Integrated Optimization of IT Service Performance and Availability Using Performability Prediction Models

Sascha Bosse, Hendrik Müller, and Klaus Turowski

Very Large Business Applications Lab
Faculty of Computer Science, Otto-von-Guericke University Magdeburg
{sascha.bosse|hendrik.mueller|klaus.turowski}@ovgu.de

Abstract. Optimizing the performance and availability of an IT service in the design stage are typically considered as independent tasks. However, since both aspects are related to one another, these activities could be combined by applying performability models, in which both the performance and the availability of a service can be more accurately predicted. In this paper, a design optimization problem for IT services is defined and applied in two scenarios, one of which considers a mechanism in which redundant components can be used both for failover as well as handling overload situations. Results show that including such aspects affecting both availability and performance in prediction models can lead to more cost-effective service designs. Thus, performability prediction models are one opportunity to combine performance and availability management for IT services.

Keywords: IT Service Management, Availability Management, Capacity Management, Performability Modeling, Redundancy Allocation Problem

1 Introduction

IT service providers are faced with the challenge of designing high-quality and cost-effective IT systems in order to stay competitive [1]. In particular, degraded quality of service may cause the violation of service level agreements, leading to penalty costs and loss of reputation for the service provider [2]. Two of the most crucial quality aspects of an IT service are performance and availability [3]. In order to ensure that a service achieves the desired quality level at a minimum of costs, capacity management for performance as well as availability management have to be performed carefully [3].

However, managing the quality and costs of an IT service is difficult as these are complex systems with specific requirements, an architectural mix of diverse hardware and software components as well as skewed workload [1], [4]. In order to have accurate estimates about service quality, an IT service is usually tested before its deployment [5], [6]. Nevertheless, there are decisions to be made in the service design stage and their correction in the deployment or operation stage may be very costly [7], [8]. This applies especially for architectural decisions which have the greatest impact on quality attributes [9].

13th International Conference on Wirtschaftsinformatik,
February 12-15, 2017, St. Gallen, Switzerland

Bosse, S.; Müller, H.; Turowski, K. (2017): Integrated Optimization of IT Service Performance and Availability Using Performability Prediction Models, in Leimeister, J.M.; Brenner, W. (Hrsg.): Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik (WI 2017), St. Gallen, S. 76-90

Therefore, performance and availability modeling techniques are effective in the service design stage, as they provide a first estimate about the service's quality level [1], [5], [10]. On the basis of these models, different design alternatives can be compared to minimize service costs [1], especially if alternatives are recommended automatically [4]. For that purpose, optimization approaches can be applied. For instance, in [3] a heuristic for service-component allocations is presented to support capacity management. Other examples include optimizing the redundancy design for availability management by defining redundancy allocation problems (RAP), e.g. in [11], [12].

Despite its advantages, a recent report of Gartner comes to the conclusion that most enterprises under-invest in performance modeling for capacity management [13]. One reason for that may be that most existing approaches for capacity management consider performance in isolation, disregarding dependencies to other quality attributes such as reliability, availability, or scalability [14], which can be subsumed under the term dependability [15]. Thus, performance prediction models overestimate the ability of a service to satisfy its consumers [16]. On the other hand, applying pure dependability approaches will often lead to conservative quality estimates so that cost saving potentials are not addressed [16]. Hence, combined models for availability and performance prediction could depict an opportunity to integrate capacity and availability management in order to achieve a cost-effective and high-quality service design.

For approaches in which both performance and dependability aspects are considered, the term performability modeling can be used. While performability prediction models have been developed to analyze a single system, their possible impact on design optimization considering a large variety of systems has not been researched. Therefore, the question remains if an optimization approach on the basis of performability models would allow for higher cost-effectiveness of IT services than if isolated approaches are applied. In order to answer this question, an optimization problem is formulated in this paper for the capacity and redundancy design of an IT service. A simple Petri net performability model is used to estimate the quality and costs of a service design. In an illustrative example, it can be demonstrated that performability models provide the flexibility to include mechanisms affecting both performance and availability of a service such as an elastic standby redundancy derived from cloud computing concepts, which is not only used in failure but also in overload situations. This leads to a more effective peak-load handing [17] and, thus, to design suggestions with less costs while satisfying performance and availability service level objectives.

2 Related Work

In this section, relevant basics of capacity and availability management as well as performability modeling are introduced as the basis for the optimization problem.

2.1 Capacity Management and Performance Models

The capacity management process is an important part of the design stage in IT service management frameworks such as the ITIL [18]. Its purpose is to ensure that an

IT service meets the current and future performance-related requirements in a cost-effective manner [18]. Related terms are capacity planning and software performance engineering. The process of capacity management relies on performance models estimating the relation between input (workload model, design alternatives) and output variables (e.g. response time, throughput, or utilization) [7]. Performance models can be classified into measurement- and model-based approaches [5].

