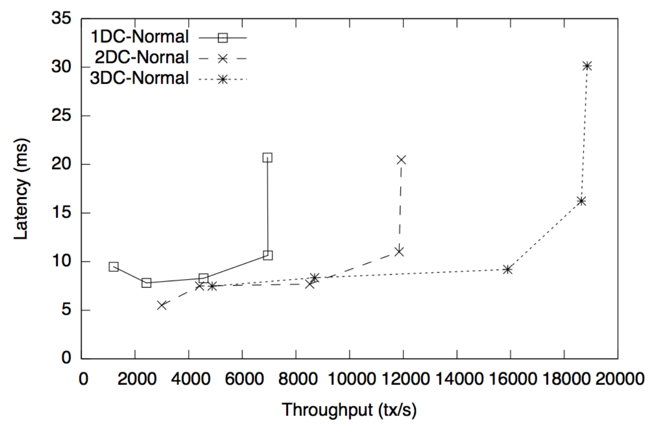


**Figure 2: Antidote performance comparison with varying number of data centers.**

in Antidote keeps this additional load low, allowing the throughput to almost double when we add the second DC.

Figure 3 shows the detailed results for a single experiment, where it is possible to observe the evolution of throughput and latency during the complete experiment. These graph are generated by Basho Bench, and are very useful to understand the behavior of the system as the database size increases.

## 4 CONCLUSION AND FUTURE WORK

In this paper we introduced a new benchmark for data stores providing weak consistency, which is modeled after a wide-area healthcare production system for managing medical prescriptions. We briefly presented the data model and operations for this benchmark. We have described our initial prototype and reported preliminary performance results obtained with Antidote database.

As next step we plan to provide a precise specification of the FMKe operations and their functional requirements. From this specification we will derive a reference implementation of the benchmark with bindings for multiple languages. Further, we will define a set of tests that developers can run to assess the consistency and availability properties of their system. For instance, these tests would allow checking whether operations executed atomically, or if the

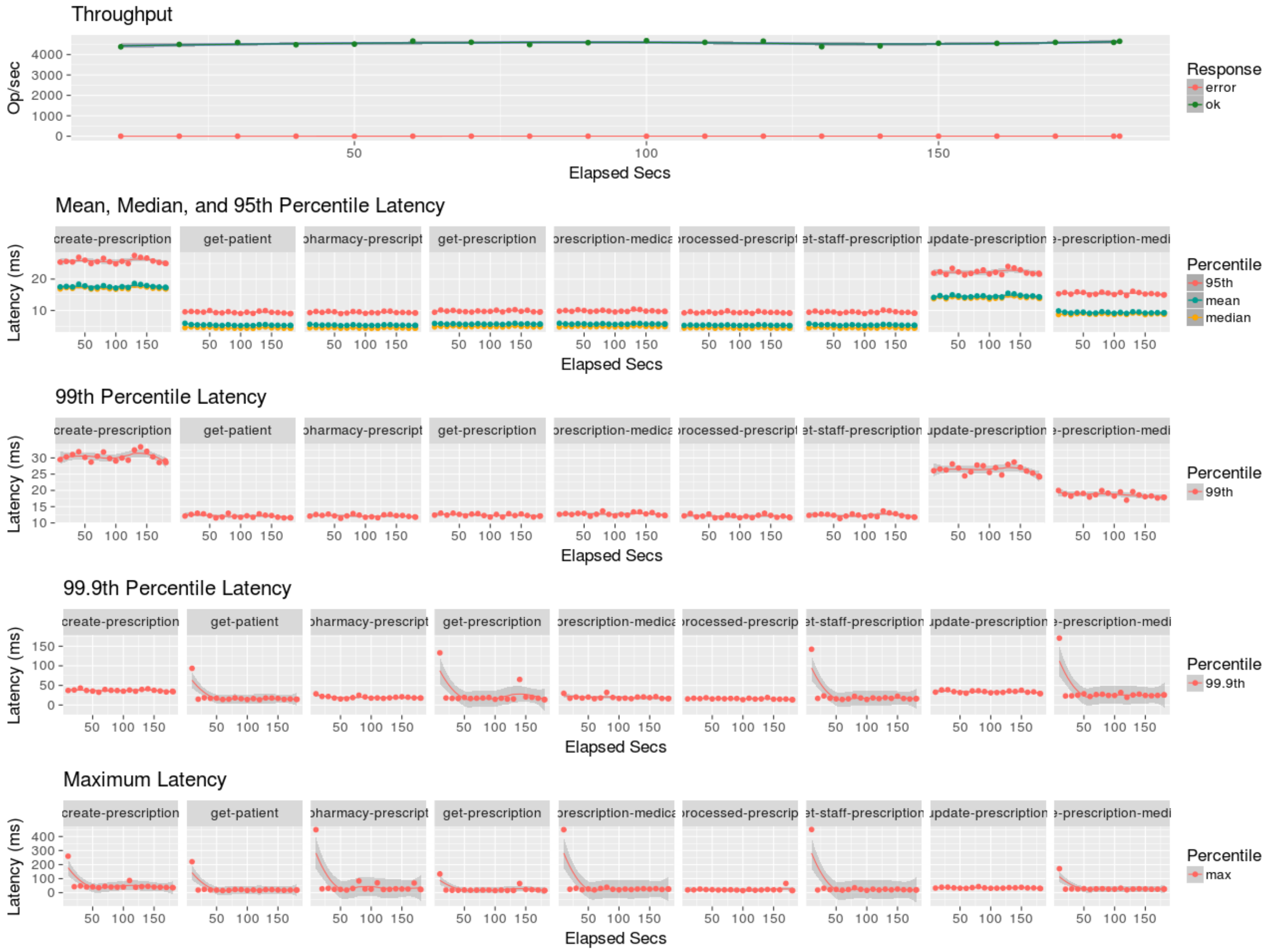


**Figure 3: Results for a single experiment (1 DC, 32 clients).**

system provides causality. We also aim for mechanisms to measure data staleness, which is a relevant trade-off for storage systems providing weak consistency guarantees and high availability.

*Acknowledgements.* This work was partially supported by FCT/MCTES: NOVA LINCS project (UID/CEC/04516/2013) and the European Union, through projects SyncFree (grant agreement number 609551) and LightKone (grant agreement number 732505). We would like to thank Kresten Thorup (Trifork) for his help in the design of the benchmark.

## REFERENCES

[1] The Transaction Processing Performance Council, Benchmark C. http://www.tpc.org/tpcc/default.asp. Accessed: 2017-02-15.
[2] The Transaction Processing Performance Council, Benchmark C. http://www.tpc.org/tpcw/. Accessed: 2017-02-15.
[3] Fusion Ticket Solutions Limited. https://github.com/fusionticket. Accessed: 2017-02-16.
[4] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.
[5] AntidoteDB. http://antidotedb.eu. Accessed: 2017-02-15.
[6] FMKe code repository. https://github.com/goncalotomas/fmke. Accessed: 2017-02-15.
[7] Basho Bench. https://docs.basho.com/riak/kv/2.2.0/using/performance/benchmarking/. Accessed: 2017-02-15.