Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH

Open Access

Evaluating cloud database migration options using workload models

Martyn Ellison*, Radu Calinescu and Richard F. Paige

Abstract
A key challenge in porting enterprise software systems to the cloud is the migration of their database. Choosing a cloud provider and service option (e.g., a database-as-a-service or a manually configured set of virtual machines) typically requires the estimation of the cost and migration duration for each considered option. Many organisations also require this information for budgeting and planning purposes. Existing cloud migration research focuses on the software components, and therefore does not address this need. We introduce a two-stage approach which accurately estimates the migration cost, migration duration and cloud running costs of relational databases. The first stage of our approach obtains workload and structure models of the database to be migrated from database logs and the database schema. The second stage performs a discrete-event simulation using these models to obtain the cost and duration estimates. We implemented software tools that automate both stages of our approach. An extensive evaluation compares the estimates from our approach against results from real-world cloud database migrations.

Keywords: Database modelling, Cloud migration, Enterprise systems, Model-driven engineering

Introduction

The benefits of hosting an enterprise system on the cloud — instead of on-premise physical servers — are well understood and documented [1]. Some organisations have been using clouds for over a decade and are considering switching provider [2], while others are planning an initial migration [3]. In either case, the most challenging component to migrate is often the database due to the size and importance of the data it contains. However, the existing cloud migration work focuses on the software components and gives minimal consideration to data. For instance, the ARTIST [4] and REMICS [5] cloud migration methodologies refer to the database but do not support any database specific challenges. Similarly, cloud deployment simulators like CDOSim [6] focus only on compute resources. The limitations of these existing cloud migration methodologies are described further in "Related work" section.

Migrating large relational databases from physical infrastructure into the cloud presents many significant challenges, e.g., managing system downtime, choosing suitable cloud instances, and choosing a cloud provider. The database could be deployed on a database-as-a-service offered by one of several public cloud providers, or installed and configured on a virtual machine(s). With either option, selecting the appropriate cloud resources requires knowledge of the database workload and size. The infrastructure of the source database may impact the migration duration; if it has limited available capacity or bandwidth, then it will take longer to extract the data. An organisation may wish to upgrade the existing database hardware to speed up migration, or schedule downtime to migrate the database while it is idle.

In this work, we assist with this decision-making process via a tool-supported approach for evaluating cloud database migration options. Our approach has two stages—database workload and structure modelling, and database migration simulation—and estimates migration duration, migration costs, and future cloud running costs.We assume the source and target databases have an identical: schema, type (e.g., relational or NoSQL), vendor (e.g., Oracle or MySQL), and software version. Changing any of these parameters is a complex activity, which organisations tend to perform separately (as discussed in "Approach overview" section).

Given logs and a schema of a candidate database, the database modelling stage generates: (i) a workload

*Correspondence: mhe504@york.ac.uk
[1]Department of Computer Science, University of York, Deramore Lane, York, UK

model conforming with the Structured Metrics Metamodel (SMM) [7], and (ii) a structure model conforming with the Knowledge Discovery Metamodel (KDM) [8]. The second stage of the approach uses these models, alongside a cost model of the target cloud platform, to perform a discrete-event simulation of the database migration and deployment. To ease the adoption of the new approach, we implemented two software tools that automate the main tasks.

We carried out an extensive evaluation of the approach using several open-source enterprise applications, and a closed-source system from our industrial project partner Science Warehouse [9]. In particular, our database modelling method and tool were applied to 15 systems (including Apache OFBiz, and MediaWiki) to obtain workload and structure models. In each case, the system was installed on a server and configured with an Oracle or MySQL database. The experimental results (detailed later in the paper) show that our tool can extract models from a broader range of systems and with lower overheads than the leading existing tool (Gra2MoL) [10]. Furthermore, we carried out a case study which showed that our database modelling tool could be extended to support a Microsoft SharePoint schema with less effort than Gra2MoL. Furthermore, we performed a case study that showed DBLModeller could be extended to support a Microsoft SharePoint schema with less effort than Gra2MoL.

To evaluate the accuracy of the database migration simulation stage of our approach, we compared its predictions to four real cloud database migrations. We migrated the Science Warehouse Oracle database: (1) from AWS to Microsoft Azure, and (2) from Microsoft Azure to AWS. The first migration used Amazon's EU Ireland region, while the second used the EU Frankfurt region. Furthermore, we migrated an Oracle database containing synthetic data for the Apache OFBiz ERP system between the same regions. The relative error between these real migrations and our predictions was between 4% and 22%. As explained in the threats to validity section later in the paper, fewer experiments were feasible for this stage of our approach. Mirroring real-world migrations requires databases containing tens to hundreds of gigabytes of data. The costs and time associated with this are significant.

To the best of our knowledge, our work represents the first tool-supported approach for the systematic, end-to-end evaluation of cloud database migration options. The main contributions of our paper are:

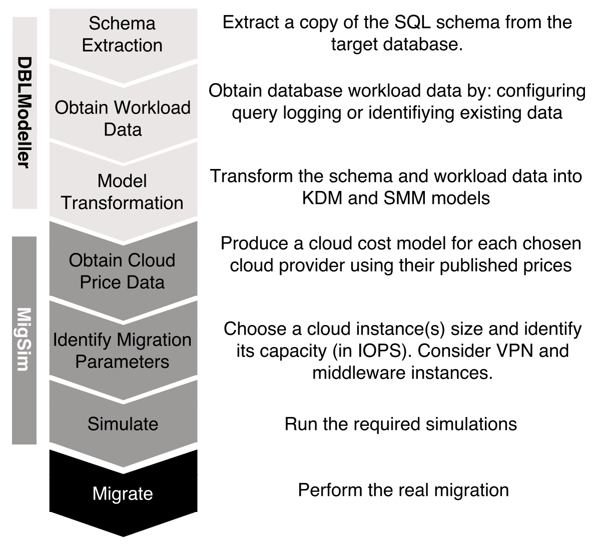
1. A new database modelling method for generating workload and structure database models.
2. A new method for simulating cloud database migration to estimate migration duration, migration costs, and future cloud running costs.
3. Software tools that automate our database modelling and cloud migration simulation methods.
4. An extensive evaluation of our approach and its tool-supported methods.

These contributions significantly extend our preliminary results on database model extraction [11] in several ways. First, the new approach includes a database migration simulation stage (described in "Migration simulation stage" section) that was missing from [11]. Second, we provide an algorithm for our model transformation technique (in "Model refinement algorithm" section). Third, we include a more detailed analysis of the SQL keyword survey results from [11] (in "SQL support" section). Fourth, we present new experimental results that evaluate the complete end-to-end approach by migrating real enterprise systems in the cloud (in "Migration" section). Finally, we make the tools used by our approach and (to enable replication) the experimental data and the results from their evaluation freely available on GitHub [12, 13].

The rest of the paper is structured as follows. "Approach overview" section provides an overview of our approach. "Database modelling stage" section presents our database workload and structure modelling method and tool implementation. "Migration simulation stage" section describes our simulation method, which uses the database workload and structure models to evaluate cloud migration options. "Evaluation" section presents our evaluation of the approach. Finally, "Conclusion and future work" section concludes the paper with a summary and suggests future work directions.

Approach overview

The main stages of our approach are 1) database modelling and 2) migration simulation. These are highlighted


DBLModeller
MigSim
Schema Extraction
Extract a copy of the SQL schema from the target database.
Obtain Workload Data
Obtain database workload data by: configuring query logging or identifiying existing data
Model Transformation
Transform the schema and workload data into KDM and SMM models
Obtain Cloud Price Data
Produce a cloud cost model for each chosen cloud provider using their published prices
Identify Migration Parameters
Choose a cloud instance(s) size and identify its capacity (in IOPS). Consider VPN and middleware instances.
Simulate
Run the required simulations
Migrate
Perform the real migration
Fig. 1 Two-stage evaluation of cloud database migration

in Fig. 1 as light grey and dark grey respectively. Each has three sequential tasks, which are supported by our DBLModeller and MigSim tools.

The database modelling method we devised for stage 1 is technology-independent and enables organisations with legacy enterprise systems to understand their database workload and structure. It semi-automatically synthesises a workload model (conforming to the Structured Metrics Metamodel [7]) and a structure model (conforming to the Knowledge Discovery Metamodel [8]). These enable the identification of read-heavy tables, write-heavy tables, and daily load patterns.

Both models are obtained using text-to-model transformations. The source of the structure model is a database schema extracted from the database. While the source of the workload model is existing or newly created database workload data (e.g., query log files). The metamodels they conform to are common in model-based software modernisation approaches (e.g., [4–6]), which ensures the interoperability of the models generated by our method with future technologies.

The second stage of our approach simulates the migration of a database to a new cloud provider. This consumes the structure and workload models from the previous stage. First, a cloud cost model is obtained for each cloud provider/platform considered as a potential migration target. To keep this task simple, we developed a cloud cost metamodel that captures only the cloud charges relevant for database migrations. The types of cloud charges vary significantly between providers, and our use of this simpler, domain-specific metamodel instead of a generic metamodel like [14] or [15] helps overcome the complexity due to this heterogeneity.

The next task is choosing the database migration parameters. These include the capacity of the target cloud database, server(s) specification, and whether a middleware (e.g., [16]) or VPN is required. Finally, a simulation is performed using our MigSim discrete-event simulator for each set of parameters.

Relational databases are prevalent in the cloud migration literature, and they are therefore the focus of our work. However, the approach (Fig. 1) does not have a strict dependency on one database type. We require a cloud cost model, a source of workload data, and a schema, which could also be obtained from NoSQL databases [17, 18]. The schema is used to interpret the workload model (i.e., the frequency data structures/tables are accessed) and track migration progress in the simulation. Using an inferred NoSQL schema will not affect the cost estimates.

While our approach is generic, the DBLModeller and MigSim tool implementations are specific to Oracle and MySQL out-of-the-box. The exact versions and supported constructs, and the extension mechanisms for other databases are discussed later in the paper.

