Experiments or Simulation? A Characterization of Evaluation Methods for In-Memory Databases

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*—The recent growth of interest for in-memory databases poses the question on whether established prediction methods such as response surfaces and simulation are effective to describe the performance of these systems. In particular, the limited dependence of in-memory technologies on the disk makes methods such as simulation more appealing than in the past, since disks are difficult to simulate. To answer this question, we study an in-memory commercial solution, SAP HANA, deployed on a high-end server with 120 physical cores. First, we apply experimental design methods to generate response surfaces that describe database performance as a function of workload and hardware parameters. Next, we develop a class-switching queueing network model to predict in-memory database performance under similar scenarios. By comparing the applicability of the two approaches to modeling multi-tenancy, we find that both queueing and response surface models yield mean prediction errors in the range 5%-22% with respect to mean memory occupancy and response times, but the accuracy for the latter deteriorates in response surfaces as the number of experiments are reduced, whereas simulation is effective in all cases. This suggests that simulation can be very effective in performance prediction for in-memory database management.**

***Index Terms*—Simulation; In-memory Database; Performance Model; Response Surface; Approximation; SAP HANA.**

I. Introduction

The growing interest for in-memory analytics has attracted industry and academia to use in-memory databases and related technologies as part of cloud-based offerings [1], [2]. Nonetheless, research still widely considers traditional disk-based database systems. Evaluation methods for such systems have mostly revolved around experiment-driven approaches that rely on Latin-hypercube designs, adaptive sampling or Kriging methods in order to construct underlying response surfaces of database performance [3]–[7]. In contrast, with in-memory analytics systems we experience fewer types of performance contention, since analytical data processing is mostly independent of disk access rates [8]. This simplifies the modeling process and introduces new potential for simulation-based methods to service management as an alternative to well established experiment-driven approaches. Accordingly, the question arises, if one, the other, or a joint use of both approaches is preferable.

Our work addresses this question with a comparative analysis that explores the applicability of simulation and response surfaces to model multi-tenancy performance for in-memory databases. Both approaches can help to anticipate database performance under different workload and hardware configurations and drive database management decisions that guarantee sustainable profits to cloud providers. However, there is no rule-of-thumb intuition that guides the choice for one specific method, since both come along with their advantages and disadvantages. To further explore the suitability of both methods we focus on the consolidation of large in-memory databases. This constitutes a new research problem, and thus performance impacts under these scenarios have not been studied yet. In particular, we use response surfaces to analyze the impact of analytical workloads and hardware parameters on tenant interference and memory occupancy in such systems. Since experiment time is a crucial factor for the accurate construction of these models, we also consider an evaluation of response surfaces based on a reduced set of experiments. To complement our analysis, we develop a queueing network based simulation model and explore its capability to describe response times and memory occupancy. We expect a simulation model to be a competitive alternative with little requirements on experimental time and broader applicability to ad-hoc evaluation scenarios.

During our study we observed that in-memory database performance seems mainly workload driven. We also noticed contrasting effects on response times and memory occupancy depending on hardware resource sharing strategies. This raises interesting questions for future work on database resource management. Further, we found our simulation model highly competitive to response surfaces, indicating potential for a joint use of both approaches as an effective aid to database management and capacity planning.

Summarizing, our main contributions are:

- A performance analysis of in-memory databases across different workload and hardware parameters using response surfaces
- Definition of a queueing network model to predict performance under multi-tenant workloads
- A comparative evaluation of response surface and simulation model

The remainder of our paper is structured as follows: Section II provides an overview of experiment- and model-driven system analysis approaches and describes our experimental testbed.

978-3-901882-77-7

Section III puts focus on the analysis of our chosen experiment design using response surfaces, while we present an alternative model based on a class-switching queueing network in Section IV. We give a comparative evaluation of our predictive models in Section V and discuss related work in Section VI. Finally, we conclude the paper and outline future work in Section VII.

II. Methodology

A. Approaches to In-Memory Database Analysis

In-memory databases are a new class of systems able to process multi-client and multi-tenant workloads with improved speedups compared to traditional hard disk based systems. However, analyzing the performance of these systems under varying workloads and hardware configurations can be costly and time consuming. Hence, we explore the applicability of both experiment-driven and model-driven performance analysis techniques. To provide a short comparison, Table I lists advantages and disadvantages of both approaches. In particular, we are interested if an experiment-driven approach is worth the large experimental time that comes with it compared to simulation. Therefore, the focus of our methodology is first set on experimental design methods, such as full factorial designs [9], to analyze in-memory database performance across various dimensions. We then explore the accuracy of regression-based approximations over different fractions of our experimental data which allows us to extend our considerations to fractional factorial designs. Lastly, we provide a comparison to a class-switching based queueing model for in-memory databases. The goal of this comparison is to clarify applicability of both methods to predicting queueing and non-queueing related performance metrics. With the former we mean response times, query throughputs and server utilization, performance measures that can be directly obtained from queueing networks. However, non-queueing related metrics, such as memory occupancy or power consumption, have to be inferred from direct measures as detailed in Section IV.

TABLE I
Comparison of System Analysis Approaches

Experiment-driven Approach	Model-driven Approach
Planning and Modeling	
+ Established experimental designs	+ Many options for model definition
- Dependent on good factor selection	- Choice of model complexity
- High effort in setting up testbed	- Experiments for model validation
Execution	
- Experiment time (high variability)	+ Relative short execution times
- Susceptible to experimental interruptions & measuring inaccuracy	+ Independent from experiment interruptions
Analysis	
+ High accuracy over design space	+ Only few experiments required for trace-driven parameterization
+ Good for metrics difficult to model	
+ Can feed trace-driven models	+ Re-usability
- Adding more metrics is very costly	+ Exploration of additional metrics trivial
- Potentially long trace analysis	
- Poor accuracy outside design space	- Accuracy-complexity trade-offs

B. Testbed Description

1) Hardware and Software Platform: Our experimental setup is composed of a TPC-H benchmark suite that drives analytical workloads on multiple SAP HANA in-memory database containers and manages monitoring tools to capture low-level hardware usage profiles. The hardware platform we use is an IBM x3950 X6 server running SLES 11 SP3. This test server features 8 processor sockets with a total of 120 physical cores and provides a total of 6 TB RAM corresponding to 750GB per socket. The software stack of our benchmark driver comprises shell and python scripts for performing database resource allocation, managing experiment specific parameter settings, submitting TPC-H queries and actuating monitoring tasks. In particular, we focus on modeling the computational time to process queries, but not the cost of result fetch operations that require both CPU and network resources. Dealing with networks adds a new layer of complexity in our models that we leave for future work. Therefore, in our experiments we kept benchmark driver and database containers on the same server and issued queries without fetching the result set. However, this does not imply workloads are becoming light-weight. Due to multiple concurrently active users and the CPU intensive character of analytical workloads we are still experiencing high contention for resources and query interference, keeping our test system heavily utilized.

2) Resource Allocation and Measurement Tools: During our experimentation we make use of HANA's multi-tenant database containers. More specifically, we define a tenant as a customer that has exclusive access rights to one or more database containers/processes. The workload of a tenant is generated by one or more users/clients that relate to this specific tenant. The advantage of a multiple-container system that manages several tenant database containers lies in the possibility of isolating tenant database resources for performance and security reasons, while avoiding additional overhead by sharing non-tenant specific database processes across all containers. In particular, we make use of Linux *cgroups* to specify the number of CPU sockets a tenant database container is assigned to. This allows us to control the degree of overlapping across tenant databases, e.g. full, partial or isolated. Moreover, we employ three monitoring tools during our experimentation to collect micro architectural statistics and database memory usage profiles. First, we use Intel PCM [10] to retrieve information about hardware performance counters for CPU core utilization, core and DRAM power consumption, CPU temperature and other measures. Next, we use a tool that continuously monitors core-affinity of database threads [11], from which we infer threading levels of database queries. Lastly, we implement a monitor that samples the total memory used by all active tenant database processes. To ensure sampling accuracy with short query execution times, we set the sampling granularity of all monitoring processes to 50 ms, for which monitoring overheads remained below 0.1% during our experiments.

C. Experimental Design

For our study, we are interested in six different workload and resource configuration parameters that we believe have

TABLE II
CHOICE OF DESIGN FACTORS

Factor	Description
x_1	Client think time (in seconds)
x_2	Number of active tenant databases
x_3	Database scale factor, uncompressed (in GB)
x_4	CPU-sockets a tenant database is assigned to
x_5	Degree of resource overlapping between two tenants
x_6	Number of clients/users per tenant

a large impact on database performance. We list our choice of these parameters, in experimental design terminology also called *factors*, in Table II.

First, we consider workload parameters that directly correspond to the TPC-H workload intensity. These are think times x_1 that define the delay between database requests sent by each client and the number of clients x_6 that are active at the same time. Both parameters are indirect proportional, meaning that database response times surge with an increase of clients or a decrease in think times. Furthermore, with factor x_3 we consider the TPC-H database scale factor (SF). The scale factor describes the uncompressed size of a tenant dataset that has to be processed during query execution, and thus we expect this factor to be a main driver of database response times and memory occupancy. With the latter we mean the total memory used by a tenant database process. In addition, we are interested in how the number of CPU sockets x_4 that a tenant database is assigned to affects its performance. We think that while not necessarily beneficial under light load, a higher number of CPUs is a welcome processing resource under heavy workload scenarios. Furthermore, our considerations include the number of tenant databases x_2 that are deployed on the same server and the degree of resource sharing between several tenants x_5. These two factors are of importance for consolidation scenarios, since we can pack more tenants onto a server when processing resources are shared. In Figure 1 we illustrate this with an exemplary packing under different overlapping strategies on a 4-socket system. Since overlapping forces tenant performance interference it becomes an interesting factor for our analysis.

Analyzing our choice of six factors is a difficult task since an experimental exploration of all their combinations is time consuming. For example, a complete analysis of six factors each at four different settings would require more than 17 months worth of experiments at execution times of 3 hours per experiment. Experiment designs can drastically reduce the number of experiments required to efficiently explore such factors settings. Hence, we focus on 2^k full factorial designs [9], with $k = 6$. These designs are restricted to two settings for each of the k factors. As a first step to obtain the corresponding experiment design matrix, we define respective lower and upper factor bounds $\{X_l, X_h\}$, listed in Table III. We can then construct the design matrix using the senary Cartesian product over the six sets $\{X_l^i, X_h^i\}$, $i = 1, ..., 6$ formed by columns x_i in Table III. This results in 64 experiment runs that account for a total experiment time of eight days.

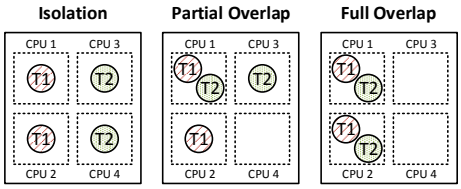


Fig. 1. Different Resource Assignment Strategies for 2 consolidated Tenants (T1 and T2) that have access to 2 Sockets on a 4 Socket System.

TABLE III
DESIGN FACTORS AND THEIR BOUNDS

Bound	x_1	x_2	x_3	x_4	x_5	x_6
Lower Bound (X_l)	0	2	30	2	full	8
Upper Bound (X_h)	10	4	300	4	isolated	16

III. FULL FACTORIAL DESIGN

Based on our chosen 2^6 design we want to determine the effects on in-memory database performance for each design factor. Hence, we explore the variation that each of the main factors and two-factor interactions contributes to the total variation of response times, utilization, memory occupancy and CPU power consumption. To do so, we fit a second-order regression model with two-factor interaction terms on each of the response variables across our datasets from all 64 experiments, whereby input variables x_i are mapped into the [-1,1] domain [9]. Subsequently, we use the regression coefficients to determine individual factor contributions. Our regression model has the following form:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k} \sum_{j=1}^{k} \beta_{i,j} x_i x_j + \epsilon, \tag{1}$$

with k regression coefficients $\beta = (\beta_0, ..., \beta_k)$ and an additive error ϵ, estimated with ordinary-least squares (OLS). While the first term β_0 gives an estimate for the mean model response, the second term expresses the main effects for each of the design factors and the third term corresponds to two-factor interaction effects. We determine the variation per factor SSX_k compared to the total variation SST of each response according to [9]:

$$\mathrm{SSX}_k = 2^k \frac{\beta_k}{\mathrm{SST}} \tag{2}$$

$$\mathrm{SST} = \sum_{i=0}^{2^k} (y_i - \bar{y})^2, \tag{3}$$

where 2^k with $k = 6$ accounts for the number of experiments and $\bar{y}$ denotes the mean over the set of measured responses $y_i, i = 1, ..., 2^k$. Furthermore, we used a common logarithmic transformation, suggested also in [9], where we fit the natural logarithm $\ln y$ of the respective response variable, also known as multiplicative model. The reason for this approach is that we can explain the total variation in response times due to main factor and two-factor interaction effects to 99.8% using a multiplicative model. An additive model would explain a total variation of only 93.3%.

TABLE IV
VARIATION OF RESPONSES (IN %) DUE TO FACTORS AND THEIR INTERACTIONS

Factor	RTime	Util	M_{mean}	M_{peak}	Power
x_1	1.01	17.17	2.86	0.20	13.37
x_2	1.51	0.51	0.03	0.01	0.06
x_3	81.01	13.23	90.04	95.98	8.93
x_4	1.06	3.70	0.40	0.19	6.36
x_5	8.90	15.97	0.09	0.01	31.84
x_6	3.81	8.05	4.09	2.73	6.68
x_1x_2	0.00	0.38	0.02	0.00	0.44
x_1x_3	0.58	12.47	1.74	0.37	10.35
x_1x_4	0.00	0.64	0.04	0.02	0.41
x_1x_5	0.02	6.16	0.09	0.02	4.37
x_1x_6	0.00	2.21	0.14	0.01	1.70
x_2x_3	0.00	0.32	0.02	0.00	0.33
x_2x_4	0.01	0.05	0.00	0.00	0.44
x_2x_5	1.46	0.41	0.03	0.00	1.22
x_2x_6	0.01	0.17	0.00	0.01	0.18
x_3x_4	0.00	0.40	0.01	0.01	0.18
x_3x_5	0.01	4.32	0.03	0.07	2.79
x_3x_6	0.00	1.36	0.06	0.01	0.97
x_4x_5	0.13	1.15	0.00	0.05	1.15
x_4x_6	0.06	0.79	0.06	0.04	0.54
x_5x_6	0.18	2.41	0.00	0.00	1.58

We present the results of our analysis in Table IV. More specifically, we chose one reference tenant and report the factor importance across our experiment design for this tenant with regard to mean response times, mean memory occupancy and peak memory occupancy. In addition, we report results for the mean utilization and CPU power consumption per socket. We observe a distinct behavior across response times, utilization and memory occupancy. While for our chosen design response times and utilization seem primarily affected by the database scale factor, number of clients and tenant overlapping, memory occupancy seems relatively insensitive to the latter. We were interested if our system measurements can support the predicted factor importance and looked at the scenario with scale factor 30, think time 10 seconds, eight clients and two sockets. We give the results for this scenario in Figure 2, which shows the increase in three different responses when varying the number of tenants that overlap. As suggested by Table IV, response times are strongly affected by tenant overlapping, while peak memory occupancy seems indeed independent from the degree of overlapping. We also noticed that power consumption is more sensitive to overlapping than CPU utilization. This can be partially explained by an increased power consumption of cache units since likewise overlapping causes an increase of L3 cache misses with a contribution of 21.4%. Next to the importance of main factors we further noticed that CPU utilization and power consumption show large two-factor interaction effects. Thus, if a more detailed analysis of these metrics is required, an experimenter should avoid particular resolution IV designs and any resolution III design [9], in which important two-factor interactions are confounded with other two-factor interactions.

Summarizing the results, we found that across our experiment design space in-memory database performance is mainly workload driven. However, we also observed a large contrasting effect of resource overlapping strategies on response times and memory occupation. This opens an interesting question for future research, since from a memory perspective overlapping can be an effective strategy to increase tenant packing.

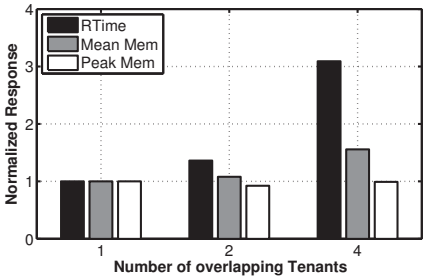


Fig. 2. Increase of different Responses based on Overlapping. Scenario: SF30, 2 Sockets, 8 Clients. Overlapping corresponds to Utilizations of 0.34, 0.64 and 0.98. Values normalized by Single Tenant Scenario.

IV. QUEUEING NETWORK BASED PREDICTION MODEL

A. Model Description

As alternative to the regression model presented in Section III, we define a multi-class closed queueing network (QN) model to capture in-memory database characteristics and estimate database performance contention under different design factors. Figure 3 provides a structural overview of our in-memory database model. In particular, we use a closed QN with class-switching to model the closed-loop execution of TPC-H workloads. Using a class-switching model we can simulate the behavior of a TPC-H client that recurrently submits a permuted sequence of all 22 TPC-H query classes. This means that for each client only one query job is in the system at the same time, but after execution it changes its query class. For a more detailed view on the concept of class-switching see [12, Sec 7.3.6]. Furthermore, we use delay centers to model the client think time between single query submissions. To account for tenant database specific threading limits, we employ admission queues that delay query submissions once the corresponding finite capacity region is full. During our experimentation we observed that threading levels can become highly variable with analytical workloads. Hence, we use a fork-join construct that captures the database internal query processing mechanism on multi-core systems. With the fork-join feature we can model the parallelization of query plans into l subtasks that are handled by query engine specific worker threads and their synchronization during a parallel aggregation phase. To model the execution of worker threads, we use processor sharing (PS) queues, where service times are generally distributed i.i.d. random variables [13]. Further, we use the concept of class-chains [12, Sec 7.3.6] to distinguish between client activity across multiple tenant databases. Hence, for each tenant we separately define 22 classes and one class-switch to ensure client jobs remain within their respective chain.

To further emphasize the importance of class-switching we compared our simulation model against the simulator in [14] that does not consider such a concept. The advantage of our model is that we can define the total workload population N as

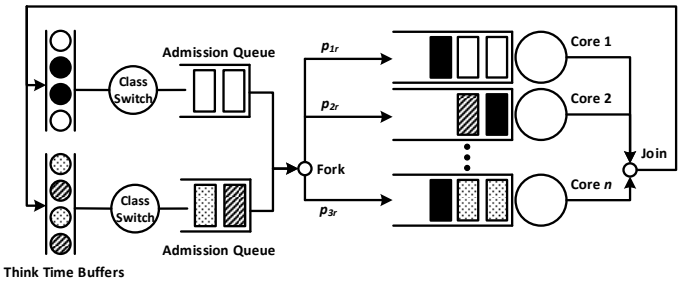


Fig. 3. Multi-tenancy Queueing Model of an In-memory Database Server

the maximum number of TPC-H clients that submit query requests. For the simulator in [14] however, we have to oversize the maximum workload population and set N for each query class separately to the maximum number of TPC-H clients. As shown in [14] this approach requires think-time estimates from trace logs dependent on the number of active clients, from which our class-switching model is independent. To analyze the accuracy of both simulation models, we compared their response time predictions against trace response times for the 16 user scenario of our traces in [11]. In particular, we used the simulation framework JMT [15] to solve our queuing model. The results confirmed the accuracy of the non-class-switching model at a relative error of 25%, while our model was able to reduce this gap down to 6%.

B. Predictive Functionality

This section presents the methods we use to obtain queueing- and non-queueing related performance estimates from our in-memory database specific model. In fact, we are not only interested into mean performance estimates but also want to consider the more expressive probabilistic measures. The difficulty of using analytical methods to obtain mean estimates from fork-join models that consider variable threading levels is discussed in [14], [16], [17]. However, obtaining probabilistic estimates for such models adds additional layers of complexity. Hence, we employ the simulation framework JMT to solve the queueing model given in Figure 3 and obtain queueing related measures, such as server utilization, response times and queue lengths directly from simulation logs. To obtain non-queueing related metrics, we infer memory occupancy or power consumption from these measures. In particular, we propose a modification to our memory occupancy model in [17] to estimate the distribution of memory occupancy from JMT trace logs:

$$M_i = \sum_{c=1}^{C} \frac{Q_{c,i}}{l_c} m_c \min(1, \omega_i). \tag{4}$$

Here, M_i denotes the estimation of memory occupancy expressed as sum of products of per-class queue lengths $Q_{c,i}$ and per-class memory consumption m_c over the sampling interval i, with $i = 1, ..., I$. In our case, we split the JMT simulation logs into I sampling intervals at a granularity of one second per interval of simulated time. Further, we note that since JMT samples queue lengths on the level of forked jobs, we have to approximate the number of actual query jobs that are active in the system. To do this we first divide $Q_{c,i}$ by the corresponding per-class threading levels l_c. We justify this scaling by an observation we made in our single user experiments, indicating that per-class memory occupancy is independent from the threading level. Second, we provide a correction term $\min(1, \omega_i)$ with

$$\omega_i = \frac{TL}{\sum_{c=1}^{C} Q_{c,i}} \tag{5}$$

to approximate the number of active jobs in cases where the simulator predicts a total number of threads that exceeds the database threading limit *TL*. Theses cases occur since currently JMT does not allow to define tenant-specific finite capacity regions that share the same processing resources. Hence, we run the simulation without finite capacity regions and scale down $Q_{c,i}$ by ω_i for intervals i during which the number of forked jobs $Q_{c,i} > TL$. Once we have estimates for $Q_{c,i}$, peak and mean memory occupancy can be determined as follows:

$$M_{\text{peak}} = \max_i M_i \tag{6}$$

$$M_{\text{mean}} = I^{-1} \sum_{i}^{I} M_i \tag{7}$$

To obtain power consumption estimates based on CPU utilization recent work suggests linear and non-linear models [18]. Hence, we considered the following two formulations:

$$P_{\text{linear}} = c_1 U + c_0 \tag{8}$$

$$P_{\text{quadratic}} = c_2 U^2 + c_1 U + c_0, \tag{9}$$

that model power consumption based on the mean CPU utilization U. To calibrate these models, we determined the coefficients c_i using linear regression over power and utilization measurements from our full-factorial experiments. We obtained c_1 and c_0 for a linear model with 1.33 and 37.02 respectively and $c_2 = 0.0039, c_1 = 0.79, c_0 = 52.91$ for a quadratic model. However, we noticed a slightly better coefficient of determination of 0.932 for a quadratic model over 0.929 for a linear model, and thus we will use (9) during our evaluation.

V. Evaluation

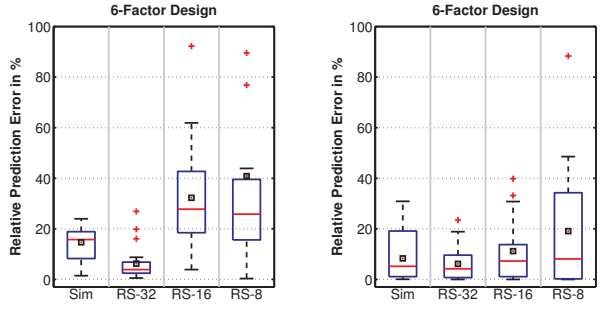
A. Methodology and Evaluation Scenarios

1) Overview: Though restricted to two levels, 2^6 full factorial designs still come with a high experimental effort. In fact, the corresponding 64 design points of our 2^6 design account for eight days worth of experiments, at an experiment time of three hours including overhead incurred by tenant restarts. Hence, we want to explore at what cost in model accuracy this effort can be reduced using different fractional factorial designs [9]. In particular, we are focusing on three 2^{6-p} designs, where $p = 1, 2, 3$. These designs correspond to 32 (four days), 16 (2 days) and 8 (one day) experiments respectively. We rely on Matlab's *fracfact* function to generate 2^{k-p} designs and use the regression model in (1) to fit a

TABLE V
DISTINCT CHOICE OF DESIGN POINTS FOR EVALUATION

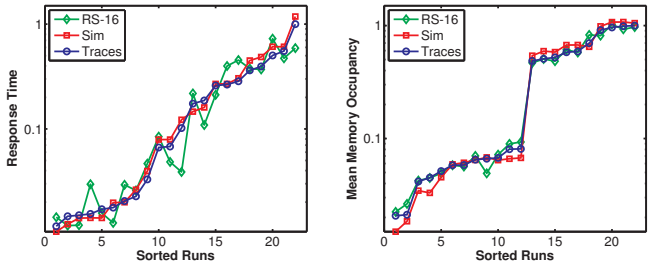
Heavy Load (x_1=0)					Medium Load (x_1=10)				
x_2	x_3	x_4	x_5	x_6	x_2	x_3	x_4	x_5	x_6
2	30	2	full	16	2	30	2	full	8
2	30	2	isolated	8	2	30	2	isolated	16
2	30	4	full	8	2	30	4	full	16
2	300	2	full	8	2	300	4	full	8
2	300	4	full	16	2	300	4	isolated	16
2	300	4	isolated	8	4	30	2	full	16
4	30	2	full	8	4	30	2	isolated	8
4	30	2	isolated	16	4	30	4	isolated	16
4	30	4	isolated	8	4	300	2	isolated	16
4	300	4	full	8	4	300	4	full	16
4	300	4	isolated	16	4	300	4	isolated	8

x_1: Think Time in s, x_2: Number of Tenants, x_3: Database Scale Factor, x_4: Sockets per Tenant, x_5: Degree of Overlapping, x_6: Number of Clients



(a) Response Time (b) Utilization

Fig. 4. Distribution of Prediction Errors for Response Times and Utilization based on the 6 Factor Design.



(a) Response Time in Seconds (b) Mean Memory Occupancy in GB

Fig. 5. Predicted Response Times and Mean Memory Occupancy, Evaluation Runs sorted ascending by Trace Values - normalized by respective maximum Trace Value, logarithmic Scale

performance measure of interest. For the actual evaluation, we use a distinct subset over the remaining $2^k - 2^{k-p}$ data points that all three models have to predict. Table V list the corresponding scenarios that have not been used during the model training process. In addition, we compare all three regression models with our class-switching simulation model. In contrast to all regression models, our simulation model does only require service demands and threading levels obtained from isolated query runs. We have obtained both parameters from single user experiments and refer to [17] for more detailed information about the extraction process.

B. Prediction of Queueing Related Metrics

At first we are interested in the accuracy of mean response time and utilization estimates. Since we look across many different scenarios, we prefer to give the distribution of errors that have to be expected when using each model. Figure 4 shows the corresponding results for our simulation model and the three regression models built from 32 experiments (RS-32), 16 experiments (RS-16) and 8 experiments (RS-8). In particular, we use Matlab's standard box plots, augmented with squares to represent the mean for each response. The results suggest that our simulation model achieves a good performance for both mean response times and server utilization with a 75th percentile below a relative error of 20%. Interestingly, the regression models that use a fraction of 1/4 (RS-16) and 1/8 (RS-8) of the original full-factorial design result in large prediction errors. We attribute this to the confounding of important two-factor interactions with other two-factor interactions. For example, in the case of RS-16, the response time interaction BE (Table IV) is confounded with two other two-factor interactions, and can thus not be estimated properly.

TABLE VI
VARIATION OF RESPONSES (IN %) DUE TO MAIN FACTORS

Factor	Response Time				Utilization			
	RS-64	RS-32	RS-16	RS-8	RS-64	RS-32	RS-16	RS-8
x_1	1.01	1.02	1.21	0.57	17.17	16.02	12.57	14.99
x_2	1.51	1.37	0.94	0.50	0.51	0.96	1.15	12.41
x_3	81.01	81.42	82.04	78.09	13.23	12.61	9.39	11.81
x_4	1.06	0.90	1.04	0.73	3.70	4.54	6.10	15.66
x_5	8.90	9.14	8.64	10.74	15.97	15.32	16.32	16.72
x_6	3.81	3.82	3.35	8.85	8.05	8.41	8.06	15.88

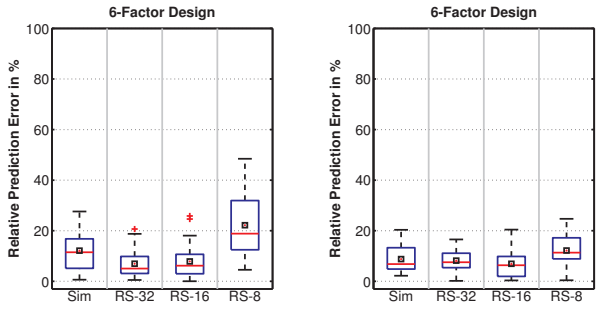
x_1: Think Time, x_2: Number of Tenants, x_3: Database Scale Factor, x_4: Sockets per Tenant, x_5: Degree of Overlapping, x_6: Number of Clients

A further analysis of the residuals between simulator response times and actual measurements revealed that estimates are slightly optimistic under light load scenarios and mostly pessimistic under heavy load. We can observe a similar behavior in Figure 5(a), where optimistic predictions tend to occur in scenarios with small or medium response times. We also discovered that response time estimates for scale factor 30 are generally more optimistic than for scale factor 300. We explain the cause for this in measurement inaccuracies for threading levels of scale factor 30 and expect improvements using a smaller sampling intervals.

Due to the large prediction errors for RS-16 and RS-8, we were interested if these methods can at least be used to estimate the main factor importance of the corresponding 2^6 full factorial design. Table VI presents a comparison of these estimates. While factor estimates for response times seem reasonable under all fractions, we noticed that particularly for CPU utilization main factors cannot be estimated accurately with a 2^{6-3} fractional factorial design (RS-8) due to severe confounding of main factors with important two-factor interactions.

C. Prediction of Non-Queueing Related Metrics

Since the focus of our work is on in-memory databases, we are also interested in analyzing the prediction quality for a second group of metrics, such as mean and peak memory



(a) Mean Memory Occupancy (b) Peak Memory Occupancy

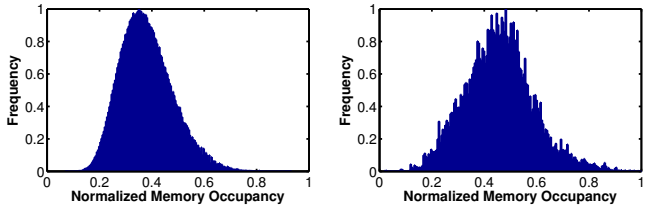
Fig. 6. Distribution of Prediction Errors for Mean and Peak Memory Occupancy based on the 6 Factor Design.

occupancy as well as CPU power consumption. These metrics are not directly available from queueing theoretic models, and thus we use the approximations given in (4) and (9) to obtain estimates for these.

1) Memory Occupancy: Figure 6 presents the results for our analysis of mean and peak memory predictions across all four predictive models. First we observe, in contrast to query response times and CPU utilization, that regression estimates for memory occupancy seem generally more accurate. We suspect this is due to a less severe confounding, since memory occupancy shows fewer important main factor and two-factor interaction effects (Table IV). Interestingly, our simulation model seems to be competitive with regression models fitted over 1/2 (RS-32) and 1/4 (RS-16) of the corresponding full factorial design. Moreover, for our simulation model the analysis in Figure 5(b) suggests a similar prediction trend compared to response times: slightly optimistic for scale factor 30 and slightly pessimistic for scale factor 300, whereby both scale factors are be distinguished by the significant jump in memory occupancy.

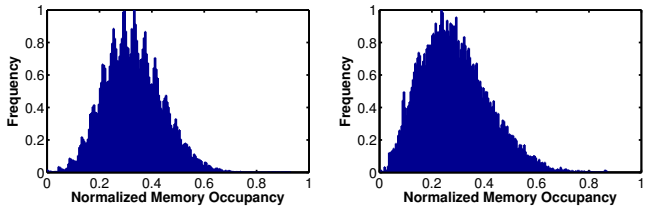
Since our simulation model yields promising estimates for mean and peak memory occupancy, we will further investigate if a similar accuracy can be retained for probabilistic estimates. For this analysis we examine the memory occupancy histogram under scenarios with two SF30 tenants that are sharing the same two CPU sockets and run 16 concurrent clients with 1) no think times (Figure 7) and 2) a think time of 10 seconds (Figure 8). For comparison purposes, we normalized all four histograms by the maximum trace memory occupancy from scenario 1). First, we observe a similar general trend between simulation and trace logs. Second, the simulation seems more optimistic under high load. The reason for this can be found in the jobs queue length correction in (4), where our scaling seems to strong. We suspect a more detailed approximation could lead to better estimates. Under the more practical scenario in Figure 8(a), where think time delays occur between query submissions, the results indicate a better approximation of the long tail. Concluding, the probabilistic estimates look promising for a more detailed analysis as part of our future work.

2) Power Consumption: Finally, we want to include an analysis for estimates of CPU power consumption. Models that



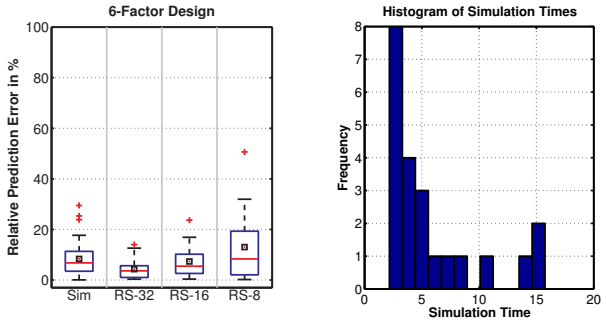
(a) Simulation (b) Traces

Fig. 7. Predicted Memory Occupancy Histogram vs. Traces for Scenario: Zero Think Time, 2 Tenants, Scale Factor 30, 2 Sockets, Full Overlapping, 16 Clients - (normalized by max value from corresponding trace scenario)



(a) Simulation (b) Traces

Fig. 8. Predicted Memory Occupancy Histogram vs. Traces for Scenario: 10 Second Think Time, 2 Tenants, Scale Factor 30, 2 Sockets, Full Overlapping, 16 Clients - (normalized by max value from trace scenario in Figure 7(b))



(a) Power Consumption (b) Simulation Times in Minutes

Fig. 9. (a) Distribution of Prediction Errors for Power Consumption based on the 6 Factor Design, (b) Simulation Times across all 22 Evaluation Scenarios.

can accurately capture this metric have potential in scheduling and resource allocation scenarios. Before we analyze the results, our first anticipation is that prediction accuracy for power consumption should follow the trend in corresponding utilization estimates. We expect this behavior, since there is a direct relationship between these two metrics. Figure 9(a) presents the distribution of relative prediction errors across all four models. As expected, we observe a similar accuracy for all models compared to utilization estimates in Figure 4(b). A more detailed analysis for our simulation model revealed that prediction errors above 20% correspond to light and medium load scenarios. We noticed a similar trend for utilization estimates and expect that solving the pessimistic prediction of utilizations, will lead to better estimates for power consumption.

TABLE VII
MEAN RELATIVE PREDICTION ERRORS (%) ACROSS ALL METRICS

Method	RTime	Util	M_{mean}	M_{peak}	Power
Sim	13.77	10.16	11.60	9.21	9.14
RS-32	6.27	6.14	6.97	8.17	4.28
RS-16	32.27	11.20	7.85	6.94	7.32
RS-8	40.87	19.10	22.19	12.18	13.02

D. Conclusion

Summarizing our results, we observed that:

- Response surfaces over fractions 1/4 and 1/8 can be effectively used to restore a corresponding full-factorial design for memory occupancy
- Response times and utilization seem most sensitive to two-factor interactions and require more experiments to be captured accurately
- With regard to Table VII simulation models can retain a good accuracy across all classes of metrics and show potential for probabilistic estimates for memory occupancy
- Simulation poses little requirements on experimental time, Figure 9(b), compared to our 2^6 design with 2-3 hours per experiment

VI. RELATED WORK

The majority of studies that evaluate database performance have focused on system measurement and machine learning techniques. For example, the authors in [19] propose linear regression models to predict response time violations for multi-tenant in-memory databases, but consider only small tenant datasets < 204 MB and servers with 2 compute units. Other works consider the joint use of machine learning and adaptive sampling techniques [3]–[5]. For example [3], [5], focus on multivariate regression methods and support vector machines to capture contention effects under analytical query mixes and model performance effects that disk and memory contention impose on database performance. In [4], the authors further propose collaborative filtering and adaptive sampling to construct response surfaces of database query throughputs. However, all three approaches are evaluated on small scale PostgreSQL systems only, do not consider large multi programming levels and put focus largely on I/O contention models. In [20] support vector machines and kernel canonical correlation analysis are used to predict query performance under analytical workloads. The approach is evaluated on top of PostgreSQL for scale factors TPC-H 1 and 10, but requires fine-grained information on query operator level. Kriging-based techniques using Gaussian process regression are leveraged in [21] and [7] to estimate response surfaces of query performance under TPC-H workloads on IBM DB2 and PostgreSQL systems. However, both works do neither consider hardware resource assignment parameters nor multi-tenancy aspects and limit their evaluation to small database scale factors. Experiment design methods, such as fractional factorial and response surface designs are successfully used in [22] to approximate and optimize Markovian models, but the focus is put on optimization of a theoretical model rather than a real system.

