SLA-driven Planning and Optimization of Enterprise Applications

Hui Li
SAP Research Karlsruhe
Vincenz-Priessnitz-Strasse 1
76131 Karlsruhe, Germany
hui.li@computer.org

Giuliano Casale
SAP Research Belfast
Newtownabbey, BT37 0QB
United Kingdom
giuliano.casale@sap.com

Tariq Ellahi
SAP Research Belfast
Newtownabbey, BT37 0QB
United Kingdom
tariq.ellahi@sap.com

ABSTRACT

We propose a model-based methodology to size and plan enterprise applications under Service Level Agreements (SLAs). Our approach is illustrated using a real-world Enterprise Resource Planning (ERP) application, namely SAP ERP. Firstly, we develop a closed queueing network model with finite capacity regions describing the SAP ERP application performance and show that this model is effective and robust in capturing measured response times and utilizations. Secondly, we propose an analytical cost model of ERP hosting that jointly accounts for fixed hardware costs and dynamic operational costs related to power consumption.

Based on the developed performance and cost models, we propose to use multi-objective optimization to find the Pareto-optimal solutions that describe the best trade-off solutions between conflicting performance and cost-saving goals. Experimental validation demonstrates the accuracy of the proposed models and shows that the attained Pareto-optimal solutions can be efficiently used by service providers for SLA-driven planning decisions, thus making a strong case in favor of the applicability of our methodology for deployment decisions subject to different SLA requirements.

1. INTRODUCTION

Enterprise resource planning (ERP) applications are a class of software systems that provide the core business functionalities for an enterprise which aims to improve its productivity and efficiency. For instance, SAP ERP is used for business activity coordination and resource management in large and midsize enterprises [23]. Traditionally, industrial enterprise software systems such as ERPs are purchased, provisioned, and maintained on-premise, especially in large organizations where capacity planning decision are updated infrequently and rely on simple sizing rules. However, the ongoing trend of cloud computing and Software-as-a-Service (SaaS) [1] is showing an increasing trend toward configuring enterprise applications as *on-demand* services. This trend makes increasingly complex the sizing and provisioning of applications because of dynamic business-driven requirements on capacity, responsiveness, and operational costs that must be met in combination to make the on-demand approach both scalable and cost-effective. Clearly, it is very difficult to take into account all cost and performance requirements without properly engineered methodologies. This paper proposes a solution to this problem by developing a model-based methodology for application deployment that is capable of accounting for multiple cost and performance constraints using multi-objective optimization [10].

Binding contracts on the service levels between customers and service provider are ubiquitous in modern service-oriented applications. Service Level Agreements (SLA) are a common way to specify such contractual terms, including both functional and non-functional properties [19]. SLAs can be used by customers and service providers to monitor if an actual service delivery complies with the agreed terms. In case of SLA violations, penalties or compensations can be directly derived. From the service provider perspective, it is important to guarantee the service level objectives specified in the SLA in terms of performance and availability requirements. At the same time, it is crucial for the service provider to reduce the Total Cost of Ownership (TCO) as quantified by hardware and operational costs related to power consumption and IT management.

This paper proposes a new methodology for an enterprise application provider to optimally plan and size its application deployment according to stated SLA objectives. On one hand, a *performance model* is developed for a real-world ERP enterprise application, showing how a simple analytical queueing-theoretic model can describe the performance of real-world industrial applications that are much more complex than simplified systems considered for performance modeling exercises in the literature. On the other hand, a *cost model* is developed to quantify the tangible costs for hosting such applications. We show that these two models are able to predict the performance and cost objectives. The performance and cost goals are conflicting with each other, and we apply a multi-objective optimization (MOO) technique to find the so-called *Pareto-optimal* (best trade-off) solutions. The Pareto-optimal set enables the service provider to define, evaluate, and decide on the performance goals in SLAs with respect to the cost factors. Recommendations on the best deployment decisions are readily derived from the Pareto-optimal solutions. To the best of our knowledge, this is the first time that a comprehensive methodology that jointly uses queueing network models and multi-objective optimization is proposed for deployment of enterprise applications under SLA constraints. Specifically, the novel contributions of this paper can be summarized as follows:

• We propose a closed queueing network model with finite capacity region (*FCR queueing network*) for the ERP enterprise application. This model describes a multi-station, multi-tier architecture, which takes into account software threading levels affecting multiple resources and their impact on the underlying hardware. We show that this performance model is able to predict the end-to-end response time of the ERP application in a way that reflects well measurements of the real system in operation.

• We develop a cost model that is able to quantify two tangible cost components of the TCO, namely fixed hardware costs and dynamic server power consumption, which is an operational cost determined by application usage. For server hardware, we propose a pricing model that is function of per-core performance and the number of cores, thus accounting for the current trend of using multi-core architectures in enterprise servers. Server power consumption, on the other hand, is modeled as a function of CPU utilization. Server hardware and power consumption are weighted to reflect different cost structures.

• We adopt a multi-objective optimization (MOO) approach for SLA-driven planning and use a state-of-the-art MOO algorithm, called SMS-EMOA [6], for its implementation. A multi-objective approach enables the planner to evaluate design tradeoffs and balance performance and costs from the service provider perspective. It also provides a systematic way to optimally specify service level objectives (SLOs), and translate such objectives into both software and system level parameters.

The rest of the paper is organized as follows. Section 2 describes the queueing network model of the SAP ERP application together with validation results proving its accuracy. Section 3 develops the cost model based on publicly available benchmark results and real measurement data of enterprise systems. CPU costs and power consumption are modeled separately and later included into a comprehensive cost model. Section 4 presents a SLA-driven planning framework that builds on top of the SMS-EMOA multi-objective optimization algorithm. The concept of Pareto front and MOO are introduced, and the SMS-EMOA algorithm is described including its parallel implementation used in experiments. Section 5 presents the experimental results of applying multi-objective optimization in SLA-driven capacity planning; the computational efficiency of SMS-EMOA is discussed, and the use of Pareto-optimal set in decision making is illustrated. Section 6 overviews related work. Finally, Section 7 gives conclusions and outlines future work.

2. THE PERFORMANCE MODEL

We use SAP ERP as a case study for our SLA-driven capacity planning and optimization methodology [23]. ERP software applications are challenging to provision, therefore they represent a difficult test case for capacity planning methodologies. Complexity stems from the heterogeneity of ERP workloads, which encompass thousands of transaction types associated to different business areas (e..g, sales, distribution, financial, supply chain management), and from the characteristics of the software architecture, which is much more complex than that of simple web systems considered in the performance evaluation literature for benchmarking and modeling exercises. ERP transactions are processed by different software subsystems running on top of a middleware or a software integration platform, such as SAP NetWeaver [23]. This integration platform needs to be accounted explicitly in the performance model, in addition to the characteristics of the underlying hardware resources, to achieve good performance predictions. This requires a different from hardware resource consumption modeling, which does not explicitly account for the software architecture characteristics and yet is sufficient to achieve very good system modeling predictions on web systems [25, 15]. Workload complexity is usually tackled in ERP application sizing by considering simplified transaction mixes that stress specific business functions known to be representative of system usage for a given customer. Throughout the experiments reported in the paper, we have used a workload composed by sales and distribution transactions, including order creations, order listing, and delivery decisions[1]. The sequence of transactions submitted to the ERP system is identical for all clients and repeated cyclically 20 times, which corresponds to an experiment duration of about 1 hour excluding initial and final transients. Requests are sent to the system by a closed-loop workload generator which issues a new request after completion of the previous one and following an exponential think time with mean $Z = 10s$.

The goal of this section is to describe the general architecture of the SAP ERP application (Section 2.1) and outline the proposed modeling approach based on queueing networks with finite capacity regions [7, 18] (Section 2.2). We also discuss experimental results on the real-system proving that our model is in good agreement with observed system performance (Section 2.3).

