Scalable User Data Management in Multi-Tenant Cloud Environments

Pieter-Jan Maenhaut*†, Hendrik Moens†, Veerle Ongenae* and Filip De Turck†
*Ghent University, Faculty of Engineering and Architecture, Dept. of Industrial Technology and Construction
Valentin Vaerwyckweg 1, 9000 Ghent, Belgium
†iMinds – INTEC, Ghent University, Dept. of Information Technology
Gaston Crommenlaan 8 bus 201, 9050 Ghent, Belgium
Email: pieterjan.maenhaut@intec.ugent.be

***Abstract*—The rise of cloud computing and its elastic, on-demand resource provisioning introduces the need for a flexible and scalable multi-tenant architecture. In a multi-tenant application every tenant (client) makes use of shared application instances, but each tenant typically has its own user data. The shared application instance behaves like a private instance by guaranteeing both data separation and performance separation for every tenant. As the number of tenants increases, the amount of data grows. A scalable solution for the storage is needed, allowing tenant data to be divided over multiple database instances, but taking into account performance isolation and custom data assurance policies.**

In this paper we introduce an abstraction layer for achieving high scalability for the storage of tenant data. This layer uses data allocation algorithms to determine an acceptable allocation of tenant data to different databases. We describe a mathematical model for the allocation of tenant data which can be optimized using existing linear programming techniques, and introduce the BDAA-n and FDAA, two algorithms that will find an optimal allocation of data by iterating over the possible permutations. The proposed solutions are evaluated based on their flexibility, complexity and efficiency. The flexibility of the BDAA and FDAA makes them easy to customize and extend to fit most scenarios, but the algorithms will achieve best results for tenants with a limited number of subtenants. Linear programming is an alternative for tenants with a higher number of subtenants, but the customizability of the algorithm for specific use cases is limited due to the need for linear functions.

I. Introduction

In a multi-tenant architecture, a software application is designed to virtually partition its data and configuration, and each tenant works in a virtual application instance. Cloud computing [1] is a technology that enables elastic, on-demand resource provisioning. As the infrastructure provider usually charges for the number of instances used, an optimal usage of available resources is desired to reduce operating costs. Adding multi-tenancy to the application reduces the operating cost, but as the number of tenants grows, a scalable architecture for both the application and data is needed, with a minimal impact on the performance by other tenants.

Custom policies can put additional constraints to the allocation of tenant data in multi-tenant cloud applications. Such policies can be used to meet legal and business data archival requirements for both persistent data and records management. Policies about data privacy, migration and retention need to be supported by the data allocation algorithms, as these have a direct impact on the possible reallocation of data.

In this paper, we focus on the scalability of the tenant data in multi-tenant applications. This scalability can be achieved using existing techniques such as partitioning and NoSQL databases, but we will present a solution for allocating data to separated database instances, taking into account performance isolation and custom data assurance policies for each tenant. In this paper we present a generic solution for scalable storage independent of the underlying database system. By doing so, applications that need a traditional SQL interface are also supported.

The remainder of this paper is structured as follows. In the next section we will discuss related work. Afterwards, in Section III we will present our architecture. In Section IV we will introduce data allocation algorithms and discuss possible extensions. In Section V, we will evaluate our algorithms and in Section VI we state our conclusions and discuss avenues for future research.

II. Related Work

The work presented in this paper is related to both scalable storage solutions and custom data policies. In previous work [2], we presented a hierarchical method for organizing tenants and storage of tenant data, and characterized the impact on the performance in a theoretical way. Tenants and subtenants can be logically structured using the *tenant tree*, and a mapping can be made to the physical storage of the tenant data. The theoretical analysis was verified by experiments on different environments, using common relational SQL databases. The outcome of this research is used in this paper to build a scalable database layer. In other previous work [3], [4] we have focused on customizability and performance isolation of multi-tenant computational components of cloud services. This paper extends the previous work by focusing on the data storage components of applications rather than on computational components.

Data assurance policies can be used to meet legal and business data archival requirements for both persistent data and records management. Compliance with regulatory policies on data remains a key hurdle to cloud computing [5]. Jun Li et al. [6] [7] propose a policy management service that

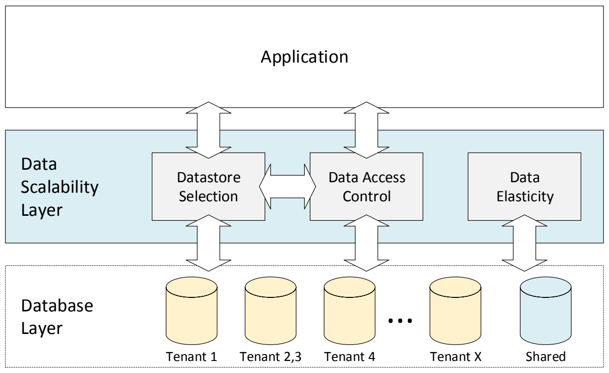

Fig. 1. Architecture of the data scalability layer and possible communication with other layers.

offers scalable management of data assurance policies attached to data objects stored in a cloud environment. With GEO-DAC [8], the authors provide a policy framework that enables the expression of both the service providers' capabilities and customers' requirements, and enforcement of the agreed-upon policies in service providers' environments. The data assurance policies described in their work add additional constraints on the allocation of data and therefore the data allocation algorithms presented in this paper need to be flexible enough to support such policies.