Measurement-based performance models require existing systems to be observed for at least some of the possible design alternatives [19]. The properties of non-existing systems can be estimated by using machine learning methods such as support vector machines or random forests [20] (black-box approach). Nonetheless, these performance models often lack transferability to other problem classes [5]. In model-based approaches, analytical performance models are used to describe the relationship between input and output variables (white-box approach). Examples are queuing networks [7], queuing Petri nets [4], (generalized) stochastic Petri nets [10], or stochastic reward nets [16]. However, constructing these models requires insight into the performance characteristics of the used system components, which may not be accessible especially for third-party software components [21]. Besides the pure measurement- or model-based approaches, a grey-box approach can be applied, e.g. by using gained knowledge of existing components and design patterns [5] to combine low-level measurements and high-level analytical models [8].

In order to evaluate performance models, the following techniques can be used: prototyping, testing, simulation, or analytical evaluation [5]. While prototyping and testing provide more accurate results, these approaches are also very costly and can only be applied in later lifecycle phases [6]. On the other hand, analytical and simulation techniques can be applied in the design phase of the service lifecycle [5]. Analytical evaluation is faster, but some aspects of the system performance may not be considered (e.g. G/G/n queues). In this case, simulation approaches are more effective [4], [5], [7] which also allow for dynamic analysis of performance metrics [22].

After performance models can be constructed and evaluated, many possible design alternatives should be analyzed in order to find a suitable cost-performance tradeoff [1]. Since these alternatives should be compared quickly [7], an optimization approach can be applied. For instance, in [3] an optimization problem is defined to find the minimum number of computing nodes for a set of IT services, and a component-allocation heuristic is developed to identify (sub)optimal solutions.

2.2 Availability Management and the Redundancy Allocation Problem

Similar to capacity management, availability management is an IT service design process aiming at meeting the agreed availability requirements at minimum costs [18]. A related term is software reliability engineering, which is, however, more focused on single software components than on complex systems of (third-party) components [23]. As in capacity management, black- and white-box availability models can be constructed and evaluated by different methods. White-box approaches for availability modeling include combinatorial, state-space-based, or hierarchical techniques [24].

Basically, the availability of a system can be increased by applying fault removal, fault prediction, fault prevention, and fault tolerance approaches [15] while the latter two can be considered in the design phase [25]. However, preventing faults by increasing component reliabilities is limited [26] and, thus, fault-tolerance has to be applied in order to achieve high-availability, e.g. by introducing redundancy mechanisms [27]. Redundancy means to provide additional components with equal functions to cover faults of the original components. Redundant components may be purchased from another manufacturer or developed by different teams to decrease the probability of common-cause failures (heterogeneous redundancy). Additionally, active and passive (or standby) redundancy can be distinguished.

A component in active redundancy is ready to instantly takeover for a defect component. Passive components are in a lower state of readiness and need some time for takeover. Depending on the state and the time to activation, standby components can be classified into cold-, warm-, and hot-standby [28]. With increasing time to activation, usually failure rate and operational costs of a passive component are decreased [27]. In order to decide which components may require redundancy mechanisms and which type of redundancy has to be used, a redundancy allocation problem (RAP) can be defined. A RAP is a NP-hard optimization problem [29] and is often characterized by the following aspects:

1. A system consists of n required subsystems.
2. In each subsystem, a number of components can be operated in redundancy.
3. Components fail and recover according to random time distributions.
4. For each subsystem, a number of components has to be identified so that availability is maximized or costs are minimized subject to constraints.

In recent years, RAP definitions with increasing complexity have been developed to include characteristics of IT systems. For instance, standby redundancy and its effects have been considered in [12], [28]. RAP are usually solved using (meta-)heuristics [30] such as genetic algorithms, e.g. in [11], [12], [28].

2.3 Performability Modeling and Design Optimization

While isolated performance and availability models are still subject of research, several approaches try to combine both dependability and performance aspects as there are significant relations between those. One reason for this is that availability is not defined as the absence of failures, but as the probability of success [31]. Thus, unavailability is not only caused by failures but also by overload situation which lead to request rejections [19]. Furthermore, when response times increase, users will abort waiting for a reply and will consider the service as unavailable [7]. On the other hand, component faults may lead to degraded capacity of a service.