2.1 Architecture

We provide an high-level overview of the architecture of the SAP ERP system, the interested reader can found additional information in [23]. A basic ERP installation is composed by an application server and a database server. Clients interact with the ERP system through a graphical user interface (GUI) on client-side which exchanges data with the application server through a proprietary communication protocol. The interaction model is stateful and workflow-oriented: to complete a complex function, such as a delivery, the GUI guides the user through an ordered sequence of dialog windows. These windows show data that is dynamically retrieved from the ERP system in response to atomic client-initiated requests called *dialog steps*. These dialog steps are also used to send data updates to the ERP system. Throughout the rest of the paper, we always refer to the dialog step as the basic element of computation of the SAP ERP system (i.e., atomic request) and we give response time and performance index estimates on a per-dialog-step basis.

2.1.1 Dialog Step Response Time Components

Table 1 lists the components of the response time of a dialog step; qualitative description is given below.

Wait time (R_{wait}). Upon arrival to the ERP system, a

[1]Due to the non-disclosure agreements that are in place, we cannot provide in the paper additional information on the detailed characteristics of the transactions of the sales and distribution workload used in the experiment. However, we stress that the workload mix used is strongly representative of typical SAP ERP usage profiles.

R_{wait}	Dispatcher waiting queue latency
Z_{lgr}	Load, generation, and roll-in time
R_{wp}	Non-idle time spent in work processes
R_{db}	Data provisioning time

Table 1: Components of response time in SAP ERP

dialog step first joins an admission control queue. The dispatcher forwards requests from this waiting buffer to server threads with processing capabilities, called *work processes* (WPs). Admission takes place when there is an idle WP and according to a first-come first-served (FCFS) scheduling rule. WPs run as independent operating system processes that share a memory area managed by the application server; this area stores table buffers and client session information. Service into a WP is offered in a non-preemptive manner, thus a dialog step that starts execution in a WP does not leave until completion of its activity cycle and it is always served by the same WP. As a result of non-preemptive scheduling, wait time tends to become the dominating component of the end-to-end response time as the number of active users is large with respect to capacity.

Load, generation, and roll-in times (Z_{lgr}). Upon admission of a dialog step into a WP, the application server builds and stores in memory user context, object code, and data required for executing the dialog step transaction. We denote by Z_{lgr} the sum of all latencies related to these initialization activities. Since these overheads are mostly due to memory-bound operations, an increase of the number of cores and of the number of WPs (*software threading level*) does not significantly affect Z_{lgr}, which may therefore be seen as a constant delay suffered on the end-to-end path of dialog steps as opposed to the other components of the response time that increase with the load.

Time in work process (R_{wp}). This is the response time component due to computations performed within a WP. Most ERP transactions break down into a cycle of CPU processing phases followed by synchronous calls to the database to provision new data for the upcoming computations. During this period, the WP remains idle waiting for new data. Following these observations, we represent with the R_{wp} term the time spent by the dialog step in the WP when this is *not* blocked waiting for data. That is, in presence of synchronous calls to the database, we assume that the related time spent idle waiting for DB response is *not* included in R_{wp}. Summarizing, the R_{wp} term captures CPU-bound activities and hence is significantly affected by changes in the number of cores or in the software threading level.

Database response time (R_{db}). This is the cumulative response time due to the provisioning of data from the database server. We ignore the load placed on the database by background operations for two reasons: first, these are executed with lower priority than dialog step operations, hence they do not impact on dialog step end-to-end response times; additionally, in our experiments they are responsible of very small utilization (DB utilization is about 3% for a system at 90% utilization). This make them also negligible for power consumption prediction.

2.2 SAP ERP Performance Model

We now define a performance model with the aim of predicting end-to-end response times of dialog steps under different hardware and software configurations. To this end, we need to have in the model as explicit input parameters both the number of CPUs K (or vCPUs in virtualized environments) and the software threading level, as specified by the number of work processes W used in the ERP configuration.

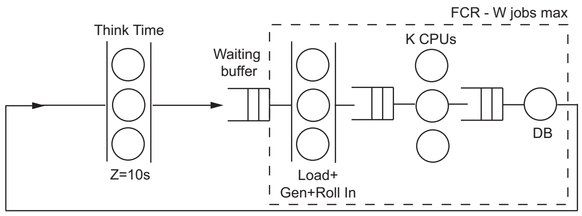


Figure 1: FCR queueing model of SAP ERP

2.2.1 Models with Finite Capacity Regions

The starting point of our modeling analysis is that jointly accounting for the software threading level W and for the number of CPUs K using a product-form queueing network model [18, 20] is a difficult task. Product-form networks are standard capacity planning model enjoying efficient solutions algorithms such as Mean Value Analysis (MVA) [21]. Although product-form models support the definition of multi-server queues with exponential service times through the load-dependent formalism [18], they cannot account for two simultaneous constraints affecting the processing activity of a same resource. In our case, they could not represent the joint software and hardware parallelism constraints on the CPUs given by W and K. A number of modeling techniques exist that are capable of overcoming these issues by explicit representation of constraints in resource usage, noticeably Layered Queueing Networks [22, 13] (LQNs) and Queueing Petri Nets (QPNs) [16] are two popular formalisms for these constraints. In this work, we focus instead on the simpler formalism of queueing networks with finite capacity regions (FCR queuing networks) [7, 18, 17, 2, 5]. These are models that are similar to ordinary product-form networks, but that can also place constraints on the maximum number of jobs circulating in a subnetwork of queues called the finite capacity region. FCR queueing networks enjoy several approximation schemes that make them appealing for analytical evaluation [18, 2]. Indeed, FCR queueing networks may be seen as basic specializations of LQNs or QPNs, therefore we stress that they are not more expressive than these formalisms. Nevertheless, our preference for FCR queueing models is motivated by the fact that they are the simplest class of models that offers the features we need to describe performance scalability as a function of the software threading level W and for the number of CPUs K.

2.2.2 SAP ERP Model

Starting from the above discussion, we have defined a performance model of the SAP ERP application using the FCR queueing network model shown in Figure 1. The model features a FCR that represents the dispatcher admission control policy by imposing a limit of W requests circulating in the stations inside the FCR: thus, each request represents a dialog step being admitted into a work process. The FCR models the software threading limit imposed by the number of work processes W specified in the ERP configuration.

The arrival process to the FCR waiting buffer is instead controlled by the workload generator. This is modeled in Figure 1 as a delay station ($-/M/\infty$ queue) with exponential think times having mean $Z = 10s$. Within the FCR, the load, generation and roll-in times are modeled as a passage through a delay station with mean service time equal to $Z_{lgr} = 0.015s$ as estimated from measurement. Note that this modeling decision makes the number of CPUs K uninfluential on the Z_{lgr} term; this is desired because Z_{lgr} is a memory-bound latency term. The remaining stations within the FCR are a multiserver $-/M/K$ queue representing the WP usage of the K CPUs and a $-/M/1$ queue modeling software contention at the database server. This is an accurate modeling abstraction whenever application server and database run on separate tiers. However, as we explain in the next subsection, we have found this to be a very accurate approximation also in the case of a two tier deployment where application server and database share the same CPUs. This can be explained by the fact that both work processes and database spend a part of their service time in I/O-bound and memory-bound operations that allow other processes to gain access to the CPU in the meanwhile. This makes the parallelism of the ERP system effectively greater than the number of processors K. Rather than explicitly modeling I/O and memory effects, which would introduce additional complexity in model parameterization and approximation, we prefer to decouple the database as a separate queue since this is the resource with most frequent access to I/O and storage resources and thus contention at this server is often not due to limited CPU capacity. Validation experiments reported in the next subsection indicate that the proposed model provides an accurate abstraction of the system performance under changes of the number of CPUs and WPs.

