Institute of Architecture of Application Systems

Evaluating Caching Strategies for Cloud Data Access using an Enterprise Service Bus

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Steve Strauch

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{santiago.gomez-saez, vasilios.andrikopoulos, frank.leymann, steve.strauch}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{GomezSaez2014,
  author    = {Santiago G{\'o}mez S{\'a}ez and Vasilios Andrikopoulos and Frank
               Leymann and Steve Strauch},
  title     = {Evaluating Caching Strategies for Cloud Data Access using an
               Enterprise Service Bus},
  booktitle = {Proceedings of the IEEE International Conference on Cloud
               Engineering (IEEE IC2E 2014)},
  year      = {2014},
  pages     = {289--294},
  doi       = {10.1109/IC2E.2014.49},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

2014 IEEE International Conference on Cloud Engineering

Evaluating Caching Strategies for Cloud Data Access using an Enterprise Service Bus

Santiago Gómez Sáez, Vasilios Andrikopoulos, Frank Leymann, Steve Strauch
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Stuttgart, Germany
{santiago.gomez-saez, vasilios.andrikopoulos, frank.leymann, steve.strauch}@iaas.uni-stuttgart.de

***Abstract*—Nowadays different Cloud services enable enterprises to migrate applications to the Cloud. An application can be partially migrated by replacing some of its components with Cloud services, or by migrating one or multiple of its layers to the Cloud. As a result, accessing application data stored off-premise requires mechanisms to mitigate the negative impact on Quality of Service (QoS), e.g. due to network latency. In this work, we propose and realize an approach for transparently accessing data migrated to the Cloud using a multi-tenant open source Enterprise Service Bus (ESB) as the basis. Furthermore, we enhance the ESB with QoS awareness by integrating it with an open source caching solution. For evaluation purposes we generate a representative application workload using data from the TPC-H benchmark. Based on this workload, we then evaluate the optimal caching strategy among multiple eviction algorithms when accessing relational databases located at different Cloud providers.**

Keywords*—*Enterprise Service Bus (ESB); Cache; Relational Databases; Performance Optimization

I. Introduction

The successful introduction of the Cloud computing paradigm generated the need for *Cloud-enabled applications*, i.e. applications that are migrated to the Cloud. Most of the literature on the subject discusses Cloud migration as the re-packaging of applications into one or more virtual machines (VMs). However, the increasing number of available Cloud services on the Platform as a Service (PaaS) level like Google App Engine[1] allow for the partial migration of the application [1]. In this context it becomes possible to migrate only some of the components of the application off-premise (in the Cloud), e.g. its database, or some of the application functionality, e.g. backing up of data, while the remaining of the application remains on-premise.

Focusing in particular on the Data Layer of the application, as defined by Fowler [2], i.e. as the data accessing, manipulation and persistence aspect of the application, there are two different types of migration options: in one type the usually, but not necessarily, relational database management system (RDBMS) used by the application is migrated as a whole to a VM and accessed remotely by the application through conventional means, e.g. SQL commands over a JDBC connection. The other type relies on a predefined (usually RESTful) API provided by a Database as a Service (DBaaS) like Amazon SimpleDB[2].

[1]Google App Engine: http://developers.google.com/appengine/
[2]Amazon SimpleDB: http://aws.amazon.com/de/simpledb/

In any case, accessing the application database over a network, and relying on exogenous to the application resources due to hosting the database off-premise has potentially a significant negative impact on the QoS of the application, e.g. due to unpredictability in the network latency between the application and the data, or to variability in the performance of the Cloud service provider [1]. Furthermore, as discussed more extensively in [1], migrating (any part of) the application to the Cloud most probably involves a degree of adaptation involved. In the best case only a degree of re-wiring may be necessary to redirect the application to the migrated off-premise database. In the worse case, significant changes may be required to the application logic in order to cope with incompatibilities in the way the data is accessed, e.g. when moving to a DBaaS from a traditional RDBMS.

