Transparent Access to Relational Databases in the Cloud using a Multi-tenant ESB

Steve Strauch, Vasilios Andrikopoulos, Santiago Gómez Sáez and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany

Keywords: Data Access Layer, Relational Databases, Enterprise Service Bus (ESB), Multi-tenancy.

Abstract: In the last years Cloud computing has become popular among IT organizations aiming to reduce their operational costs. Applications can be designed to run in the Cloud, or can be partially or completely migrated to it. Migrating the data layer of an application to the Cloud, however, implies that existing applications might need to be adapted in order to access their migrated to the Cloud databases. In this work we examine how we can use an existing ESB to enable transparent access to the relational data store running either in the Cloud or on-premise. The goal of our approach is to minimize the effort required to adapt the application. In particular, we discuss the requirements and prototype realization of a Cloud aware data access layer for transparent data access, using an existing open source and multi-tenant aware ESB as the basis. We then evaluate the performance of our proposed solution by considering different Cloud providers and using example data from an existing benchmark as application workload.

1 INTRODUCTION

With the advent of Cloud services beyond Infrastructure as a Service (IaaS) solutions (Mell and Grance, 2011), the migration of individual application layers or even individual architectural components to the Cloud, becomes possible. Applications are usually built using the three layers pattern: Presentation, Business Logic, and Data (Fowler et al., 2002). The migration of the Data layer to the Cloud is the focus of this work. Application data is typically moved to the Cloud for Cloud bursting, data analysis, or backup and archiving purposes. We distinguish between two different types of interaction with databases in the Cloud. On the one hand there is the direct interaction with a database hosted on the Cloud as a *data store* that takes place on a fine granular level, e.g., by using SQL commands. The second interaction type relies on a service interface to interact with the Cloud database, as in the case of e.g. Amazon SimpleDB[1], which becomes a *data service*. This requires interaction on the level of the service interface that is more coarse grained compared to the interaction when using SQL for instance. We therefore need to consider both types of interactions in the following. In addition to relational databases (Codd, 1970), widely established in industry and research *Not Only SQL* (NoSQL)[2] solutions have become increasingly popular in recent years (Sadalage and Fowler, 2012). For the scope of this work we focus only on relational databases.

Looking at the Data Layer in particular, the layer can be subdivided into the Data Access Layer (DAL) and the Database Layer (DBL). The DAL is an abstraction layer encapsulating the data access functionality, while the DBL provides persistence and retrieval capabilities. As discussed in (Andrikopoulos et al., 2013), the migration of the Data Layer to the Cloud includes two main steps: the migration of the DBL to the Cloud, and the adaptation of the DAL to enable Cloud data access. The DBL can either be migrated completely or partially to the Cloud. In the latter case, the DBL is distributed using both non-Cloud technologies (local databases), and data stores and services. Thus, the distribution of the DBL essentially changes the border of the application compared to traditional applications. The assumption in our work is that the DBL has already been migrated to the Cloud, and therefore we focus on the DAL. As a consequence of the tight coupling of the Business Logic layer with the DBL in traditional applications built without using any service-oriented or Cloud technology, the Business Logic layer must be explicitly aware of the location of the data and the data store it is interacting with.

[1]Amazon SimpleDB: http://aws.amazon.com/de/simpledb/

[2]List of NoSQL Databases: http://nosql-database.org

Strauch S., Andrikopoulos V., Goméz Sáez S. and Leymann F..
Transparent Access to Relational Databases in the Cloud using a Multi-tenant ESB.
DOI: 10.5220/0004835900050014
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (CLOSER-2014), pages 5-14
ISBN: 978-989-758-019-2
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

Therefore, any change to the location of the DBL results in the need for adapting the Business Logic layer, not only to rewire the connection to the DBL, but also to cope with potentially less fine granular interaction types when moving to data service solutions.

In order to minimize the impact of this adaptation, in this work we propose the use of the Enterprise Service Bus (ESB) technology (Chappell, 2004) as the means to decouple the Business Logic from the Database Layer by leveraging the Data Access Layer. The ESB technology as the messaging infrastructure between applications and services addresses the essential requirement of application integration while ensuring loose coupling. In the recent years it has become as a result ubiquitous in enterprise computing environments. ESBs control message handling during the interaction between applications and services, allowing for dynamic routing, and are at the core of each Service-Oriented Architecture (SOA) (Josuttis, 2007). It is only natural therefore to enable the distribution of the DBL in the Cloud by taking advantage of the dynamic routing features of ESBs, allowing for hosting the database both on-premise and off-premise. As to the extend of our knowledge there is no solution so far that supports this, we want to enable the communication between the Business Logic and the DBL via the ESB (DAL) using the native database communication protocol. For this purpose, we build on an existing open source ESB previously extended for multi-tenancy (Strauch et al., 2012a; Strauch et al., 2013b) to enable transparent access to multiple data sources both locally, and in the Cloud.