2.3 Model Validation

The validation of the performance model proposed in the previous subsection has been done using an installation of SAP ERP on a two-tier configuration. The ERP system runs in a virtual machine under VMware ESX server; this virtual machine is configured with 32GB of memory, 230GB of provisioned storage space, and $K \in \{1, 2, 4\}$ vCPUs each running with 2.2GHz frequency. The underlying hardware supports each vCPU with a separate physical processor. We have been running a sales and distribution workload on the ERP system configured with $W \in \{1, 2, 4, 8, 16, 32\}$ work processes. For given choice of number of vCPUs K and work processes W, we have first performed a dummy experiment with $N = 1$ to load system caches thus accounting for system reboot after the configuration change, followed by experiments for $N = 300$ users; all think times are exponential with mean $Z = 10s$. The value $N = 300$ is chosen because under this load the system reaches heavy-usage conditions where the number of clients requesting simultaneously dialog step processing is greater than both the number of processor and work processes. Therefore, this heavy-load experiment demonstrates the effects of both the K and W parameters on the scalability of the ERP application.

Model estimates of the application performance are obtained as follows. The service demand parameters used in the definition of the FCR queueing models are $D_{cpu} = 0.072s$ for the CPU service demand, $Z_{lgr} = 0.015s$ and $D_{db} = 0.032s$ for the database demand. These values are obtained by measurement of the hardware infrastructure and from the internal performance monitor of the SAP ERP system available via the STAD transaction [23]. To keep the model and its parameterization simple, we assume throughout this paper that service demands are independent of the load. The FCR model has been here solved by simulation using the Java Modeling Tools suite [5] that supports the analysis of FCR queueing networks: response times are immediately computed by the tool; conversely, we approximate the utilization of the CPUs by the utilization U_{cpu} of the multi-server station only, which is consistent with our observation that the DB server has low impact on the utilization (3% at 90% utilization) and that performance degradation at the DB is due to I/O and memory overheads.

K	W	N	R (Model)	R (Meas.)	U (Model)	U (Meas.)
2	1	300	25.86	28.76	0.30	0.51
2	2	300	9.15	9.32	0.56	0.65
2	4	300	3.57	7.62	0.79	0.66
2	8	300	1.76	2.41	0.92	0.86
2	16	300	1.11	1.37	0.97	0.97
2	32	300	1.05	0.91	0.98	0.99
4	1	300	25.68	26.8	0.15	0.28
4	2	300	9.19	10.6	0.28	0.37
4	4	300	1.61	1.67	0.46	0.51
4	8	300	0.47	1.16	0.52	0.65
4	16	300	0.41	0.43	0.53	0.63
4	32	300	0.37	0.38	0.53	0.62

Table 2: FCR queueing model validation results against measurement. Legend: R is response time; U is utilization; N is number of users; K number of processors; W is software threading level.

Table 2 shows validation results for the proposed model as compared to measurement of 12 experiments on the real ERP application. As we can see, the model produces response time estimates that are in good agreement with experimental data. In particular, for low and high number of WPs the results are tight to the measured ones and make a case for the effectiveness of the proposed model in capturing the salient effects of W and K on system performance. Some deviations as soon as W exceed K, where for both cases with two and four CPUs the results are optimistic estimates on the measured performance. Two considerations are possible regarding these configurations: first, these are system configurations where the software threading level starts exceeding the number of processors in the system, thus it is problematic to predict if the bottleneck is due to hardware or software components and thus performance prediction is clearly harder than in the other cases. A second remark is that, despite some optimistic results, the overall trend of the performance model reflects quite well the one measured on the real system. This is important because real-world systems deployed under SLA constraints systematically over-provision capacity with the aim of avoiding SLA penalties. In this context, we believe that the general performance trend and the order of magnitude of an index is more important that highly-accurate point estimates in all evaluations, since eventually an over-provisioning gap will be anyway applied to the recommended solution. As such, configurations where W is close to K can be expected to be more critical and should either discarded or would require

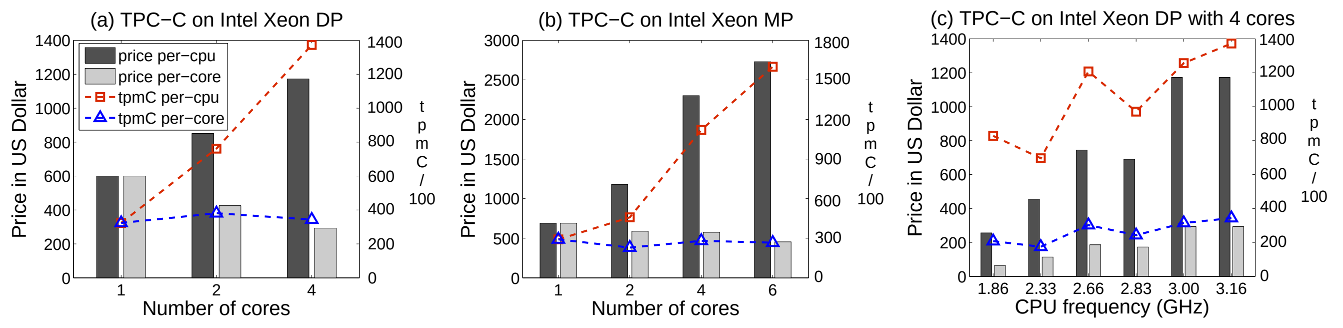


Figure 2: 117 certified TPC-C benchmark results [24] run on Intel Xeon DP/MP platforms within the timeframe between 7/2002 and 12/2008. TPC-C is measured in transactions per minute (tpmC).

stronger over-provisioning gaps for safety margin.

Concerning utilization estimates, the model appears quite robust on all experiments with an error that is typically less than 15%. In particular, the results in heavy load appear tighter to the measured values. Since utilization is responsible for our estimates of the power consumption, these result indicate that our performance model is expected to have good prediction accuracy on system energy consumption.

Summarizing, the results proposed in this section illustrate that our FCR queueing network model is successful in capturing the scalability of the ERP under different number of processors K and work process configurations W. Only configurations around the critical point $K = W$ are harder for response time prediction; utilization estimates are generally good in all cases.

3. THE COST MODEL

After developing a performance model for the enterprise applications, we focus on the cost factors and economic impacts of hosting such applications. For the service providers to specify SLAs and optimize their service/infrastructure landscapes, it is important to analyze, understand, and model the cost components within the so-called "Total Cost of Ownership" (TCO). TCO is intrinsically complex and involves a great number of tangible/intangible factors. As is pointed out in [3], the TCO of a large-scale hosting center can be broken down into four main components: hardware, power (recurring and initial data-center investment), recurring data-center operations costs, and cost of the software. Normally the operations costs (incl. human capital/consulting) and software constitute a large percentage of TCO for commercial deployment, however, it is very difficult to develop a generic quantitative cost model for these components. In this paper we focus on more tangible cost factors such as server hardware, and we incorporate power consumption into the cost model as a server's energy footprint becomes an increasingly important cost factor in large-scale hosting environments.

Not aiming at a comprehensive TCO model, this paper focuses on the quantitative aspects and develop an analytic cost model that consists of two tangible cost components: server hardware and power consumption. Firstly, a pricing model for CPU is proposed as a function of per-core performance and the number of cores. The per-core performance is based on the published results of industry-standard OLTP (online transaction processing) benchmark TPC-C [24] on Intel DP/MP platforms[2]. The fitted CPU pricing model also manifests the current multi-/many-core trend. Secondly, server power consumption is modeled as a function of CPU utilization using a customized Power function. By combining the fitted models for both server costs and power consumption, we develop a simplified analytic model that can be used in the studies of optimizing the enterprise system landscape with multiple objectives.