The approach and tools require both databases to be identical apart from the data. Changing the database version, schema (e.g., to modernise its design), or database type (e.g., relational to NoSQL) are significant independent challenges [19–21] and outside the scope of our work. We expect many organisations to perform separate projects for cloud database migration and data conversion. Our industrial partner for this work, Science Warehouse, took this approach. Conversion of the data and database structures will require cloud or in-house compute resources. Determining the cost of this is a complex activity, which we propose in "Conclusion and future work" section as future work.

Database modelling stage

Systems being migrated to the cloud (and their databases) will usually have been in continuous development for a long time. This can result in unused tables or data, which engineers are reluctant to clean-up for fear of unintended consequences. This issue is compounded when there is poor documentation or when knowledgeable engineers leave the team. However, an accurate understanding of the database must be obtained to choose cloud migration parameters and approaches. This information is also required for other common activities, such as database refactoring, archival of old data, and potential transition to a NoSQL datastore.

A key challenge when developing a database modelling method is heterogeneity, as the available tools and features that the new method can exploit are different between each database provider. For example, MySQL Enterprise Monitor [22] can generate statistics on the workload and MySQL Workbench [23] can model the structure. Other databases have similar tools with varying functionality. As described next, our database modelling method and DBLModeller tool adopt a platform-independent approach that utilises a SQL schema dump and a SQL query log (if necessary). As we show later in the evaluation section, this approach overcomes many of the differences between SQL dialects.

Our database modelling method has five steps, as described below. The 'Obtain Workload data' task from Fig. 1 has been decomposed into Workload Source Identification, Query Logging, and Workload Extraction.

1. Schema Extraction. A user obtains a SQL dump from the target database using existing tools (e.g., [24] or [25]). Scripts from version control systems should be avoided as they may not be accurate.
2. Workload Source Identification. A user identifies any existing sources of workload data, for example, from

database monitoring tools. The goal is to produce a sequence of measurements containing: data being read or written, and the target (i.e., database table).

3. Query Logging. This task is only performed when workload sources identified in Step 2 provide limited information or are completely unavailable. Here an interceptor/spying library, such as the multi-platform P6Spy [26], should be used to record the queries. This needs to be in place for long enough to capture several usage cycles, i.e., minimum to maximum load variations.
4. Workload Extraction. A user processes the gathered data from Step 2 or Step 3 into a sequence of workload measurements. The DBLModeller tool can automate this step for P6Spy or Oracle query logs [27].
5. Text-to-Model Transformation. The sequence of workload measurements from Step 4 is automatically transformed into a SMM-based workload model [7]. Furthermore, the schema from Step 1 is automatically transformed into a KDM-based [28] structure model.

Deciding whether to perform Step 3 depends on the quantity and quality of workload information identified in Step 2. This information must provide a sequence of time-boxed measurements for the amount of data inserted into, and read from, various database entities. A *database entity* can be a set of columns, a table, a set of tables, or a schema.

Step 5 uses grammar-to-model mapping to perform the transformations required to obtain the KDM and SMM models. We significantly extended the approach from [29] with the ability to handle multiple SQL dialects as input. This is possible because we require high-level models and many of the differences between dialects are at the implementation level. Furthermore, we have used novel *executable annotations* rather a model-to-model transformation to restructure the model. Typically a text-to-model transformation (T2M) would be required to produce an intermediate model, then a model-to-model (M2M) transformation would be applied to get the desired model. Our executable annotations avoid this, meaning that only a single transformation is required. Furthermore, we used the standardised KDM and SMM metamodels to provide interoperability with other leading tools and approaches. KDM and SMM are already used extensively in REMICS, ARTIST, and other cloud migration projects [30, 31].

Based on our database modelling method, we implemented the DBLModeller command line tool to automate the 'Workload Extraction' and 'T2M Transformation' steps of the method. This transforms a SQL schema to a KDM structure model and a SQL query log to an SMM workload model. The query log must be in the P6Spy [26] or Oracle [27] format; alternatively, a CSV file containing workload measurements can also be used as input. The SQL schema must be in the format produced by Oracle Database Developer or MySQL Workbench. This allows database models to be easily obtained for the second stage of our approach.

KDM Compliance

The KDM is a large metamodel with twelve packages distributed across four abstraction layers. Each package allows some legacy system artefact to be modelled. The KDM's developers — the Open Management Group — proposed compliance levels [28] for tools using the metamodel, this reflects the fact that many tools (such as DBLModeller) do not wish to implement all of the KDM. Level 0 complaint tools support the: core, kdm, source, code, and action packages. A level 1 tool adds support for one or more of the: platform, data, event, UI, build, structure, and conceptual packages. Finally, level 2 tools support the entire metamodel.

Our database modelling method and DBLModeller tool only support the KDM data package, and therefore do not comply with any level. We made this decision because our goal differed from that of the KDM's developers, who focused on the model-driven re-engineering of a software system. If DBLModeller was to be used for this purpose, we envisage that it would be an "add-on" to other model extraction tools (e.g., [32]).

Model refinement algorithm

A key difference between DBLModeller's 'T2M Transformation' step and Gra2MoL [10] is our use of *executable annotations* to restructure (i.e., refine) the model. As such, during the text-to-model transformation (shown

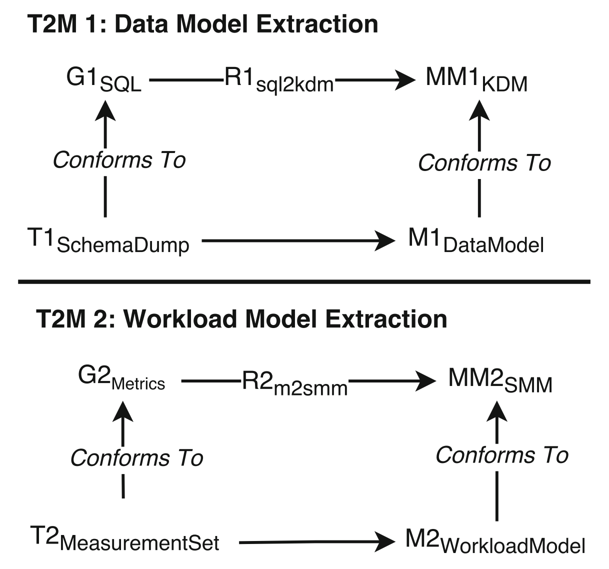

T2M 1: Data Model Extraction
$$\begin{tikzcd} G1_{SQL} \arrow[r, "R1_{sql2kdm}"] & MM1_{KDM} \\ T1_{SchemaDump} \arrow[u, "Conforms To"] \arrow[r] & M1_{DataModel} \arrow[u, "Conforms To"'] \end{tikzcd}$$
T2M 2: Workload Model Extraction
$$\begin{tikzcd} G2_{Metrics} \arrow[r, "R2_{m2smm}"] & MM2_{SMM} \\ T2_{MeasurementSet} \arrow[u, "Conforms To"] \arrow[r] & M2_{WorkloadModel} \arrow[u, "Conforms To"'] \end{tikzcd}$$

Fig. 2 Text-to-model transformations in DBLModeller

in Fig. 2) annotations are introduced into the KDM and SMM models. Afterwards, the models are searched to find, execute, and remove the annotations. The three types of annotations used by DBLModeller are described below.

1) Move annotations. These annotations act on the line on which they are placed, moving it to within another element in the model. The target element is identified by its name and type, which are included as parameters in the annotation. With the standard Gra2MoL framework, the element order in the model will match the input text.

2) Add annotations. These annotations create new model elements in a specific model location, this can either be the same location as the annotation or a different one. This is used to handle 'ALTER TABLE' statements and keys within 'CREATE TABLE' statements, as both would be impossible with a single Gra2MoL T2M transformation. For example, processing an ALTER TABLE statement requires some existing model element to be modified (e.g. add a primary/key foreign relationship). This goes against the traditional function of a T2M transformation which creates a model element when a statement is found in the text.

3) Reference annotations. These annotations serialise name-based references (e.g. SchemaName. TableName) to model paths, e.g.

$$//@element.1//element.3//element.0$$

Although Gra2MoL has a mechanism for tasks like this, an annotation is needed because model elements can be created after the grammar-to-model mapping is complete.

The DBLModeller model refinement is formalised by function REFINEMODEL from Algorithm 1. Given an *annotatedModel*, this function first invokes GETANNOTATIONS to obtain a list of Move annotations that is passed to EXECUTEMOVES (line 2). This ensures that all Move annotations, which will change the location of the other annotations, are executed before obtaining the Add and Reference annotations and passing them to EXECUTENONMOVES (line 3).

Migration simulation stage

The second stage of our approach uses discrete-event simulation to estimate the database migration cost, migration duration, and future cloud running costs. Our simulation method, called MigSim, focuses exclusively on the database and considers: (1) its size and workload, (2) growth trends, and (3) compute instance/virtual machine performance.

Accurate estimates of the database migration costs enable organisations to plan, budget and investigate trade-offs. From a planning perspective, the organisation will typically want to know how much time a migration requires, as this may rule out an Internet-based migration. One common trade-off an organisation might want to investigate is duration versus cost. Additional bandwidth or increased database performance could speed-up the data transfer into the new database. Similarly, they can look at the cost benefits of 'cleaning-up' the database before migration, i.e., identifying and removing unneeded tables or archiving old data. Our simulation method can be equally applied to migrations between clouds, or from on-premise databases to the cloud.

We employ simulation to estimate these migration values because of the large number of variables associated with cloud database migrations, and the complexity of their relationships. These variables include the source database workload, workload growth, database size, cloud instance performance, and cloud usage charges. In particular, the database workload and cloud instance performance are stochastic variables [33–35] which meant we ruled-out a model transformation/query to obtain results directly from the models.

From the numerous discrete-event simulators available, we decided to extend the CloudSim Framework [36]. This framework has existing functionality for simulating data migration between software applications running on different hosts. Furthermore, it has been used extensively to simulate other aspects of cloud systems (e.g., [6, 37]). The key extension we made to CloudSim was to associate costs with resource consumption. These costs are defined in a cloud cost model for the target cloud platform, which is also input into the simulation. The migration duration is dependant on the infrastructure capacity/performance; this performance data is provided to the simulation as a set of parameters. These contributions are highlighted in Fig. 3. Several useful components of CloudSim are reused, notably: (1) the underlying simulation engine, it's (2) time-shared resource allocation algorithms, (3) networking functionality, and (4) the VM, Host, and Datacenter classes.