III. Architecture Overview

Most web applications today use databases for storing persistent user data that can be logically grouped together as a database layer in the application architecture. To achieve better scalability of the database layer, we define the data scalability layer between the data access layer and the databases, that is responsible for load balancing and selecting the correct database. By using this abstraction layer, the physical data distribution of the underlying databases is hidden towards the data access layer.

Figure 1 illustrates the architecture of the data scalability layer. As most software architectures will have a clear distinction between the persistent storage and the application, it is possible to add this extra layer in between. Three components can be defined inside the data scalability layer:

- The **Datastore Selection** component is responsible for selecting the correct database and locating the data in one of the possible databases.
- The **Data Access Control** component will verify if the current tenant user has the required permissions to read and/or modify the selected data.
- The **Data Elasticity** component is responsible for achieving high scalability, by increasing and/or decreasing the number of database instances and reallocation of tenant data.

A. Data Elasticity Component

The data elasticity component is responsible for achieving high scalability by allocating the required amount of resources, in this case the number of database instances. This component will evaluate the current load on the different instances, add or remove additional instances and reallocate tenant data, taking into account the different constraints such as data retention policies. The data elasticity component makes use of data allocation algorithms to evaluate the current load and to decide if additional instances should be added or removed.

B. Database Layer

In the presented architecture, multiple database instances exist and tenants (and subtenants) can either have a dedicated instance or share an instance with other (sub)tenants. One database instance is shared between all tenants. This instance holds all general tenant information such as tenant specific configuration parameters, feature configuration (if the multi-tenant variability approach from [3] and [4] is used), billing information and data assurance policies. For every tenant, a representation of the current allocation of data is also stored. The shared database instance shouldn't become the bottleneck of the application as the amount information in it is limited, because only general information about the tenants is stored.

IV. Data Allocation Algorithms

The goal of the data allocation algorithms is to determine an allocation of tenant data to the different database instances resulting in a minimal cost. This is one of the tasks of the data elasticity component of the data scalability layer. Because of this, a cost function is needed to evaluate a possible allocation given certain metrics. The first metric is the number of database instances as more instances will result in a higher cost. The next metric is the average response time for each tenant and for the whole system, which can be calculated using the Equations introduced in [2]. Another useful metric is the current load on the existing database instances.

All algorithms are designed to work for a single tenant and its child nodes (two levels of the tenant tree described in [2]), but in some scenarios the tenant tree might have more than two levels. In this case, the algorithms can be executed multiple times, starting at the leaf nodes of the tree (the lowest level) and continuing towards the root. This concept is illustrated in Figure 2, where the execution order is denoted by numbers. In this example, the selected algorithm will be executed four times.

A. Linear Programming Data Allocation Algorithm (LPDAA)

The data allocation algorithms determine the amount of data for every node that needs to be moved to the parent node, resulting in a minimal cost using the cost function described earlier in this paper. This cost function together with the possible ranges for the variables and additional bounds describe a mathematical model which can be solved using Linear Programming (LP) or Integer Linear Programming

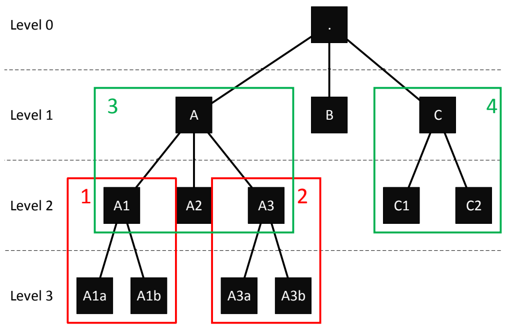

Fig. 2. Using the data allocation algorithms with a tenant tree consisting of more than two levels. In this example, the selected algorithm will be executed four times. The order of algorithm invocations is shown using numbers.

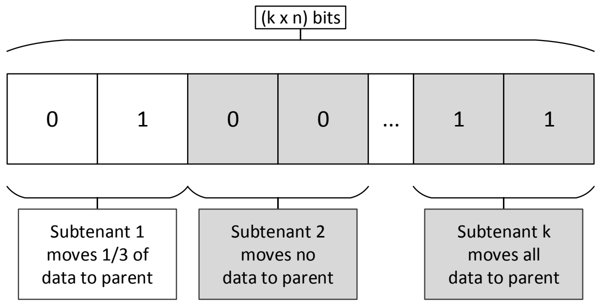
(k x n) bits
0
1
0
0
...
1
1
Subtenant 1 moves 1/3 of data to parent
Subtenant 2 moves no data to parent
Subtenant k moves all data to parent

Fig. 3. An example representation of a single permutation when using the BDAA algorithm for $n = 2$ with k subtenants.

