Energy-Efficient Resource Allocation and Provisioning for In-Memory Database Clusters

Karsten Molka
Department of Computing
Imperial College London & SAP Belfast, U.K.
k.molka13@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London, U.K.
g.casale@imperial.ac.uk

***Abstract*—Systems for processing large scale analytical workloads are increasingly moving from on-premise setups to on-demand configurations deployed on scalable cloud infrastructures. To reduce the cost of such infrastructures, existing research focuses on developing novel methods for workload and server consolidation. In this paper, we combine analytical modeling and non-linear optimization to help cloud providers increase the energy-efficiency of in-memory database clusters in cloud environments. We model this scenario as a multi-dimensional bin-packing problem and propose a new approach based on a hybrid genetic algorithm that efficiently handles resource allocation and server assignment for a given set of in-memory databases. Our trace-driven evaluation is based on measurements from an SAP HANA in-memory system and indicates improvements between 6% and 32% over the popular best-fit decreasing heuristic.**

***Index Terms*—Nonlinear optimization; genetic algorithm; in-memory database; SAP HANA.**

I. Introduction

Traditionally, in-memory databases have been offered as part of large on-premise systems that can handle highly processing intensive workloads. In the recent past, however, in-memory databases have become part of on-demand offers from small Amazon Web Service instances, such as HANA One, to high-end large scale systems in private, managed cloud environments, such as HANA Enterprise Cloud[1]. On-demand systems, particularly in the Software-as-a-Service (SaaS) and Database-as-a-Service (DaaS) model, relieve customers from the burden of configuring, managing and operating database servers. Conversely, cloud providers can leverage the economy of scale and reduce data center cost by consolidating database workloads and even entire database systems [1]. Improving data center cost has been focus of many works, with reduction of energy cost, resource utilization and load-balancing being just some in a wide range of optimization goals [2].

In this paper, we specifically consider the problem of optimizing the energy-efficiency of an in-memory database cluster by means of consolidation. In general, this can be handled by either static [3], [4], or dynamic resource allocation [5]–[9]. While static resource allocation aims for providing resources over a set period of time, its dynamic variant adapts resource allocations depending on fluctuations of workload demands in either a reactive or proactive manner. In our case, the static approach is highly relevant, as we consider private clouds with large scale in-memory database deployments, for which dynamic re-configuration would be very time consuming. From a modeling perspective, it is common to express consolidation scenarios as bin-packing problem using integer-linear programming (ILP) or mixed-integer non-linear programming (MINLP) formulations. Both are known to be NP-hard, and in practice they often limit the applicability of standard ILP and MINLP solvers due to long optimization times or the difficulty of expressing certain ILP/MINLP problems in canonical form ready to use for such solvers. Many recent works therefore consider greedy heuristics, such as Johnson's first-fit (FFD) and best-fit decreasing (BFD) [10], or modifications of these [1], [11]. A reason for this can be ascribed to their efficiency in finding good solutions, which reported in literature are often as close as 20% to global solutions of the corresponding optimization problems [1], [12]. With respect to existing works that focus on cost reduction, such as [1], [5]–[9], [11], [13], [14], we consider large datasets and explicitly model tenant workload interference and the multi-core aspect of in-memory database servers, the latter two of which are often ignored despite their presence in open research challenges [15].

Our resource allocation approach assumes that users provide performance constraints in form of response time SLOs (service level objective). This approach is generally more attractive than making specific resource reservations, as from a user perspective workload dynamics and physical resource requirements of in-memory databases are difficult to understand [16]. Conversely, cloud providers have the burden of optimizing infrastructure in a cost-effective way while still being able to meet stipulated SLOs. To solve this problem and maximize revenue, we want to aid cloud providers by guiding the following two resource management decisions:

- the type of servers that should be provisioned and
- the way server resources need to be split amongst co-located database instances.

To do so, we propose an energy-aware framework that provisions server resources for a set of in-memory databases and consolidates workloads in an interference-aware manner. Our framework is lightweight, easily adaptable and achieves substantial energy savings in cloud data centers. Summarizing, our main contributions in this work are as follows:

- a novel methodology based on a non-linear optimization

[1]https://hana.sap.com/implementation/deployment/cloud.html

program to minimize energy consumption for in-memory database clusters
- a hybrid heuristic that combines the best-fit decreasing heuristic with a genetic algorithm (GA) to handle scenarios that render standard BFD unusable
- the use of novel analytical performance models for efficient search of global optima
- a trace-driven experimentation using real measurements from a commercial in-memory database, SAP HANA, that reveals significant improvements in energy cost over the popular BFD heuristic

The remainder of this paper is structured as follows. Section II surveys related work. In Section III we introduce our system model and formulate the problem statement. The proposed optimization approach is discussed in Section IV and evaluated in Section V and VI. Lastly, the paper is concluded and possible extensions are outlined in Section VII.

II. Related Work

Cost reduction in data centers has been subject to research over the past decades, but to our knowledge none of the existing works have considered consolidation approaches to optimize the energy-efficiency of heterogeneous in-memory database clusters. Typically, consolidation scenarios are modeled as bin-packing problem and over the years, many variations of this problem have been in the focus of research [17], [18]. While many recent approaches consider VM consolidation, e.g. [4], [11], [13], others focus on the consolidation of database workloads [1], [14], [19], [20]. Due to the complexity of such problems, the presence of local minima and the large number of variables and resource constraints, most works prefer to use greedy heuristics, such as first-fit and best-fit decreasing [10]. These, however, do not guarantee global solutions. To over come this problem, the authors in [4] apply truncated singular value decomposition, which reduces large constraint matrices to a manageable size that can be handled by standard ILP solvers. But while this approach is promising, it cannot handle the type of non-linear constraints required to model in-memory database performance. The static approach in [14] considers linear models for combining tenant resource demands on MySQL databases, while the authors in [1], [19] facilitate more expressive non-linear models to predict database performance. Both approaches use greedy algorithms to solve their optimization problem, as do the authors in [21] for adjusting the number of database servers required to handle particular workload intensities. In contrast, we provide a global search heuristic that combines best-fit decreasing and algorithms from the field of evolutionary computation to improve solutions over BFD. Other works present tenant placement strategies for multi-tenant databases, and consider techniques such as replica swapping to minimize tenant interference [20], [22]. However, they rely on simple additive models to approximate resource utilizations and cover transactional workloads only. The recent survey in [15] has revealed that explicit modeling of interference effects between VMs / tenants on modern multi-core hardware is largely neglected and still part of open research questions. We consider these aspects with our approach and in contrast to the above we evaluate using large scale datasets.