In order to minimize the adaptation effort required, and to mitigate the negative impact on the QoS of the application by migrating its Data Layer to the Cloud, we propose the use of an Enterprise Service Bus (ESB) solution as the means to provide transparent data access to potentially multiple databases migrated to the Cloud. While similar efforts, e.g. [3] have proposed the use of ESBs in accessing data services, they rely on wrapping the database as a Web services and therefore require additional development effort. Furthermore, these approaches do not utilize caching support on the level of data requests in order to cope with network latency, as traditionally supported by middleware solutions. The solution we discuss in the later sections has built-in caching support, allowing for multiple caching strategies to be selected from a list of available ones. Since the efficiency of caching strategies relies heavily on the nature of the application workload [4], a major goal of this work is to provide an evaluation of the efficiency of our proposal for one representative application workload before generalizing the discussion across different loads in the future.

The contributions of this work can therefore be summarized as follows:

- The design and realization of CDASMix, a multi-tenant aware ESB solution with caching support that enables transparent data access to databases both on-premise and off-premise.
- A performance evaluation of our proposal, with the dual purpose of showing the impact of introducing CDASMix to the performance of the application, and identifying the optimal caching strategy for CDASMix

978-1-4799-3766-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IC2E.2014.49

for different deployment options across Cloud service providers.

- A set of initial findings stemming from this evaluation, that can be valuable for related efforts.

The remaining of this paper is structured as follows: Section II discusses the architecture and implementation of our proposal. Section III discusses the methodology, experimental setup and results, and the most important findings of the evaluation of our proposal. Section IV summarizes related work. Finally, Section V concludes with future work.

II. CDASMix: Cloud Data Access Support in ServiceMix

In this section we present *Cloud Data Access Support in ServiceMix (CDASMix)*, an architecture and implementation approach enabling *transparent data access* to relational databases both on-premise and off-premise. We reuse and extend *ESB^{MT}*, a multi-tenant aware ESB solution [5], as the basis for the design and realization of our approach. ESB^{MT} enhances the Apache ServiceMix 4.3 JBI container [3] with multi-tenant communication support for service endpoints deployed in the ESB, and multi-tenant aware dynamic endpoint deployment capabilities. As ESB^{MT} focuses on enabling multi-tenancy at the communication level between application services rather than on application's data access, the remaining of this section describes the extensions realized for CDASMix.

The ESB^{MT} provides a *Web UI* and a *Web services API* in order to offer customization, administration, and management functionalities to its users [5]. The Web services API was extended with a set of operations for configuring the transparent data access. No modifications on the Web UI were required. The Business Logic layer of ESB^{MT} enables role-based access control for administration and management of the *JBI Container Instance Cluster* in the Resources layer [5]. The realization of transparent Cloud data access support in CDASMix does not require architectural modifications on this layer, but extensions for enabling administration and persistence of information related to the tenant's off-premise data stores are needed. The Resources Layer of CDASMix consists of a set of registries and multiple JBI containers which constitute a JBI Container Instance Cluster. CDASMix uses the existing ESB^{MT} registries described in [5]. The service registry database schema has been extended to store Cloud data access management information in a tenant aware manner. SQL statements Routing support required this extension to include both connection-related information and database-specific information, e.g. remote database URL and database schema details respectively.

In the following we focus on the required extensions of each instance of the JBI container cluster in order to enable transparent Cloud data access support (Fig. 1). JBI containers provide integration support with external services via various protocols and message processing facilities such as routing, as in traditional ESB solutions. ESB^{MT}, and by extension CDASMix, is based on the OSGi Framework [4] in order to allow components to be deployed in a loosely coupled manner. The ServiceMix solution at the core of ESB^{MT} is equipped for this purpose with several OSGi components which realize the ESB functionality complying to the JBI specification. For example, the ServiceMix-HTTP Binding Component (BC) supports interactions with external services via HTTP, and the Apache Camel [5] Service Engine (SE) realizes a set of known Enterprise Integration Patterns [6]. The Normalized Message Router (NMR) provides integration support between JBI and OSGi components within the ESB through a message-based interface. JBI components provided by ServiceMix support multiple communication protocols (e.g. HTTP, JMS, etc.) by marshaling and demarshaling protocol-specific messages into a standardized internal Normalized Message Format (NMF) [7]. Nevertheless, database system native communication protocols are not supported, as they are solution-specific.