(ILP). We will refer to these algorithms as the Linear Programming Data Allocation Algorithm (LPDAA) and the Integer Linear Programming Data Allocation Algorithm (ILPDAA) respectively.

B. Basic Data Allocation Algorithm (BDAA)

The Basic Data Allocation Algorithm (BDAA) algorithm will find the best allocation of tenant data for a single tenant with k subtenants by evaluating all possible allocations within a defined search space. Each tenant with its subtenants will have at least one database instance and there is a clear separation of data belonging to different tenants. For a single permutation, we will represent every subtenant by n bits. Larger values for n will result in a larger search space. As a result, a permutation will be represented using $n \times k$ bits with a value between 0 and $2^{k \times n}$. The value of the n bits denote the percentage of data stored at the parent database. For example, if we would use 2 bits for every subtenant, $n = 2$ and we will refer to the algorithm as the BDAA-2. Figure 3 illustrates the representation of a single example permutation using the BDAA-2.

Custom policies such as data assurance policies might put constrains on the possible allocation of tenant data. The BDAA algorithm can be easily extended to support such policies by marking permutations that are violating some of the defined constraints as invalid. In the last step of the BDAA algorithm, the valid permutation with minimal cost is returned.

C. Fast Data Allocation Algorithm (FDAA)

The complexity of the BDAA algorithm is mainly dependent on the number of different permutations that have to be evaluated. Although searching for the optimal allocation can happen offline, large values of either k (the number of subtenants) or n (the number of bits used to represent a single subtenant) can rapidly make the algorithm unusable as it would take too much time to evaluate all permutations. To reduce the number of permutations, the Fast Data Allocation Algorithm (FDAA) adds a pre-processing step in which some of the subtenants are removed from the input. After this, the FDAA solves the data allocation problem using the BDAA. The algorithm also contains a post-processing step in which the removed subtenants are re-added to the output.

V. Evaluation Results

A. Execution time of LPDAA and ILPDAA

The mathematical model described above was implemented using the Java API of the IBM ILOG CPLEX Optimization Studio [9] V12.4 for Linux x86-64. The results of the experiments show that LPDAA tends to be faster as the ILPDAA.

B. Execution time of BDAA-n and FDAA

The BDAA-n and FDAA were implemented in Java and executed on a Linux server with an Intel Core i7 CPU (2.80 GHz) with 8 GiB of memory. In order to calculate the average execution times every subtenant was assigned a random distinct number of database records and this was repeated 50 times for every value of k (the number of subtenants). Figure 4 illustrates the measured average execution times for a single tenant with an increasing number of subtenants. As the number of subtenants increases, the probability increases that one or more subtenants can be removed in the pre-processsing step of the FDAA. As a result, the FDAA tends to have a lower complexity than the BDAA-1.

C. Algorithm Comparison

To compare the different algorithms, all algorithms were configured with the same constraints and bounds, and the same cost function. In our simulations, the tenant and subtenants were assigned an initial amount of data and each iteration the amount of data for each tenant was slightly increased. Figure 5 illustrate the calculated cost of an experiment for the different data allocation algorithms. Similar experiments were executed with different iterations based on realistic scenarios, but these experiments provided similar results.

As there are only small differences in the results, all algorithms could be good candidates for the implementation of the data elasticity component. However, there is a significant difference between the flexibility of the different algorithms. As custom policies add additional constraints on the allocation of data, the data allocation algorithm should be flexible enough

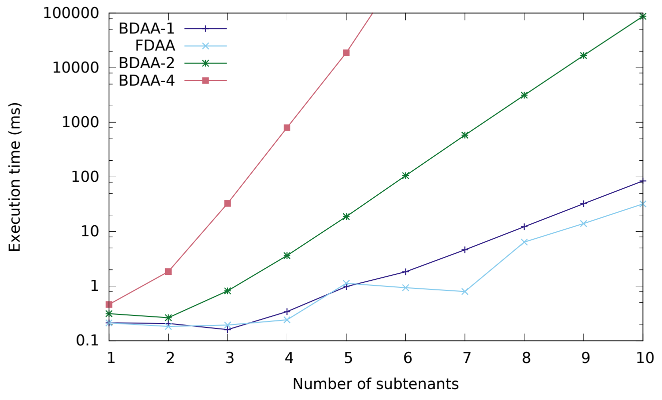

Fig. 4. A comparison of the average execution times of 50 experiments for the BDAA and the FDAA with an increasing number of subtenants.