3.1 Modeling CPU Costs with Multi-Core

Among the many components of server hardware, namely CPU, memory, storage, and network, we focus on the CPU costs in this paper and make simplified assumptions that costs of other components remain constants or scale with the CPU costs. We are particularly interested in the price-performance relationship on multi-/many-core platforms, as the general trend in processor development has been from single-, multi-, to many cores. Our goal is to investigate and model the relationship between the objective, namely the price per-CPU (C_{cpu}) or price per-core (C_{core}), and the two related parameters: number of cores (N_{core}) and benchmark results per-core (T_{core}). T_{core} also corresponds to the processing speed of the core, and thus the resource demands of the measured OLTP applications.

We examine the certified TPC-C [24] benchmark results on Intel DP/MP platforms and associate them with CPU price information[3] [14], which are shown in Figure 2. As there are two independent parameters (N_{core} and T_{core}) we study one of them by fixing the value of the other, and vice versa.

3.1.1 *Price, Performance, and Number of Cores*

Firstly let us look at the price versus the number of cores given a similar per-core performance. In 2(a), we can see

[2]The sales and distribution workloads used for performance model evaluation are also studied and the results are not published here due to the non-disclosure agreements in place. However, we stress that the TPC-C results are representative for data fitting purposes.

[3]Disclaimer: The performance is measured in tpmC (transactions per minute), which is defined as how many New-Order transactions per minute a system generates while executing other transactions types. Such a performance measure is influenced by CPU performance and additional factors like machine architecture, cache sizes, memory size/latency/bandwidth, operating system, storage system characteristics, DBMS, TPC-C version/settings as well as other factors not mentioned here.

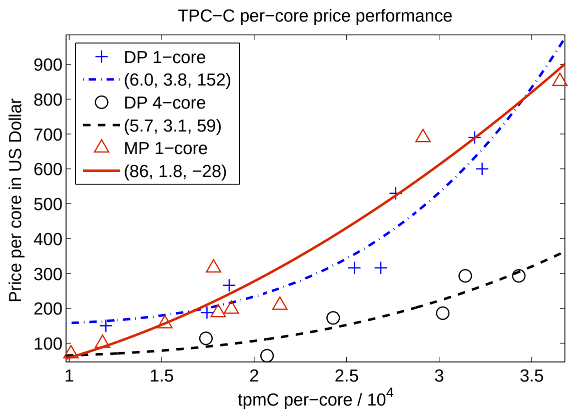


Figure 3: Power function fitting of different TPC-C data. Parameters are (c_1, c_2, c_3) in (1).

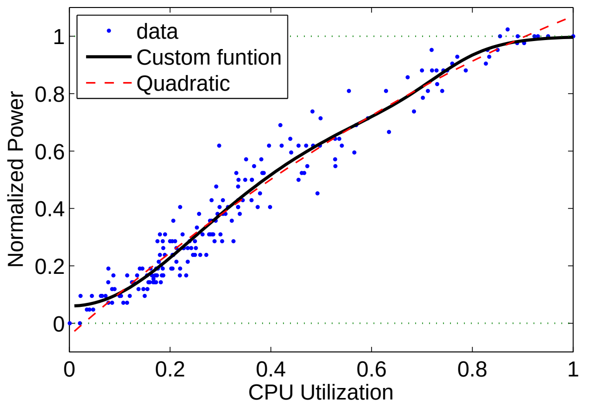


Figure 4: Normalized power vs CPU utilization.

model param.	c_1	c_2	c_3	c_4	c_5
TPCC/DP	36	2.0	261	-0.9	-105

Table 3: CPU cost model parameters for TPC-C benchmarks on Intel Xeon DP (3).

model param.	c_1	c_2	c_3	c_4	c_5
business app.	276.7	15	7	2.1	1.1

Table 4: Power consumption model parameters for a customized business workload (5).

that the per-core price decreases as the number of cores per CPU increases on the Intel Xeon DP platform. As the per-core performance of TPC-C remains the same, the price/performance ratio improves by adding more cores. Generally this trend is also observed for TPC-C on Intel MP, as is shown in Figure 2(b). We notice that the per-core tpmC decreases slightly as the number of cores increases. This is because that the core frequency scales down as the number of cores scales up. Nevertheless, as the chip design becomes more efficient, the per-core performance/frequency ratio improves with the evolution of CPU generations.

Secondly let us examine the price versus the per-core performance given the same number of cores. In Figure 2(c), as predicted, we can see that the price increases as the CPU frequency and throughput numbers increase. Some abnormal behavior happens between 2.33 GHz and 2.83 GHz. This may be explained partially by the noise in the data as there is only one available measurement each for CPU frequency at 2.33 GHz and 2.83 GHz. Nevertheless, the general trend of price increasing with speed (core frequency) still holds. Figure 3 gives a better view on the pattern of how price changes with the per-core performance for TPC-C. On both DP and MP platforms with different cores, the per-core price scales with the per-core throughput like a power function. We studied different functions for curve fitting, including polynomial, exponential, power, and other custom functions. It is found that the power function, shown in (1), gives the overall best fit for different data sets.

$$f(x) = c_1 x^{c_2} + c_3 \tag{1}$$

It is also shown that the price per-core decreases like a power function while increasing the number of cores per-CPU. This indicates that the power function in (1) can be used to model the relationships between price per-core and T_{core} or N_{core} individually.

3.1.2 A CPU Price Model

The next step is to study per-core performance (T_{core}) and number of cores (N_{core}) jointly and model their relationship with price. Since the power function is the best fitted model for T_{core} and N_{core} individually, we can extend this model to a multi-variable case[4]. A power function with two variables can be formulated as follows:

$$C_{core} = g(T_{core}, N_{core}) = c_1 T_{core}^{c_2} + c_3 N_{core}^{c_4} + c_5, \tag{2}$$

where $(c_1, ..., c_5)$ are the parameters to be fitted. The price per-CPU C_{cpu} is readily obtained by multiplying price per-core with the number of cores:

$$C_{cpu} = N_{core} C_{core} = N_{core} g(T_{core}, N_{core}). \tag{3}$$

A non-linear least-squares method in the Matlab Optimization toolbox (lsqcurvefit) is used for curve fitting, and the fitted parameters are shown in Table 3. The fitted model gives an overall good interpolation of real benchmark results. Although different benchmarks on different platforms may yield different parameters[5], the model shown in (3) is general and flexible enough for estimating a wide range of CPU cost information.

It should be noted that the power-function based model for CPU costs developed in this section depends on the Intel pricing schemes for its multi-/many-core platforms. Our contribution is to fit such price information with mathematical models, in relationship to real OLTP benchmark results. This gives the planners/architects at the provider side a convenient tool for estimating hardware costs given the desired performance level of their applications.

[4]An informal proof for this extension can be described as follows: When x or y is constant, either $f(x)$ or $f(y)$ takes the form $ax^b + c$. This means there is no x or y components of any form in the function other than x^b or y^d. So $f(x, y)$ can be written as $ax^b + cy^d + e$.

[5]There are no sufficient data for curve fitting of TPC-C benchmark on Intel MP platform.

3.2 Modeling Power Consumption

Power consumption and associated costs become increasingly significant in modern datacenter environments [12]. In this section we analyze and model the server power consumption of business applications. We study the relationship between system power consumption (P_{sys}, measured in Watts) and CPU utilization (U), which is used as the main metric for system-level activity. We run a customized application similar to sales and distribution business processes (the same workload used in Section 2.3), on a 64-bit Linux server with 1 Intel dual-core CPU and 4 GB main memory. The system power is measured using a power meter connected between the server power plug and the wall socket. The CPU utilization data is collected using Linux utilities such as *sar* and *iostat*. Monitoring scripts in SAP performance tools are also used for correlating power and CPU utilization data.