We focus our approach on providing communication support for a well known and widely used RDBMS, MySQL [6]. The same approach however can be used to provide support for other systems such as PostgresSQL [7]. Figure 1 zooms in the architecture of one ESB instance in the cluster in order to illustrate how transparent message routing and transformation of application data requests was realized for enabling transparent access through one physical endpoint in CDASMix to potentially multiple backend data stores. The developed *MySQL Proxy* is an OSGi component that implements an OSGi- and JBI-compliant version of a Java-based MySQL proxy and serves as the physical endpoint to communicate via the native communication protocol of MySQL. We extended and integrated the existing open source Java-based MySQL Proxy in Myosotis Tungsten Connector[8] for this purpose. The MySQL Proxy is fully integrated with the JBI container via the *Normalized Message Routing (NMR) API*. A *Cache Cluster* ensures high performance of SQL read operations in CDASMix by dynamically generating cache keys based on the information type received in the MySQL Proxy, and by deleting cache records when SQL statements involve data modification. For the realization we reused and extended Ehcache 2.6.0 [9] by integrating it into CDASMix also as an OSGi bundle. The ESB^{MT} Camel SE 2011.11 (denoted as *SMX-Camel-mt* in Fig. 1) provides multi-tenancy and integration support between the JBI environment and the Enterprise Integration Patterns realizations of Apache Camel. A custom Camel component named *CamelcdasmixJDBC* was developed in CDASMix in order to dynamically connect to off-premise data sources via multiple database system communication protocols. The CamelcdasmixJDBC component is accessed through a Camel endpoint in the *SMX-Camel* OSGi component (Fig. 1). The utilization of the SMX-Camel OSGi component enables loading of *CamelcdasmixJDBC* component packages during runtime, e.g. in order to support a new database system.

Focusing on caching, the version of Ehcache currently used supports three popular cache eviction algorithms: *Least Recently Used (LRU)*, *Least Frequently used (LFU)*, and *First In First Out (FIFO)*. *LRU* deletes the element with the last used timestamp, *LFU* the element with the least number of hits, and the *FIFO* algorithm deletes the elements from the cache in the same

[3] Apache ServiceMix: http://servicemix.apache.org
[4] OSGi Version 4.3: http://www.osgi.org/Download/Release4V43/
[5] Apache Camel: http://camel.apache.org
[6] MySQL: http://www.mysql.com
[7] PostgresSQL: http://www.postgresql.org
[8] Tungsten Replicator: http://myosotis.continuent.org
[9] Ehcache: http://ehcache.org