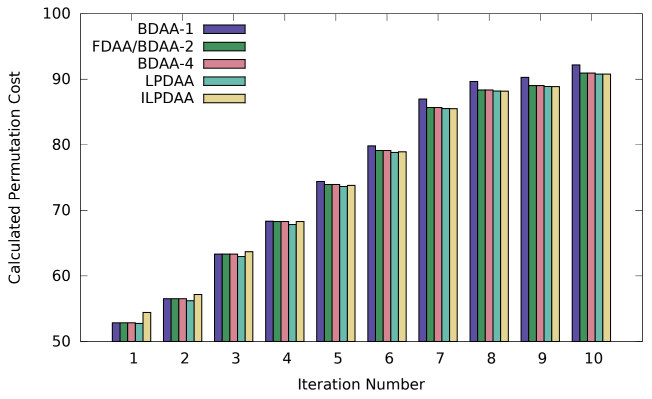

Fig. 5. The calculated cost of the optimal allocation for the different data allocation algorithms over the different iterations. In this experiment, the algorithms were configured to return the valid data allocation resulting in a minimum average response time.

to support them. The BDAA and FDAA are very flexible, as they are easy to extend and customize. Any cost function can be used allowing the algorithm to support most scenarios. The LPDAA and ILPDAA on the other hand are less flexible, as they are depending on linear problems. This puts some limitations to the cost function and constraints. The LPDAA for example can not count the number of database instances, which is needed for a useful cost function. The ILPDAA also can not support all possible scenarios. Some extensions will need additional decision variables, but this increases the complexity of the model significantly.

VI. Conclusions

Scalable multi-tenant applications which will be hosted on the cloud need a scalable architecture for both the application and data. In this paper, we focused on the scalability of the tenant data in such applications. We started with extending the architecture of a typical application by adding the data scalability layer and defined the most important components inside this layer. The data elasticity component is the main component responsible for achieving high availability of the database layer by allocating tenant data to multiple database instances. This component invokes data allocation algorithms to find the optimal allocation of tenant data.

The goal of the data allocation algorithms is to find a possible allocation with minimal cost. The cost function can take different metrics into account, such as the average response time of the system and the number of database instances. The BDAA tends to find an optimal allocation by enumerating the possible permutations. The FDAA reduces the execution time of the BDAA with an exponential factor by introducing a pre-processing step. For tenants with a high numbers of subtenants, linear programming techniques offer an alternative for the FDAA algorithm by using a mathematical model. The ILPDAA is preferred as it offers more possibilities, but the customizability of the algorithm for specific use cases is limited due to the need for linear functions. In scenarios where performance isolation and support for custom data assurance policies are essential, the FDAA should be used instead.

In future work, the model described in this paper will be used to construct a flexible middleware for building multi-tenant applications on the cloud. The goal of this middleware layer is to allow developers to build multi-tenant applications in a traditional way, whereas the middleware layer provides all multi-tenant related functions such as scalability, performance isolation and security in a transparent way.

References

[1] M. Armbrust, R. Fox, Armandoand Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds : A Berkeley view of cloud computing,” University of California at Berkley, Tech. Rep., 2009.

[2] P.-J. Maenhaut, H. Moens, M. Decat, J. Bogaert, B. Lagaisse, W. Joosen, V. Ongenae, and F. D. Turck, “Characterizing the performance of tenant data management in multi-tenant cloud authorization systems,” in *Proceedings of the 14th Network Operations and Management Symposium (NOMS2014)*, Krakow, Poland, may 2014.

[3] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, “Feature placement algorithms for high-variability applications in cloud environments,” in *Proceedings of the 13th Network Operations and Management Symposium (NOMS 2012)*, 2012, pp. 17–24.

[4] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, “Cost-effective feature placement of customizable multi-tenant applications in the cloud,” *Journal of Network and Systems Managemement*, Feb. 2013.

[5] M. Henze, M. Grossfengels, M. Koprowski, and K. Wehrle, “Towards data handling requirements-aware cloud computing,” in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2, Dec 2013, pp. 266–269.

[6] J. Li, S. Singhal, R. Swaminathan, and A. Karp, “Managing data retention policies at scale,” in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, 2011, pp. 57–64.

[7] ——, “Managing data retention policies at scale,” *Network and Service Management, IEEE Transactions on*, vol. 9, no. 4, pp. 393–406, 2012.

[8] J. Li, B. Stephenson, H. Motahari-Nezhad, and S. Singhal, “GEODAC: A data assurance policy specification and enforcement framework for outsourced services,” *Services Computing, IEEE Transactions on*, vol. 4, no. 4, pp. 340–354, 2011.

[9] IBM ILOG CPLEX Optimization Studio. [Online]. Available: http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud