Improving Kieker's Scalability by Employing Linked Read-Optimized and Write-Optimized NoSQL Storage

Armin Moebius
a.moebius@ibak.de
IBAK GmbH & Co. KG, Kiel, Germany

Sven Ulrich
s.ulrich@ibak.de
IBAK GmbH & Co. KG, Kiel, Germany

Abstract

Kieker's monitoring output can be persistently saved into logs by utilizing relational databases or file systems. Currently, there is no support for noSQL storage. As part of our Regression Benchmarking Execution Environment (RBEE) we introduce a self-contained system offering noSQL storage capabilities and acting as gateway between Kieker and RBEE.

We show, how polyglot persistence can increase Kieker's scalability by employing separate read-optimized and write-optimized noSQL storage. For this purpose we extend Kieker to store its monitoring output in Apache Cassandra, which is a write-optimized wide-column noSQL database. For the analysis of the generated monitoring output we are using ElasticSearch, a read-optimized document store noSQL storage. We are interlinking read-optimized and write-optimized noSQL storage within RBEE. To ensure scalability, we are employing a container infrastructure. The mentioned noSQL storages and the linker are operating within one single Docker container which scales horizontally.

For generating reference values, we instrument a Java SE application with Kieker's file system writer and measure throughput and method's execution times. Consecutively, we instrument the same Java SE application with our Apache Cassandra writer and measure throughput and method's execution time. Finally, we will compare measurement results of Kieker's file system writer with the measurement results of our Apache Cassandra writer.

1 Introduction

Kieker [9, 7] allows saving its monitoring logs by utilizing relational databases as well as file systems. There is no support for utilizing noSQL [12] storage. Our Regression Benchmarking Execution Environment (RBEE) [15] uses Kieker's monitoring output as input for its regression benchmarking process. Therefor, we extend Kieker to handover its monitoring output to RBEE's monitoring log. RBEE consists of several self-contained Systems (SCS) [16]. In the remainder of this paper, we will describe the RBEE monitoring log SCS in more detail.

2 Kieker Extension

For enabling Kieker to directly store its monitoring output in Apache Cassandra, we extend Kieker's `SyncDbWriter` and `AsyncDbWriter` classes [8]. These classes provide access to different databases. We built four additional classes and extend the available abstract classes and interfaces from Kieker. These classes are Apache Cassandra specific, but work with any monitoring record. At first, there is the `CassandraDb` class, which is a basic implementation to access Apache Cassandra with the help of the DataStax Java driver [2]. It provides methods for creating tables and inserting data. Further, there is the `CassandraSyncDbWriter` class, which extends Kieker's `AbstractMonitoringWriter`. We override the method newMonitoringRecord(), which is called from the Kieker framework, when a new record is available. `CassandraAsyncDbWriter` is the asynchronous implementation of our writer which extends Kieker's `AbstractAsyncWriter`. The worker threads, which are persisting the monitoring data, were built with the help of the class `AbstractAsyncThread` in our class `CassandraWriterThread`. In order to get the Kieker framework to work with our implementation we had to add some configuration into the `kieker.monitoring.properties` file. Support for Apache Cassandra will be integrated in one of the next Kieker releases.

3 Data Storage

RBEE requires a Docker [3] container infrastructure and implements polyglot persistence [6]. Its monitoring log is a containerized system, which scales horizontally. The data storage is based on two linked noSQL storages. Kieker writes its monitoring output directly to Apache Cassandra [1], which is a write-optimized wide-column noSQL database. When Kieker has finished generating monitoring output, this data is pushed asynchronous to ElasticSearch [5], a read-optimized document store noSQL storage which resides in the same container. For this purpose, we provide our `rbee_cte` Java-based command-line tool [13], which provides import and export algorithms within our RBEE monitoring log container image [14]. As database driver we use the one provided

by DataStax [2]. For ElasticSearch, we employed the standard Java API. Our `rbee_cte` command-line tool needs the IP of both, Cassandra and ElasticSearch as parameter. Furthermore, the keyspaces and table names, which have to be transferred to ElasticSearch must be specified. If all necessary parameters are submitted, `rbee_cte` tries to connect to both. After that, the transfer starts with gathering the required data out of Cassandra for each keyspace and each table. For this, we use a result set with a size of 10,000 entries. The next step is to convert each row into a `Map<String, Object>`. The keys of the map are the column names of the table. If the data is converted it is send as bulk request to ElasticSearch. For this we use the BulkProcessor of the Java API with a bulk size of 10,000 and three concurrent request. So, a maximum of 30,000 records at once were transferred.