Before data fitting and modeling we first perform a data pre-processing step called *normalization*. Instead of directly modeling P_{sys} we use a normalized power unit P_{norm}, which is defined as follows:

$$P_{norm} = \frac{P_{sys} - P_{idle}}{P_{busy} - P_{idle}}, \tag{4}$$

where the measured P_{idle} ($U = 0$) and P_{busy} ($U = 1$) for our test system are 42W and 84W, respectively. Different systems may have different idle and peak power consumptions. The normalized measurement results are shown in Figure 4.

Generally speaking the server power consumption increases as the CPU utilization grows. One important finding from the measurement data is the so-called power capping behavior [12], which means there are only a few times that the highest power consumption is reached by the server. Additionally we find that such highest power points are drawn mostly when the CPU utilization is higher than 80% and they have very similar peak values. Most of the functions, such as quadratic polynomial, power, exponential, and Gaussian, cannot fit such flat curve of power values in the high-utilization interval (see the quadratic fitting in Figure 4).

We developed a model that can fit such power-capping behavior well. The model is inspired by the frequency response curve of a linear filter called Butterworth filter [28]. It has such desired "flat" behavior in the passband of the frequency. We replace the polynomial part of the transfer function with the following customized power function with two U components:

$$h(U) = c_1U^{c_2} + c_3U^{c_4} + c_5, \tag{5}$$

where $(c_1, ..., c_5)$ are the parameters to be fitted. The model that relates normalized power (P_{norm}) and CPU utilization U can be formulated as follows:

$$P_{norm}(U) = 1 - h(U)^{-1}. \tag{6}$$

The fitting result is shown in Figure 4 and the fitted model parameters are listed in Table 4. We can see that the proposed power model fits the measurement data well, especially during the high utilization period. Given the measurements for P_{idle} and P_{busy}, the overall system power consumption P_{sys} can be obtained by substituting P_{norm} (6) in (4).

3.3 A Cost Model for Enterprise Applications

By combining the cost models for CPU and power consumption in previous sections (equations (3), (4), and (6)), we developed a cost model for business applications:

$$Cost(T_{core}, N_{core}, U, I) = \tag{7}$$
$$p_0 + p_1C_{cpu} + p_2 \int_{t\in I} P_{sys}(U(t))dt,$$

where t is the measurement time, I is the measurement period ($t \in I$), p_0 is an adjusting constant, p_1, and p_2 are the weighting parameters that scale the individual model outputs. If during the measurement period only average utilization is available, the output can be written as $P_{sys}(\overline{U})I$. The model in (7) uses an additive form to combine server hardware costs and operational costs, in which parameters p_1 and p_2 have to be set properly to reflect different cost structures.

To summarize from a mathematical modeling perspective, we can conclude that the power function ($c_1x^{c_2} + c_3$) and its variants have attractive properties for fitting a wide range of curves, including both single- and multi-variable case. Thus, the power function family represents a general and flexible modeling library from which different cost models can be fitted and derived.

In practice when using the cost model for the optimization of enterprise systems, we need to determine the weighting parameters p_1 (fixed cost) and p_2 (operational cost). These parameters are chosen in a way to reflect the real numbers obtained in case studies in [4]. There are two situations under study in this paper. On one hand, for a typical "classical" data center the ratio of fixed cost versus operational cost (r) is set to $7:3$, which indicates that the high server capital costs dominate overall TCO by 70%. For a modern commodity-based data center, on the other hand, the ratio r is set to $3:7$. This means operational costs including power consumption and cooling become the dominating factor. The cost model outputs of (7) for these two situations are illustrated in Figure 5, where differences can be clearly identified. For instance, the total cost increases significantly with the increasing system utilization for the high operational cost situation ($r = 3 : 7$), which is not the case for the high fixed cost counterpart($r = 7 : 3$). We also observe that the discontinuity of cost model outputs along the performance/core axis in the $r = 3 : 7$ situation. This is because the settings of P_{idle} and P_{busy} take discrete values like a piecewise constant function. The CPU performance per core is divided into three ranges and the values of P_{idle} and P_{busy} are set accordingly. For instance, for a 2-core system from low to high performance, P_{idle} and P_{busy} have been set to $[40, 60, 80]$ and $[65, 95, 150]$, respectively. Such settings are made in accordance to the CPU power consumption characteristics on Intel platforms. In the $r = 7 : 3$ situation, however, such effects is dramatically reduced as the operational cost is no longer dominant. We investigate both situations in the optimization phase to see how different cost structures impact the planning results.

4. MULTI-OBJECTIVE OPTIMIZATION

We adopt a multi-objective approach towards SLA-driven planning of enterprise applications. A framework is introduced for formulating the problem with multiple objectives and describing the design paradigm. What lies in the core of the framework is a multi-objective optimizer, and we apply

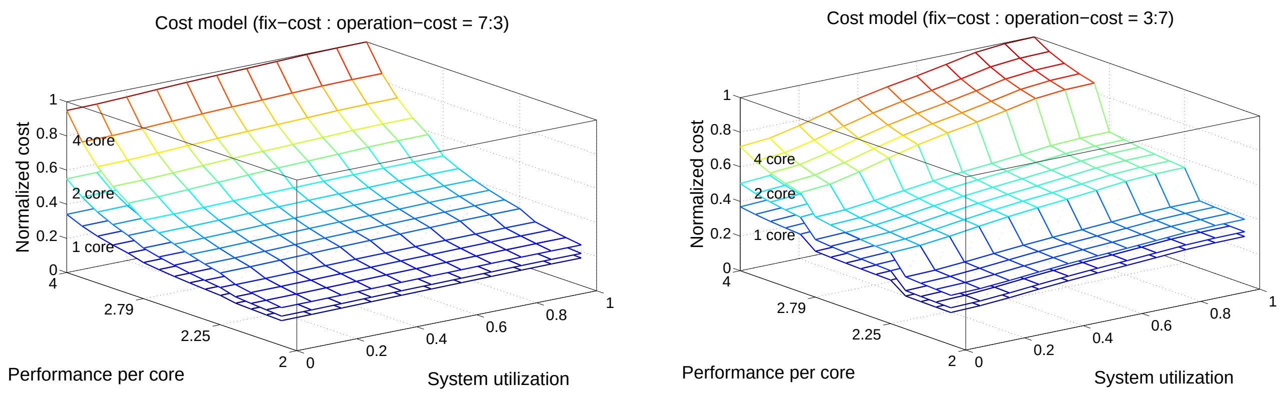


Figure 5: Cost model structures: For a typical "classical" data center, the ratio of fixed cost versus operational cost (r) is set to $7:3$. For a modern commodity-based data center, the ratio r is set to $3:7$.

a state-of-the-art evolutionary multi-objective optimization (MOO) algorithm. We show how the performance and cost models can be used in an optimization process of the planning phase.

4.1 A SLA-Driven Planning Framework

Firstly we present a framework for SLA-driven planning and optimization, which is shown in Figure 6. The system planner interacts with the planning tool via a dashboard-based User Interface (UI). The planner starts with defining the *objectives*, namely, system end-to-end *response time* and *infrastructure cost*. In this case the problem is formulated as a minimization problem: minimizing both response time and cost. The planner then follows several main steps in the planning phase:

1. Define default *constraints* or extract them from the customer SLAs. Such constraints are considered as fixed constants in the optimization process, and they are mostly related to the user workloads. For a closed queueing network model used in this paper, the constraints of interest are *number of users* and *think time*.