Our contribution therefore can be summarized by offering:

- The identification of requirements for enabling transparent data access to data stores and data services.
- The design and realization of CDASMix, a multi-tenant aware ESB solution that enables transparent data access to databases both on-premise and off-premise.
- A performance evaluation of our proposal, establishing the impact of introducing an ESB as the DAL between the application and the database across different deployment options.

The remaining of the paper is structured as follows: Section 2 identifies the requirements for supporting transparent data access for databases that have been migrated to the Cloud. Based on these requirements, Section 3 discusses the architecture and implementation of our proposal. Section 4 provides a performance evaluation for this realization for different database and deployment options considering different providers. The paper closes with Section 5 and Section 6 summarizing related work, and concluding with some future work, respectively.

2 TRANSPARENT CLOUD DATA ACCESS REQUIREMENTS

The set of *functional* and *non-functional* requirements for transparent Cloud data access we present in this section has been identified during our work in various EU research projects and especially during the collaboration with industry partners. The requirements are geared towards minimizing the impact to the Business Logic and Presentation application layers of adapting the application to cope with the migration of the database to the Cloud (Andrikopoulos et al., 2013).

2.1 Functional Requirements

The following functionalities must be offered by any DAL that provides transparent access to databases in the Cloud.

FR_1 *Interaction with Data Stores and Data Services*: The DAL must support both fine- and coarse-grained types of interactions, through SQL and service APIs, respectively.

FR_2 *Management and Configuration*: The DAL must provide management and configuration capabilities for both data stores and data services, e.g. registration of a new data store, including its configuration data, e.g. database schemas, database system endpoint URL, etc.

FR_3 *On-premise and Off-premise Support*: The DAL has to support data stores and data services that are either hosted on-premise or off-premise and using Cloud or non Cloud technologies.

FR_4 *Transparent Data Access*: As the database layer might be distributed between on-premise and off-premise using Cloud and non-Cloud technologies, the business logic is no longer aware where the data is located compared to traditional applications. Thus, the DAL must determine autonomously the data store or data service the request should be forwarded to, based on the request sent by the Business Logic layer.

FR_5 *Caching*: As the migration of the database to the Cloud introduces a potentially high distance between the different application layers measured in network hops, the DAL has to support caching in order to mitigate the consequences of network latency.

2.2 Non-functional Requirements

In addition to the required functionalities, a DAL enabling Cloud data access should also respect the following properties.

- NFR_1 *Loose Coupling*: The DAL must allow for Cloud data stores or data services to be changed without affecting the Business Logic layer, except from the necessary reconfiguration of the connection to the data store or data service.
- NFR_2 *Security*: As the DAL stores information on the available data stores and data services including their internal structure, e.g. database schemas and access credentials, third parties have to be prevented from gaining access to these data.
- NFR_3 *Backward Compatibility & Extensibility*: DAL components have to be compatible with directly wiring the business logic with a specific data store, as in traditional applications. Furthermore, extensibility has to be ensured in order to integrate the components with future technologies and solutions, e.g. new types of data services in the Cloud.

As discussed in the previous section, the proposed approach builds on the use of the ESB technology as the realization of these requirements. However, given the fact that multiple organizational domains may have access to the same data stores or services, it has to be ensured that each organization is confined to their own data. While modern Relational Database Systems (RDBMSs) have been offering this capability for a while, *multi-tenancy* on the level of middleware is still an open issue. For this purpose, in (Strauch et al., 2012b) we identify a set of requirements for multi-tenancy for ESB solutions. These requirements, summarized in the following, have also to be taken into consideration when designing and implementing a transparent DAL solution using an ESB.

More specifically, *tenant-based identification* and *hierarchical access contro*l have to be supported, as the ESB has to manage and identify multiple tenants (organizational domains) and their corresponding users. For the administration and management of both tenants and users, *customizable interfaces* have to be provided, including both Web service interfaces and GUIs. Through these interfaces, the tenant-based deployment and configuration of the ESB and the services available for a specific tenant should be managed in a transparent way. As other multi-tenant components in a PaaS environment request similar information, e.g. workflow engines, the ESB should offer a *shared registry* of services and of tenants and users for other PaaS components. Furthermore, services and applications that are not multi-tenant aware should still be able to seamlessly use and transparently interact with the ESB.