4 Test Environment

Our system under test (SUT) is part of a Java-based employee work time registration system. In more detail, we focus on the process of logging employee's working times. The SUT provides a Swing-based GUI which is presented to the employee on a touch-screen.

For generating Kieker monitoring output, we employ Marathon's [10] open source edition for automated Java Swing GUI testing. We created an exploratory test script, which enables us to run the same tests multiple times and exactly reproducible. All tests were executed on identical Hardware. In more detail, a HP Workstation Z420 with Intel Xeon E5-2670 CPU, 32 GB RAM, a SATA hard drive and 1 GBit/s network connectivity.

For evaluating scalability, performance and monitoring overhead of our approach, we set up different scenarios. So, there is a non instrumented setup (a) for generating reference values. In the following, we set up Kieker file based monitoring (b) as well as a non containerized environment with read optimized and write optimized data storage but no containerization (c). In order to examine the impact of containerization, we set up the following scenarios additionally:

(d) 1 container on 1 container host

(e) 6 containers on 1 container host

(f) 6 containers on 3 container hosts with 2 containers per container host

All setups were executed on identical hardware. In more detail, HP Proliant DL380 G7 Servers with two Intel Xeon X5650 CPUs, 144 GB RAM, 16 SAS 10K HDDs and 1 GBit/s network connectivity.

5 Evaluation

Within the defined scenarios, we gathered measurement data by executing the relevant SUT's methods with Marathon. Our exploratory test script is identical for each scenario. For each scenario the test script is executed for 3,000 times. Scenario (a) is used for determining reference values. There is no Kieker instrumentation. In scenario (b), we instrumented the SUT and used Kieker's `AsyncFsWriter` for storing the monitoring output in the local file system.

We set up Cassandra and ElasticSearch on Ubuntu Server [17] in scenario (c). There is no virtualization and no containerization in this scenario. Kieker sends its monitoring output directly to the Cassandra instance using Cassandra's native transport.

In scenario (d) we employed Docker as container infrastructure on top of Ubuntu Server. Further, we employed our RBEE monitoring log Docker image [14] and executed one single container instance.

In addition to scenario (d) we increased the number of RBEE monitoring log container instances in scenario (e) from 1 instance to 6 instances. All container instances were still executed on one container host. All Cassandra instances were part of one ring topology with a replication factor of 2. All ElasticSearch instances were part of one cluster.

Finally, we used three container hosts for operating six container instances in scenario (f). Each container host operates two container instances. For container cluster management, we employed Docker Swarm [4] and used the same Docker RBEE monitoring log image as used in scenario (d).

Figure 1 shows the determined execution times for the SUT running our Marathon test script for each scenario. Hence, there is no monitoring enabled, scenario (a) has the least execution time.

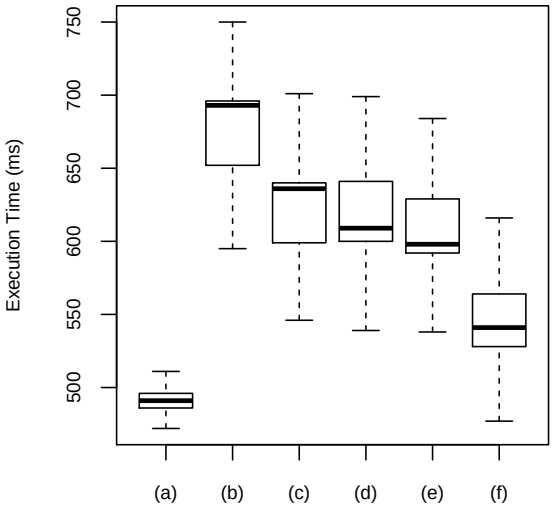

Figure 1: Execution Time of SUT's Methods

In scenario (b) we used Kieker monitoring with data storage on local hard disk. Compared to the scenarios (c-f), scenario (b) has the longest execution times. Scenario (c) shows, that utilizing write-optimized noSQL storage alone decreases execution time. Using containerization on one single container host in scenarios (d-e) decreased the execution times further. Finally, employing multiple container hosts in

scenario (f) has the highest impact on the execution times.

For linking read-optimized and write-optimized noSQL storage, we provide our Java-based command-line tool `rbee_cte` within the RBEE monitoring log container image. In scenarios (d-f), we traced transfer times for transferring data from Apache Cassandra to ElasticSearch for several amounts of data. Figure 2 shows the traced transfer times. In all of the scenarios (d-f) the transfer time scales proportional to the number of transferred records. Employing multiple container hosts in scenario (f) has the most impact on transfer time.