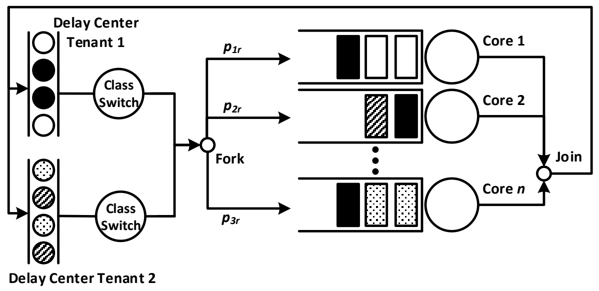


Fig. 1. Queueing network model of a multi-core database server

Works on optimizing energy efficiency, e.g. [11], [23], typically move nodes to an offline state to conserve energy under low system loads. [11] for example employs a Wiener predictor to anticipate future demand patterns and combines it with a modified BFD heuristic to guide the choice of required servers. Other works on resource allocation apply collaborative filtering and profiling-based methods to create models that estimate the impact of resource assignment on application performance [16], [24], and compliance with stringent response time SLOs under transactional and analytical multi-tenant workloads [25], [26]. None of these approaches, however, considers hardware configurations relevant for processing large in-memory workloads.

III. System Model and Problem Formulation

A. System Model

We begin by introducing the system model relevant for our consolidation problem. This model is composed of a server cluster and multiple in-memory database tenants. Our goal is to determine resource allocation and assignment policies that minimize the energy consumption for such a server cluster. With resource allocation we refer to determining the relative share of CPU time allocated to a tenant. Resource assignment relates to choosing a specific server with sufficient resources to host a particular set of tenants. By tenant we mean a customer, for which multiple clients issue a set of business analytical queries to a customer-exclusive in-memory database.

We model a set of tenants that is hosted on a multi-core in-memory database server as multi-class closed queueing network (QN). The QN consists of a delay node, representing the client think time, a fork-join subsystem to handle request parallelism, a queueing station and R request classes. We further employ a class switch construct to model the change of class for requests that are successively issued by a client. We require such a construct to capture the execution model of the TPC-H workload used for our experimentation, where each client has only one job in the system at any given time, but changes class after every job execution. Figure 1 provides a graphical representation the respective QN model for two tenants, each with two different request classes (circles) and their subtasks (rectangles).

W_t	Predicted response time for tenant t
M_k	Predicted memory occupancy for server k
U_k	Utilization of server k
P_k	Power consumption of server k

(a) Performance Measures

T	Number of tenants
$T^{\max}$	Maximum number of tenants per server
K	Number of servers
$W_t^{\max}$	Maximum mean response time for tenant t
$M_k^{\max}$	Maximum available memory on server k
C_k	Operational cost per server k in \$

(b) Additional Parameters

Fig. 2. Relevant notation

B. Decision Variables and Problem Statement

Solving our consolidation problem requires the allocation of servers, performing server assignments and splitting CPU resources between co-located tenants. These three steps are handled by the following decision variables:

- $a = [a_t], 1 \leq a_t \leq K, \ \forall t, 1 \leq t \leq T$. Resource assignment vector: assigns a server index to tenant t, when K servers are available.
- $s = [s_t], s^{\text{lb}} \leq s_t \leq s^{\text{ub}}, \ \forall t, 1 \leq t \leq T$. Resource share: sets a preferred CPU share for tenant t and is bounded by the lower and upper thresholds $s^{\text{lb}}, s^{\text{ub}}$. Note that partitioning memory may not be a good idea for in-memory DBs due to large peak memory usage, so a hard partitioning would severely under-provision for peaks.
- $y = [y_k], y_k \in \{0, 1\}, 1 \leq k \leq K$. Switching server k on (1) or off (0).

Our goal is to decide a triplet (a, s, y) that minimizes the operational cost of an in-memory database cluster. By operational cost we denote the hourly cost of power consumption generated by the cluster. The optimization problem can be classified as multi-dimensional bin-packing problem, which we express with a mixed-integer non-linear program:

$$C_{\min}^{\text{total}} = \min_{a,s,y} \sum_{k=1,\ldots,K} C_k y_k \tag{1a}$$

$$\text{s.t.:} \quad W_t(a,s) \leq W_t^{\max}, \ \forall t \tag{1b}$$

$$M_k(a,s) \leq M_k^{\max} y_k, \ \forall k \tag{1c}$$

$$U_k(a,s) \leq U_k^{\max} y_k, \ \forall k \tag{1d}$$

$$T_k \leq T^{\max}, \ \forall k \tag{1e}$$

We determine the cost C_k of an individual server k as the sum of the power consumption P_k for CPU, memory and other system components, multiplied by the electricity cost C^{kWh} per kWh: $C_k = C^{\text{kWh}}(P_k^{\text{CPU}} + P_k^{\text{DRAM}} + P_k^{\text{other}})$. Here, we assume CPU power consumption to be the only factor that significantly changes during workload execution and consider power consumption of main memory and other system components to be constant. Server power consumption can then be modeled as function of CPU utilization plus a constant offset for the remaining system resources. The non-linear functions $W_t(a,s)$, $M_k(a,s)$ and $U_k(a,s)$ are used to calculate mean response times for tenant t, as well as server k memory and CPU utilization. These functions are aware of all decision variables and additional parameters from Table 2, which we here omit to simplify the notation. Constraints (1b-1d) ensure that tenant response times, server memory and CPU utilization do not exceed their respective maximums. And with constraint (1e) we limit the number of tenant databases that can be co-located on a single server.