Cloud cost metamodel

We developed a metamodel to define the structure and content of our cloud cost model. The wide ranges of services typically offered by cloud providers and the regional variations in pricing are complex; this makes a model-based approach for including cost data in the simulation desirable.

Our metamodel — implemented in the Eclipse Modelling Framework — has been designed to support charges from major public clouds which may be incurred during database migration. Each instance of our metamodel

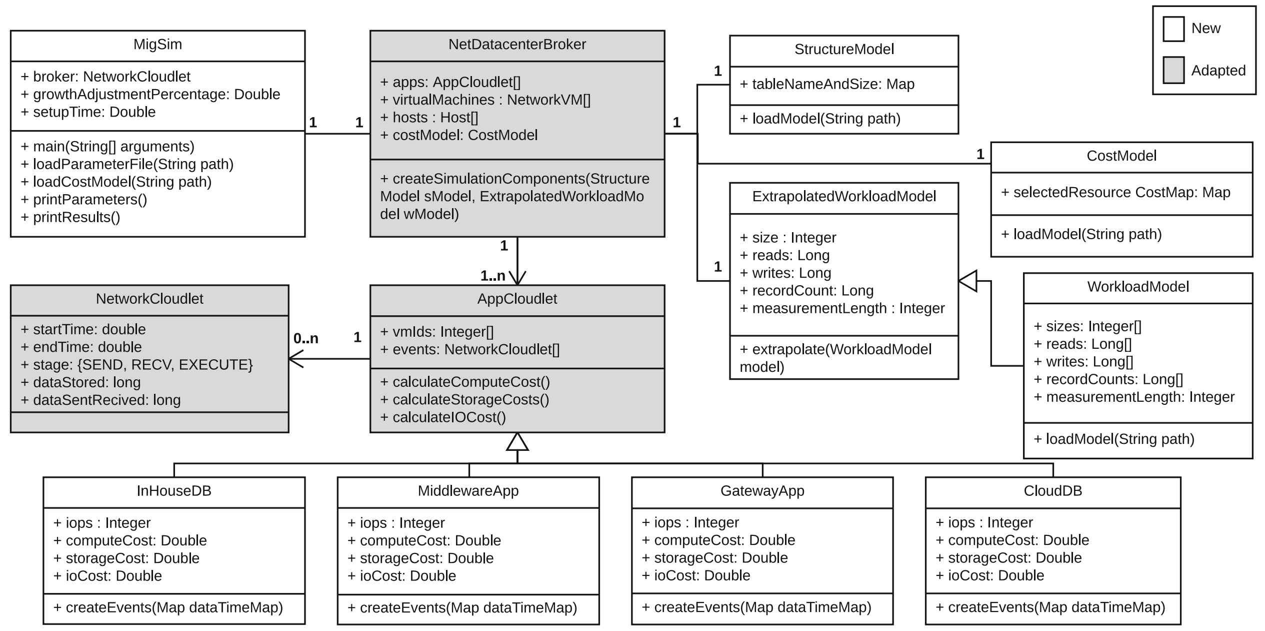


Fig. 3 UML2 class diagram showing the new major new components in MigSim, as well as those adapted from CloudSim

corresponds to a single cloud provider and covers a set of cloud services (e.g., Amazon RDS and EC2). Finally, each cloud service is associated with multiple cloud charges, as shown in Fig. 4.

We have chosen to exclude negotiable and market-based cloud charges from the scope of our metamodel. The majority of organisations are unlikely to deploy, and thus to pay for a relational database deployment in this way. For example, Amazon's EC2 spot instances are automatically terminated when the current market price exceeds an organisation's bid price. They are intended for transient workloads or those which can be accelerated when compute instances are cheaply available. Furthermore, negotiated/private pricing arrangements are typically the domain of extremely large-scale systems (i.e., hundreds of millions of users worldwide). These are outside the scope of MigSim due to their scarcity and unique characteristics.

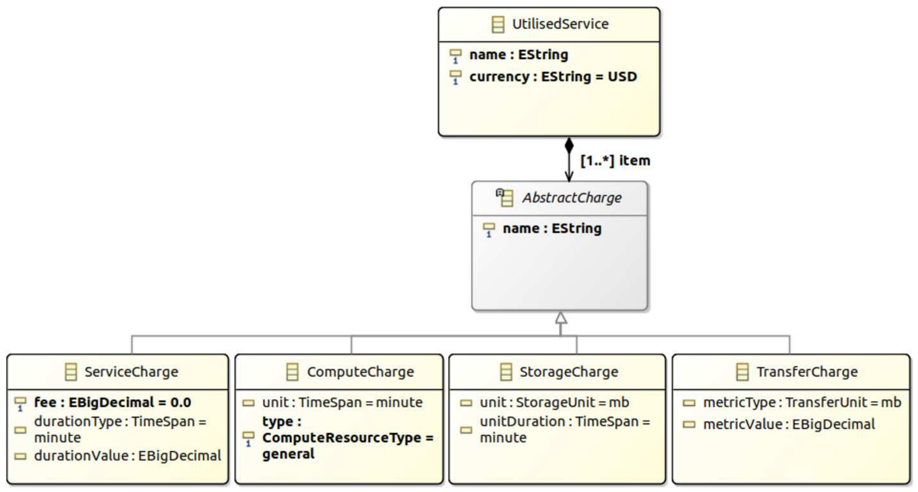


Fig. 4 A snippet of the cloud cost metamodel showing its key components

Four types of charge are relevant to database migration and therefore captured by our cloud cost metamodel: service, compute, storage, and transfer. A 'ServiceCharge' represents non-infrastructure charges, e.g, an AWS Support Plan, static IP addresses, or VPN connections. This consists of a fee and a duration which can be specified in minutes, hours, days, months, or years. A 'ComputeCharge' models the cost of a cloud Virtual Machine running for a given time (e.g., Amazon EC2), while a 'StorageCharge' corresponds to the cost of persistent data storage (e.g., Amazon S3). A 'TransferCharge' represents the cost for data sent or received across a network. Charges may be tiered so that a per-unit discount is applied for heavy usage or to allow regional price differences.

As part of our MigSim implementation, we manually created instances of our metamodel for Amazon Web Services and Microsoft Azure. These models contained real values taken from the providers' websites and enabled us to simulate the migration of multiple databases between the two platforms.

Several existing metamodels also capture cloud costs, such as those proposed by Leymann et al. [15] and Maximilien et al. [14]. However, these metamodels are not specific to the cloud migration of databases and model elements unnecessary for our use case are present in each metamodel. Rather than using an existing metamodel 'out-of-the-box', we could have tailored/modified it to suit our use case. After investigation this was ruled out because: (1) the effort for approach users would be greater, and (2) the tailored metamodel would be less intuitive. As our approach requires users to populate a cloud cost model from data on a provider's website, we considered ease-of-use to be essential.

Design

The migration environment within MigSim contains five key components as shown in Fig. 3: in-house database, middleware, gateway, cloud database, and the 'MigSim' simulation controller. The controller uses the structure and workload models to add data transfer tasks (called 'NetworkClouldets') to the other components. These tasks represent data migration and future database load. The data is extracted from the in-house database by the middleware, then sent through the gateway (e.g., a VPN) to the new cloud database. This transfer takes place over a simulated TCP/IP network to represent a migration over the Internet.

MigSim has four inputs: a workload model, structure model, cost model, and parameter set. Within the parameter set a user specifies: the cloud resources for each database or application (whose cost is defined in the model) and their performance. Optionally, they can override the workload model and specify the expected future growth percentage and any additional hours the migration infrastructure must be running (e.g., during non-working hours). The results provided when the simulation finishes are: future running costs, migration cost, and migration duration.

A 'middleware' and 'gateway' are included in MigSim as they are common migration components [16, 38, 39] and impact upon the cost and/or migration duration. For example, a Middleware component will introduce a bottleneck if its performance is worse than that of the databases. Either component can in-effect be excluded from the simulation by setting its cost to zero and configuring its performance to be higher than that of the other components. This supports migrations without middleware (e.g., if database replication functionality is performing the migration) or those where a VPN is not required.

MigSim has been implemented by using and extending the CloudSim framework. Each database or migration application from Fig. 3 inherits CloudSim's AppCloudlet object, which can be used to represent an application running in the cloud. Each AppCloudlet is assigned to a Virtual Machine, running on a separate physical Host, in a CloudSim Datacenter. This allows us to use CloudSim's functionality of simulating data transfer, although we extended this to associate costs with resource consumption.

The Workload Model specifies historical growth trends and usage patterns. This data must be extrapolated to predict the future database traffic if these trends continue. The storage space consumed by the database in the future is dependent on the amount of new data inserted. Furthermore, the provisioned database throughput required depends on future database traffic. We have used linear regression, specifically the ordinary least squares (OLS) method [40], to estimate the number of future read and write queries received by the database.

The OLS method produces a regression equation for the existing data, which minimises the sum of the squared errors (i.e., the error between each data point in the model and the equation). By default, the number of estimated future workload measurements is equal to the number of data points in the model. For example, a workload model covering the previous six months with one measurement per month will be used to produce estimates for the next six months. The state of the InHouseDB migration component (from Fig. 3) is the same as it was at the last measurement in the workload model. The NetDatacenterBroker component generates load for the InHouseDB, which matches workload values estimated with the regression equation. As the simulation runs, the

MiddlewareApp component migrates the data from the InHouseDB to the CloudDB using slack capacity.

As an alternative to extrapolation from the model, a user can specify an expected growth rate by setting a simulation parameter. For example, if the system will soon be launched to customers from a new country, then the growth rate can be increased accordingly. This allows business knowledge and plans to be included in the simulation.

The capacity of the source database, middleware, and target database, is defined in IOPS and input into the simulation. IOPS (or Input/output operations per second) is a performance measurement of storage devices like SSDs and cloud storage services. The throughput of a relational database depends on the performance of the persistent storage, CPU, and available RAM. However, in MigSim we have chosen to focus on the persistent storage IOPS and assume that other database server components are not a bottleneck. Typically in large enterprise-scale cloud databases, the storage costs are greater than those associated with CPU or RAM. The required peak IOPS for an existing database can be measured using the approach in [41].