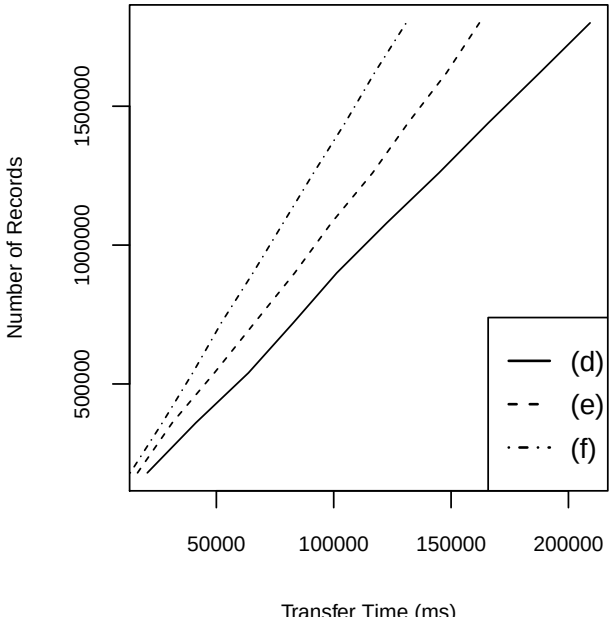

Figure 2: Transfer Time Between Read-Optimized and Write-Optimized NoSQL Storage

All evaluation data is published on Zenodo [11]. Our RBEE monitoring log Docker container image is available on DockerHub [14]. The source code of our Java-based command-line tool (`rbee_cte`) can be found on GitHub [13]. For further information visit our RBEE website [15].

6 Conclusion

Employing polyglot persistence and utilizing a container infrastructure enables us to improve Kieker's scalability. We have shown, how the usage of write-optimized noSQL storage reduces execution times of a SUT's methods. Containerization and Clustering led to further decreased execution times. We analyzed the impact of a asynchronous connection between our read-optimized and write-optimized storage. Furthermore, we evaluated the throughput and performance of our provided import and export algorithms, too. When summing up the minor execution times and the time required for asynchronous transport between read-optimized and write-optimized storage, the SUT's total run time is still reduced. Finally, our RBEE monitoring log, built upon Docker, Cassandra and ElasticSearch improves Kieker's data storage scalability.

References

[1] *Apache Cassandra*. `http://cassandra.apache.org/`.

[2] *DataStax Cassandra Driver*. `https://github.com/datastax/java-driver/`.

[3] *Docker*. `http://www.docker.com/`.

[4] *Docker Swarm*. `https://www.docker.com/products/docker-swarm`.

[5] *ElasticSearch*. `https://www.elastic.co/`.

[6] Wilhelm Hasselbring. "Microservices for Scalability: Keynote Talk Abstract". In: *ICPE 2016*, pp. 133–134. URL: `http://eprints.uni-kiel.de/31829/`.

[7] André van Hoorn, Jan Waller, and Wilhelm Hasselbring. "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis". In: *ICPE 2012*, pp. 247–248. URL: `http://eprints.uni-kiel.de/14418/`.

[8] *Kieker Cassandra Extension*. `https://kieker-monitoring.atlassian.net/browse/KIEKER-1452`.

[9] *Kieker Monitoring Framework*. `http://www.kieker-monitoring.net/`.

[10] *Marathon*. `http://marathontesting.com/`.

[11] Armin Moebius and Sven Ulrich. *Data for: Improving Kieker's Scalability by Employing Linked Read-Optimized and Write-Optimized NoSQL Storage*. Aug. 2016. DOI: `10.5281/zenodo.61227`. URL: `http://dx.doi.org/10.5281/zenodo.61227`.

[12] Jaroslav Pokorny. "NoSQL Databases: A Step to Database Scalability in Web Environment". In: *Proceedings of the 13th International Conference on iiWAS*. 2011, pp. 278–283. URL: `http://doi.acm.org/10.1145/2095536.2095583`.

[13] *RBEE Monitoring Log Command Line Tool*. `https://github.com/rbee-dev/mlog/`.

[14] *RBEE Monitoring Log Docker Image*. `https://hub.docker.com/r/rbee/mlog/`.

[15] *Regression Benchmarking Execution Environment*. `http://www.rbee.io/`.

[16] *Self-Contained Systems Architecture*. `http://scs-architecture.org/`.

[17] *Ubuntu Server*. `http://www.ubuntu.com/server`.