IV. Optimization Approach and Implementation

Before we discuss our solution approach in detail, we will first present our reference method for evaluating solution candidates and outline the modifications we have made to the BFD heuristics that will serve as baseline for our evaluation.

A. Evaluating Solution Candidates

When evaluating the energy efficiency of a cluster, we need to determine the energy cost of each individual server. Hence, we start by looking at the resource assignment vector a of a solution candidate and find out which server is assigned to a particular tenant. Subsequently we compute all relevant performance measures by solving the corresponding queueing network. For greater efficiency, we use our formulation Task-Placement AMVA (TP-AMVA, [27]) and combine it with a quadratic power model to estimate CPU power consumption [28]. In order to handle an non-uniform split of CPU resources amongst tenants, we consider the approach suggested in [29] and [30]. The authors propose a service differentiation mechanism called biased processor sharing, an extension to egalitarian processor sharing that can model asymmetric resource shares. The authors give an AMVA-based formulation, which applied to our TP-AMVA response time equation, results in adding a weight π_r for request class r to express the fraction of server capacity allocated to that class. This modification has the following form:

$$W_r(\vec{N}) = d_r \left(1 + \sum_{v=1}^{R} \frac{\pi_v}{\pi_r} A_v(\vec{N})\, p_v^{core}\, p_{rv}^{cont}\right), \tag{2}$$

where the response time W_r is determined by multiplying the class r service demand d_r with a specific service rate degradation factor. The latter is dependent on the arrival queue length $A = \sum_{r=1}^{R} A_r$, seen on arrival of a class r job at the system, given a vector of job populations $\vec{N} = [N_r], r = 1, \ldots, R$ and a total of R job classes. Our formulation uses the Bard-Schweitzer approximation [31] to estimate A_r and applies core routing and contention probabilities $(p_v^{core}, p_{rv}^{cont})$ to correct A_r for queries with variable threading levels and to account for query class interference, as described in [27]. The CPU utilization of a server k is determined by $U_k = \sum_r^R X_{kr} d_{kr}$, where X_{kr} is the class-r throughput. Memory is computed as sum over the product of per-class queue lengths Q_r and the respective class-r memory occupancy m_r. For our evaluation we make a conservative assumption and consider the 95th percentile of memory occupancy, see [37]. As mentioned

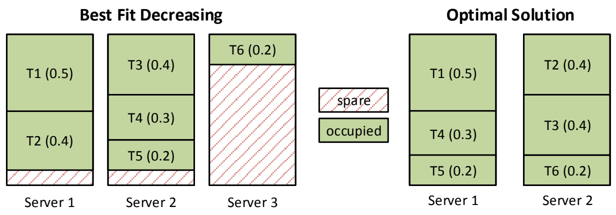


Fig. 3. Required number of servers under BFD and optimal solution, when packing a set of tenants T1,...,T6, each with a workload intensity ≤ 1

above, all four performance measures W, U, X and Q are obtained by solving TP-AMVA. When evaluating these with TP-AMVA during the optimization process, we obtain the weights π_r depending on the number of co-located tenants that reside on the same server as follows: $\pi_r = s_t / \sum_{t \in T} s_t$, and $r \in R(t)$, where $R(t)$ returns all class indexes that belong to tenant t. In our evaluation we will show that using this approach brings a large improvement over the popular BFD bin-packing heuristic, which is oblivious to such weights.

B. Best-Fit Decreasing and First-Fit

1) BFD Overview: Since the BFD heuristic will serve as a baseline for our evaluation, we will briefly outline how it works. Let us consider the example in Figure 3, a scenario with six tenants $T_1, ..., T_6$, each with a given workload intensity ≤ 1 and three servers, each of which can handle a total workload intensity of 1. In order to increase the packing density and to place tenants on as few servers as possible, BFD sorts all tenants in decreasing order of their workload intensity and puts tenants iteratively onto the server with the highest utilization. In case the latter does not provide sufficient resources to handle the additional workload, the server with the next highest utilization will be chosen. If no suitable server can be found, BFD has failed (in case of a fixed set of servers) or a new server needs to be added to the server pool. In Figure 3 we can see how BFD performs on our exemplary scenario, which affirms that BFD does not guarantee global optima.

2) Baseline Heuristics: For our evaluation in Section VI we will use a modification of BFD, similar to [11]. As sorting criteria for the list of servers we will consider the product of server utilization and 1/(number of server CPUs). This choice follows the intuition that in the heterogeneous case servers with a smaller number of sockets are prioritized over larger systems, which thus makes the cluster more energy-efficient. In the homogeneous case servers are sorted solely based on their CPU utilizations, which similar to the heterogeneous case leads to a higher packing density and allows us to switch off idle servers. Since out of scope, we leave the analysis of other metric combinations, which could serve as sorting criteria, for future work. In addition to BFD, we also consider the first-fit heuristic (FF), one of the simplest approaches, oblivious of any order of tenants or servers. FF simply works by picking the first server that matches a tenant's requirements and assigns it to this tenant. We employ TP-AMVA to evaluate if a server matches these requirements and to check whether resource utilization constraints can be satisfied. Note that both heuristics are not aware of resource shares, which requires us to equally split CPU resources between co-located tenants.