In order to consider these aspects in a more accurate model of performance and availability, the concept of performability models can be applied [31]. Examples can be found in [5], [10], [16], [19]. However, there is a lack of design optimization approaches that are based on performability models although some RAP definitions are able to address performance aspects superficially since system capacity is formed by the components operated in active redundancy. For instance, multiple component states have been introduced to model performance degradation of components. In order to compute the system performance from component states, the loss of load probability (LOLP) index is used as a measure for unavailability (e.g. in [32]) which has been originally defined for electrical power systems as the probability that demand exceeds capacity. In an IT system, throughput is a related concept [33]. In a parallel subsystem of (active) redundant components, the throughput can be computed as the sum of the components' throughputs. The throughput of a series system is defined as the minimum of the subsystems' throughputs. In order to estimate the availability of the system, the load of the system is set in relation to its throughput.

The disadvantage of this approach is that performance is not considered in detail so that, for instance, response times cannot be analyzed. Furthermore, all subsystems have the same load at every time. In an IT service, however, a subsystem may be visited by a request more than once and the processing time in a subsystem can differ significantly [7]. In addition to that, the LOLP approach leads only to estimates about the mean utilization of components which makes it difficult to get accurate estimates for operational costs which may depend strongly on current utilization as e.g. the power consumption of a CPU. Thus, this approach is not suitable for an effective design optimization for a combined capacity and availability management.

3 The Redundancy and Capacity Allocation Problem (RCAP)

Our study of related work revealed that no suitable design optimization problem has been defined on the basis of performability models. In order to investigate the feasibility of performability modeling for design optimization, a new optimization problem is presented in this section on the basis of the RAP. The objective of the optimization problem is to minimize the costs of an IT service while satisfying mean response time and availability constraints. Although the capacity of a service is determined by the components operated in active redundancy, the problem is named redundancy and capacity allocation problem (RCAP) to underline the differences to classical RAP approaches in which performance aspects are not sufficiently addressed.

In order to demonstrate the opportunities of performability models, two different strategies for components in passive redundancy are considered: in the classical standby case, passive components are activated if and only if another component has failed. In the elastic standby scenario, these components are activated only in case of overloads, which may be caused by load peaks as well as component failures.

3.1 Assumptions

An IT service is consumed by customers that send independent, comparable requests. In order to respond to a request, a sequence of operations has to be performed. An operation is executed in a subsystem consisting of functional equivalent components. Each of these components can serve a number of operations concurrently and the processing time depends on the capacity of the component.

A component can have three states: active, standby (passive), and failed. Initially, a component is either active or standby. In active mode, a component fails and is recovered again after random time intervals. A component does not provide capacity if it is failed and cannot fail if it is in standby mode. If a standby component has to be activated, a random time elapses before the component is set to active mode and may fail. In addition to component failures, the whole service can also be affected by a failure, which leads to the rejection of all requests currently processed (disaster case). All failure, recovery and activation times as well as inter-arrival and processing times can be described by parametric random distributions.

If all servers are busy in a subsystem, operations to be processed are queued until a server becomes available (FIFO strategy). An admission control can be enabled to avoid unnecessary processing if a user will likely abort waiting for a reply by automatically rejecting requests after a certain time in the system (timeout).

The costs of the service are the sum of capital and operational costs. While the former is determined by the acquisition costs for components, the latter costs are compound of the costs arising for recovery activities and the components' operational costs that depend linearly on their utilization levels. Another linear dependency is assumed between components and system operational costs, which is characterized by the power usage effectiveness (PUE) that ranges in practice from 1.2 to 3.0 [34].

3.2 Problem Definition

A redundancy and capacity allocation problem (RCAP) is characterized by a timestep Δt, a factor PUE as well as by random variables R, TTF and TTR describing inter-arrival times, times to failure and times to recover for an IT service. Furthermore, a number of operations o and subsystems n have to be defined as well as a function ops mapping operations to subsystems. Another function S maps the operations to random variables describing the standard service time. In addition to that, a timeout to can be defined for admission control.

A set of components is associated with each subsystem. Each component i is characterized by its number of servers for concurrent operations s_i, a service time norm factor f_i, random distributions TTF_i and TTR_i, a random distribution TTA_i for its time to activation, its acquisition costs ac_i, recovery costs rec_i as well as operational costs for Δt in standby state $standby_i$, in idle mode $idle_i$ and if fully utilized $full_i$.

A solution candidate x to the RCAP consists of $2n$ sets of component selections describing the components to be operated in active as well as passive redundancy for each subsystem. Thus, the optimization problem can be defined as follows:

$$\min_x C(x,t) \text{ s.t. } A(x) \geq A_0 \wedge RT(x) \leq RT_0 \tag{1}$$

Since the costs of a service C depends on the time interval to be considered, a parameter t has to be defined as well as the service level objectives for availability A_0 and response time RT_0.

An overview of all required parameters to define a RCAP are presented in Table 1.