In addition, both *data isolation* and *performance isolation* have to be ensured, in order to avoid that tenants acquire access to data or computing resources of other tenants. In order to address the *security* aspect, corresponding confidentiality, authorization, authentication, and integrity mechanisms have to be realized on tenant and user level. Finally, the ESB components must be extensible and reusable by other components in the PaaS environment to ensure independence from a specific technology or solution. All of these requirements are considered in the following discussion.

3 CLOUD DATA ACCESS SUPPORT IN ServiceMix

In this section we discuss *CDASMix (Cloud Data Access Support in ServiceMix)*, an architectural and implementation approach which fulfills the requirements identified in the previous section. As the basis for the design and implementation of our approach, we reuse and extend the ESB^{MT} multi-tenant aware ESB solution[3], as presented in (Strauch et al., 2013a). ESB^{MT} enhances the Apache ServiceMix 4.3 JBI container[4] with multi-tenant communication support within service endpoints deployed in the ESB, and multi-tenant aware dynamic endpoint deployment capabilities. As shown in Fig. 1, the ESB^{MT} architecture contains three layers: Presentation, Business Logic, and Resources. The Presentation and Business Logic layers define a generic interface and logic design approach of a multi-tenant aware administration and management application for ESBs. The Resources layer contains the ESB-specific instances cluster, databases for tenant-related information, and an intermediary component to mediate the management communication between the upper layers and the ESB instance cluster. However, ESB^{MT} focuses on enabling multi-tenancy at the communication level between application services, rather than on application's data access. Therefore, the remaining of this section describes the extensions realized for CDASMix in each of the system's layers highlighted in Fig. 1 to enable transparent Cloud data access support, and provide a deeper insight on the realized components. An extended description of the architectural and implementation ap-

[3] ESB^{MT}: http://www.iaas.uni-stuttgart.de/esbmt/
[4] Apache ServiceMix: http://servicemix.apache.org

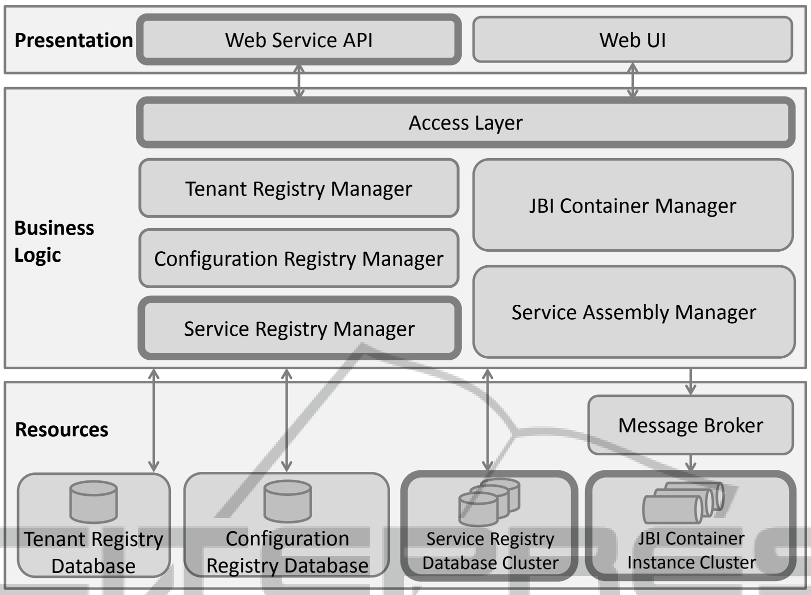

Figure 1: Architecture of CDASMix based on ESB^{MT}.

proaches used is provided in (Gómez Sáez, 2013).

3.1 Resources Layer

The Resources Layer of CDASMix consists of a set of registries and multiple JBI containers which constitute a *JBI Container Instance Cluster*, as described in Fig. 1. JBI containers provide support for message routing and transformation, as in traditional ESB solutions. ESB^{MT}, and by extension CDASMix, relies on the OSGi Framework[5] in order to allow components to be deployed in a loosely coupled way. 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 routing of HTTP requests, and the Apache Camel[6] Service Engine (SE) realizes a set of known Enterprise Integration Patterns (Hohpe and Woolf, 2003). 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, SMTP, JMS, etc.) by marshaling and demarshaling protocol-specific messages into a standardized internal Normalized Message Format (NMF) (Rademakers and Dirksen, 2008). Nevertheless, database system native communication protocols are not supported, as they are solution-specific. We focus our approach by providing communication support for a well known and widely used database system, MySQL[7]. The same approach however can be used to provide support for other systems too, e.g. PostgresSQL[8].

Figure 2 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 in CDASMix based on the ESB^{MT} architecture. More specifically, the developed `MySQL Proxy` is an OSGi component that implements an OSGi- and JBI-compliant version of a Java-based MySQL proxy (NFR$_1$). As the default MySQL Proxy shipped with MySQL distributions is implemented in the C programming language, the existing open source Java-based MySQL Proxy in Myosotis Tungsten Connector[9] was reused and extended. The MySQL OSGi bundle implements the native MySQL communication protocol and is fully integrated with the JBI container via the Normalized Message API, an OSGi bundle which exposes an interface for accessing the NMR endpoint registries and routing operations. In order to mitigate the latency introduced by using CDASMix as an intermediate communication component between the application architecture layers, the caching mediation pattern presented in (Rao et al., 2006) is realized within CDASMix. Thus, a caching component ensures high performance of SQL read op-

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

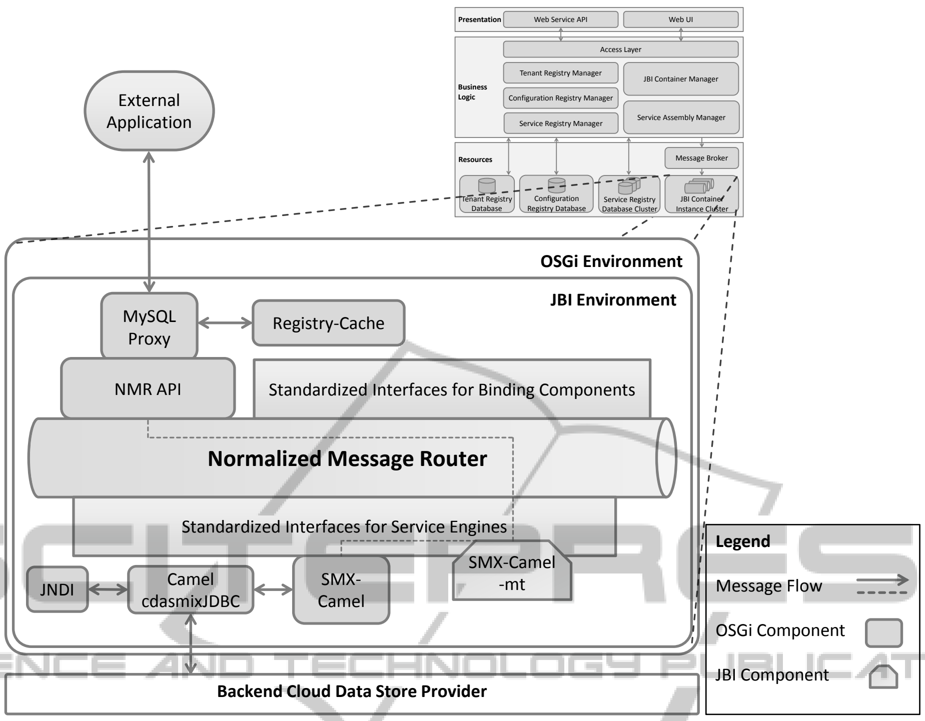

Figure 2: Architecture of an ESB Instance.

erations in CDASMix by utilizing a Least Recently Used (LRU) caching policy configured in Ehcache 2.6.0[10], and by deleting cache records when SQL statements involve data modification (FR_5). Therefore, the caching strategy does not only aim to reduce the latency of accessing data through the ESB, but also intends to reduce the NMR system resources consumption.

The ESB^{MT} Camel SE 2011.11 (denoted as `SMX-Camel-mt` in Fig. 2) provides multi-tenancy and integration support between the JBI environment and the Enterprise Integration Patterns realizations of Apache Camel. In CDASMix, a custom Camel component named `CamelcdasmixJDBC` (see Fig. 2) was developed for dynamically connecting to backend data sources via multiple database system communication protocols (FR_4, NFR_1). This component is accessed through a camel endpoint in the `SMX-Camel` component (Fig. 2). Database connections are registered in the JNDI component of ServiceMix to decrease the latency produced by creating one database connection per user per received request. The JBI specification limits the inclusion of referenced packages in the SU during build time, rather than during runtime as in OSGi. Therefore, extensions realized on the `CamelcdasmixJDBC` component, e.g. supporting a new database system, require updating, building, and redeploying the service units referencing the `CamelcdasmixJDBC` component. Utilizing the ServiceMix Camel SE OSGi component enables loading of `CamelcdasmixJDBC` component packages during runtime (NFR_3).