2. Define *parameters* to be optimized. In the context of this paper most of the parameters are configuration parameters in the enterprise system landscape. These include hardware resource specifications, namely, *Resource Demand (D)* and *number of cores* K. It also includes application server configurations such as W, *number of WPs* (dialog work processes).

3. Formulate the problem for optimization. The performance and cost models developed in previous sections can take configuration parameters as inputs and generate/predict performance and cost outputs. The utility functions scale the model outputs as utilities for a unified representation of objective values. The decoder, on the contrary, maps the encoded parameters into model-specific formats.

4. Run the optimization and interpret the results. With the set of "optimal" trade-off solutions obtained via optimization, the planner can make educated decisions for planning the system landscape according to different levels of SLAs.

The central component of the framework is an evolutionary MOO algorithm called SMS-EMOA, which will be elaborated in the next section. Here we give more explanations on

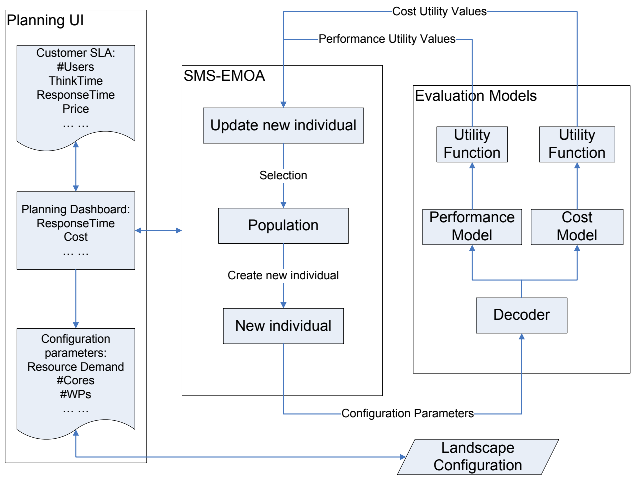


Figure 6: A SLA-driven planning framework.

utility and encoding/decoding functions. Firstly, for scaling the diverse objective values into unified utilities (e.g. $[0,1]$), we adopt Derringer's individual desirability function [11]. In case of a minimization problem, to which our problem belongs, the desirability value is increasing along with the value of the objective, bounded by a maximum value. For the sake of simplicity linear scaling is used in practice. Secondly, like other evolutionary algorithms the configuration parameters is encoded in the individuals as continuous double values. The number of WPs is discretized by rounding up to the closest small integer. The number of cores is encoded as a double variable $x \in (0,3)$, and is decoded by $2^{floor(x)}$ (1, 2, or 4 cores).

4.2 A Multi-Objective Optimizer

In the introduction of the planning framework the MOO algorithm is treated as a black-box: iteratively evaluate the objective values, generate new parameters, and hopefully after some generations (sub)optimal solutions could be found. In this section we explain the rationale behind a true multi-objective optimization and describe how a state-of-the-art evolutionary MOO algorithm works.

Multi-objective optimization (MOO) is the process of simultaneously optimizing two or more objectives. Most problems in nature have several, possibly conflicting, objectives. In the context of this paper, for instance, we are aiming at maximizing the system performance at the same time minimizing the infrastructure cost. On one hand, common ways of dealing with MOO problems include treating them as single-objective by turning all but one objective into constraints, or combining multiple objectives into one. A MOO algorithm, on the other hand, tries to find good compromises (or trade-offs) rather than a single global optimum. Therefore the notion of "optimum" in multi-objective optimization changes accordingly, and the most commonly accepted term is called *Pareto optimum* [10].

The concept of Pareto optimum and Pareto front are explained as follows. Given a parameter vector $X \in \mathbb{R}^n$, an evaluation function $\mathbf{f} : X \to Y$ evaluates the quality of the solution by mapping the parameter vector to an objective vector $Y \in \mathbb{R}^m$. The comparison of two parameter vectors $\mathbf{x}$ and $\mathbf{x}'$ follows the well-known concept of *Pareto dominance*. We say that an objective vector $\mathbf{y}$ *dominates* $\mathbf{y}'$ (in symbols $\mathbf{y} \prec \mathbf{y}'$), if and only if $\forall i \in \{1, \dots, m\}$: $y_i \leq y_i'$ and $\mathbf{y} \neq \mathbf{y}'$. The set of non-dominated solutions of a set $Y \subseteq \mathbb{R}^m$ is defined as: $Y_N = \{\mathbf{y} \in Y | \nexists \mathbf{y}' \in Y : \mathbf{y}' \prec \mathbf{y}\}$. Given a multi-objective optimization (minimization) problem

$$f_1(\mathbf{x}) \to \min, \dots, f_m(\mathbf{x}) \to \min, \mathbf{x} \in X \subset \mathbb{R}^m, \qquad (8)$$

the image set $Y(S)$ of this problem is defined as $\{\mathbf{y} \in \mathbb{R}^m | \exists \mathbf{x} \in X : f_1(\mathbf{x}) = y_1, \dots, f_m(\mathbf{x}) = y_m\}$. The non-dominated set of $Y(X)$ is called *Pareto front*. In other words, the Pareto front consists of a set of optimal solutions representing different trade-offs among the objectives. The knowledge of Pareto front helps the decision maker in selecting the best compromise solutions.

In order to approximate a continuous Pareto front that typically consists of infinitely many points, we can compute an approximation set that covers the Pareto front. In general, an *approximation set* is defined as a set of mutually non-dominated solutions in $Y(X)$. A common indicator for the quality of an approximation set, measuring how well it serves as a well-distributed and close approximation of the Pareto front, is the *hypervolume indicator* (or: S-Metric) [6]. The problem of finding a well distributed approximation of the Pareto front can be recasted as the problem of finding an approximation set that maximizes the S-Metric.

Evolutionary algorithms possess several characteristics that are naturally desirable as the search strategies for multi-objective optimization [10]. Among other indicator-based MOO algorithms, the S-Metric Selection Evolutionary Multi-objective Optimization Algorithm (SMS-EMOA) approximates such S-Metric maximal approximation sets. The SMS-EMOA algorithm implements a steady-state $(\mu + 1)$ evolutionary strategy: keep a population of μ individuals, remove one "bad" individual and add a new one in each generation. SMS-EMOA can also be parallelized by distributing function evaluations to different processors. We follow the algorithmic details for the hypervolume computation and variation operators as described in [6], and integrated both sequential and parallel SMS-EMOA implementation in SLA-driven planning.

5. EMPIRICAL EVALUATION

In previous sections, we have shown the experimental results for performance model validation and cost model derivation. In this section we present the simulation results of applying multi-objective optimization to SLA-driven planning.

We implemented the performance model using the Java Modeling Tools (JMT) mentioned in Section 2.3. The cost models and utility functions are implemented in Java. These models evaluate the input parameters and feed objective values into the SMS-EMOA optimization engine (implemented in C++). On a 3 GHz dual-core Intel machine, a single evaluation of our simulation-based performance model takes around $T_{perf} = 8$ seconds. For the sequential version of the optimization engine, the total run time is proportional to the number of generations N_{gen} ($T_{total} = T_{perf} N_{gen}$). The parallel version of optimization scales sub-linearly with the number of processors, especially for relatively long model evaluations. In the case 1000 generations it takes approximately 25 minutes on 4 processors.