Table 1. Parameter overview for the RCAP

Problem Parameters	Description
$\Delta t \in \mathbb{R}$	Timestep for operational costs definition
$PUE \in \mathbb{R}$	Power usage effectiveness of the IT service
$R: \mathbb{R} \to \mathbb{R}$	Distribution of request inter-arrival times
$TTF, TTR: \mathbb{R} \to \mathbb{R}$	Distribution of time to failure/recovery for the service
$o, n \in \mathbb{N}$	Number of required operations and subsystems
$ops: \mathbb{N} \to \mathbb{N}$	Function mapping operations to subsystems
$S: \mathbb{N} \to (\mathbb{R} \to \mathbb{R})$	Function mapping operations to standard service time distributions
$to \in \mathbb{R}$	Timeout after which admission control is applied
$s_i \in \mathbb{N}$	Number of servers in a component *i*
$f_i \in \mathbb{R}$	Service time norm factor for a component *i*
$TTF_i, TTR_i, TTA_i: \mathbb{R} \to \mathbb{R}$	Distributions of time to failure, recovery, and activation for a component *i*
$ac_i, rec_i \in \mathbb{R}$	Acquisition and recovery costs of a component *i*
$standby_i, idle_i, full_i \in \mathbb{R}$	Operational costs per timestep in standby and idle mode as well in full operation for a component *i*

3.3 A Simple Performability Prediction Model

On the basis of the RCAP definition, a simple performability prediction model is developed that can be used to estimate availability, mean response time, and costs of the solution candidates. Due to their modeling power, a solution candidate is modeled in a generalized stochastic Petri net (GSPN), cf. e.g. [35]. In order to consider arbitrary random variables, these models are evaluated by Monte Carlo simulation.

Thus, a GSPN has to be automatically generated for each solution candidate. First, an arrival transition is defined that generates request tokens. To each subsystem, a queue place is assigned with infinite capacity for collecting incoming requests. Another place collects all rejected requests for the service. For a subsystem, a place and a corresponding processing transition are created for each server of the assigned components and are connected to the queue of the subsystem performing the next operation. The subsystem queue is connected to the server places by immediate transitions which are only active if the component is working. In the case that the corresponding component fails, another immediate transition fires tokens to the rejection place instead. To all transitions, random firing time distributions are assigned according to the component's definition.

In

Figure **1**, an example GSPN of a single subsystem of one active component with two servers is presented. Activation functions (dashed rectangles) are used to prevent requests to be rejected if the component is working or to be processed if it is failed. Request are created by the arrival transition and collected in the queue place. Depending on the component state (failed/working cycle below), requests are rejected or processed and assigned to the next operation queue.

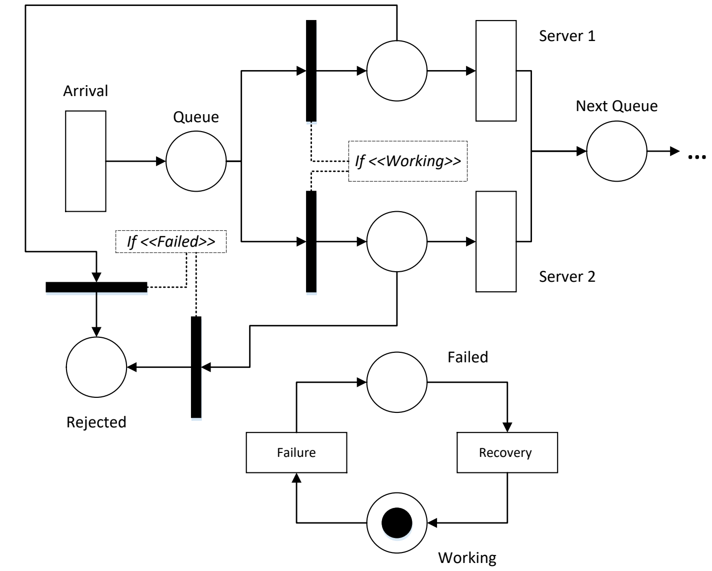

Figure 1. GSPN processing model example with an active component

In order to implement the operation sequence as well as the admission control, tokens are distinguishable (colored Petri net) by their arrival time and the current operation. Based on this information, all tokens that have been longer in the system than the defined timeout are moved to the rejection place. Additionally, a subsystem model such as presented above can map more than one operation since firing times depend on the current operation of the token. After all operations are performed, the token is moved to a place in which completed requests are stored.

While the state of an active component can be modeled by a simple failed/working cycle, for standby components, a standby place is introduced to indicate the current state of the passive component as displayed in Figure 2.