Once the simulation is complete, the costs are calculated from the migration duration and the utilised cloud resources. MigSim only calculates the costs associated with the new infrastructure, i.e., the migration middleware, the VPN gateway on the new cloud platform, and the new database. The migration will increase the load on the existing database and utilise its Internet bandwidth, although the cost of this is heavily dependant on the organisation so it is not considered within our simulation method.

Additional compute time is included in final predicted costs for set-up, configuration, tear-down. As part of these activities the infrastructure may need to be kept running during non-working hours (e.g., overnight, weekends, or holidays). The *Additional compute time* parameter is manually specified by a user at the start of the simulation. It should be set according to the experience of the team; if they have previously worked with the system and cloud platform less time may be required. These activities are a necessary part of any migration and can significantly increase the total migration cost for small databases.

Evaluation

This section presents the extensive experiments we carried out to evaluate three aspects of our approach. First, we evaluate the extraction of database models in "Modelling" section. Next, we present and assess the accuracy of the database migration cost and duration predictions produced by the approach, and its estimation of cloud database running costs in "Migration" and "Running costs" sections, respectively. We conclude with a discussion of the threats to the validity of our findings in "Threats to validity" section.

Modelling

Our database model extraction method aims to support the widely-used [42] Oracle and MySQL dialects of SQL, and to be easily extensible to new SQL constructs or versions. In this section, we evaluate how effectively our method achieves these aims. Additionally, we evaluate the performance, completeness and correctness of the database models produced by our DBLModeller tool.

SQL support

Fully supporting every SQL dialect is impractical due to the number that exist and the size of the language. Therefore, our DBLModeller tool supports a subset of two SQL dialects: Oracle and MySQL. Whilst it is straightforward to identify which dialects to support (many organisations report on the estimated market share [42]), it is harder to select statements and keywords to support within these.

We obtained MySQL and/or Oracle schemas for the 15 real-world databases shown in Table 1. Every schema was obtained by deploying an instance of the system then using MySQL Workbench or Oracle SQL Developer to extract a SQL schema dump. This gave a sample size of 15 schemas which were analysed to identify the MySQL or Oracle SQL keywords being used.

Figures 5 and 6 show the DBLModeller support for the 25 most used keywords in our database schema sets for MySQL and Oracle, respectively. None of the keywords

Table 1 Database schemas used for keyword analysis

System	Type	Domain
Science Warehouse	Oracle	E-Commerce
University of Murcia Record System	Oracle	Record System
Apache OFBiz	Oracle & MySQL	Business Management & E-commerce
MediaWiki	Oracle & MySQL	Collaboration
Confluence	Oracle & MySQL	Collaboration
Joomla	Oracle	Website Management
Magneto	Oracle	E-commerce
SonarQube	Oracle	Software Engineering
Mantis	Oracle	Software Engineering
WordPress	Oracle	Website Management
Moodle	Oracle	Education
OrangeHRM	Oracle	Record System
SuiteCRM	Oracle	Business Managment
RefBase	Oracle	Education
OpenMRS	Oracle	Record System

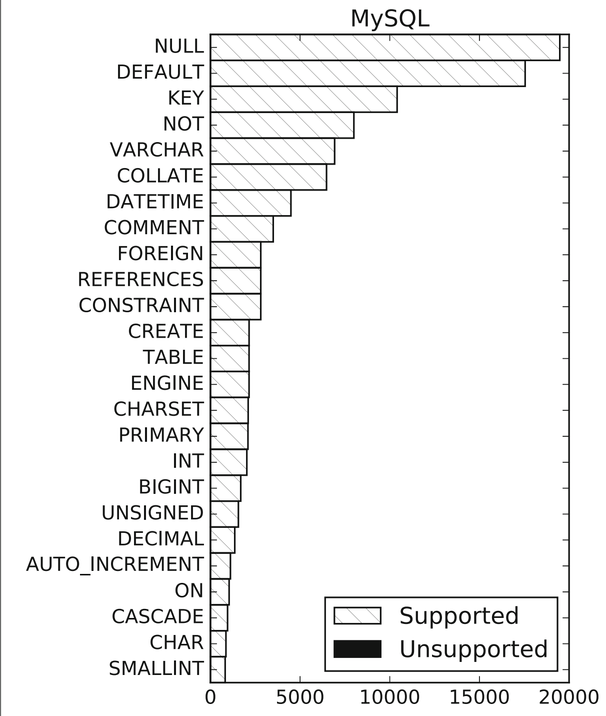


Fig. 5 Most frequent SQL keywords for the MySQL

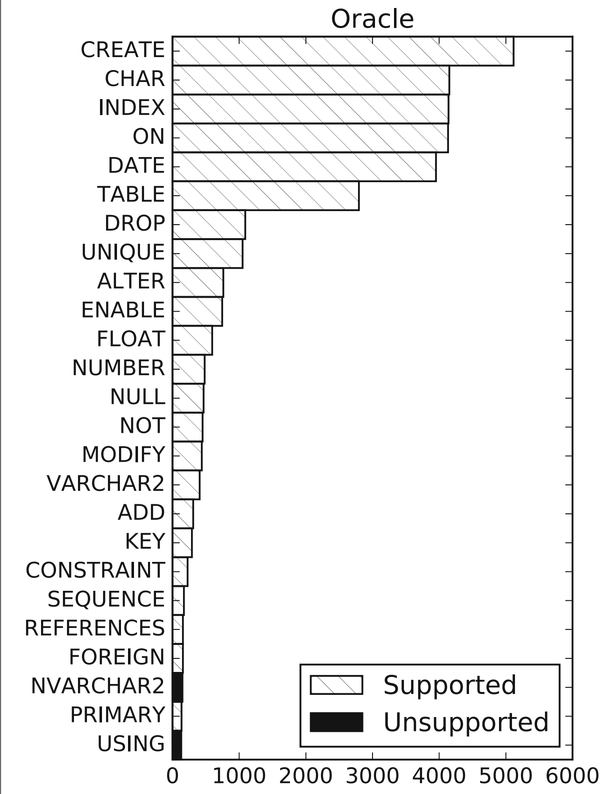


Fig. 6 Most frequent SQL keywords for Oracle

which appear in the MySQL top 25 are unsupported, while only two keywords in the Oracle top 25 are unsupported: NVARCHAR2 and USING. NVARCHAR2 is only used by Confluence (albeit extensively), however this means the data type for NVARCHAR2 columns will be null. If necessary, support for NVARCHAR2 could be added by modifying one line of the SQL grammar (to map it to the KDM:String data type). The lack of support for USING is not an issue because it specifies whether an index is enabled or disabled in the Confluence schema, and this detail is lost when abstracting to a KDM model. Our analysis used the published keyword lists for each dialect [43, 44].

Tailoring SQL support to the language constructs used in schemas is a time-effective approach for developing a structure model extraction tool. Alternatively, we could have supported all Data Definition Language (DDL) statements (as defined in the Oracle and MySQL documentation). DDL statements are those that allow the creation, deletion, and alternation of schema objects. A complete SQL dialect also includes Data Manipulation, Transaction, and System Control Statements. However, our keyword analysis showed that only 41% of DDL keywords were used in our sample of database schemas. This supports our claim that development effort can be saved through selective keyword support with minimal impact on system support.

Extensibility

It is inevitable that DBLModeller — or any similar model extraction method — will need to be extended. Our decision to implement common SQL keywords means some organisations will have to add to our implementation even when modelling Oracle or MySQL databases. Some systems will use less well-know SQL-based databases, and new versions of popular databases will be released.

We evaluated the extensibility of DBLModeller by comparing it against Gra2MoL [10, 45], the leading SQL-to-KDM extraction tool. For this purpose, we carried out a case study where we extended both tool-supported methods to accommodate a Microsoft SharePoint schema.

The schema was obtained from a Microsoft SharePoint 2013 instance installed on a Microsoft SQL Server database. This installation was created specifically for the case study. The schema rather than the data was needed, so our results are unaffected by the system being unused. SharePoint uses 16 schemas, and the largest of these was selected; this schema contains 7 KLOC consisting of 136 tables, 5442 columns, and 61 indexes. Our goal when choosing a system for this case study was to have a large schema unsupported by both tools in equal measure.

The changes needed to DBLModeller to support the schema were determined by attempting to extract a

Table 2 Lines of code to support the SharePoint schema

	G2M new grammar		DBLModeller extension	
	New	Updated	New	Updated
Lexer	70	0	25	6
Parser	11	0	6	14
T2M	25	85	24	1
M2M	10	5	0	0
Total	*116*	*90*	*55*	*21*

model, then noting any errors produced. These were then fixed incrementally and the number of modified lines of code were counted. However, with Gra2MoL a new ANTLR grammar had to be developed to parse the schema dump. This makes it possible to compare the work required to extend a grammar against the work required to develop a new grammar.

Table 2 presents the results and shows that the extension of DBLModeller required fewer code changes. The use of our annotated T2M transformation meant that no M2M rule changes were needed. The use of a multi-dialect grammar meant it was unnecessary to write a new grammar for the Microsoft SQL dialect. Instead, we modified various rules in our existing grammar. However, when comparing the development time/effort in extending the two tools it is important to consider whether a new LOC represents the same effort in each. As identical technologies are used in the Gra2Mol PLSQL example and DBLModdler (ANTLR for the Lexer/Parser, and the G2M DSL for the T2M rules), the results should be comparable. We conclude that the changes made in DBLModeller have had a positive effect on extensibility. Furthermore, the similarities between SQL dialects meant that extending the DBLModeller was a straightforward task.

Completeness, correctness, and performance

Model completeness, model correctness, and model extraction performance have been evaluated together due to their interdependence (e.g., extracting a complete model will require more time than an incomplete model). DBLModeller was compared to Gra2MoL's PLSQL2KDM example [45] as this had the highest level of SQL support at the time of writing. We extracted KDM models from the database schemas of four systems: Apache OFBiz, MediaWiki, Science Warehouse, and a student record system [10]. With OFBiz and MediaWiki we obtained Oracle and MySQL versions of the schema by installing them on both databases. Additionally, SMM models were extracted from Wikipedia (using six months' worth of data from [46, 47]) and from Science Warehouse's system.