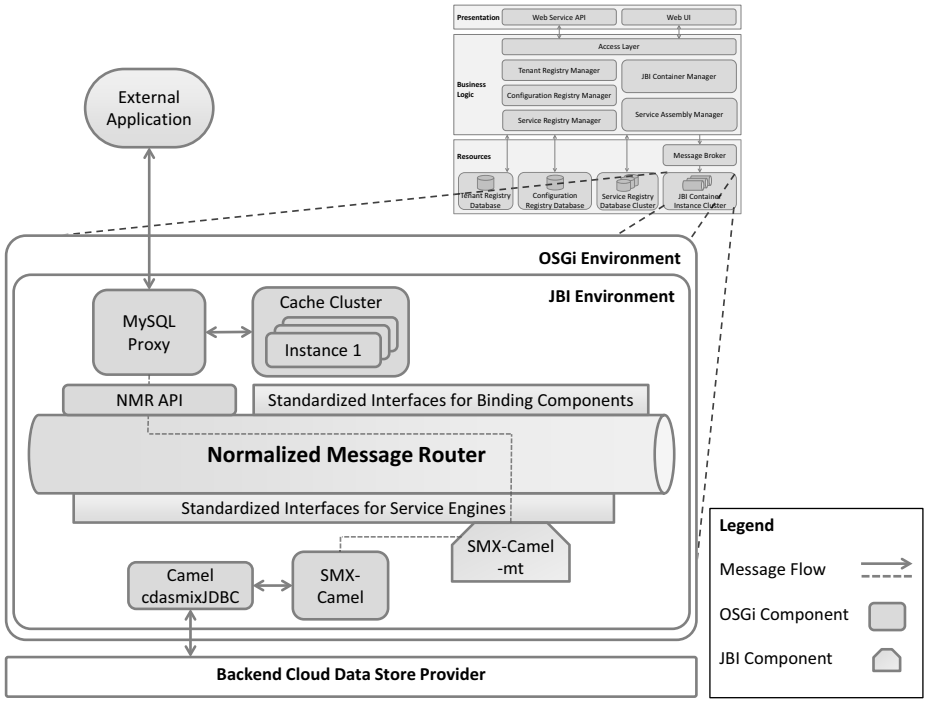


Figure 1. Architecture of an ESB Instance in CDASMix

order they were put into the cache [8]. The effectiveness of these algorithms in comparison to the absence of cache, and to the direct accessing of off-premise data sources (without CDASMix intervention), is discussed in the following section.

III. Evaluation

A. Methodology

In this work we focus on measuring the impact of introducing an ESB between the application and its migrated data to off-premise Cloud services, and on introducing multiple caching strategies to ameliorate negative performance effects. In general, the best caching strategy is the one that minimizes the end-to-end client latency of the application. Therefore, we do not only measure the latency (e.g. in terms of throughput or response time), but also the performance of the cache eviction algorithm (e.g. cache hit rate vs. cache miss rate).

The caching strategy selection when accessing data in a multi-tier architecture is heavily dependent on the application workload [4]. CDASMix is meant to be used for databases that are accessed transparently over the network. For the purposes of this work we therefore focus the workload generation towards maximizing the data volume transfered through CDASMix, effectively simulating a read-intensive application [9], i.e. an application that mostly executes complex read operations in its database system. The TPC-H[10] benchmark provides a set of complex queries for a predefined database schema. However, the benchmark does not include a load driver, and the queries are not distributed across a predefined workload. Therefore, in order to evaluate CDASMix we generate an application workload reusing a fixed set of N TPC-H queries, but organizing them towards building a *discrete uniform workload distribution*, where each query has a probability of $1/N$ of being included in the workload.

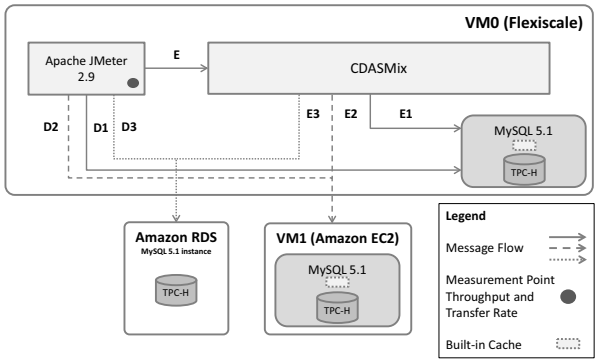


Figure 2. Overview of the Experimental Setup

In addition, three main scenarios are taken into consideration in this evaluation, based on different Cloud infrastructures where the database can be deployed, as shown in Fig. 2:

I. in an RDBMS on the same VM as CDASMix (VM0 on Flexiscale[11]);
II. in an RDBMS hosted on a VM of a different Cloud provider (CDASMix on VM0 (Flexiscale) and MySQL hosted on Amazon EC2[12] (VM1)); and
III. in a DBaaS solution (Amazon RDS[13]) with the CDASMix hosted on VM0.

For each of the scenarios, we first compare the direct access to the database, with a transparent access to the database through

[10]Transaction Processing Performance Council: http://www.tpc.org

[11]Flexiant Flexiscale: http://www.flexiscale.com
[12]Amazon EC2: http://aws.amazon.com/ec2/
[13]Amazon RDS: http://aws.amazon.com/rds/

CDASMix without caching support enabled (see Fig. 2). In a second step, we enable caching support in CDASMix, and evaluate the effect on performance of the available cache eviction algorithms: *LRU*, *LFU*, and *FIFO*. Performance is measured in terms of average *throughput* (Req./s) and average *cache hit ratio* (% cache hit). For simplifying the discussion, one tenant is used across all scenarios.