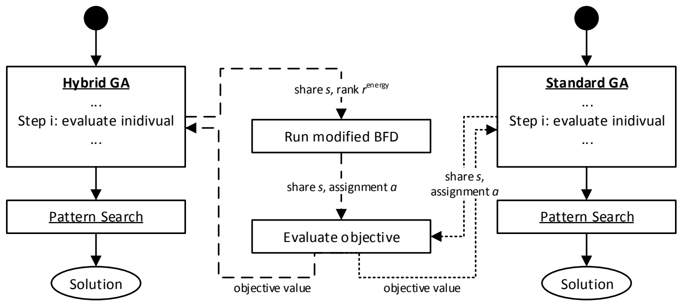


Fig. 4. Comparison of optimization process between hybrid and standard GA

C. Pattern Search and Standard Genetic Algorithm

We are further interested in the performance of pattern search (PS), a standard direct search method whose mechanics can be found in [32]. While this method does not perform a global search, it can be beneficial for refining existing solutions found in an exploratory search phase [33]. Hence in our case, we want to investigate if PS is able to improve solutions found by BFD. To initialize PS we provide a starting point consisting of the resource assignment a, determined by BFD, and the equally split resource shares s. For our evaluation we denote this approach by BFD-PS. In addition to the latter, we also consider a non-conventional optimization method, called genetic algorithm (GA). GAs belong to a group of non-deterministic global search methods and are increasingly used for cloud resource scheduling problems [34]. From a motivational point of view, we chose GA because we found other solvers hard to apply to our MINLP problem, as many of them do not support black-box functions with mixed-integer domains. We plan, however, to investigate other solvers in future work. On the right-hand side of Figure 4 we give an overview on the workflow of our standard GA method, which we will simply denote by GA. Both BFD-PS and GA traverse the search space spanned by the resource assignment vector a and share allocation vector s. To reduce the problem complexity, we exclude y from the decision variables and instead compute y_k from a as follows: $y_k = \max_{t \in 1..T} I(a_t, k), \forall k$, where $I(a_t, k) = 1$ if $a_t = k$ and 0 otherwise. This however does not change the solution space of (1a).

D. Hybrid Genetic Algorithm

As we have pointed out in Section IV-B, method BFD requires preset values for tenant resource shares s. We will therefore combine it with GA, in the sense that GA will search for optimal values of s and feed them into BFD during the evaluation of an individual, whereas BFD determines the assignment of servers to tenants. For later use we will refer to this method with GA-BFD. The latter, however, still has limitations. We thus want to further improve this method by loosening BFD's greedy strategy and assigning servers more intelligently. This second approach is another hybrid variant of GA, with its workflow depicted on the left-hand side of

Algorithm 1 Modified Best-Fit Decreasing

```
1: function $BFDMod(r^{\text{energy}}, s)$
2:     $ts \leftarrow sort([1..T], '\text{descend}')$
3:     $a, P_k^{\text{total}} \leftarrow [],\ \ P^{\text{total}} \leftarrow 0$
4:     for all $t \in ts$ do
5:         for all $k \in 1, ..., K$ do
6:             $ts' \leftarrow getTenantsFromServer(k, a)$
7:             $P_k \leftarrow evalPerformance(k, [t, ts'])$
8:             $P_k^{\text{total}}(k) \leftarrow P^{\text{total}} + P_k$
9:         end for
10:        $P' \leftarrow sort(P_k^{\text{total}}, '\text{ascend}')$
11:        $idx \leftarrow chooseServer(P', r^{\text{energy}}(t))$
12:        $a(t) \leftarrow idx,\ P^{\text{total}} \leftarrow P'(r^{\text{energy}}(t))$
13:    end for
14:    return $a, s$
15: end function
```

evalPerformance	Evaluates the performance of a server, returns the server power consumption P_k and adds a penalty in case of SLO violations.
getTenantsFromServer	Returns the set of tenants assigned to a server.
chooseServer	Returns the server index that corresponds to the $r^{\text{energy}}(t)$th position in P'

Fig. 5. Auxiliary functions to Algorithm 1.

Figure 4. We will call this method GA-BFDMod since it uses a modification of BFD, BFDMod. Recall that standard BFD orders servers based on their utilization and number of sockets as described in Section IV-B. In contrast, BFDMod chooses servers that contribute least or only little to the overall cluster energy consumption P^{total}. However, since BFDMod cannot optimize along the continuous domain either, tenant resource shares are determined by GA. In addition, GA provides a recommendation (rank r) that BFDMod considers when choosing a server, to make standard BFD more flexible. Hence, in each optimization iteration, GA provides the following decision variables to BFDMod and runs BFDMod, which computes the actual placement:

- s - the tenant resource share vector
- $r^{\text{energy}} = [r_t^{\text{energy}}], 1 \leq r_t^{\text{energy}} \leq K$ - the rank of the server to be chosen for tenant t.

The main steps of BFDMod are summarized in Algorithm 1 and Figure 5. During the search phase for an appropriate server BFDMod ranks each server k according to P_k^{total}, the total cluster power consumption under condition that server k is chosen for tenant t. A suitable server is then picked corresponding to the rank suggested by GA. Note that servers with a higher rank correspond to a lower P_k^{total}. Once BFDMod has finished the assignment for all tenants, it returns the complete assignment and resource share vector for a final evaluation of the objective function. In all three cases, GA, GA-BFD and GA-BFDMod, we consider a final optimization run with pattern search to further refine the obtained solutions.

V. Evaluation Methodology

Before we present the results of our evaluation in Section VI, we will provide a brief description of the testbed we used and give an overview on the evaluation scenarios.

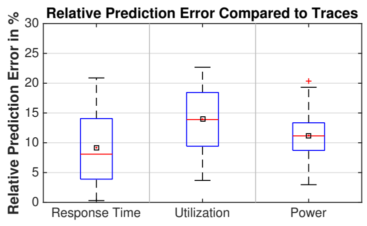
A. Hardware and Software

Our experiments were performed on a 2.27 GHz Intel Xeon X7560 system with 32 physical cores running SUSE Linux Enterprise Server 11 SP2 and MATLAB R2016a. We implemented the heuristics FF, BFD and BFDMod as MATLAB functions and use the pattern search and genetic algorithm solvers provided by MATLAB for our methods BDF-PS, GA, GA-BFD and GA-BFDMod. During our evaluation we will also compare method GA-BFDMod with its equivalent PSO-BDFMod, which uses particle swarm optimization (PSO) instead of the GA. However, MATLAB's PSO solver only works on real numbers and MATLAB's PS solver currently does not accept mixed-integer problems. We therefore applied the commonly used method of mapping the discrete space of assignment vector a and and rank vector r to the continuous domain by rounding, as described in [36]. For both methods, BFD-PS and GA, we handled the non-linear constraints (1b-1d) by adding an exponential penalty to the objective function. The penalty is determined from the sum of exponentials of constraint violations: $\sigma \sum_{i=1}^{L} \exp(\sigma x_i)$, where the penalty parameter σ controls the severity of the penalty, L accounts for the number of violated constraints and x_i holds the value of constraint violation i. We preferred using this method over providing non-linear constraint functions because of two reasons. First, a notable slowdown of the pattern search and genetic algorithm solver occurred when trying to solve the non-linear constraints. Second, the GA solver got stuck in poor local optima more often.