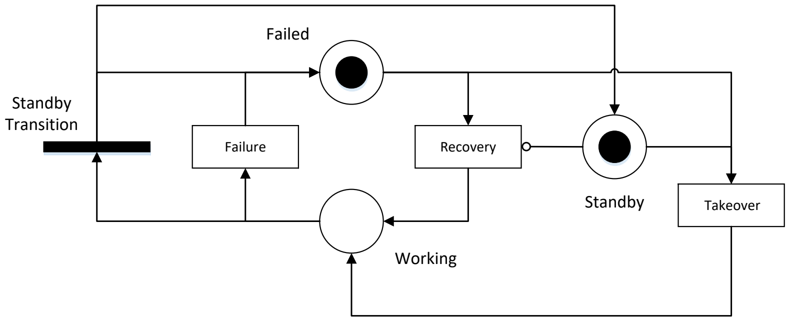

Figure 2. GSPN state model example of a passive component

In standby mode, the state of the component is equivalent to a failure, but recovery is disabled. To the takeover and standby transitions, activation functions are assigned depending on the subsystem characteristics and the chosen standby strategy. In case of the classical standby, the takeover transition is activated if an active component fails. If the active components are recovered, the standby transition is activated and the passive component changes to standby mode. In elastic standby, takeover is activated if the subsystem queue is not empty and all working servers are busy (overload situation). The component is deactivated again if none of its servers are busy.

For each completed request, the response time is the difference between its completion and its arrival time. The mean response time of a solution candidate is computed from all request response times at the end of a simulation run. Availability is defined as the ratio of completed requests to arrived requests minus those in completion. The costs are the sum of acquisition, recovery, and operational costs. Impulse rewards for recovery costs are assigned to recovery transitions. The utilization of a component is computed after each timestep Δt and is used to compute its current operational costs c_i linearly based on values $idle$ and $full$. This value is used in order to estimate the service operational costs C_{op} up to a time *t*:

$$c_i(t) = \begin{cases} standby_i & \text{, if} \ll Failed \gg \\ idle_i + u_i(t) \cdot full_i & \text{, otherwise} \end{cases}$$
$$C_{op}(x,t) = \frac{PUE}{\Delta t} \cdot \int_0^t \left(\sum_{i \in x} c_i(\tau) \right) \mathrm{d}\tau \tag{2}$$

By simulating the behavior of the solution candidate in several independent runs, mean values for response time, availability, and costs can be computed to characterize a solution candidate with statistical significance.

3.4 A Genetic Algorithm to Solve the RCAP

The RCAP defines an infinite search space that has to be efficiently explored for identifying (sub)optimal solutions. Therefore, a genetic algorithm is defined, which is characterized by problem-specific operations for evaluating a solution's fitness, encoding a solution, initializing random solutions, altering (mutation) and recombining solutions as well as selection and termination criteria.

The suitability of a solution to solve the defined RCAP is characterized by its fitness to be maximized. According to [11], a penalty function is used to avoid infeasible solutions in the end. Since costs has to be minimized, the fitness is defined as the negative costs minus penalty:

$$f(x) = -C(x) - \delta \left(\frac{\max\big(0, A_0 - A(x)\big)}{NFT_A} + \frac{\max(0, RT(x) - RT_0)}{NFT_{RT}} \right) \tag{3}$$

In order to scale the penalty, pre-defined near-feasibility thresholds (NFT) are used as well as a factor δ, which is the difference between the minimal cost of all solutions and the minimal cost of feasible solutions (cf. [36]).

For the encoding, a solution candidate is represented by two vectors for the active and the passive configuration. Each vector consists of n integer vectors corresponding to the indices of the defined components for a subsystem. An example encoding for three subsystems may look as follows:

$$\Big(\big((2,1),(1),(2,2)\big),\big((\ ),(1),(\ )\big)\Big)$$

In this solution candidate, two different components are operated in active redundancy in the first subsystem (of the second and the first type in this subsystem), but no component is operated in passive redundancy (empty bracket in the second vector). In the second subsystem, however, two components of the first type in this subsystem are used whereas one is in active and the other in passive redundancy.

Solution candidates are generated by choosing random components for the subsystems. The number of components in a subsystem is randomly chosen according to a pre-defined distribution. If a solution is to be mutated, a subsystem is randomly chosen and a random operation is applied to active or passive sets: with 20% probability, a component choice is exchanged. A component is either added or removed with 40% probability. The remove operation is disabled if a subsystem consists only of one active or zero passive components. For the recombination of two solutions, separate uniform crossovers are applied for the active as well as the passive configuration. Thus, subsystem configurations are exchanged between both solution candidates.

In the genetic algorithm, first a number μ solutions are generated and their fitness is evaluated. In the main loop, λ solutions ($\lambda > \mu$) are created by recombination (whereas fitter solutions are more frequently recombined) and mutation is applied with a certain probability p_{mut}. After the fitness of the new solution candidates is evaluated, μ solutions are selected from those by performing tournament selection for the next loop (generation). In tournament selection, μ tournaments of t solutions are performed in which the kth fittest solution is selected with probability $p_t(1-p_t)^{k-1}$. The algorithm results in the fittest solution after $maxGen$ generations.