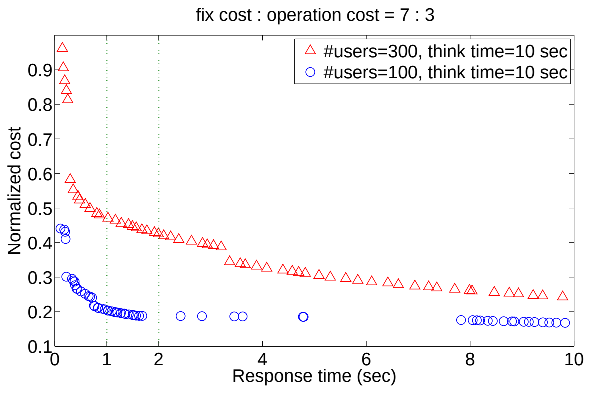
5.1 SLA-Driven Planning

We conducted experiments to validate the MOO-based approach and illustrate how to use the Pareto optimal solutions (Pareto front) for SLA-driven system planning. The questions of interests are listed and addressed as follows.

5.1.1 *How does the proposed multi-objective optimization approach work in practice?*

Figure 7 shows the Pareto fronts attained by the SMS-EMOA algorithm for different cost structures. Overall we can see that the algorithm is able to find a well-balanced approximation set for the Pareto front. For $r = 7 : 3$ case (high fixed cost) with 100 users, the relatively large gap between response time R of 2 and 8 seconds is because that most of the points in this region are dominated by the left-most point. It means that most of the interesting decision points reside in the left of $R = 2$ seconds, in a relatively light-loaded situation ($users = 100$). When there are more users in the system ($users = 300$), the decision points spread out and cover the whole front evenly. Figure 8 further shows the full history of an MOO run with 1000 generations for the $r = 7 : 3$ case, with both dominant and non-dominant solutions. By comparing the Pareto front and the full history in this case, it is reasonable to believe that the algorithm is able to approximate closely with the true Pareto front, and the final set of points are the optimal trade-off solutions. Another attractive property we observe is the so-called "kneeling" behavior. From the coverage of the design space in Figure 8, it shows that the heuristic search strategy of the algorithm is not only able to find a well-balanced solution set, but also able to concentrate solutions around the "knee" point. The knee point is located in the lower-left part of the Pareto front, which represents the most interesting trade-off solutions in the front. For $r = 3 : 7$ case (high operational cost), on the other side, we are also able to obtain a well-balanced Pareto front. However, the front is clearly divided into multiple, discrete sets of points. This is explainable by the cost model structure of the high operational cost case, as the power consumption has a piecewise-like pattern with respect to the processor speed (see Figure 5). It becomes particularly interesting to interpret such results together with the parameters, which are discussed in detail in the next section.

5.1.2 *How can the attained Pareto front and parameters be used for decision making?*



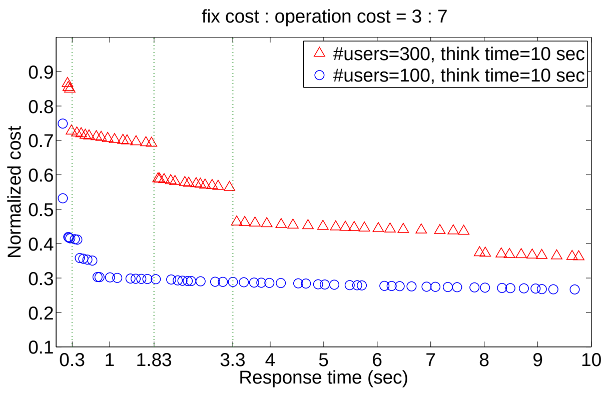


Figure 7: Pareto fronts attained by MOO optimization with different cost structures.

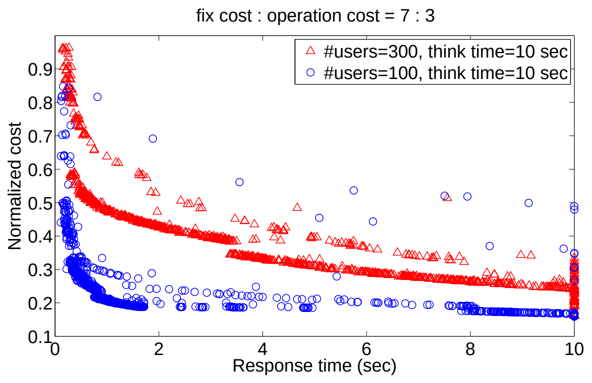


Figure 8: History of an MOO run showing both dominant and non-dominant solutions.

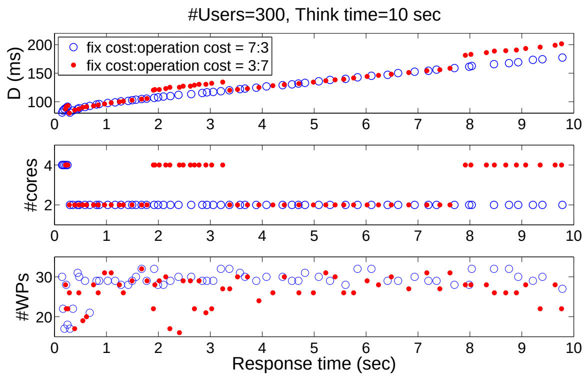


Figure 9: The solution space showing the Pareto-optimal configuration parameters.

Given a Pareto front, we now focus on how to use it for planning and decision making. A system planner wants to plan enterprise systems on the provider's landscape according to different customer SLAs, such as different workload constraints (number of users) and performance guarantees (response time R). The attained Pareto front enables the planner to evaluate and explore different trade-off solutions, and such solutions are the optimal compromises. For instance, in the $r = 7 : 3$ case with 300 users shown in Figure 7, guaranteeing $R = 1$ or $R = 2$ seconds have quite some differences in costs, so it makes sense to set different price categories as well (e.g. bronze or silver). For a light-load situation with 100 users, however, it is not necessary to consider $R = 2$ second as a threshold because the service provider can easily guarantee $R = 1$ second without much increase in infrastructure costs. If the customer wants even tighter bound on response time $R < 1$ second (e.g. gold), the pricing should increase dramatically as the cost goes up a lot. Using this example we can see that the interpretation of the Pareto front (values and shapes) can greatly assist in decision making in the SLA-driven planning process. One a decision is made, the corresponding parameters (number of cores, number of WPs, and resource demand D, shown in Figure 9) can be readily used for system configuration. Any decision taken from the front will be guaranteed to be a Pareto-optimal solution. It is up to the planner to assess the trade-offs for the service provider.

It becomes even more interesting when we examine the high operational cost case ($r = 3 : 7$). We can see that the Pareto fronts are divided into discrete sets of points. Within each set of points the costs remain relatively similar. Therefore the boundaries between sets represent the interesting points where decisions can be made. For instance, in the case of 300 users, $R = 0.3$, $R = 1.83$ and $R = 3.3$ seconds are appropriate threshold values for dividing the SLA offers with three levels of pricing and response time guarantees (e.g. gold, silver, and bronze). By cross-checking with the parameters in the solution space, we are able to find the corresponding patterns for configuring the system. For the gold customer ($R <= 0.3$ seconds), the planner needs to configure a “high-end dual-core” system ($D = 80$ms). For the silver customer ($R <= 1.83$ seconds), a “middle-end quad-core” system ($D = 120$ms) is need. And for the bronze customer ($R <= 3.3$ seconds) it requires a “middle-end dual-core” system ($D = 120$ms). Some margins will have to be included in the actual settings, but it gives the general idea of SLA-driven planning assisted by the Pareto front. For the application server we modeled, a setting of number of WPs> 20 (dialog work processes) should be good as the improvement in response time flattens out after the number of work processes exceeds a certain threshold (say 20).

To sum up, The results provided by multi-objective optimization can be easily interpreted and offer intuitions on important sizing questions such as how to map different SLA classes (e.g., gold, silver, bronze) into actual system configurations. The proposed model-driven methodology is also flexible and extensible: we can introduce additional parameters and objectives under different scenarios, which can be readily plugged into the optimization framework.