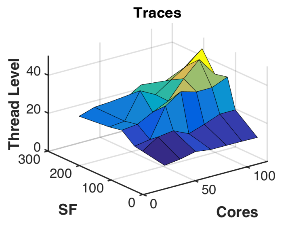
B. Server and Application Model

1) Evaluation Parameters: To better relate the evaluation to real world scenarios, our system model is based on measurements from TPC-H query runs in isolation, executed on an industrial in-memory database system, SAP HANA. The data for this model, the queueing network representation and its relevant parameters are derived from [27], [28] and [37]. The execution of database workloads on a server is modeled by a closed multi-class queueing network. The QN is combined with a fork-join system to model parallel query execution on a multi-core system, and a delay node is used represent the user think time. With database workloads we refer to multiple tenants that have exclusive access to an in-memory database process. Each tenant is associated with one or more concurrent clients also called users, which issue analytical requests. In our case we considered analytical requests of type TPC-H with 22 different request classes, characterized by their execution time, thread level parallelism and memory requirement. Based on these considerations, the three main workload parameters relevant for our optimization model are:

- T: the number of tenants
- N_t: number of concurrent users that belong to tenant t



(a) Prediction errors



(b) Threading levels

Fig. 6. (a) Relative prediction errors of TP-AMVA for response times, CPU utilization and power consumption, (b) Threading level obtained from traces for an exemplary TPC-H query

- SF_t: the database scale factor of tenant t (uncompressed database size in GB under TPC-H).

The two main hardware configuration parameters that we will vary during our evaluation are:

- K: the number of servers
- S_k: the number of CPUs of server k.

To show that our QN model is adequate for handling variations of these parameters during the optimization we evaluated its accuracy under multi-tenant scenarios with regard to response times, utilization and power consumption. In particular, we considered the dataset from [28], which contains a variation of T, N_t, SF_t and S_k over 64 experiments. Figure 6(a) gives the corresponding results that suggest an average prediction accuracy ranging between 9% and 14%. We refer to [37] for details on the prediction accuracy for memory occupancy, which lies in a similar range.

2) Parameter Settings: With respect to the evaluation of our main contribution, the hybrid optimization approach, we have limited the maximum number of tenants that can be co-located on the same server by setting $T^{\max} = 5$. This threshold seemed appropriate, as we observed that three tenants with a high workload, i.e. 30 concurrent users, already fully utilize the CPU resources of an eight socket system. We have further bounded the resource shares s_t by: $s^{\text{lb}} = 0.1$ and $s^{\text{ub}} = 0.9$ to avoid that our solvers choose extreme values for s_t. The number of concurrent users N_t is varied from 6 to 24 for each tenant in each scenario and, similar to [38], we have added a Gaussian noise to N_t with standard variation $\sigma = 2$. Moreover, for each evaluation scenario we choose tenant scale factors SF_t from the range [30,300] and tenant response time constraints $W_t^{\max}$ from $[0.4 \times SF_t, 1.2 \times SF_t]$ using a uniform distribution. Think times are set by $Z_t = 10 \times SF_t/30, \forall t$, to reflect that analyses with larger scale factors, e.g. quarterly or yearly data, are somewhat less frequent. Finally, we have measured the static part of server energy consumption, including P_k^{DRAM} and P_k^{other}, on our 8-socket system and have scaled down accordingly for systems with fewer number of sockets and less main memory. During our analysis we assume energy cost with 0.15 per KWh.

3) Interpolation of QN Parameters: Our QN model requires three specific parameters: the service demand d, thread level parallelism p and memory occupancy m. These parameters are defined for every request class and they depend on the database scale factor and the number of CPU cores. One might expect these to scale linearly with an increase in scale factor and CPU cores. Interestingly, this is not always the case. While service demands and memory occupancy generally showed a linear behavior, we noticed non-linear effects in particular for the threading level. We illustrate this in Figure 6(b) for the threading level of one exemplary TPC-H query from our traces. We think that this behavior partly stems from the SAP HANA query planner that tries build query plans with an optimal set of parallel execution phases based on the dataset size and the number of CPU cores available to the database.

Since we want to consider a mix of different scale factors and server configurations for our evaluation, we ran a set of 15 experiments at five different scale factors and three different server configurations (30, 60 and 120 cores) and use these as a basis to obtain d, p and m for other scale factors in the range [30,300] or CPU cores in [30,120]. To do so, we applied regression to fit a Gaussian process model, a linear model and a quadratic model on the corresponding dataset, all three of which are suitable for an interpolation of d, p and m. However, we found Gaussian process interpolation to be the preferable choice, and will therefore consider it for our evaluation.

VI. Experiments and Results

We evaluate the approaches discussed in Section IV on the problem of improving energy efficiency of an in-memory database cluster. Our evaluation scenarios are characterized by a variable problem size to analyze the scalability of our approach as well as by homogeneous and heterogeneous server setups with varying utilization constraints to investigate the behavior in more challenging situations. For every scenario we report the required number of servers and the improvement of our approaches over the BFD baseline. Due to the non-deterministic nature of our GA-based methods, we considered five runs, each with a differently initialized random number generator, and picked the best solution found.