4 An Illustrative Example: Sales and Distribution

The advantages of the performability optimization approach are demonstrated in a small and illustrative example describing an IT service with eight operations. For that purpose, the GA, model generation and simulation have been implemented in the java-based simulation framework *AnyLogic 6.8.2*. The simulation results have been verified by comparing test instances with results from Markov-based queuing and availability models.

The operations to be performed describe a standard business process for a delivery from stock scenario with prior lead generation and subsequent material requirements check. Therefore, five different subsystems are involved in this process (cf. Table 2). In order to define service times, data from a standard benchmark for the sales and distribution modules of SAP ERP business applications has been used. In this context, the capacity of components can be measured in the normalized metric SAPS (SAP Application Performance Standard), where 100 SAPS equal 2,000 fully processed order line items per hour. Standard service times have been computed from mean response times of various benchmark results that had been performed over the last ten years and are presented in Table 2. Using the reciprocal of these values as a rate, exponential random variables have been defined for the standard service time of each operation. These times are valid for the mean processing power of all analyzed components which is 5332.62 SAPS.

Table 2. Operations, subsystems, and standard service times of the illustrative example

Operation	Subsystem	Mean standard service time in ms
Maintain Leads	Customer Relationship Management (CRM)	2,451.65
Create Sales Order	Sales and Distribution (SD)	1,415.69
Create Outbound Delivery	Logistics Execution (LE)	1,309.62
Display Sales Order	Sales and Distribution (SD)	883.65
Change Outbound Delivery	Logistics Execution (LE)	1,101.17
Create List of Sales Orders	Sales and Distribution (SD)	1,923.90
Create Billing Document	Billing (B)	2,121.83
Display Stock/Requirements Situation	Production Planning and Control (PP)	557.71

In each subsystem, three possible components can be chosen that are presented in Table 3. The norm factor for the service times is computed by relating the SAPS per server to the mean value of the standard service times. For the computation of operational costs, an energy price of 0.2814 €/kWh is assumed.

Table 3. Available components for the subsystems

Label	*ac* in €	Capacity in SAPS	*s*	*f*	*standby* in ct/h	*idle* in ct/h	*full* in ct/h
A	2,494	11,500	8	3.70	0.02	3.83	6.22
B	21,580	150,000	72	2.57	0.28	2.43	15.67
C	4,669	43,000	16	1.99	0.14	1.58	5.52

All components share the same transition times: time to failure is exponentially distributed with a mean value of 8,760h, time to recover normally distributed with mean 1.73h and a variance of 0.5h and time to activation normally distributed with mean 30s and variance 5s (cold standby). The simulation is run for three years to map depreciation of acquisition costs (26,280h). A *PUE* of 2.0 is assumed, meaning that for every unit of energy consumed by the components a unit of energy is consumed for supporting systems. For an arrival rate of 18,000 requests per hour (exponential distribution of inter-arrival times), the mean response time should be lower than 30s and the availability should be at least 99.95%. The timeout for requests is 100s.

For both the classical and the elastic standby scenario, ten iterations of the genetic algorithm are performed. In each of these iterations, ten generations are conducted with $\mu = 30, \lambda = 60, p_{mut} = 0.2, t = 4, p_t = 0.9, NFT_A = 0.0001$ and $NFT_{RT} = 5s$. Solution candidates are initialized by choosing a normally distributed number of components in each subsystem (mean 3, variance 2 for active and mean 1, variance 0.5 for passive components). In order to evaluate a solution, 25 simulation replications are performed. In Table 4, the mean and standard deviation of the fitness values in the ten iterations are presented. It can be stated that in the elastic standby case solutions with less cost are identified on average.

Table 4. Fitness statistics for ten iterations in both scenarios

	Classical	**Elastic**
Mean fitness	-215,180.52	-128,415.66
Standard deviation of fitness	54,668.21	49,718.86

A comparison of the fittest solutions of both scenarios over all ten iterations is given in Table 5, using the labels of Table 3 to indicate recommended components for the subsystems presented in Table 2. For each subsystem, the recommended components in active (a) and passive (p) mode are displayed. While availability and response time are comparable in the elastic case, the costs of the service are significantly reduced in comparison to the classical standby scenario. The main reason for this is that fewer components are used in active mode.