CDASMix utilizes the existing ESB^{MT} registries in the Resources layer, described extensively in (Strauch et al., 2012b; Strauch et al., 2013b). A database schema modification in the Service Registry Database Cluster was necessary (Fig. 1), as it must be extended to persist tenant-related Cloud data access management information. We identify two required types of information when executing SQL statements on RDBMSs: connection related, and database related. The former contains the required information to connect to a database instance, e.g. access credentials, database URL, etc., while the latter includes information about the database schema, e.g. table identifiers. Therefore, we create a data structure which persists the tenant's backend database connection and structure information for enabling transparent access through one physical endpoint in CDASMix, to multiple backend data stores or data services located either on-premise or off-premise (FR_1, FR_3, FR_4).

3.2 Business Logic Layer

The Business Logic layer of ESB^{MT} enables role-based access control for administration and manage-

[10]Ehcache 2.6.0: http://ehcache.org

ment of the *JBI Container Instance Cluster* (Strauch et al., 2012b; Strauch et al., 2013b) (NFR_2). Enabling transparent Cloud data access support in CDASMix does not require architectural modifications on this layer, but demands implementation extensions for administrating access and persistence operations on the information related to tenant's backend data stores. Therefore, we extend the components highlighted in Fig. 1. The *Access Layer* is extended with operations executable at the tenant operator role level. The *Service Registry Manager* component is extended with operations for persisting and accessing the CDASMix-related data in the *Service Registry Database Cluster*.

3.3 Presentation Layer

The Presentation layer contains the *Web UI* and the *Web services API* components which allow the customization, administration, management, and interaction with the other layers (Strauch et al., 2012b; Strauch et al., 2013b). In CDASMix, no modifications on the Web UI were required, but a set of operations for configuring the transparent data access through CDASMix was included in the Web services API (FR_2).

4 EVALUATION

In this section we present the performance evaluation of our proposal. Therefore, we define the method and the experimental setup (Section 4.1), and discuss the experimental results (Section 4.2).

4.1 Evaluation Method & Experimental Setup

The evaluation of CDASMix in this work focuses on measuring the impact of introducing the ESB between an application and its migrated to the Cloud database. Three main scenarios are taken into consideration for this purpose, based on different Cloud infrastructures where the database can be deployed, as shown in Fig. 3:

1. in an RDBMS on the same VM as CDASMix (VM0 on Flexiscale[11]);
2. 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
3. in a DBaaS solution (Amazon RDS [13]) with the CDASMix hosted on VM0.

For each of the scenarios, we compare the direct access to the database with a transparent access to the database through CDASMix. In order to simplify the discussion, we only use one tenant for database requests across all scenarios. Evaluating the multi-tenancy capabilities offered by CDASMix are the subject of ongoing future work.

For purposes of creating a data set and queries over it, we use the TPC-H benchmark[14] which emphasizes large volumes of data, queries with high degree of complexity, and answering critical business questions. 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. The benchmark was used to create the database schema and populate it with a fixed data volume, which was consequently imported into the two MySQL deployments (on Flexiscale and Amazon) and on Amazon RDS. The generated data set has a total size of 1GB distributed among 8 tables, conforming to the TPC-H database schema.

A set of three complex (SQL) `SELECT` queries extracted from the benchmark was combined with two simpler `UPDATE` statements and used as the basis for an artificial work load. Figure 4 lists one of the select queries used as part of this process. The work load used consists of arbitrarily executed queries in a set of ten data requests in a loop of ten iterations, for a total of a hundred queries, where queries are distributed in a N updates to M selects ratio, with $(N, M) = (3, 7)$. The ratio used was selected in order to force the cache of CDASMix to flush relatively often (roughly once every three queries).

Apache JMeter[15] is used for load driving purposes from a local computer, using the JDBC Connection Configuration and MySQL Connector/J (5.1.22) driver 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. 3). When CDASMix is used, JMeter is configured only once with an ESB endpoint (connection E in Fig. 3) and the redirection of the queries to the various databases through connections E1 to E3 uses the configuration mechanisms offered by CDASMix, as explained in the previous section. JMeter is set up to measure the elapsed time for each request to be pro-

[11] Flexiant Flexiscale: http://www.flexiscale.com

[12] Amazon EC2: http://aws.amazon.com/ec2/

[13] Amazon RDS: http://aws.amazon.com/rds/

[14] Transaction Processing Performance Council Benchmark^{TM} H http://www.tpc.org/tpch/

[15] Apache JMeter 2.9: http://jmeter.apache.org

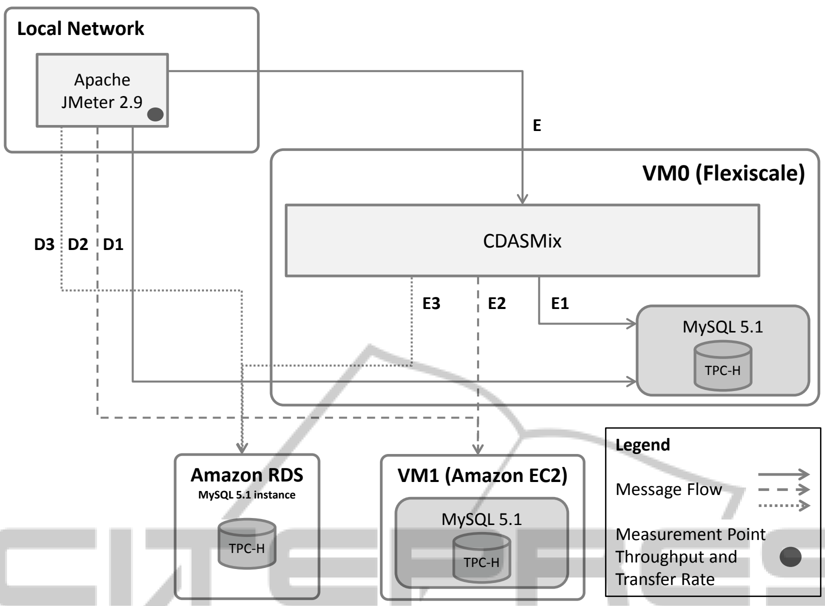

Figure 3: Overview of the Experimental Setup.

```
select s_acctbal , s_name , n_name ,
    p_partkey , p_mfgr , s_address , s_phone
    , s_comment from part , supplier ,
    partsupp , nation , region
where p_partkey = ps_partkey and
    s_suppkey = ps_suppkey and p_size
    = 47 and p_type like '%BRASS' and
    s_nationkey = n_nationkey and
    n_regionkey = r_regionkey and
    r_name = 'MIDDLE_EAST' and
    ps_supplycost = ( select min(
    ps_supplycost ) from partsupp ,
    supplier , nation , region where
    p_partkey = ps_partkey and
    s_suppkey = ps_suppkey and
    s_nationkey = n_nationkey and
    n_regionkey = r_regionkey and
    r_name = 'MIDDLE_EAST' )
order by s_acctbal desc , n_name ,
    s_name , p_partkey limit 100;
```

Figure 4: Sample SQL Query Used.

cessed, 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. 3, 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. 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.

4.2 Experimental Results

Figure 5 summarizes the measurements taken for the defined work load across the three database deployment scenarios. In addition, and for purposes of establishing a baseline, the average throughput with JMeter installed directly on VM0 (i.e. on the same VM) as the database was measured to be 0,15 transactions per second for the defined work load. When compared with the direct access to the same database from a local machine using the same load (leftmost column in Fig. 5) it can be concluded that network latency between the local machine and Flexiscale results in an average of 3,52% decrease in throughput. The effect of this latency has to be factored in for the remaining analysis whenever CDASMix is used.

In particular, the following conclusions can be drawn when examining Fig. 5:

- When comparing direct access to MySQL from

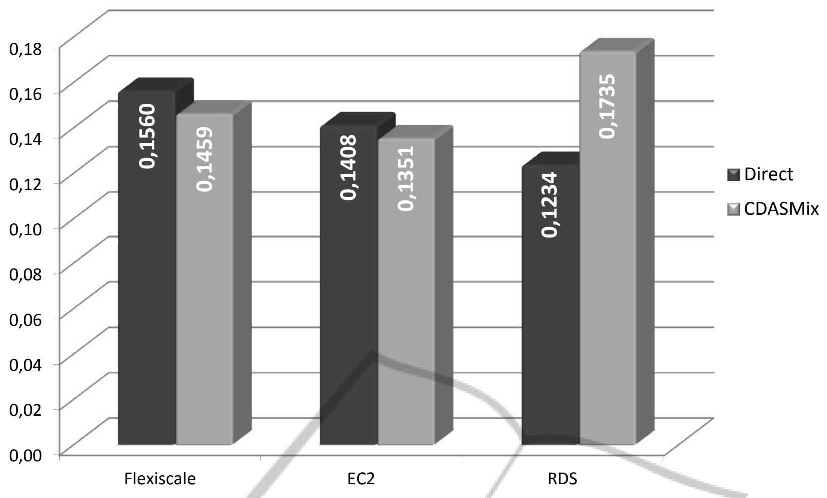

Figure 5: Average Throughput (Transactions per Second): Without (Direct) and With Using CDASMix

the (local) loader with indirect access through CDASMix, a 6,47% decrease in throughput is observed. Taking into account the 3,52% decrease due to network latency, we can conclude that the actual impact of introducing CDASMix amounts to 3%, which can be considered acceptable for many applications.