A. Varying Scenario Scale

In this experiment we consider three different scenario scales, determined by the number of tenants T and the total number of servers K with $T, K = 6$, 12, 24. For these scenarios we equally split the K servers into a set of heterogeneous servers with $sock_k = 2$, 4 and 8 sockets and order them randomly. Resource utilization constraints are set for each server k to $U_k^{\max} = 0.99$ and $M_k^{\max} = 1000 \times sock_k/8$ GB.

Looking at Figures 7, 8 and 9, we can see that increasing the number of users per tenant results in a proportional increase of servers required to host all tenants. More interesting however is that our hybrid approach GA-BFDMod clearly outperforms the other solution methods. The improvements in energy cost achieved by GA-BFDMod over BFD range between 6% and 32% depending on the scenario. Method GA-BFD in contrast seems to find good solutions only under larger scenarios and higher workload intensities (Figure 9). Our guess is that under these scenarios there is more leeway for reducing the number

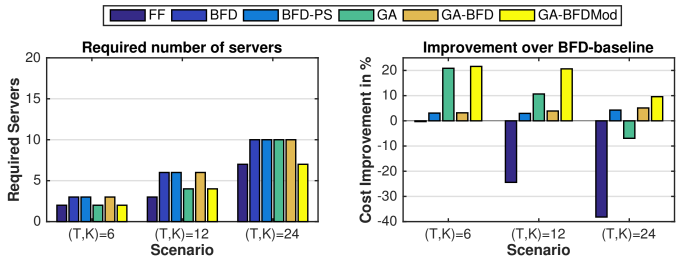


Fig. 7. Experiment results when varying scenario scale (T,K) - light load

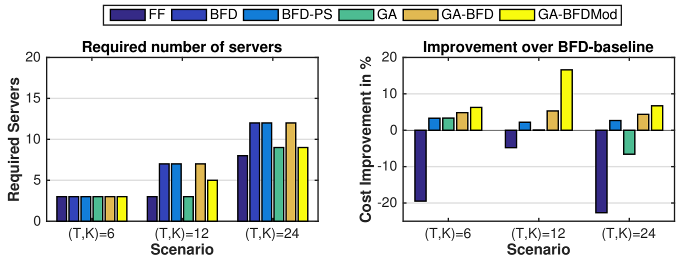


Fig. 8. Experiment results when varying scenario scale (T,K) - medium load

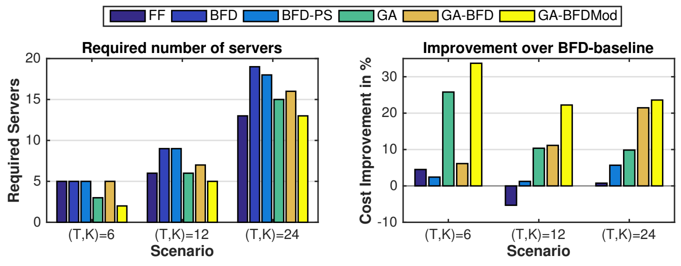


Fig. 9. Experiment results when varying scenario scale (T,K) - high load

of servers, keeping in mind that in GA-BFD the GA part only guides the search for resource shares but is limited by BFD in terms of server assignment. The corresponding number of required servers for GA-BFD over BFD supports this guess. Further, we observe that method GA can be competitive to GA-BFDMod, though this seems to be the case for small scale scenarios and light load only. We attribute this to the higher number of constraints under larger scenarios, which are difficult to satisfy. Our hybrid GAs mostly avoid that problem, since BFD and BFDMod are likely to find a feasible solution in every iteration. A look at method PS-BFD indicates a constant improvement over BFD, which yet remains between 2% and 6%. As expected, the simple first-fit heuristic FF produces the worst results. There are a few cases where the random server order works in favor of FF over BFD by a small percentage, which is negligible in the respective scenario, though. Finally, we noticed that generally BFD requires more servers than FF. The reason behind this becomes clear when looking at the type of servers FF allocates. FF uses more eight socket systems that on one hand can host more tenants, but on the other hand increase the total cluster energy cost. We expect such an effect to diminish under scenarios with homogeneous servers.

We also considered another set of experiments where we varied the noise added to the number of users N_t per tenant and found similar results to those in Figure 7-9.

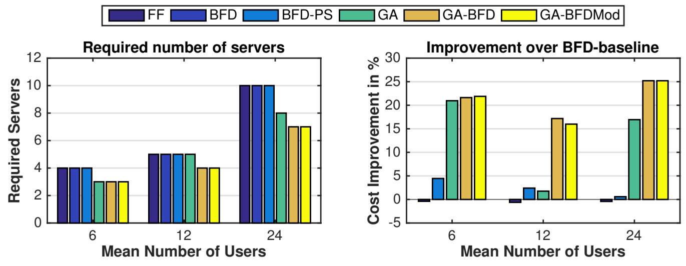


Fig. 10. Experiment results when varying workload intensities under scenario (T,K) = 12 - using homogeneous servers

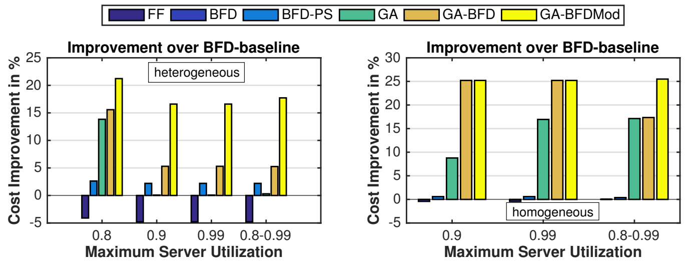


Fig. 11. Experiment results when varying utilization constraints under scenario (T,K) = 12. (left) - medium load and heterogeneous servers, (right) - high load and homogeneous servers

B. Homogeneous Servers

In this set of experiments we analyze the behavior of our approach under scenarios with four socket server configurations and give the corresponding results in Figure 10. We can see that method GA-BFD becomes competitive to GA-BFDMod. Since all servers have the same configuration, the order of them seems to play a minor role here. The results for FF and BFD confirm this, as there is virtually no difference between these two. Nevertheless, there is still a considerable amount of improvement to expect from our hybrid GA-based methods.