Model completeness was assessed by comparing the number of model elements and input elements, while for correctness the properties of the model elements and input elements were compared. We developed a small model checking tool to automate this analysis. DBLModeller was able to extract models from the six schemas successfully, and from the output of our tool we concluded: that the input text and the output model had the same number of elements; that all table, column, and sequence names were correct; and that relationships between tables were correct. Furthermore, we confirmed that the models conformed to KDM and SMM using the Eclipse Modelling Framework.

The performance of DBLModeller was assessed by extracting a KDM model for each schema and measuring the time taken. This process was repeated 20 times per schema. We expected that the removal of the M2M transformation from the model extraction process will have significant performance gains. A virtual machine on the Digital Ocean cloud platform with 4GB of RAM and two Ivy Bridge based Intel Xeon cores was used to perform the experiment.

The performance results are presented in Table 3, which shows that DBLModeller can extract a KDM model in less time than Gra2MoL for every schema from our experiments. As Gra2MoL supports fewer SQL statements than DBLModeller, in order to obtain results it was necessary to modify the schemas by removing unsupported content until they could be processed by Gra2MoL. Simple calculations based on the results from Table 3 show that the DBLModeller model extraction times *per KLOC* were up to 86% smaller for Oracle schemas and up to 84% smaller for MySQL schemas. Finally, we note that the DBLModeller model extraction times do not exceed 3 minutes: extracting the model for the largest of our schemas took only 174s. This is fairly insignificant in the context of a cloud database migration, which is a positive aspect of our approach.

Table 3 Model extraction times using DBLModeller and Gra2MoL

Schema	Size (KLOC)	Tool	Mean (Secs.)	Std. Dev.	sec/ KLOC
Oracle OFBiz	31.5	DBLM	174	2.35	6
	10.3	G2M	237	3.4	24
Oracle MediaWiki	2	DBLM	7	0.23	4
	0.8	G2M	14	0.68	18
Oracle Sci-ware	1	DBLM	5	0.19	5
	0.4	G2M	14	0.62	35
Oracle UoM	0.3	DBLM	5	0.21	17
	0.3	G2M	10	0.62	33
MySQL OFBiz	21.7	DBLM	104	2.13	5
	9.5	G2M	230	9.73	24
MySQL MediaWiki	1	DBLM	5	0.24	5
	0.4	G2M	13	0.53	33

Migration

Prediction of migration cost and duration

The key goal of our approach is to accurately estimate cloud database migration cost and duration. This would enable different migration options and parameters to be evaluated, therefore supporting decision-making. We measured the achievement of this goal by comparing our predictions against (1) real cloud database migrations and (2) cost calculators from the cloud providers [48, 49]. The database migrations used a closed-source system from our industrial partner Science Warehouse [9] and the open-source ERP System Apache OFBiz [50].

Both databases (Science Warehouse and OFBiz) were migrated twice: from the existing cloud platform to a new cloud platform, then back again. This provided two data points per system for our evaluation. While we used public clouds as the source and target infrastructure, our approach can also be applied to in-house to cloud migrations. The Amazon Database Migration Service was used as middleware to perform both migrations. Additionally, the Science Warehouse migration required a VPN between the two clouds to secure the data during transfer.

The Science Warehouse system is an enterprise procurement system for making purchases in business-to-business scenarios. At the centre of this is a product catalogue which is populated by 'supplier' organisations, from which 'buyer' organisations can make purchases. The vast majority of users are in the United Kingdom and Ireland, resulting in peak loads during business hours in these countries.

Apache OFBiz (Open For Business) is an ERP system which contains applications for: e-commerce/online shopping, fulfilment of orders, marketing, and warehouse management. Unlike with Science Warehouse we did not have a real instance to use; therefore we populated the system with 200GB of synthetic data to represent the use-case of large online retailer. This was random data which matched the purpose of the column, e.g., 16-digit numbers following the Mastercard and Visa format were inserted into a credit card number column. All database tables were populated.

The Science Warehouse database was migrated while idle and the OFBiz database was migrated with synthetic load applied. This reflects how some of our approach users can perform the migration with the system shutdown. However, many larger systems would be critical to the organisation and this would be impossible. The load represented a user base within a single country, with two daily peaks at approximately 1100 and 1500. A daily total of 1.3 million queries were made; 90% between 0600 and 2000. Two Amazon EC2 instances were used to send these queries to the database server.

Our four migrations are modest in terms of size and cost. Many organisations migrating enterprise system databases will be moving data between database clusters rather than single servers, making the cost more significant. A common reason for database migration is scalability, where database load is reaching capacity. Limited capacity would therefore be available to migrate the database while it is being used. Inducing such large databases and loads was not feasible in this evaluation due to the high costs. However, we expect the accuracy of the experimental results to be similar for larger systems (such as those migrated in [2, 51], which have similar characteristics).

The Amazon Simple Monthly Calculator [48] and the Microsoft Azure pricing calculator [49] are used for each migration to provide a cost baseline, as shown in Table 4. These are often the first tools an organisation will use when planning a cloud migration. However, compared to our approach they have significant limitations. Most notably they require a user to accurately identify the cloud resource they require. As workload information not directly considered by these tools, there is no indication when the selected cloud resources represent over or under provisioning. Furthermore, the Amazon Cost Calculator does not include the Amazon Database Migration Service and can only predict costs for one month.

The lack of support for determining the inputs to the cost calculators can cause an organisation to make coarse-grained estimates based on the size of the existing database servers. We based the calculated costs in Table 4 on running the migration infrastructure for one week. This represents a typical estimate for systems of this size without knowing detailed workload information [52]. For the calculations, the Science Warehouse database is migrated to a 'db.m4.2xlarge' AWS instance and a 'D4v2' Azure instance. The OFBiz database is migrated to a 'db.m4.4xlarge' and 'D5v2' respectively.

Table 4 Predicted migration costs

System	Migration	Size	Mean migration duration (Std Dev.)	Total cost	Cost calculator baseline
Science Warehouse	AWS → Azure	38GB	417 Min (7 Min.)	$32.96	$264.68
	AWS ← Azure	18GB	144 Min. (2 Min.)	$26.41	$408.59
Apache OFBiz	AWS ← Azure	200GB	1147 Min. (12 Min.)	$12.10	$821.71
	AWS → Azure	163GB	901 Min. (8 Min.)	$9.68	$508.93

Table 5 Actual migration costs

System	Migration	Size	Migration duration	Total Cost
Science Warehouse	AWS → Azure	38GB	402 Min.	$40.12
	AWS ← Azure	18GB	147 Min.	$30.75
Apache OFBiz	AWS ← Azure	200GB	1176 Min.	$13.30
	AWS → Azure	163GB	888 Min.	$6.96

The predicted migration cost and duration obtained with our modelling and simulation approach are shown in Table 4 alongside the cost calculator baseline. Each simulation was performed 20 times on a laptop PC with a Intel i5-6200 (dual core, 2.3GHz) and 8GB of RAM. For comparison, the costs for the real migrations we performed are shown in Table 5. These values were obtained using data from Amazon CloudWatch and the Microsoft Azure Active Log, which are services that record the creation and deletion times of each cloud resource.

The Science Warehouse outbound migration took 0.3 hours less than our prediction and its "return" migration took 0.15 hours less; a relative error of of 4% and 22%, respectively. In contrast, the outbound OFBiz migration took 1.6 hours longer then predicted and the return migration took 0.8 hours longer; these figures correspond to a small relative error of 8% and 5%, respectively.

The simulation Execution times ranged from 8 minutes for the 18GB simulation (best case) to 144 minutes for the 200GB simulation (worst case). When computing the cost, MigSim rounds the migration duration up to the nearest full hour, in line with the cost model of many cloud providers. Therefore, the cost did not vary between runs, although small differences may arise for larger datasets.

The 'Total Cost' columns in Tables 4 and 5 include *Additional Time*. As discussed previously in "Design" section, migrating a database often requires the underlining infrastructure to be running for longer than the data transfer. Common tasks include: set-up, VPN configuration, tear-down, and non-working hours where it is not possible to the delete the cloud infrastructure. MigSim allows such tasks to be accounted for when predicting migration costs. We manually estimated additional time for each migration and input this as a migration parameter (1 hour for Apache OFBiz and 16 hours for Science Warehouse due to security requirements).

A side-effect of database migration is that the target database will consume less storage space and perform better than the source database (despite identical data). This is due to different amounts of data being inserted and removed during routine usage of the source database, creating fragmented free space [53]. Furthermore, all schema objects are (tables, indexes, etc.) are essentially rebuilt in the target database. We have included the size column to show the storage space consumed by the database before it was migrated. This value was provided as input to our simulation via each system's workload model.

For each migration (simulated and actual) the databases' SSD/HDD was matched to its workload, as would be expected for any existing system. However, every SSD/HDD type in AWS or Azure has its own price and performance characteristics which impacts our results. On AWS the Science Warehouse database used magnetic storage which cost of $0.0002 per GB-Hour and a published performance of 100 IOPS. On Azure a 'P6 Premium Managed Disk' was used (240 IOPS, $0.001). Each cloud provider has a different way of abstracting from the physical SSD/HDD in the their datacentre and the virtualised storage devices they sell to users. As a result, it challenging to have mirror the performance characteristics on the source and target side of a migration.

The Apache OFBiz database used a provisioned IOPS SSD on AWS (1000 IOPS, $0.0017) and a P20 Premium Managed Disk on Azure (2300 IOPS, $0.0002). Both databases have higher performance levels than for the Science Warehouse system due to the increased size. The impact of this extra performance can be seen in Table 4 and 5 as the migration time is not proportional to size.

Our evaluation compares like-for-like hardware so the performance differences do not affect the results. For example, the AWS to Azure migration of the Science Warehouse database uses magnetic storage (Table 5). The simulation of this migration models the performance of magnetic storage (Table 4).