Our approach has its limitations as well. We validated the performance model and the cost model individually via benchmarking. Assuming that both models can accurately predict the targeted objectives, the simulation results show the effectiveness of applying multi-objective optimization to SLA-driven planning. In real-world deployment scenarios, nevertheless, variations can occur and the service level objectives (SLOs) have to be calibrated accordingly. Our performance model can predict the average system response times under exponentially-distributed workloads. Real systems, however, could exhibit different levels of burstiness and heavy-tail behavior. From a planning perspective some additional system capacity has to be reserved to anticipate unforeseen situations. We believe that good run-time techniques can complement design-time methodologies, for instance, by introducing penalties and adaptiveness [8].

6. RELATED WORK

We briefly discuss the related work on the topic of SLA-aware service and system planning. As is shown in Section 2 performance modeling has been extensively investigated for system capacity planning and lots of literature are available on this topic. Chen et al. [9] propose a multi-station queueing network model for multi-tier Web applications. By utilizing the performance model they further developed an approach to translate service level objectives such as response time into system-level parameters. Our work is different from theirs in the following aspects. Firstly, they deal with both open and closed workloads for Web applications, and developed an approximate MVA solver for handling multi-station queues and multi-class users. Our performance model, however, is developed for transactional business-critical applications. We developed a closed multi-station queueing network with finite capacity regions that is able to model additional factors (software threading level: number of WPs in the application server). Solving the model requires simulation in our approach, which is relatively slower than analytic solvers. Secondly, they apply a (single-objective) constraint satisfaction algorithm to derive system-level parameters from high-level constraints. We developed a cost model in addition and adopt a multi-objective approach for similar purposes.

Planning and optimization techniques have been studied for web services composition. Zeng et al. [29] present an approach for QoS-aware service selection for DAG-like (directed acyclic graph) service composition. Multiple QoS attributes are considered (e.g. price, latency, availability) as selection criteria and a simple additive weighting (SAW) method is used to add all weighted attributes into one ranking score. Another way of combining attributes is Derringer's desirability function [11]. It scales the individual attributes into desirabilities and combines them into one overall desirability using the geometric mean. For a similar web services composition problem, Wada et al. [26] apply an evolutionary multi-objective optimization technique for service selection. They focus on the service level and assume that objectives such as response time, throughput, and cost are readily measurable entities. Our solution differs from theirs in that we take a holistic approach and aim at mapping service-level objectives into system-level parameters. Therefore we have to explicitly model the system performance and cost. Our multi-objective algorithm SMS-EMOA also has an efficient parallel implementation. Multi-objective trade-off analysis and design space exploration have also been applied in other domains such as embedded systems and component-based software [27].

7. CONCLUSIONS AND FUTURE WORK

In this paper we developed a performance model for the ERP enterprise application and a cost model of hosting such applications. The performance model is a closed queueing network model with finite capacity regions (FCR), which is able to predict the SAP ERP application performance and is validated with empirical data. It is difficult in general for modeling the TCO of an application hosting provider. Our cost model is a simplified analytic model that quantifies the server hardware cost and power consumption as operational cost. The two cost factors can be scaled and combined to reflect different datacenter cost structures. We apply a multi-objective optimization technique to find the best tradeoff solutions (i.e. Pareto front) for the performance and cost goals. The attained Pareto front can be utilized to assist decision making of SLA-driven planning, and the corresponding parameters can be readily used for configuring the provider's system landscapes. With the assumption of realistic performance and cost models, the benefits of a multi-objective approach are clearly shown: instead of reaching a global optimal point, a set of best tradeoff solutions are provided to decision makers. They can explore different performance targets and the cost/pricing strategies, and the corresponding solutions are guaranteed to be the optimal compromises.

In future work we plan to solve FCR models with non-iterative approximations instead of simulations. We expect that it will greatly reduce the computational times of performance model evaluation, thus also the optimization program. We also plan to map the Resource Demand D into hardware configuration profiles, including both virtualized and non-virtualized environments. A prototype SLA-driven planning toolkit is being developed that implements the framework described in this paper.

Acknowledgments

The authors would like to thank Michael Emmerich and Alexander v.d. Kuijl for their discussions on multi-objective optimization topics. We also want to thank Daniel Scheibli for his contributions on power measurement in cost models. This research is funded partly via project SLA@SOI in the EU 7th framework programme, grant agreement no. FP7-216556 and by the InvestNI/SAP MORE project.

8. REFERENCES

[1] Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, 2009.

[2] J. Anselmi, G. Casale, and P. Cremonesi. Approximate solution of multiclass queuing networks with region constraints. In *Proc. of IEEE MASCOTS*, pages 225–230, 2007.

[3] L. Barroso. The price of performance: An economic case for chip multiprocessing. *ACM Queue*, 3(7):48–53, 2005.

[4] L. A. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[5] M. Bertoli, G. Casale, and G. Serazzi. Jmt: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[6] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 3:1653–1669, 2007.

[7] A. Brandwajn and W. M. McCormack. Efficient approximation for models of multiprogramming with shared domains. In *Proc. of ACM SIGMETRICS*, pages 186–194, 1984.

[8] J. S. Chase, D. C. Anderson, P. N. Thakar, A. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centres. In *Proc. of SOSP*, pages 103–116. ACM, 2001.

[9] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai. Translating service level objectives to lower level policies for multi-tier services. *Cluster Computing*, 11:299–311, 2008.

[10] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley Series in Systems and Optimization. John Wiley & Sons, 2001.

[11] G. Derringer and R. Suich. Simultaneous Optimization of Several Response Variables. *Journal of Quality Technology*, 12:214–219, 1980.

[12] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *proc. of the 34th Intl. Sym. on Computer Architecture*, pages 13–23. ACM Press, 2007.

[13] G. Franks, T. Omari, C. M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Trans. Software Engineering*, 35(2):148–161, 2009.

[14] Intel. Intel processor pricing, 2007-2009. Online available: `http://www.intc.com/priceList.cfm`, accessed Mar 2009.

[15] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proc. of the 6th Intl. Conf. on Autonomic Computing and Communications (ICAC)*, 2009.

[16] S. Kounev. Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Trans. on Software Engineering*, 32(7):486–502, 2006.

[17] A. E. Krzesinski and P. Teunissen. Multiclass queueing networks with population constrained subnetworks. In *Proc. of ACM SIGMETRICS*, pages 128–139, 1985.

[18] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[19] H. Li, W. Theilmann, and J. Happe. SLA Translation in Multi-layered Service Oriented Architectures: Status and Challenges. Technical Report 2009-08, Univ. of Karlsruhe, Germany, 2009.

[20] D. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, and Methods*. Prentice Hall, 1998.

[21] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):312–322, 1980.

[22] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. on Software Engineering*, 21(8):689–700, 1995.

[23] T. Schneider. *SAP Performance Optimization Guide*. SAP Press, 2003.

[24] TPC. TPC-C: on-line transaction processing benchmark V5, 2009. Online available: `http://www.tpc.org/tpcc/`, accessed Mar 2009.

[25] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi. An analytical model for multi-tier internet services and its applications. In *Proc. of ACM SIGMETRICS*, pages 291–302. ACM Press, 2005.

[26] H. Wada, P. Champrasert, J. Suzuki, and K. Oba. Multiobjective Optimization of SLA-Aware Service Composition. In *proc. of IEEE Congress on Services*, pages 368–375, 2008.

[27] E. Bondarev, M. Chaudron, and E. de Kock. Exploring performance trade-offs of a JPEG decoder using the deepcompass framework. In *proc. of WOSP*, pages 153–163, 2007.

[28] S. Winder. *Analog and Digital Filter Design, Second Edition*. Newnes, 2002.

[29] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Software Engineering*, 30(5):311–327, 2004.