C. Varying Utilization Constraints

With our next set of experiments we want to get an idea on how our methods perform when varying certain resource constraints. Hence, we consider different choices of utilization constraints and evaluate these for a scenario with heterogeneous and homogeneous servers, as shown in Figure 11. For the heterogeneous case, we again observe GA-BFDMod performing best, no matter whether the sever utilization constraint is fixed or randomly picked out of the interval [0.8,0.99], as is the case for the last group of columns. Under the homogeneous scenarios we observe a performance for all methods similar to the experiments in Figure 10. The interesting part here is that the performance of GA-BFD deteriorates when using variable CPU utilization constraints. Such constraints exacerbate the search for an adequate server order for BFD. We expect to see similar effects when using different memory constraints for each server. We omit the results for the utilization constraint of 0.8, since none of our methods could find a feasible solution in that case. This means the number of available servers is not sufficient to run the tenant workloads without violating the utilization constraint.

Method	T,K=6	T,K=12	T,K=24
GA-BFDMod[†]	0.15	0.83	5.38
PSO-BFDMod[†]	0.08	0.39	2.23
Improvement	0.49	0.38	1.49

[†] *timeout 5 hrs + 30min for PS*

Fig. 12. Optimization times (hrs) and PSO improvement over GA (%)

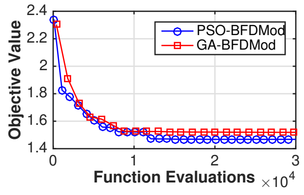


Fig. 13. Objective value over function evaluations, (T,K) = 24

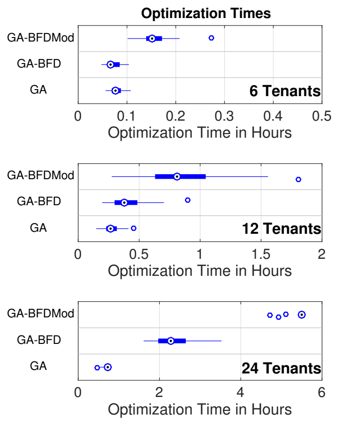


Fig. 14. Optimization times across different scenario sizes

D. Particle Swarm Optimization vs. Genetic Algorithm

Lastly, we are interested in a comparison of GA and particle swarm optimization (PSO), another widely considered EC method [2]. For this comparison, we exchanged the genetic algorithm solver in GA-BFDMod against MATLAB's PSO solver, and call this method PSO-BFDMod. Due to the faster convergence reported in literature, we limited PSO-BFDMod to a maximum of 300 optimization iterations and did not consider a final run with pattern search. The corresponding results are given in Figure 12 and 13. First, we notice that generally PSO-BFDMod finds a slightly better final solution in shorter time, performing on average between 0.38% and 1.49% better than GA-BFDMod, and up to 2.5% in single scenarios. Second, while optimization times for both methods are of little relevance for our static scenario, an early solver stop can still provide a good solution. We can see this for an exemplary high load scenario in Figure 13, where both methods found a solution close to their respective final solution within about 12000 function evaluations, which correspond to one hour in optimization time on our test system. For completeness we also provide the optimization times for our basic GA implementations in Figure VI-D.

E. Summary

In this analysis we have seen that our hybrid approach GA-BFDMod outperforms the other algorithms and strongly reduces cluster energy consumption over the popular best-fit decreasing heuristic. We conclude that both, adaption of the resource shares and guiding the server choice for BFDMod, have a positive effect on the quality of found solutions. While the slightly faster method GA-BFD seems competitive under homogeneous scenarios, its performance deteriorates when resource constraints are not set uniformly, which often is the case in real world scenarios. For scenarios under which our methods perform similarly, we believe either method is suitable. However, as possible improvements are not known a priori, we feel GA-BFDMod is likelier to deliver better results. Further, we believe that our results reflect realistic energy figures, which are implicitly included in operational costs, since our energy models are fitted to real system measurements on SAP HANA, see ref [28]. Lastly, we found that our hybrid method GA-BFDMod can be slightly improved by using particle swarm optimization. Recall, that our optimization methodology focuses on static consolidation of large in-memory databases in private clouds, for which on-line migrations on short time scale are impractical. Optimization times are thus of less importance. However, if required, our method can still be applied to more frequent cluster reconfiguration, given that an early solver stop still results in good solutions. For larger cases with more than 40 servers our method should remain suitable, but it may require hierarchical application to reduce computational complexity.

VII. Conclusions and Future Work

In this paper we have proposed an optimization methodology to solve the problem of energy-efficient in-memory database consolidation. Our solution combines popular greedy heuristics with more advanced EC algorithms to solve the corresponding resource allocation and provisioning problem, while taking into account characteristics of analytical workloads, response time SLOs as well as tenant interference on multi-core hardware. We have provided a trace-driven evaluation using measurements from a commercial in-memory database system SAP HANA, indicating cost improvements of up to 32% over the popular BFD heuristic. Our approach shows its main benefits in private cloud scenarios with a stable customer base and predictable demand patterns and can help respective cloud providers to manage their infrastructure in a cost-effective way, while being able to meet stringent performance SLOs.

Possible extensions for future work include the consolidation of tenants with mixed analytical and transactional workloads, as well as an evaluation of the applicability of our approach with respect to other in-memory systems and general cloud applications. Since the latter can have substantial disk I/O, a further performance model extension may then be required. Another promising direction is the combination of our methods with techniques based on singular value decomposition, which increasingly gain relevance for large-scale provisioning with hundreds of servers running many different cloud services. Other directions include the optimization of dynamic scaling scenarios. In particular, we found existing methods to be weak already in static scenarios. Our models could also give further insights into public cloud use cases and help to determine an optimal trade-off between allocations of small versus large instances, including pricing combinations provided by different public cloud vendors.

