Resources Planning in Database Infrastructures

Eden Dosciatti[1], Marcelo Teixeira[1], Richardson Ribeiro[1], Marco Barbosa[1], Fábio Favarim[1], Fabrício Enembreck[2] and Dieky Adzkiya[3]
[1]*Federal University of Technology-Paraná, Pato Branco, Brazil*
[2]*Pontificial Catholical University-Paraná, Curitiba, Brazil*
[3]*Delft University of Technology, Delft, The Netherlands*

Keywords: Modeling, Simulation, Resources Planning, Performance, Availability.

Abstract: Anticipating resources consumption is essential to project robust database infrastructures able to support transactions to be processed with certain quality levels. In Database-as-a-Service (DBaaS), for example, it could help to construct Service Level Agreements (SLA) to intermediate service customers and providers. A proper database resources assessment can avoid mistakes when choosing technology, hardware, network, client profiles, etc. However, to be properly evaluated, a database transaction usually requires the physical system to be measured, which can be expensive an time consuming. As most information about resource consumption are useful at design time, before developing the whole system, is essential to have mechanisms that partially open the black box hiding the in-operation system. This motivates the adoption of predictive evaluation models. In this paper, we propose a simulation model that can be used to estimate performance and availability of database transactions at design time, when the system is still being conceived. By not requiring real time inputs to be simulated, the model can provide useful information for resources planning. The accuracy of the model is checked in the context of a SLA composition process, in which database operations are simulated and model estimations are compared to measurements collected from a real database system.

1 INTRODUCTION

Transaction processing is a crucial part of the development of modern web systems, such as those based on *Service-Oriented Architecture* (SOA), a new paradigm to compose distributed business models. In SOA, an entire transaction is usually composed by distinct phases, such as networking, service processing, database processing, third-part processing, etc. For resources planning, it is usual that each particular phase is individually approached. In this paper, we concentrate on evaluating database transaction processing, especially for SOA systems (although not only), complementing previous results focused on the other phases of SOA (Rud et al., 2007; Bruneo et al., 2010; Teixeira et al., 2015).

In SOA, transactions are directly related to *Quality of Service* (QoS), and *Service Level Agreements* (SLAs) are mechanisms used to legally express commitments among service customers and providers (Sturm et al., 2000). Performance and availability of database operations are examples of clauses that can be agreed in SLA, specially when the database itself is provided as a service (DBaaS).

The effects of not being able to fulfill a database SLA are many. This kind of transaction commonly appears in the context of a service composition, as a particular stage of an SOA application. Therefore, if it fails to fulfill the metrics accorded in an SLA, this will probably affect the overall web service behavior and, as a consequence, the overall service orchestration, in a ripple effect, breaching one or more SLAs. Thus, for an entire SOA process, it is important to prevent a database transaction to fail or, at least, to be able to anticipate when it is susceptible to happen.

This task may not be so easy, as the ratio of load variation in web applications can reach the order of 300% (Chase et al., 2001), making it difficult to anticipate QoS. What is observed is that applications are entirely developed to be then stressed and measured, which can be quite expensive and time consuming. Recent works have suggested that SOA QoS can be estimated by modeling (Rud et al., 2007; Bruneo et al., 2010; Teixeira et al., 2015), but they have basically focused on networking and processing stages, assuming that database time consumption is implicit, which may be a strong assumption, as illustrated in (Teixeira and Chaves, 2011).

Dosciatti, E., Teixeira, M., Ribeiro, R., Barbosa, M., Favarim, F., Enembreck, F. and Adzkiya, D.
Resources Planning in Database Infrastructures.
In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016) - Volume 1*, pages 53-62
ISBN: 978-989-758-187-8

In this paper, we propose a stochastic modeling approach to estimate performance and availability of database transactions susceptible to intense workloads. By adopting *Generalized Stochastic Petri Nets* (GSPNs) as modeling formalism, we construct a formal structure that can be simulated and estimations can be used to anticipate resource consumption of database operations running under different load profiles. Based on these estimations, it is furthermore shown how to construct, at modeling time, realistic contracts for database transactions, which can be naturally combined as part of the estimations provided in works such as in (Rud et al., 2007; Bruneo et al., 2010; Teixeira et al., 2015).

The main advantage of our approach is not requiring real-time measurements nor the complete system implementation to be simulated. These information may not be available at design-time, when resources allocation is conducted. Instead, the model supports high level parameters collected from the *Data Base Management System* (DBMS) and statistics collected from samples of database query execution. For this reason, database technology, infrastructure or particular type of operation to be simulated, are implicit into the simulation scheme.

An example of a contract composition process is presented to illustrate the proposed approach. Using parts of a real database system and samples of relational database operations, we collect the input parameters to the model, which is then simulated and estimations are collected. Afterwards, we validate the estimations. This could be done by comparing them to benchmark data. In this paper, however, we are more interested on the uncertainty observed in the real-time behavior of transactions, e.g., how transactions behave when parameters change, or what is the performance degradation when workload increases, or what is the rate of requests queueing for a load profile, etc. These informations are not directly available from benchmarks, since they focus mostly on best and worst cases, for example. To be possible to check the accuracy of the proposed model so, we compare its estimations to measurements collected from a real database system. Results indicate that it is possible to trace the real behavior keeping a stochastically-reasonable average of 80% accuracy.

The paper is organized as follows: Section 2 discusses the related work; Section 3 introduces the basic concepts of SOA, SLA and GSPN; Section 4 presents the proposed GSPN model. Section 5 presents an example and some final comments are discussed in Section 6.

2 RELATED LITERATURE

Performance of databases has been a concern since the firstly proposed technologies and relational models (Elhardt and Bayer, 1984; Adams, 1985). From the web advent, however, advanced features have been combined to the existent DBMSs, attempting to support emergent requirements such as parallelism, distribution (Dewitt and Gray, 1992), object (Kim et al., 2002) and service-orientation (Tok and Bressan, 2006), etc. Although the interest on new technologies has recently grown, it has become more and more difficult to estimate their behavior.

In particular, when a database is part of a service, or when it is provided as a service itself, it is usually exposed to a highly variable and data-intensive environment, which makes it critical to estimate its QoS levels. In (Ranganathan et al., 1998), it has been discussed the impact of radically different workload levels on the database performance and how it becomes a concern when the database is immersed in QoS-aware frameworks that require QoS guarantees (Lin and Kavi, 2013). In general, the literature tackle this concern using run-time policies to filter and balance the database load (Lumb et al., 2003; Schroeder et al., 2006; Krompass et al., 2008). When connecting business partnerships, however, the negotiation of QoS criteria starts much earlier, at the service design phase, as it is necessary to plan and compose SLA clauses to be agreed.

An option to cover this gap is by adopting analytic models. For example, in (Tomov et al., 2004) it has been proposed a queuing network model to estimate the response time of database transactions. Furthermore, in (Osman and Knottenbelt, 2012) it has been compared the performance of different database designs via modeling. Queue time is predicted by using heuristic rules in (Zhou et al., 1997). Besides not being natively constructed for web environments, this approaches are also predominantly deterministic, which often does not match the characteristics of the real web environments (Teixeira et al., 2011) and can compromise the accuracy when estimating transactions with variable workloads. In addition, they are not usually flexible enough to be quickly converted in practical tools, or to be modified to analyze different system orchestrations, etc.

Thus, the need for supporting database QoS estimation remains. This is a quite challenging task, as web environments practically lack execution patterns and can present highly variable workloads, making it critical for a transaction to be estimated (Nicola and Jarke, 2000). In the same way, it is conceivably difficult to ensure that database queries will execute quickly enough to keep the process flow, avoiding it to be delayed more than expected (Reiss and Kanungo, 2005). The modeling approach to be presented in this paper is an option to face these challenges and implement database resources planning.

3 RELATED CONCEPTS

SOA comprises a set of principles for software development, fundamentally based on the concept of *service* (Josuttis, 2008). A service is a self-contained component of software that receives a request, processes it, and returns an answer. Eventually, a particular step of a service execution involves to access a database structure an process a data transaction. It may happen that a database transaction is itself offered as a service (DBaaS). In this case, the transaction processing is even more critical, as it is susceptible to a data-intensive environment, and its behavior becomes difficult to be estimated.

In SOA, legal commitments on services, including database transactions, are expressed by a mechanism known as *Service Level Agreements* (SLA) (Sturm et al., 2000). An SLA expresses obligations and rights regarding levels of QoS to be delivered and/or received. SLA clauses usually involve metrics such as response time, availability, cost, etc., and also establish penalties to be applied when a delivered service is below the promised standard (Raibulet and Massarelli, 2008).

In practice, ensuring that a SOA system will behave as expected is very difficult, and so it is difficult to compose, at design time, realistic SLAs. An alternative to probabilistically estimate the behavior of a service is given by modeling approaches. A model enables to observe the service behavior under "pressure", without exactly constructing the whole system.

The model described in this paper serves to this purpose and it is modeled by Petri nets. *Petri net* (PN) (Reisig and Rozenberg, 1998) is a formalism that combines a mathematical foundation to an intuitive modeling interface that allows to model systems characterized by concurrency, synchronization, resources sharing, etc. These features appear quite often in SOA systems, which make PNs a natural modeling choice.

Structurally, a Petri net is composed by *places* (modeling states), *transitions* (modeling state changes), and *directed arcs* (connecting places and transitions). To express the conditions that hold in a given state, places are marked with *tokens*.

Extensions of Petri nets have been developed to include the notion of time (Murata, 1989), which allows to represent time-dependent processes, such as communication channels, code processing, hardware designs, system workflows, etc. *Generalized Stochastic Petri Nets* (GSPNs) (Kartson et al., 1995), for example, is an extension that combines timed and non-timed PNs. In GSPN, *time* is represented by random variable, exponentially distributed, which are associated to *timed transitions*. When, for a given transition, the time is irrelevant, then one can simply use *non-timed* (or *immediate*) transitions.

Formally, a GSPN is a 7-tuple $GSPN = \langle P, \mathcal{T}, \Pi, I, O, M, W \rangle$, where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places;
- $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions;
- $\Pi : \mathcal{T} \to \mathbb{N}$ is the priority function, where:

$$\Pi(t) = \begin{cases} \geq 1, & \text{if } t \in \mathcal{T} \text{ is immediate;} \\ 0, & \text{if } t \in \mathcal{T} \text{ is timed.} \end{cases}$$

- $I : (\mathcal{T} \times P) \to \mathbb{N}$ is the input function that defines the multiplicities of directed arcs from places to transitions;
- $O : (\mathcal{T} \times P) \to \mathbb{N}$ is the output function that defines the multiplicities of directed arcs from transitions to places;
- $M : P \to \mathbb{N}$ is the initial marking function. M indicates the number of tokens[1] in each place, i.e., it defines the state of a GSPN model;
- $W : \mathcal{T} \to \mathbb{R}^+$ is the weight function that represents either the immediate transitions weights (w_t) or the timed transitions rates (λ_t), where:

$$W(t) = \begin{cases} w_t \geq 0, & \text{if } t \in \mathcal{T} \text{ is immediate;} \\ \lambda_t > 0, & \text{if } t \in \mathcal{T} \text{ is timed.} \end{cases}$$

The relationship between places and transitions is established by the sets ${}^\bullet t$ and $t^\bullet$, defined as follows.

Definition 1. *Given a transition $t \in \mathcal{T}$, define:*

- ${}^\bullet t = \{p \in P \mid I(t,p) > 0\}$ *as the pre-conditions of t;*
- $t^\bullet = \{p \in P \mid O(t,p) > 0\}$ *as the post-conditions of t.*

A state of a GSPN changes when an enabled transition fires. Only enabled transitions can fire. *Immediate* transitions fire as soon as they get enabled. The *enabling rule* for firing and the *firing semantics* are defined in the sequel.

Definition 2 (Enabling Rule)**.** *A transition $t \in \mathcal{T}$ is said to be enabled in a marking M if and only if:*

- $\forall p \in {}^\bullet t, M(p) \geq I(t,p)$.

[1] Black dots are usually used to graphically represent a token in a place.

When an enabled transition fires, it removes tokens from input to output places (its *pre* and *post* conditions).

Definition 3 (Firing Rule). *The firing of transition* $t \in \mathcal{T}$ *enabled in the marking* M *leads to a new marking* M' *such that* $\forall p \in ({}^{\bullet}t \cup t^{\bullet}), M'(p) = M(p) - I(t,p) + O(t,p)$.

A GSPN is said to be *bounded* if there exists a limit $k > 0$ for the number of tokens in every place. Then, one ensures that the state-space resulting from a bounded GSPN is finite.

When the number of tokens in each input place p of t is N times the minimum needed to enable t ($\forall p \in {}^{\bullet}t, M(p) \geq N \times I(t,p)$, where $N \in \mathbb{N}$ and $N > 1$), it enables the transition to fire more than once. In this situation, the transition t is said to be enabled with degree $N > 0$. Transition firing may use one of the following dynamic semantics:

- *single-server*: N sequential fires;
- *infinite-server*: N parallel fires;
- *k-server*: the transition is enabled up to k times in parallel; tokens that enable the transition to a degree higher than k are handled after the first k firings.

It can be shown (Kartson et al., 1995; Marsan et al., 1984) that GSPNs are isomorphic to *Continuous-Time Markov Chains* (CTMC). However, it is more expressive, as it allows to compute metrics by both simulation and analysis of the state space. In the last case, GSPN are indeed converted into CTMC for analysis. Furthermore, GSPNs allow to combine exponential arranges to model different time distributions (Desrochers, 1994), which is useful to capture specific dynamics of systems.