Table 5. Fittest individuals in the classical (above) and the elastic standby scenario (below)

CRM		**SD**		**LE**		**B**		**PP**		***A***	***RT***	***C***	***-f(x)***
a	p	a	p	a	p	a	p	a	p	0.9…	in s	in €	
CCAA	A	CCCC		ACC		CC	C	AAAC		99407	25.18	119,041	131,370
AC	C	CC	C	CC		C	C	C	C	99490	24.60	84,231	84,233

These results demonstrate that a classical optimization approach will result in higher costs since capacity for overload situations has to be provided by the active components and standby components are only used in failure cases. In the elastic scenario, however, some of the needed capacity for overload situations can be provided by the standby components while failover is still applicable. Thus, considering a mechanism affecting both availability and performance in an integrated model leads to a more cost-effective service design.

5 Conclusion

In design optimization for IT services, most approaches rely on isolated models for performance or availability estimation. However, defining optimization problems on the basis of performability models considers design mechanics affecting both capacity and availability of an IT service. In this paper, it could be demonstrated that addressing these aspects in an integrated prediction model can lead to more cost-effective design suggestions than if performance and availability optimization are considered separately. Hence, this paper reveals some of the potential of integrated availability and capacity management for IT service management.

Although the developed simulation approach provides accurate results, the runtime of the optimization iterations may exceed several hours due to model stiffness, i.e. the great difference between transition rates of performance and availability state changes in a monolithic model [16]. This clearly affects the scalability of the approach to solve larger problem instances. One solution to this problem without affecting accuracy could be the massive parallelization of the independent simulation replications as well as of fitness evaluations. In this context, the scalability of the approach should be tested in large-scale problem instances in which service and transition times are not exponentially distributed to demonstrate the suitability of the simulation approach.

On the other hand, some assumptions made to demonstrate the feasibility of the applied approach have to be overcome in future work to increase applicability. This includes the assumption of the linear dependency between utilization and operational costs [37] as well as the use of mean response time as an objective while service level agreements normally include percentile guarantees. However, the developed approach can easily be adapted to result in arbitrary percentiles for response time. Furthermore, mean values of replications are used for fitness evaluation of solution candidates. Instead, confidence interval boundaries can be reported which would introduce uncertainty in the optimization. Another limitation is the fixed capacity of the defined components which may not reflect elastic and pay-per-use cloud computing services. This fact may be addressed by defining component capacity as a dynamic range with according costs. Additionally, the defined prediction model is not covering all aspects influencing response time and availability, for instance, operator errors which are reported to be one of the major causes for unavailability of IT systems [38]. Considering these limitations in future work as well as introducing additional quality aspects such as security could lead to even more realistic quality and cost estimates and, thus, to better service designs.

References

1. Almeida, V.A., Menascé, D.A.: Capacity planning an essential tool for managing Web services. IT professional. 4, 33–38 (2002).
2. Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Buyya, R., De Rose, C.A.F.: Towards autonomic detection of SLA violations in Cloud infrastructures. Future Generation Computer Systems. 28, 1017–1029 (2012).
3. Roy, N., Dubey, A., Gokhale, A., Dowdy, L.: A capacity planning process for performance assurance of component-based distributed systems. In: ACM SIGSOFT Software Engineering Notes. pp. 259–270. ACM (2011).
4. Kounev, S.: Performance modeling and evaluation of distributed component-based systems using queueing petri nets. IEEE Transactions on Software Engineering. 32, 486–502 (2006).
5. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software. 82, 3–22 (2009).