- A similarly low impact is also observed when comparing direct and indirect access to the same MySQL engine as before but on a different provider (EC2), with 4% decrease in throughput. The presence of caching on CDASMix is assumed to mitigate the effect of the additional network latency between the two providers (Flexiscale), resulting in better throughput when compared with the first scenario (i.e. Flexiscale only).
- The effect of caching is more eminent in the case of accessing RDS through CDASMix, where despite the additional network latency, using CDASMix actually improves throughput by 40,54%. However, in both cases, further evaluation of the effect of caching in relation to network latency is required.

In summary therefore, the effect of introducing CDASMix between an application and its database is acceptable for the majority of applications. Furthermore, any adversarial effects to performance, due to additional computations and dependence on network behavior, have to be measured against the benefits of the flexibility provided by the transparency of the data access layer.

5 RELATED WORK

To the extend of our knowledge, there is no existing approach aiming at using an ESB solution for transparent data access to databases in the Cloud. Most ESB solutions do provide support for storing data in relational database systems (RDBMS). However, all of them focus on providing access to databases through JDBC connections that are meant to be used for ESB-internal purposes. Fuse ESB, for example, provides JDBC data source support enabling users to connect to a database and make SQL based queries and data manipulations (FuseSource Corporation, 2013). However, it does not provide native support for incoming database messages, e.g. in the MySQL communication protocol, because data consumer endpoints supporting native database protocols are not deployable. The integration framework Apache Camel provides access to RDBMSs through the Camel-jdbc[16] component, but it also does not support interaction via native database communication protocol messages through consumer endpoints. Data sources in producer endpoints must be configured in both Apache Camel and existing ESBs prior their deployment, e.g. using Spring. This requirement forces the data sources' access configuration to be statically created, rather than dynamically created during runtime. JBoss Enterprise Data Service Platform[17] by Red Hat, Inc. provides data services with Service-Oriented Architecture (SOA) support and an ESB (Red Hat, Inc, 2011). Accessing the database

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

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

layer using technologies implementing an architectural paradigm like SOA, however, might require adapting the application, and in particular the upper (presentation and business logic) architecture layers, in order to support these technologies. One of the main goals of our approach is to minimize the required adaptations of the upper application architecture layers when migrating the database layer partially or completely to the Cloud by introducing an ESB as DAL.

In terms of the performance evaluation of our approach, the different benchmarks and metrics developed in the domain of Cloud computing in the recent years focus on Cloud-related features such as elasticity (Brebner, 2012), performance isolation (Krebs et al., 2012), or on virtualization technology (Makhija et al., 2006). Given the absence of a commonly agreed approach for the evaluation of Cloud-enabled middleware components (like the ESB), in previous work (Strauch et al., 2013b) we extended an ESB-specific benchmark developed by AdroitLogic (AdroitLogic Private Ltd., 2011). This work however focused on evaluating the multi-tenancy awareness aspects of our proposed ESB solution, and it is not suitable for evaluating transparent data access. On the other hand, there are various established transaction processing and database benchmarks addressing performance evaluation of relational databases, e.g. the TPC-* benchmarks from the Transaction Processing Performance Council[18], and YCSB for Cloud database services (Cooper et al., 2010). Evaluating an ESB as transparent data access layer for accessing databases on-premise or in the Cloud is not covered by these existing benchmarks. For this purpose, and as discussed in the Section 4, we use an artificial work load taken from the TPC-H benchmark and Apache JMeter as the driver for this load.

6 CONCLUSIONS AND FUTURE WORK

In the previous sections we presented our proposal for using an ESB solution as the application Data Access Layer in order to enable transparent data access to multiple data sources, both on- and off-premise (i.e. in the Cloud). A set of requirements were identified for this purpose in terms of necessary functionalities, e.g. support of both data stores, i.e. databases hosted on an IaaS infrastructure, and Databases as a Service solutions, and of additional non-functional properties like security. Using the ESB as the DAL for multiple organizational domains (tenants) also incurs additional requirements to be considered.