Methods to guarantee performance SLOs under transactional and analytical multi-tenant workloads have been proposed in [23] and [6]. Both works consider profiling-based analyses, but rather focus on non-in-memory hardware configurations. In [24] and [25] authors present management strategies for placing tenants in multi-tenant databases. [25] consider replica swapping to minimize the interference of co-located tenants and employ linear additive models for server utilization and I/O load approximations. However, in contrast to our analysis, both works focus on transactional workloads where tenants characteristics largely differ from in-memory focused customers.

Several works investigate system performance under different resource allocation strategies [26]–[28]. The authors [26] present an experimental evaluation of OLTP performance under different multi-socket database configurations and identify the variability of communication latencies under NUMA architectures as key factor that impact database performance. Management frameworks based on collaborative filtering, profiling and admission control methods are introduced in [27] and [28] to anticipate impact of resource assignments on application performance. However, neither of these approaches puts focus on large in-memory compute workloads.

VII. CONCLUSIONS AND FUTURE WORK

Summarizing our work, we have provided a comparative evaluation of response surfaces and simulation methods regarding their suitability to model in-memory database performance as function of different workload and hardware parameters. Our study was motivated by the limited dependence of in-memory databases from disk, which likewise simplifies the modeling process. In particular, we observed a high accuracy for our predictive simulation model over different classes of performance metrics, such as response times, server utilization, energy consumption and memory occupancy. Well established response surface models however, show a deterioration of prediction errors once the number of experiments is reduced. Our results suggest that simulation techniques can be an effective aid for predicting in-memory database performance.

For future work we see potential in a more detailed analysis of consolidation scenarios, where tenants are partially sharing hardware resources and plan to extend our simulation model accordingly. Another promising direction is the use of probabilistic estimates for memory occupancy to drive tenant consolidation decisions rather than relying on conservative peak memory usage. Finally, our evaluation was based on a case study using SAP HANA, leaving the open question for future research whether our models are applicable to other in-memory systems.

ACKNOWLEDGMENT

The work of Giuliano Casale is supported by the EU project DICE H2020-644869.

REFERENCES

[1] "SAP HANA Enterprise Cloud." [Online]. Available: http://hana.sap.com/deployment/sap-hana-enterprise-cloud.html

[2] "Spark - Lightning-fast cluster computing." [Online]. Available: http://spark.incubator.apache.org/

[3] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal, "Performance prediction for concurrent database workloads," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*, 2011.

[4] J. Duggan, Y. Chi, H. Hacigumus, S. Zhu, and U. Cetintemel, "Packing light: Portable workload performance prediction for the cloud," in *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, Apr. 2013, pp. 258–265.

[5] J. Duggan, O. Papaemmanouil, U. Çetintemel, and E. Upfal, "Contender: A Resource Modeling Approach for Concurrent Query Performance Prediction," in *Proceeding of the 17th International Conference on Extending Database Technology (EDBT '14)*, 2014.

[6] 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.

[7] S. Duan, V. Thummala, and S. Babu, "Tuning Database Configuration Parameters with iTuned," in *VLDB*, vol. 2, no. 1. VLDB Endowment, 2009, pp. 1246–1257.

[8] 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.

[9] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, 1991.

[10] "IntelPCM." [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[11] A. Jula, "Multicore scalability and NUMA effects on HDB with SAP-H data and TPC- H queries," SAP, Tech. Rep., 2012.

[12] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*. Wiley-Interscience, 2006.

[13] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the ACM*, vol. 22, no. 2, pp. 248–260, 1975.

[14] S. Kraft, G. Casale, A. Jula, P. Kilpatrick, and D. Greer, "WIQ: Work-Intensive Query Scheduling for In-Memory Database Systems," in *Proceedings of the fifth International Conference on Cloud Computing (CLOUD '12)*, 2012.

[15] M. Bertoli, G. Casale, and G. Serazzi, "JMT: Performance Engineering Tools for System Modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, Mar. 2009.

[16] A. Thomasian, "Analysis of Fork/Join and Related Queueing Systems," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 17:1—-17:71, Aug. 2014.

[17] K. Molka, G. Casale, T. Molka, and L. Moore, "Memory-aware sizing for in-memory databases," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS '14)*, 2014.

[18] C. Möbius, W. Dargie, and A. Schill, "Power consumption estimation models for processors, virtual machines, and servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1600–1614, 2014.

[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] M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based Query Performance Modeling and Prediction," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, Apr. 2012, pp. 390–401.

[21] M. Ahmad, S. Duan, A. Aboulnaga, and S. Babu, "Predicting Completion Times of Batch Query Workloads Using Interaction-aware Models and Simulation," in *Proceedings of the 14th International Conference on Extending Database Technology*, ser. EDBT/ICDT '11. ACM, 2011, pp. 449–460.

[22] P. Kemper, D. Mueller, and A. Thuemmler, "Combining Response Surface Methodology with Numerical Methods for Optimization of Markovian Models," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 259–269, 2006.

[23] 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.

[24] 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.

[25] 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.

[26] D. Porobic, I. Pandis, and M. Branco, "OLTP on hardware islands," *Proceedings of the VLDB Endowment*, pp. 1447–1458, 2012.

[27] 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.

[28] 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.