6. Nambiar, M., Kattepur, A., Bhaskaran, G., Singhal, R., Duttagupta, S.: Model Driven Software Performance Engineering: Current Challenges and Way Ahead. ACM SIGMETRICS Performance Evaluation Review. 43, 53–62 (2016).
7. Menasce, D.A., Almeida, V.A., Dowdy, L.W., Dowdy, L.: Performance by design: computer capacity planning by example. Prentice Hall Professional (2004).
8. Terlit, D., Krcmar, H.: Generic Performance Prediction for ERP and SOA Applications. In: Proceedings of the 18th European Conference on Information Systems (ECIS) (2011).
9. Williams, L.G., Smith, C.U.: Performance evaluation of software architectures. In: Proceedings of the 1st international workshop on Software and performance. pp. 164–177. ACM (1998).
10. Bosse, S., Schulz, C., Turowski, K.: Predicting Availability and Response Times of IT Services. In: Proceedings of the 22nd European Conference on Information Systems (ECIS). , Tel Aviv, Israel (2014).
11. Coit, D.W., Smith, A.E.: Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm. IEEE Transactions on Reliability. 45, 254–266 (1996).
12. Tokuno, K., Yamada, S.: Markovian availability modeling for software-intensive systems. International Journal of Quality & Reliability Management. 17, 200–212 (2000).
13. Head, I., Govekar, M.: Market Guide for Capacity Management Tools. Gartner (2015).
14. Williams, L.G., Smith, C.U.: PASA SM: a method for the performance assessment of software architectures. In: Proceedings of the 3rd international workshop on Software and performance. pp. 179–189. ACM (2002).
15. Laprie, J.-C.: Dependable Computing: Concepts, Limits, Challenges. In: 25th IEEE International Symposium on Fault-Tolerant Computing. pp. 42–54. , Pasadena, CA, USA (1995).
16. Ma, Y., Han, J.J., Trivedi, K.S.: Composite performance and availability analysis of wireless communication networks. IEEE Transactions on Vehicular Technology. 50, 1216–1223 (2001).
17. Ranjan, R., Zhao, L., Wu, X., Liu, A., Quiroz, A., Parashar, M.: Peer-to-peer cloud provisioning: Service discovery and load-balancing. In: Cloud Computing. pp. 195–217. Springer (2010).
18. Hunnebeck, L.: ITIL Service Design 2011 Edition. The Stationery Office, Norwich, UK (2011).
19. Woodside, M., Franks, G., Petriu, D.C.: The future of software performance engineering. In: Future of Software Engineering, 2007. FOSE'07. pp. 171–187. IEEE (2007).
20. Venkataraman, S., Yang, Z., Franklin, M., Recht, B., Stoica, I.: Ernest: efficient performance prediction for large-scale advanced analytics. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). pp. 363–378 (2016).
21. Brebner, P.C.: Performance modeling for service oriented architectures. In: Companion of the 30th international conference on Software engineering. pp. 953–954. Springer (2008).

22. Jewell, D.: Performance Modeling and Engineering. Presented at the (2008).
23. Vouk, M.A.: Software reliability engineering. In: Annual Reliability and Maintainability Symposium (2000).
24. Trivedi, K., Ciardo, G., Dasarathy, B., Grottke, M., Matias, R., Rindos, A., Vashaw, B.: Achieving and Assuring High Availability. In: Nanya, T., Maruyama, F., Pataricza, A., and Malek, M. (eds.) 5th International Service Availability Symposium (ISAS). pp. 20–25. Springer Verlag Berlin Heidelberg, Tokyo, Japan (2008).
25. Garg, H., Rani, M., Sharma, S.P., Vishwakarma, Y.: Bi-objective optimization of the reliability-redundancy allocation problem for series-parallel system. Journal of Manufacturing Systems. 33, 335–347 (2014).
26. Chi, D.-H., Kuo, W.: Optimal Design for Software Reliability and Development Cost. IEEE Journal on Selected Areas in Communications. 8, 276–282 (1990).
27. Shooman, M.L.: Reliability of Computer Systems and Networks – Fault Tolerance, Analysis, and Design. John Wiley & Sons New York, New York, NY, USA (2002).
28. Ardakan, M.A., Hamadani, A.Z.: Reliability–redundancy allocation problem with cold-standby redundancy strategy. Simulation Modelling Practice and Theory. 42, 107–118 (2014).
29. Chern, M.-S.: On the computational complexity of reliability redundancy allocation in a series system. Operations Research Letters. 11, 309–315 (1992).
30. Soltani, R.: Reliability optimization of binary state non-repairable systems: A state of the art survey. International Journal of Industrial Engineering Computations. 5, 339–364 (2014).
31. Meyer, J.F.: On evaluating the performability of degradable computing systems. IEEE Transactions on computers. 100, 720–731 (1980).
32. Ouzineb, M., Nourelfath, M., Gendreau, M.: Tabu search for the redundancy allocation problem of homogenous series–parallel multi-state systems. Reliability Engineering & System Safety. 93, 1257–1272 (2008).
33. Bosse, S., Splieth, M., Turowski, K.: Multi-Objective Optimization of IT Service Availability and Costs. Reliability Engineering & System Safety. 147, 142–155 (2016).
34. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The Cost of a Cloud: Research Problems in Data Center Networks. SIGCOMM Comput. Commun. Rev. 39, 68–73 (2008).
35. Ciardo, G., Muppala, J.K., Trivedi, K.S.: SPNP: Stochastic Petri Net Package. In: Proceedings of the 3rd International Workshop PNPM. pp. 142–151. IEEE Computer Society (1989).
36. Kulturel-Konak, S., Smith, A.E., Coit, D.W.: Efficiently Solving the Redundancy Allocation Problem Using Tabu Search. IIE Transactions. 35, 515–526 (2003).
37. Walker, E.: The Real Cost of a CPU Hour. Computer. 42, 35–41 (2009).
38. Oppenheimer, D., Ganapathi, A., Patterson, D.A.: Why do Internet services fail, and what can be done about it? In: 4th Usenix Symposium on Internet Technologies and Systems (USITS) (2003).