Running costs

In this section we evaluate the accuracy of the estimated cloud running costs produced by our tool-supported approach. Running a system on multiple clouds for a long period of time (with additional servers applying a synthetic load) was not feasible for our evaluation, so we instead focused on the auto-scaling accuracy. Our simulation considers the future database load (extrapolated from the workload model) and the capacity of the new database server. Once the load exceeds capacity an additional server is introduced i.e., auto-scaling takes place. The error

Table 6 Predicted cloud running costs for Apache OFBiz (inc. migration)

Cloud	Load	Mean cost (year 1)	Mean scaling point	Std. dev	Cloud cost calc.
	+40%	$7,084	Week 16	0.46	
AWS	+20%	$6,410	Week 24	0.39	$16,396
	+5%	$6,073	Week 28	0.46	
	+40%	$15,003	Week 16	0.39	
Azure	+20%	$13,574	Week 24	0.46	$32,564
	+5%	$12,860	Week 28	0.46	

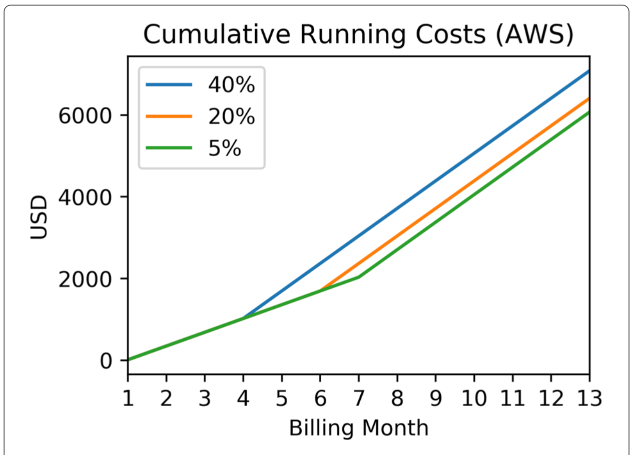


Fig. 7 Cumulative cost for an AWS deployment of Apache OFBiz

between simulated auto-scaling and actual auto-scaling is the main factor influencing the accuracy of the running costs. For example, if our simulation underestimated the database performance then servers would be added too soon and inflate the predicted costs.

Table 6 presents the predicted running costs we obtained for Apache OFBiz on Amazon Web Services and Microsoft Azure. Figure 7 shows how these costs are incurred over time. These values are based on the synthetic OFBiz models we created using the DBLModeller method before cloud migration. Additionally, we used the cloud cost calculators from Azure [49] and AWS [48] to provide a comparison baseline. Neither tool supports the selection for cloud resources for the current system's workload (or any future workload growth). We therefore estimated the cloud resources required at the end of the one-year period and ran these for the entire year without any up-scaling. This represents over-provisioning at the start of the year, although as upgrading from a single database server to multiple servers is complex we believe organisations such as our project partner would over-provision due to uncertainty. For example, an organisation would likely wish to migrate to a cluster of two database servers from the start rather than upgrade from one to two servers at the end of the first month.

By default our approach produces cost estimates based on the assumption that the observed load trends in the model will continue. The growth rate can also be manually adjusted to accommodate business plans. The results in Table 6 use this functionality to produce the 5 and 40% growth rates. These represent potential business scenarios where growth is higher or lower than expected. The results for 20% growth represent the actual trend in the model.

In order to assess the accuracy of these results, we set up a single OFBiz instance and applied the load from Table 6 for the 20% growth rate. The queue length and error logs for the Oracle database were monitored. At this load the database should be below capacity, therefore the queue should be stable and no timeout errors should be thrown (ORA-12170); these characteristics were confirmed in our experiment. The load was then ramped up by 1 query/sec every five minutes, until a timeout error was thrown by the application.

The target database performance was set to 100 IOPS, which matched the storage performance of a HDD-backed AWS RDS instance [54] and a HDD-backed 'standard disk' in Azure [55]. We selected the storage devices with the lowest level of performance to reduce the cost of our evaluation—these devices required the least synthetic/generated load to reach full capacity, and therefore we had fewer cloud instances to pay for.

In MigSim, 100 IOPS equates to an auto-scaling threshold of 24 queries per second for each database server. This threshold remains constant in every simulation, although it is reached at time moments that depend on the growth rate (sooner for high growing rate, as shown in Fig. 7). In this definition, a query is a typical CRUD (create, read, update, or delete) SQL query for an ERP system. Batch, back-up, and replication tasks are excluded. The calibration between published database performance (IOPS) and typical database queries per second is a fixed-ratio. The default value for this ratio was determined during the design phase of MigSim by analysing the database traffic from Science Warehouse and Apache OFBiz.

On Amazon AWS, the first timeout error was logged at a load of 27 queries per second (cf. Fig. 8). Database capacity was therefore reached 3 queries/sec after it was in the simulation. This is a relative error of 12% and meant that our simulation "launched" a new instance six weeks earlier than necessary. This would result in the simulated cost likely being $246.96 higher than actual running costs. However, scaling-out a database from 1 to 2 instances is a

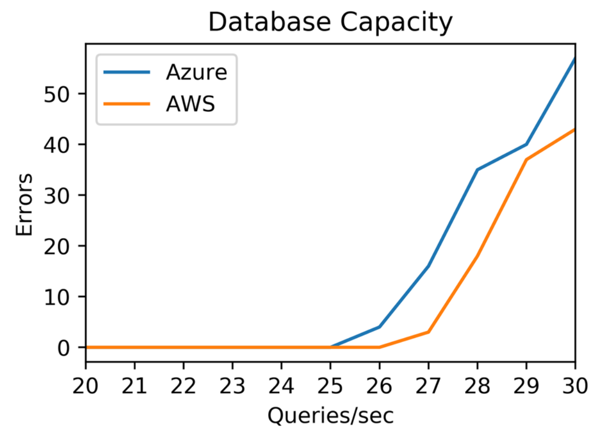


Fig. 8 Timeout errors received at different load levels

time consuming and non-trivial task. We expect that this operation would take multiple weeks, therefore reducing the error.

On Microsoft Azure the first timeout error occurred when the load reached 26 queries per second, i.e., a relative error of 8% compared to the simulation results. This would result in the predicted cost being $722.24 higher than the actual cost for this scenario.

The accuracy of auto-scaling depends on multiple factors, including the performance of the cloud instance [56], the size and structure of the database, and the queries being made to the database. However, we expect an error of up to ±12% to be typical for most enterprise web applications when modelling database performance on published IOPS.

Comparing the predicted costs from our approach with those from the existing cloud cost calculators [48, 49] shows an increase of between 53 and 60% (Table 6). This is due to over-provisioning as the calculators do not use workload information to select the required cloud resources. Cloud cost calculators are typically intended to produce cost estimates for short periods, where the resource requirements are known. However, migrating large growing database to the cloud requires a more sophisticated approach to determine costs. We have shown that MigSim can accurately predict when the database infrastructure requires scaling, and therefore the number of database instances that must be running.

Threats to validity

Construct validity threats may be due to simplifications and assumptions made when evaluating our approach. During the evaluation of the database modelling stage of the approach, 15 real-world database schemas and logs were used. Similarly, our evaluation of cloud migration costs and duration compared the predictions of our database migration simulations to real-world migrations. On the other hand, cloud running cost accuracy was inferred from database capacity accuracy (for the reasons presented in the results analysis from the previous section). Real long-term experiments to assess this cost would be necessary to increase the confidence that this result is accurate.

Internal validity threats can originate from how the experiments were performed, and from bias in the interpretation of the results. To mitigate against these threats, we repeated the experiments prone to such bias (i.e., the database migration simulations with MigSim and the DBLModeller performance results) over 20 independent runs. The code, experimental data, and results are publicly available in our GitHub repositories at [12] for DBLModeller and [13] for MigSim to enable replication. While the real-world cloud database migrations used to evaluate our estimated migration cost and duration were small-scale, we have referenced much larger cloud migrations when interpreting the experimental results. It should also be noted that migrating databases from ERP systems is an expensive and time consuming process. While we made full use of our systems by migrating them twice, experiments on other systems should performed to confirm our encouraging findings.

External validity is concerned with the generality of our approach. The characteristics of the closed-source system used in our work (Science Warehouse) have been explained in detail so the results can be generalised. Our migration and running cost evaluation focused on Oracle databases although since our approach abstracts away from the database implementation it can be applied elsewhere. MigSim considers the database capacity, load, and size; these are all implementation-independent properties. Furthermore, we used two cloud platforms during the evaluation (Amazon Web Services and Microsoft Azure) and argue that our approach is compatible with other IaaS and PaaS providers.

One other concern with the DBLModeller evaluation is whether the sample sets of schemas used in "SQL support" section are representative of the real world. This has been mitigated by choosing multiple schemas from systems of varying sizes and from different domains. A second concern is that a single case study has been used in "Extensibility" section to evaluate the extensibility. To partly address this concern, the case study used a Microsoft SharePoint schema with characteristics common to a wide range of systems. Nevertheless, additional evaluation is required to confirm generality for databases with characteristics that differ from those in our evaluation.

Related work

The Cloud Adoption Toolkit from Khajeh-Hosseini et al. [57, 58] is an approach for determining cloud migration feasibility. It considers (1) cloud adoption costs, (2) risk management, and (3) understanding tradeoffs between cloud benefits and migration risks. Compared to our work, Khajeh-Hosseini et al. have looked at the broader problem of feasibility. Their cost modelling method is high-level and focuses on current infrastructure costs, without considering utilisation. We add the ability to determine utilisation from workload models, avoiding over-provisioning on the new cloud. Their cost estimation tool (also used in [38]) has been released commercially as PlanForCloud [59]. The authors recommend modelling an application and its underlying infrastructure to estimate cloud running costs, thereby endorsing our approach. Furthermore, we consider cloud-to-cloud migrations in addition to the feasibility of the initial migration.

Binz et al. [21] present an approach for migrating enterprise applications to the cloud. Their key focus is on modelling the existing system (including all software, middleware, and hardware components) and deploying a 'cloud enabled' version on a cloud. The key tasks in this process are also automated. Our own approach complements this work by adding workload-based cost prediction. This brings two key benefits. Firstly, our approach could be used to determine financial viability and/or to select a cloud platform. Secondly, it could be used during the cloud-enablement phase of the approach to choose between cloud database types. The notable strengths of [21] include: (1) the high-level of automation, and (2) the ability to support any system or database. We aimed to incorporate these by also taking a model-based approach.

The methodology devised by the REMICS project [4] supports the migration of legacy systems to the 'service cloud paradigm', i.e., systems with a service-orientated architecture running in a cloud. Its focus is the re-design of the code, with related issues like testing, deployment, and interoperability also considered. The methodology is underpinned by a toolset to extract models from COBOL applications and generate Java from these. Similarly, the ARTIST methodology and framework [5] supports the migration of legacy systems to the cloud. Compared to REMICS, it adds feasibility assessment and business process modification. The authors argue that these are common and important activities when migrating to the cloud. Neither work investigates modelling or migration of the database explicitly.

CloudMIG [30] is a model-based tool which automatically determines a system's suitability for a cloud platform. It extracts an architecture model from the source code and a utilisation model from log files. Together with a model of the target cloud environment, these are used to identify if the system is incompatible, compatible, ready, aligned, or optimised for the cloud. However, it does not consider the database when identifying compatibility. CloudMIG extends the CloudSim simulation framework (like our MigSim), but the modelling and simulation components are not separate. This tight coupling prevents their reuse for evaluating cloud migration.

Strauch et al. [60, 61] propose a novel methodology for migrating the database of an existing system to the cloud. It focuses on complex migrations where significant quantities of data exist, and the software components remain on physical in-house servers. The strengths of this methodology include its level of detail, the evaluation with large real-world systems (SimTech SWfMS and NovaERM), and the accompanying Cloud Data Migration Tool. As a result, many organisations could easily employ it to migrate their databases. Our approach complements this methodology by accurately predicting database migration and deployment costs.

Other key approaches in the field of database modernisation and migration include Minimal Schema Extraction [62] and the business knowledge discovery framework from Normantas and Vasilecas [63]. The latter is a model-based framework for discovering *terms* and *facts* from a database. Terms are business concepts, and facts make assertions about these as defined in SBVR [64]. The researchers argue that some business vocabulary can be ingrained/implemented in a system which is not present elsewhere, and it is valuable to extract this. Their xText-based tool extracts a KDM data model from a SQL schema and derives a KDM conceptual model containing the term/fact units.

The existing approaches for extracting KDM models from SQL use an intermediate ASTM-based model [10, 63]. The Abstract Syntax Tree Metamodel is an ADM metamodel [65] which is used to standardise the syntax tree produced from the source code. The goal of the metamodel is to improve interoperability. A model-to-model transformation is then performed on the ASTM model to produce the KDM model. An intermediate model requires two transformations to be developed and maintained. However, in our DBLModeller a different approach has been used: the T2M transformation produces an annotated KDM model, then the annotations are executed to increase the level of detail. This approach reduces the lines of code needed to achieve the model extraction, bringing performance and maintainability benefits while reducing the potential for defects.

Perez-Castillo et al. propose automated approaches to extract a data schema from software source code [62] and encapsulate legacy databases using web services [66]. These will likely be of interest to organisations dealing with legacy databases. In [62] a static analysis of a systems source code is performed to identify and extract the SQL statements it contains. These SQL statements are used to generate a schema that only includes the tables/columns used by the application. However, the disadvantage of this approach is that it is difficult to implement. Many programming languages, SQL variants, and database access libraries (e.g. Hibernate) exist; and in practice, a tool will likely need to be system specific to capture all of the queries embedded in the code. In contrast, DBLModeller uses SQL dumps produced from database IDEs (e.g. MySQL Workbench or Oracle SQL Developer) to alleviate this problem and to support a wide range of systems. These produce SQL in a standard output format irrespective of how a system queries its database.

In addition to the work discussed above, several survey papers examine the field of cloud migration. Most notably, Gholami et al. [67] review 43 papers to identify their features, similarity, and research quality. The authors observe how tool support for cloud migration remains limited despite the quantity of work in the area and several papers highlighting this issue. The survey also confirms the novelty of our work as none of the papers predicted migration and deployment costs using system models. However, several did comment on the significance of the costs involved when migrating a system to the cloud.

Conclusion and future work

Three crucial pieces of information when migrating a database to the cloud are duration, migration cost, and future running costs. A detailed understanding of each allows an organisation to choose between cloud providers and different migration methods (e.g., Internet versus HDD shipping). Existing cloud cost calculators and approaches do not consider a system's current and future workload. They rely on a user accurately matching their workload to the advertised cloud resources. Their accuracy is therefore limited in real-world use. To address this gap, we developed a two-stage approach for evaluating cloud database migration options.

The first stage of our approach uses DBLModeller, a tool-supported method for modelling database workload and structure in a platform independent way. Given a database that needs to be migrated, this method generates a structure model conforming to KDM [8] and a workload model conforming to SMM [7]. These standardised metamodels ensure interoperability with exiting modelling and cloud migration tools, e.g., [4, 5]. Previous database modelling tools did not capture the properties which influence cloud migration costs, i.e., growth and query patterns. Furthermore, DBLModeller decouples the extraction of the KDM and SMM models from their use (e.g., SMM was used within CloudMIG [30] but the user cannot access the model and other SMM models cannot be used as input).

DBLModeller was evaluated using database schemas and log files from multiple real systems. Our experiments showed that DBLModeller can extract models from a wider range of systems and can be extended with less effort than the leading existing tool (Gra2MoL [10]). These key benefits were achieved by removing a model-to-model transformation from the model extraction process and by using a single multi-dialect grammar instead of using a grammar for each dialect.

The second stage of our approach uses MigSim, a tool supported-method that simulates cloud database migration from KDM and SMM models of the target database. We evaluated MigSim by migrating two enterprise systems between Amazon Web Services and Microsoft Azure. The migration cost and duration were compared against the predictions from MigSim. We also ran a database capacity experiment on Amazon's Relational Database Service to evaluate the accuracy of the auto-scaling functionality in MigSim. The estimated migration cost and duration had a relative error between 4 and 22%, while the estimated cloud running costs had a relative error of 12%. We believe these estimates would be valuable to organisations when planning a database migration from a physical server to the cloud, or between cloud platforms.

Our plans for future work include: (1) the use of additional parameters when simulating database capacity, (2) extending the approach to support relational-to-NoSQL migrations, and (3) automating the extraction of cloud cost models. Currently, MigSim horizontally scales database capacity based on the storage/IO performance. Storage performance is often the limiting factor of database performance in OLTP or enterprise systems. However, considering CPU and RAM utilisation would increase the accuracy of the simulations. Databases with analytical or business intelligence workloads would benefit the most from this enhancement, as would databases which use complex stored procedures that make greater use of the database server's CPU.

The (partial or full) migration of relational databases to NoSQL data stores is desirable for some systems [68] and supported by the Amazon Database Migration Tool [16]. However, the process is more complex than relational-to-relational database migrations. The two stages of our approach would still apply to this scenario, although we would need to consider the compute time required to perform the conversion. The existing structure and workload models already provide information about a system's size and characteristics. Additional experiments could be performed to determine the relationship between conversion time and the characteristics of the database.

A large number of public and private cloud providers exist. Currency fluctuations, the release of new services, and new hardware upgrades affect how they charge. Automating the extraction of cloud cost models would reduce error and time, and prevent lock-in [69, 70], so another area of future work is the creation of a model extraction approach and tool for our cloud cost models.

In addition to these three features, we would like to expand our evaluation to include several large-scale systems with database clusters. This would confirm our findings and provide interesting data on the impact of load on migration times. Unlike Science Warehouse and OFBiz, such systems would usually require minimal or zero downtime during migration.

Acknowledgements
This work was primarily funded by the UK EPSRC grant EP/F501374/1. Science Warehouse Ltd provided additional funding for the migration of their database during our evaluation (AWS resources only). They also provided feedback on the industrial application of the approach.

Availability of data and materials
The source code and executables for DBLModeller and MigSim are available from GitHub, [12, 13]. The SQL schemas and models used in our evaluation are available at the same location. We are unable to release the Science Warehouse SQL schema as this is proprietary. Similarly, the Science Warehouse KDM structure model has not been released as it could be used to recreate the SQL schema.

Authors' contributions
ME proposed the approach, implemented DBLModeller and MigSim, analysed related work, and performed the evaluation. He also took the lead on writing the article. RFP and RC participated in the problem definition and helped with technical revisions of the paper. RC participated in drafting the manuscript, the experimental design, and identification of threats to validity. Furthermore, RFP assisted with the analysis of the related work. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Received: 18 September 2017 Accepted: 1 March 2018
Published online: 15 March 2018