The CDASMix solution discussed in this work addresses these requirements by building on a multi-tenant-aware ESB solution called ESB^{MT} that has been presented in (Strauch et al., 2013a). Adapting ESB^{MT} to allow applications to communicate with a database through the ESB required a number of modifications to it, including database-aware routing components, the implementation of a messaging proxy for native JDBC communications, and caching mechanisms for increased performance. The resulting solution, CDASMix, allows applications to communicate using JDBC with it, while transparently redirecting their (SQL) requests to databases either on-premise or in the Cloud. As shown during the evaluation section, the impact of introducing CDASMix as the application DAL, when compared to direct access to the database from the application, results in acceptable performance degradation for the used workload and database.

Our current efforts focus on expanding the evaluation of CDASMix to include the effect of multi-tenancy on performance when multiple applications use the same CDASMix instance, and improving the generality of our findings by introducing multiple workloads. Of particular interest to our work is our ongoing investigation on the interplay between the caching done on the level of CDASMix and that of the used database engine, in order to identify the optimal caching strategy for the ESB (Gómez Sáez et al., 2014). Furthermore, in terms of future work, we plan to support further database system communication protocols beyond MySQL, and enable the SQL statement transformations between different engines, e.g. MySQL to PostgreSQL, or between their versions. Communication and statement transformation support for non-relational database engines (NoSQL) offered as services in the Cloud like Amazon DynamoDB[19], Google Cloud Storage[20], and MongoHQ[21] is also under development.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from projects 4CaaSt (grant agreement no.

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

[19]Amazon DynamoDB: http://aws.amazon.com/dynamodb/

[20]Google Cloud Storage: http://cloud.google.com/storage

[21]MongoHQ: https://www.mongohq.com

258862) and Allow Ensembles (grant agreement no. 600792) part of the European Union's Seventh Framework Programme (FP7/2007-2013) and the German government funded BMBF project ECHO (01XZ13023G).

REFERENCES

AdroitLogic Private Ltd. (2011). Performance Framework and ESB Performance Benchmarking. http://www.esbperformance.org.

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to Adapt Applications for the Cloud Environment. *Computing*, 95(6):493–535.

Brebner, P. (2012). Is your Cloud Elastic Enough?: Performance Modelling the Elasticity of Infrastructure as a Service (IaaS) Cloud Applications. In *Proceedings of ICPE'12*, pages 263–266.

Chappell, D. A. (2004). *Enterprise Service Bus*. O'Reilly Media, Inc.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387.

Cooper, B. F. et al. (2010). Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of SOCC'10*, pages 143–154. ACM.

Fowler, M. et al. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

FuseSource Corporation (2013). Fuse ESB 4.4 – Configuring and Running Fuse ESB. http://fusesource.com/docs/esb/4.4/esb_runtime/.

Gómez Sáez, S. (2013). Extending an Open Source Enterprise Service Bus for Cloud Data Access Support. Diploma Thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

Gómez Sáez, S., Andrikopoulos, V., Leymann, F., and Strauch, S. (2014). Evaluating Caching Strategies for Cloud Data Access Using an Enterprise Service Bus. In *Proceedings of IEEE IC2E'14*. IEEE Computer Society.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.

Josuttis, N. (2007). *SOA in Practice*. O'Reilly Media, Inc.

Krebs, R., Momm, C., and Kounev, S. (2012). Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In *Proceedings of QoSA'12*, pages 91–100. ACM.

Makhija, V. et al. (2006). VMmark: A Scalable Benchmark for Virtualized Systems. Technical Report VMware-TR-2006-002, VMware, Inc.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing.

Rademakers, T. and Dirksen, J. (2008). *Open-Source ESBs in Action*. Manning Publications Co., Greenwich, CT, USA.

Rao, F. Y. et al. (2006). Cache Mediation Pattern. In *Proceedings of EuroPloP'06*. Universitaetsverlag Konstanz.

Red Hat, Inc (2011). Gap Analysis: The Case for Data Services.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., and Leymann, F. (2013a). ESB^{MT}: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., and Leymann, F. (2013b). Implementation and Evaluation of a Multi-tenant Open-Source ESB. In *Proceedings of ESOCC'13*, volume 8135 of *Lecture Notes in Computer Science (LNCS)*, pages 79–93. Springer Berlin Heidelberg.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., Leymann, F., and Muhler, D. (2012a). Enabling Tenant-Aware Administration and Management for JBI Environments. In *Proceedings of SOCA'12*, pages 206–213. IEEE Computer Society Conference Publishing Services.

Strauch, S., Andrikopoulos, V., Leymann, F., and Muhler, D. (2012b). ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses. In *Proceedings of CloudCom'12*, pages 456–463. IEEE Computer Society Press.

All links were last followed on January 30, 2014.