B. Experimental Setup

The TPC-H benchmark provides a set of operations for generating different volumes of data, and a set of complex queries for a predefined database schema. For this evaluation, a sample data of 1GB was created, and imported into the relational database systems deployed in Flexiscale and Amazon, and into a database instance deployed on Amazon RDS. In terms of load driving, we generate a workload of 100 queries from 5 original queries[14]. Each query of the workload retrieves a data volume of approximately 2.5MB from the database system.

Towards minimizing the network latency and establishing the evaluation baseline of hosting the application stack in the same system as CDASMix, Apache JMeter[15] is used for load driving purposes in VM0. The workload is then imported into the load driver, and the JDBC connections are configured using the MySQL Connector/J (5.1.22) to submit the queries. In the case of direct connections to MySQL on Flexiant/Amazon EC2 and Amazon RDS, the loader has to be configured separately to address the various endpoints of the databases (connections D1, D2 and D3 in Fig. 2). When CDASMix is used, JMeter is configured only once with an ESB endpoint (connection E in Fig. 2) and the redirection of the queries to the various databases through connections E1 to E3 uses the configuration mechanisms offered by CDASMix. JMeter is set up to measure the elapsed time for each request to be processed, out of which we calculate the average *throughput* of the load in executed transactions per second.

In terms of the experimental setup shown in Fig. 2, the virtual machine VM0 (8GB RAM, 4 CPUs AMD Opteron 2GHz with 512KB cache) hosted in the Flexiscale infrastructure runs Ubuntu 10.04 Linux OS. Deploying CDASMix and JBIMulti2 in VM0 also required the deployment of the JOnAS 5.2.2 application server (for JBIMulti2), and the PostgreSQL 9.1.1 database (for the registries). The maximum JVM memory heap size of the ServiceMix process was set to 1GB. In the same VM, a MySQL 5.1.67 database system with an internal cache size of 16MB was installed. The My-SQL built-in cache in the database systems deployed in VM1 and VM0 use an LRU eviction algorithm incorporating a midpoint insertion strategy. In the Amazon AWS Cloud infrastructure, both the Amazon EC2 VM and the Amazon RDS database are hosted in the `eu-west-1a` availability zone. An `m1.xlarge` EC2 instance running the Ubuntu 12.04.2 LTS OS was created, and MySQL 5.1.67 database system was deployed on it, with an internal cache size of 16MB. In Amazon RDS, a `db.m1.xlarge` instance was created, running a MySQL 5.1.69 engine version, as the MySQL 5.1.67 engine version is not available.

[14]Generated Load: https://santiago.studiforge.informatik.uni-stuttgart.de/svn/publications/IC2E14/queries4Load/generatedLoad_5-100.csv

[15]Apache JMeter: http://jmeter.apache.org

C. Experimental Results

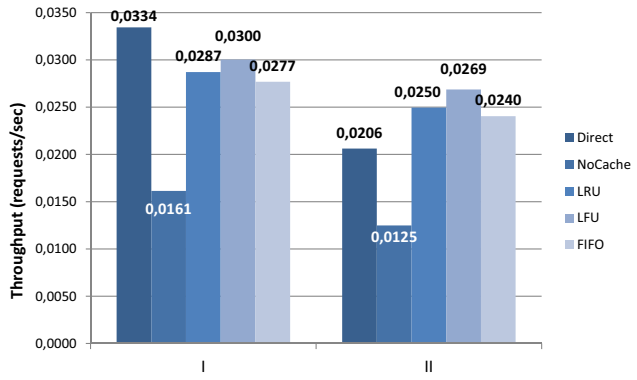
Figures 3 and 4 summarize our throughput and hit rate measurements, respectively, for all scenarios discussed in the evaluation's methodology. With respect to throughput, the following conclusions can be drawn from Fig. 3:

- Using CDASMix for transparent data access without any caching (NoCache cases in Fig. 3) always results to worse performance compared to directly accessing the database (through a JDBC connection). This is due to additional processing performed by CDASMix. However, use of caching improves performance significantly in all scenarios.
- LFU appears to be the more successful eviction algorithm in the MySQL-based scenarios (Fig. 3a) but yields slightly worse results than the other two when used in conjunction with Amazon RDS (Fig. 3b).
- For the case where the database is basically on-premise (Scenario I), even with caching enabled the throughput is (around 50%) worse when CDASMix is used. However, using LFU reduces the latency to almost 10%. In the presence of network latency (Scenarios II and III), the use of caching results in better throughput when compared to the direct connection (Scenario II) and/or the absence of caching (Scenario III).
- Based on the measurements shown in Fig. 3b, we can conclude that RDS implements an unspecified caching strategy which appears to be more effective for the generated workload. Adding (SQL) updates to the workload results in significantly less throughput; further investigation is required in this direction. Furthermore, the effect of network latency appears to be less in this scenario, due to the data-intensive nature of the service i.e. being optimized for data transfers instead of computation (as in the case of using IaaS solution in the other scenarios). As a result, all three eviction algorithms produce very similar results.

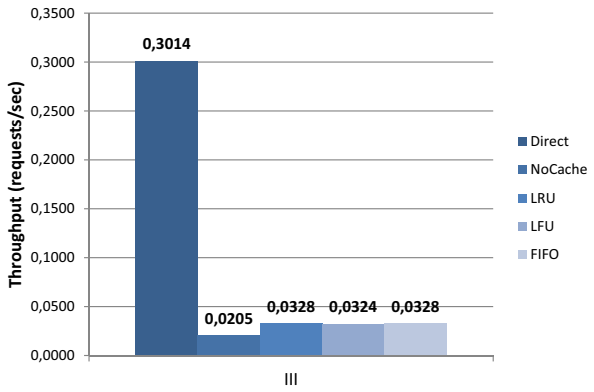
The measured hit rate is similar for all three algorithms across all scenarios (Fig. 4) and depends on the load used. In the work load that we generated, repetition of queries is (on purpose) low, which results relatively often in evicting items from the cache (almost 20% of the time). The most successful strategy for MySQL-based scenarios (LFU) has actually the worse hit rate from all algorithms, but relies extensively on the caching strategy implemented by MySQL (LRU). The impact of our findings is discussed in the following.

D. Discussion

Going back to the original goal of this work, the previous results have positively demonstrated that *a)* it is indeed possible to use an ESB solution in order to enable transparent access to data that are migrated off-premise (in the Cloud), and *b)* in order to achieve acceptable, and in some cases better performance, caching plays the most important role. Caching effectively ameliorates the impact of network latency introduced by the migration of the database, and becomes more important as latency varies unpredictably over time. However, choosing the appropriate caching strategy depends heavily on the application(s) workload. The reported results concerning



(a) Scenarios I & II (MySQL in IaaS solutions)



(b) Scenario III (DBaaS)

Figure 3. Throughput in Requests per Second for all Scenarios

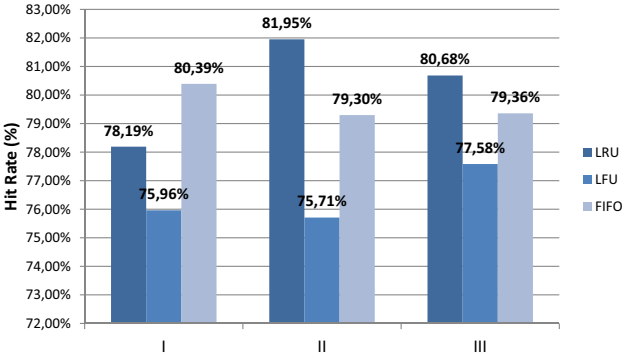


Figure 4. Average Hit Rate Per Scenario

the efficiency of the LFU eviction algorithm are applicable only to applications with read-heavy workloads. Generalizing our findings requires different type of loads also to be taken into consideration.

Furthermore, by introducing caching on the level of CDASMix, i.e. between the application and its database, we have basically implemented a two-level caching system with different caching strategies per level. In the case of Scenarios I and II, for example, our measurements show that LRU (on MySQL side) works more efficiently in tandem with LFU (at CDASMix), rather than with any other strategy. LFU at CDASMix however seems to antagonize the caching strategy implemented by Amazon RDS and does not produce better results than e.g. FIFO. It becomes therefore important to be able to identify the appropriate, meaning complementary, caching strategies for CDASMix and the RDBMS or DBaaS used. Further investigation is required for e.g. PostgresSQL and Google SQL.