References

1. Marston S, Li Z, Bandyopadhyay S, Zhang J, Ghalsasi A (2011) Cloud computing — The business perspective. Decision Support Systems 51(1):176–189
2. Hajjat M, Sun X, Sung Y-WE, Maltz D, Rao S, Sripanidkulchai K, Tawarmalani M (2010) Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In: ACM SIGCOMM Computer Communication Review, vol. 40. ACM, New York. pp 243–254
3. Alharthi A, Alassafi MO, Walters RJ, Wills GB (2017) An exploratory study for investigating the critical success factors for cloud migration in the Saudi Arabian higher education context. Telematics Inform 34(2):664–678
4. Mohagheghi P, Sæther T (2011) Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the REMICS project. In: IEEE World Congress on Services (SERVICES). IEEE, New York. pp 507–514
5. Menychtas A, Konstanteli K, Alonso J, Orue-Echevarria L, Gorroñogoitia J, Kousiouris G, Santzaridou C, Bruneliere H, Pellens B, Stuer P, Strauß O, Senkova T, Varvarigou TA (2014) Software modernization and cloudification using the ARTIST migration methodology and framework. Scalable Comput Pract Experience 15(2):131–152
6. Fittkau F, Frey S, Hasselbring W (2012) Cdosim: Simulating cloud deployment options for software migration support. In: 2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA). IEEE, New York. pp 37–46
7. Object Management Group (2016) Structured Metrics Metamodel, (SMM 1.1). Object Management Group, Needham
8. Pérez-Castillo R, de Guzmán IR, Piattini M (2011) Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Comput Standards Interfaces 33(6):519–532
9. Science Warehouse. http://www.sciencewarehouse.com. Accessed 1 July 2016
10. Izquierdo JLC, Molina JG (2010) An architecture-driven modernization tool for calculating metrics. IEEE Softw 27(4):37–43
11. Ellison M, Calinescu R, Paige RF (2016) Towards Platform Independent Database Modelling in Enterprise Systems. In: 5th International Symposium on Data to Models and Back (DataMod), Federation of International Conferences on Software Technologies: Applications and Foundations. Springer, Cham. pp 42–50
12. Ellison M DBLModeller Project Website. http://mhe504.github.io/DBLModeller. Accessed 4 July 2016
13. Ellison M MigSim Project Website. http://mhe504.github.io/MigSim. Accessed 4 July 2016
14. Maximilien EM, Ranabahu A, Engehausen R, Anderson LC (2009) Toward cloud-agnostic middlewares. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications. ACM, New York. pp 619–626
15. Leymann F, Fehling C, Mietzner R, Nowak A, Dustdar S (2011) Moving applications to the cloud: an approach based on application model enrichment. Int J Coop Inf Syst 20(03):307–356
16. AWS Database Migration Service. https://aws.amazon.com/snowball/. Accessed 8 Mar 2018
17. Ruiz DS, Morales SF, Molina JG (2015) Inferring Versioned Schemas from NoSQL Databases and Its Applications. In: International Conference on Conceptual Modeling. Springer, Cham. pp 467–480
18. Klettke M, Störl U, Scherzinger S (2015) Datenbanksysteme für Business, Technologie und Web (BTW 2015). In: Schema extraction and structural outlier detection for JSON-based NoSQL data stores. Gesellschaft für Informatike.V, Bonn
19. Kim H-J, Ko J, Jeon Y-H, Lee K-H (2018) Migration from rdbms to column-oriented nosql: Lessons learned and open problems. In: Proceedings of the 7th International Conference on Emerging Databases. Springer, Cham. pp 25–33
20. Weber J. H, Cleve A, Meurice L, Ruiz FJB (2014) Managing technical debt in database schemas of critical software. In: Managing Technical Debt (MTD), 2014 Sixth International Workshop On. IEEE, New York. pp 43–46
21. Binz T, Breitenbücher U, Kopp O, Leymann F (2014) Migration of enterprise applications to the cloud. it-Inf Techno 56(3):106–111
22. MySQL Enterprise Monitor. http://www.mysql.com/products/enterprise/monitor.html. Accessed 4 July 2017
23. MySQL Workbench. http://www.mysql.com/products/workbench. Accessed 4 July 2017
24. SQLECTRON cross platform SQL IDE. https://sqlectron.github.io. Accessed 7 Jan 2016
25. Navicat Premium database administration tool. https://www.navicat.com. Accessed 7 Jan 2016
26. P6Spy Framework. http://p6spy.github.io/p6spy. Accessed 7 Jan 2016
27. Surber D (2009) Oracle JDBC Logging using java.util.logging, Technical report. Oracle Corporation, Redwood Shores
28. Pérez-Castillo R, De Guzman IG-R, Piattini M (2011) Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Comput Stand Interfaces 33(6):519–532
29. Izquierdo JLC, Molina JG (2009) A Domain Specific Language for Extracting Models in Software Modernization. In: European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA). Springer, Berlin. pp 82–97
30. Frey S, Hasselbring W (2010) Model-based migration of legacy software systems into the cloud: The CloudMIG approach. Softwaretechnik-Trends 30(2):155–158
31. Márquez L, Rosado DG, Mouratidis H, Mellado D, Fernández-Medina E (2015) A framework for secure migration processes of legacy systems to the cloud. In: International Conference on Advanced Information Systems Engineering. Springer, Cham. pp 507–517
32. Bruneliere H, Cabot J, Jouault F, Madiot F (2010) MoDisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. ACM, New York. pp 173–174
33. Gillam L, Li B, O'Loughlin J (2014) Benchmarking cloud performance for service level agreement parameters. Int J Cloud Comput 2 3(1):3–23
34. Coulden D, Osman R, Knottenbelt WJ (2013) Performance modelling of database contention using queueing petri nets. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering. ACM, New York. pp 331–334
35. Schad J, Dittrich J, Quiané-Ruiz J-A (2010) Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endowment 3(1–2):460–471
36. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Experience 41(1):23–50
37. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Futur Gener Comput Syst 28(5):755–768
38. Khajeh-Hosseini A, Greenwood D, Sommerville I (2010) Cloud migration: A case study of migrating an enterprise IT system to IaaS. In: 3rd International Conference on Cloud Computing. IEEE, New York. pp 450–457
39. Mishima T, Fujiwara Y (2015) Madeus: database live migration middleware under heavy workloads for cloud environment. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, New York. pp 315–329

40. Dismuke CE, Lindtooth R (2006) Ordinary Least Squares. In: Methods and Designs for Outcomes Research. American Society of Health-System Pharmacists, Bethesda. pp 93–104
41. Sait AS, Jung J (2016) Determining the IOPS needs for oracle database on AWS, technical report. Amazon Web Services, Seattle
42. Feinberg D, Adrian M, Heudecker N, Ronthal AM, Palanca T (2015) Magic quadrant for operational database management systems, technical report. Gartner, Stamford
43. (2005) Database Programmer's Guide to the Oracle Precompilers. Oracle Corporation. http://docs.oracle.com/cd/B28359_01/appdev.111/b31231/appb.htm#CJHIIICD. Accessed 8 Mar 2018
44. (2005) MySQL 5.7 Reference Manual: Keywords and Reserved Words. http://dev.mysql.com/doc/refman/5.7/en/keywords.html. Accessed 4 July 2017
45. Canovas J Gra2Mol: PLSQL2ASTM example project. http://github.com/jlcanovas/gra2mol/tree/master/examples/Grammar2Model.examples.PLSQL2ASTMModel. Accessed 8 Mar 2018
46. Mituzas D Page view statistics for Wikimedia projects. http://dumps.wikimedia.org/other/pagecounts-raw. Accessed 4 July 2017
47. Wikimedia: Wikimedia Dump Index. http://dumps.wikimedia.org/backup-index.html. Accessed 4 July 2017
48. AWS Simple Monthly Calculator. https://calculator.s3.amazonaws.com/index.html. Accessed 23 Dec 2017
49. Microsoft Azure Pricing calculator. https://azure.microsoft.com/en-gb/pricing/calculator/. Accessed 23 Dec 2017
50. Apache OFBiz 16.11.01. https://ofbiz.apache.org. Accessed 4 July 2017
51. Zhang L, Wu C, Li Z, Guo C, Chen M, Lau FC (2013) Moving big data to the cloud: An online cost-minimizing approach. IEEE J Selected Areas Commun 31(12):2710–2721
52. Thakar A, Szalay A (2010) Migrating a (large) science database to the cloud. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, New York. pp 430–434
53. Fogel S (2015) Oracle Database Administrators Guide 11g Release 2. Oracle, Redwood City
54. Amazon Relational Database Service: Storage for Amazon RDS. http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html. Accessed 1 July 2016
55. Shahan R, Nevil T, Lu Y, Gries W, Squillace R, Macy M Standard Storage and unmanaged and managed Azure VM disks. http://github.com/Microsoft/azure-docs/blob/master/articles/storage/storage-standard-storage.md. Accessed 17 May 2017
56. Leitner P, Cito J (2016) Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. ACM Trans Internet Technol (TOIT) 16(3):15
57. Greenwood D, Khajeh-Hosseini A, Smith J, Sommerville I (2010) The Cloud Adoption Toolkit: Addressing the Challenges of Cloud adoption in Enterprise. In: The Scotish Infomatics and Computer Science Alliance (SICSA) PhD Conference. arXiv, Ithaca
58. Khajeh-Hosseini A, Greenwood D, Smith JW, Sommerville I (2012) The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise. Softw Pract Experience 42(4):447–465
59. PlanForCloud. https://www.planforcloud.com. Accessed 8 Mar 2018
60. Strauch S, Andrikopoulos V, Karastoyanova D, Vukojevic-Haupt K (2015) Migrating e-Science Applications to the Cloud: Methodology and Evaluation. Cloud Computing with E-science Applications. CRC Press/Taylor & Francis, Boca Raton
61. Strauch S, Andrikopoulos V, Karastoyanova D, Leymann F, Nachev N, Stäbler A (2014) Migrating enterprise applications to the cloud: methodology and evaluation. Int J Big Data Intell 5 1(3):127–140
62. Pérez-Castillo R, de Guzmán IGR, Caivano D, Piattini M (2012) Database schema elicitation to modernize relational databases. In: 14th International Conference on Enterprise Information Systems (ICEIS'12). Science and Technology Publications, Setúbal. pp 126–132
63. Normantas K, Vasilecas O (2014) Extracting term units and fact units from existing databases using the Knowledge Discovery Metamodel. J Inf Sci 40(4):413–425
64. Object Management Group (2015) Semantics Of Business Vocabulary And Rules (SBVR 1.3). Object Management Group, Needham
65. Object Management Group (2011) Abstract Syntax Tree Metamodel (ASTM 1.0). Object Management Group, Needham
66. Pérez-Castillo R, de Guzmán IGR, Caballero I, Piattini M (2013) Software modernization by recovering web services from legacy databases. J Softw Evol Process 25(5):507–533
67. Gholami MF, Daneshgar F, Low G, Beydoun G (2016) Cloud migration process—a survey, evaluation framework, and open challenges. J Syst Softw 120:31–69
68. Moniruzzaman A, Hossain SA (2013) NoSQL Database: New era of databases for big data analytics-classification, characteristics and comparison. Int J Database Theory Appl 3(4):1–13
69. Silva GC, Rose LM, Calinescu R (2013) A systematic review of cloud lock-in solutions. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol. 2. IEEE, New York. pp 363–368
70. Silva GC, Rose LM, Calinescu R (2013) Towards a model-driven solution to the vendor lock-in problem in cloud computing. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol. 1. IEEE, New York. pp 711–716