Acknowledgment

The work of Giuliano Casale has been supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 644869 (DICE).

References

[1] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs, "RTP: robust tenant placement for elastic in-memory database clusters," in *SIGMOD Conference*, 2013, pp. 773–784.

[2] Z.-h. Zhan, X.-f. Liu, Y.-j. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches," *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–33, 2015.

[3] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 266–278, 2010.

[4] T. Setzer and M. Bichler, "Using matrix approximation for high-dimensional discrete optimization problems: Server consolidation based on cyclic time-series data," *European Journal of Operational Research*, vol. 227, no. 1, pp. 62–75, 2013.

[5] A. Beloglazov and R. Buyya, "Adaptive Threshold-Based Approach for Energy-Efficient Consolidation of Virtual Machines in Cloud Data Centers," *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, no. December 2010, p. 6, 2011.

[6] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5346 LNCS, 2008, pp. 243–264.

[7] T. Setzer and A. Wolke, "Virtual machine re-assignment considering migration overhead," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, Setzer2012, 2012, pp. 631–634.

[8] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2–19, 2012.

[9] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "EnaCloud: An energy-saving application live placement approach for cloud computing environments," in *CLOUD 2009 - 2009 IEEE International Conference on Cloud Computing*, 2009, pp. 17–24.

[10] D. S. Johnson, "Near-Optimal Bin Packing Algorithms," Ph.D. dissertation, M.I.T,, 1973.

[11] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-Efficient Resource Allocation and Provisioning Framework for Cloud Data Centers," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 377–391, sep 2015.

[12] R. E. Korf, "A New Algorithm for Optimal Bin Packing," in *Eighteenth National Conference on Artificial Intelligence*, 2002.

[13] A. Wolke, M. Bichler, and T. Setzer, "Planning vs. dynamic control: Resource allocation in corporate clouds," *IEEE Transactions on Cloud Computing*, vol. 7161, no. 99, pp. 1–14, 2014.

[14] C. Curino, E. P. C. Jones, S. Madden, and H. Balakrishnan, "Workload-Aware Database Monitoring and Consolidation," in *SIGMOD*, 2011, pp. 313–324.

[15] Z. Á. Mann, "Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms," *ACM Computing Surveys*, vol. 48, no. 1, pp. 1–34, 2015.

[16] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, 2014, pp. 127–144.

[17] C. Chekuri and S. Khanna, "On Multidimensional Packing Problems," *SIAM Journal on Computing*, vol. 33, no. 4, pp. 837–851, 2004.

[18] D. S. Hochbaum and D. B. Shmoys, "A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach," *SIAM Journal on Computing*, vol. 17, no. 3, pp. 539–551, 1988.

[19] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier, "Predicting In-Memory Database Performance for Automating Cluster Management Tasks," in *Proceedings of the 27th International Conference on Data Engineering (ICDE '11)*, 2011.

[20] H. J. Moon, H. Hacigümücs, Y. Chi, and W.-P. Hsiung, "SWAT: A Lightweight Load Balancing Method for Multitenant Databases," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13. ACM, 2013, pp. 65–76.

[21] G. Soundararajan and C. Amza, "Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers," *2006 IEEE International Conference on Autonomic Computing*, pp. 231–242, 2006.

[22] Z. Liu, H. Hacigümücs, H. J. Moon, Y. Chi, and W.-P. Hsiung, "PMAX: Tenant Placement in Multitenant Databases for Profit Maximization," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13. ACM, 2013, pp. 442–453.

[23] W. Lang, J. M. Patel, and J. F. Naughton, "On energy management, load balancing and replication," *ACM SIGMOD Record*, vol. 38, no. 4, p. 35, 2010.

[24] R. Krebs, S. Spinner, N. Ahmed, and S. Kounev, "Resource Usage Control in Multi-tenant Applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, may 2014, pp. 122–131.

[25] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan, "Towards Multi-Tenant Performance SLOs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 6, pp. 1447–1463, jun 2014.

[26] H. A. Mahmoud, H. J. Moon, Y. Chi, H. Hacigümücs, D. Agrawal, and A. El-Abbadi, "CloudOptimizer: Multi-tenancy for I/O-bound OLAP Workloads," in *Proceedings of the 16th International Conference on Extending Database Technology*, ser. EDBT '13. New York, NY, USA: ACM, 2013, pp. 77–88.

[27] K. Molka and G. Casale, "Contention-Aware Workload Placement for In-Memory Databases in Cloud Environments," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 1, p. 29, 2016.

[28] ——, "Experiments or simulation? A characterization of evaluation methods for in-memory databases," in *Network and Service Management (CNSM), 2015 11th International Conference on*, nov 2015, pp. 201–209.

[29] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984.

[30] K. Kant, *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.

[31] P. J. Schweitzer, "Approximate analysis of multiclass closed networks of queues," in *Proceedings of International Conference on Stochastic Control and Optimization*, Amsterdam, 1979, pp. 25–29.

[32] R. Hooke and T. A. Jeeves, ""Direct Search" Solution of Numerical and Statistical Problems," *Journal of the ACM*, vol. 8, no. 2, pp. 212–229, 1961.

[33] J. L. Payne and M. J. Eppstein, "A Hybrid Genetic Algorithm with Pattern Search for Finding Heavy Atoms in Protein Crystals," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '05, 2005, pp. 377–384.

[34] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-Memory Big Data Management and Processing: A Survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 7, pp. 1920–1948, jul 2015.

[35] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "SAP HANA database: data management for modern business applications," *SIGMOD Record*, vol. 40, no. 4, pp. 45–51, jan 2011.

[36] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioningand Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[37] K. Molka and G. Casale, "Efficient Memory Occupancy Models for In-Memory Databases," in *Proceedings of the 24th International Symposium on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems (MASCOTS)*, 2016.

[38] J. Schaffner and T. Januschowski, "Realistic tenant traces for enterprise DBaaS," in *Proceedings - International Conference on Data Engineering*, 2013, pp. 29–35.