IV. Related Work

Accessing Cloud data services through an ESB is currently supported by the JBoss Enterprise Data Service Platform[16], by means of exposing databases as data Web services that can be accessed through an ESB [3]. However, vendor-specific communication protocols supported by relational database systems, e.g. MySQL, PostgreSQL, Oracle, usually realized in standardized APIs, e.g. JDBC[17], are not supported. Applications accessing these data services should either be adapted to use them, or designed specifically for them. CDASMix allows applications to interact with both on- and off-premise databases through the ESB in their native communication protocol with minimum adaptations required. Other ESB technologies, e.g. Fuse ESB[18], and integration frameworks such as Apache Camel[19], support configuration of connections to external relational database systems for internal processing and routing purposes. However, incoming (client-side) connections are not configurable in such technologies. CDASMix minimizes the required modifications in the application's upper layers when migrating its data to the Cloud.

Caching strategies and replication have been widely investigated in the scope of Web applications [4], [10], [11]. Caching dynamic Web content reduces the latency when getting both application data, and application Web pages [9]. Nowadays, Web applications do not statically retrieve the Web page's content, but dynamically create Web pages based on business specific requirements, e.g. customized Web pages based on predefined user preferences. In [4] the importance of characterizing the application's workload prior to the selection of the caching strategy in order to scale Web applications is remarked. Furthermore, in [9] an approach for dynamically caching Web pages at the front-end of a multi-tier J2EE architecture is presented. However, the utilization of an ESB as the application's data access layer is not considered in these works, and therefore caching support is realized within the different servers hosting the application tiers, rather than in Message-Oriented Middleware (MOM) components. Caching support in middleware systems aims to reduce the complexity of realizing caching in the application logic. In [12] a cache mediation pattern for MOM is proposed as a reusable solution for accelerating service responses. However, caching strategy definitions or developer guidelines towards selecting the most appropriate caching strategy are not included in their proposal. Furthermore, multi-tenancy awareness is not considered. The

[16]JBoss Data Services Platform: http://www.redhat.com/products/jbossenterprisemiddleware/data-services

[17]JDBC: http://www.oracle.com/technetwork/java/javase/jdbc/index.html

[18]Fuse ESB: http://fusesource.com/products/enterprise-servicemix/

[19]Apache Camel JDBC: http://camel.apache.org/jdbc.html

integrated caching solution in CDASMix realizes multi-tenancy support atop of the original caching operations by dynamically creating multi-tenant aware cache keys.

Different Cloud aware benchmarks have been created to evaluate several features such as elasticity [13] and performance isolation [14]. To the extent of our knowledge, there is no commonly agreed benchmarking approach for evaluating Cloud-enabled middleware components such as an ESB. In [5] we extended an ESB-specific benchmark to enable multi-tenancy awareness. However, neither the benchmark nor the workload are suitable for evaluating CDASMix. Transaction processing and relational database benchmarks are available for evaluation purposes, e.g. the TPC-* benchmarks from the Transaction Processing Performance Council. For example, the TPC-W benchmark generates Web application workload by emulating Web browsers [15], and TPC-H evaluates accessing large data volume through complex queries. The evaluation of transparently accessing migrated data in the Cloud through an ESB, and therefore the evaluation of performance improvement when utilizing different intermediate caching strategies, is not yet covered by such benchmarks. RadarGun[20] is a Cache Benchmark Framework which focuses specifically on caching solutions, but not on accessing data remotely, e.g. via an ESB using relational database system communication protocols. Therefore, we chose to generate an artificial workload using sample data from the TPC-H benchmark, use Apache JMeter as the driver for this load, and configure different caching strategies in CDASMix.

V. Conclusions and Future Work

In the previous sections we presented and evaluated an approach for transparently accessing data stored both on- or off-premise (i.e. in the Cloud) through an ESB solution. We proposed an architectural approach for accessing data in relational database systems, e.g. MySQL, PostgreSQL, etc., through an existing multi-tenant aware ESB solution (ESB^{MT}). The adaptation of the ESB^{MT} to allow applications to communicate with its database(s) required modifications to it, including the realization of a MySQL Proxy supporting the native MySQL communication protocol, database-aware routing and connection components, and caching mechanisms to mitigate the latency introduced by the network and routing operations. The resulting solution, CDASMix, allows applications to communicate using JDBC with it, while transparently redirecting their (SQL) requests to databases either on-premise or off-premise.

For evaluation purposes we generated a read-heavy application workload using sample data and queries from the TPC-H benchmark, and specified multiple scenarios for several Cloud services and caching strategies. We observed an expected performance degradation when using CDASMix compared to direct accessing the database from the application. However, we demonstrated that the performance degradation is mitigated, and in some cases overturned, when caching support is enabled. Furthermore, we went a step further, and found for the used database system and generated workload the most appropriate cache eviction algorithms' combination for effectively coordinating a two level cache formed by the CDASMix cache and the built-in MySQL cache. This finding however cannot be replicated when using Amazon RDS as the remote database, indicating the use of an unspecified caching mechanism at the DBaaS side.

Of particular interest to our future investigation is to include in CDASMix components to support NoSQL database systems specific communication protocols, e.g. JSON over HTTP supported in Amazon DynamoDB, etc. Furthermore, at the caching level we plan to extend the set of cache eviction algorithms actually supported by the multi-tenant aware caching component, investigate towards distributed caching mechanisms when enabling horizontal scalability of CDASMix, and optimize the performance of the integrated cache solution, e.g. based on eliminating noise in large cache systems. We also plan to extend the evaluation methodology towards analyzing and identifying different application workloads for scalability purposes using for example RadarGun[20], an open source benchmarking framework for distributed caches.

[20]RadarGun: http://github.com/radargun/radargun

Acknowledgments

This work is funded by the FP7 EU-FET project 600792 ALLOW Ensembles and the BMBF-project ECHO (01XZ13023G).

References

[1] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to Adapt Applications for the Cloud Environment," *Computing*, vol. 95, no. 6, pp. 493–535, 2013.

[2] M. Fowler *et al.*, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[3] Red Hat, Inc, "Gap Analysis: The Case for Data Services," 2011.

[4] S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso, "Analysis of Caching and Replication Strategies for Web Applications," *Internet Computing, IEEE*, vol. 11, no. 1, pp. 60–66, 2007.

[5] S. Strauch, V. Andrikopoulos, S. Gómez Sáez, and F. Leymann, "ESB^{MT}: A Multi-tenant Aware Enterprise Service Bus," *International Journal of Next-Generation Computing*, vol. 4, no. 3, pp. 230–249, 2013.

[6] Gregor Hohpe and Bobby Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.

[7] T. Rademakers and J. Dirksen, *Open-Source ESBs in Action*. Greenwich, CT, USA: Manning Publications Co., 2008.

[8] J. Handy, *The Cache Memory Book*, 2nd ed. Morgan Kaufmann, 1998.

[9] S. Bouchenak, A. Cox, S. Dropsho, S. Mittal, and W. Zwaenepoel, "Caching Dynamic Web Content: Designing and Analysing an Aspect-Oriented Solution," in *Proceedings of Middleware'06*. Springer, 2006, pp. 1–21.

[10] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Addison-Wesley, 2001.

[11] D. Wessels, *Web Caching*. O'Reilly & Associates, Inc., 2001.

[12] F. Y. Rao, R. Fang, Z. Tian, E. Lane, H. Srinivasan, T. Banks, and L. He, "Cache Mediation Pattern," in *Proceedings of OOPSLA'05*, 2005.

[13] P. Brebner, "Is Your Cloud Elastic Enough?: Performance Modelling the Elasticity of Infrastructure as a Service (IaaS) Cloud Applications," in *Proceedings of ICPE'12*, 2012, pp. 263–266.

[14] R. Krebs, C. Momm, and S. Kounev, "Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments," in *Proceedings of QoSA'12*. ACM, 2012, pp. 91–100.

[15] D. A. Menascé, "TPC-W: A Benchmark for E-Commerce," *Internet Computing, IEEE*, vol. 6, no. 3, pp. 83–87, 2002.

All links were last followed on January 22, 2014.