4 PROPOSED MODEL

The modeling proposed in this paper starts when a given web service requests a database operation. When received in the DBMS, this request is buffered, processed and buffered again, when an answer is ready to be replied back to requestor. When this happens, our modeling finishes. For this scenario, we model the subphases of a database transaction in GSPN: *Buffering* and *Processing*, as shown in Fig. 1.

Table 1 summarizes the model's notation.

Buffering Structure: The model firstly runs when the timed transition T_λ fires tokens toward the place B_I. Fired tokens model database requests and B_I models the DBMS buffer. The firing rate is defined by $1/d_\lambda$, where d_λ is the delay assigned to T_λ. The limit

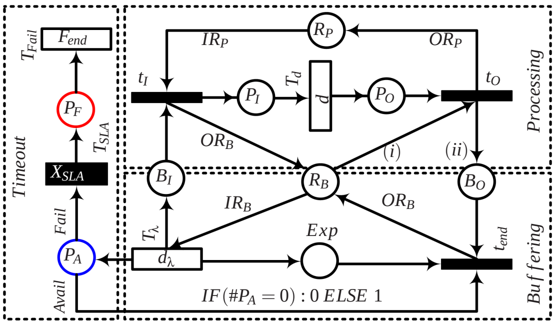

Figure 1: GSPN model.

Table 1: Notation of the GSPN model.

	Places
Exp	expectation of tokens to be processed;
R_B	resources available for buffering;
B_I	input buffer;
B_O	output buffer;
R_P	resources available for processing;
P_I	requests stored before processing;
P_O	requests stored after processing;
P_A	requests successfully attended;
P_F	requests that have failed.
	Transitions
T_λ	requests arrivals (delay d_λ);
T_d	requests processing (delay d);
T_{SLA}	requests failing (delay X_{SLA});
t_I	processing Input;
t_O	processing Output;
t_{end}	process exit point;
T_{Fail}	timeout exit point.

of tokens to be received in B_I is controlled by the number of tokens available in the place R_B, which are also shared with the output buffer B_O. In order to count the expectation of tokens into the model, and consequently to be able to estimate their performance, we create a place named Exp, that receives a copy of each token arriving in the system, and loses a token whenever the transition t_{end} fires.

Processing Structure: From B_I, tokens are moved to the place P_I, which models the processing phase. The place R_P controls the number of requests that can be concurrently processed. Tokens remain in P_I as long as it takes for them to be processed, which is modeled by the delay d of the transition T_d. After processed, T_d fires moving tokens to P_O from where the immediate transition t_O transfers them to the output buffer B_O. Remark that tokens leave the processing phase if and only if there exist enough resources in R_B. On the contrary they remain in P_O, waiting for buffering resources. From B_O, tokens immediately leave the model (by t_{end}), which represents the requestor being answered.

Timeout Structure: When the transition T_λ firstly fires, besides to send a token to B_I (performance model), it also copies it in the place P_A. The idea is to be able to estimate how many requests delay longer than a predefined response time. For that purpose, we assign to X_{SLA} the time we intend to wait until counting a failure. If the performance model reaches t_{end} first, P_A loses the token and the transaction is successfully completed. If X_{SLA} fires first, the transaction is also completed (because the arc *Avail* gets 0), but a failure is registered, i.e., a token reaches P_F.

Repository of Resources: Two repository comprise our model: R_B (buffering resources) and R_P (processing resources). From/to R_B and R_P, we connect arcs representing the number of tokens simultaneously moved when a source transition fires. We denote by IR_B and IR_P the resources consumption and by OR_B and OR_P the resources refunding from/to R_B and R_P, respectively. We assume that the number of tokens moved from/to the repositories is conservative.

Blocking: By sharing R_B with two consumers, B_I and B_O, we actually design a possibly blocking model. In fact if B_I consumes all resources in R_B, then tokens cannot leave the processing phase. At the same time, T_λ cannot fire any more tokens to B_I and, so, the model is deadlocked. We avoid this by assigning two logical conditions (*(i)* and *(ii)*) to the arcs that lead to the place B_O, where:

$$(i): \ IF \ \ (\#R_B < IR_B) : 0 \ \ ELSE \ \ IR_B;$$

$$(ii): IF \ \ (\#R_B < IR_B) : 0 \ \ ELSE \ \ 1.$$

The formulas (i) and (ii) are syntactically compliant to the *TimeNET* tool, adopted in this paper. Essentially, the condition (i) avoids the deadlock by firing t_O even without enough resources in R_B. When this is the case, the condition (ii) assigns 0 to the arc that leads to B_O and the token leaves the system. In practice, this models a situation when the DBMS rejects new transactions while the system is completely full, but as soon as any request is processed, transactions get to be received again.

4.1 Model Parameters

To be simulated, the GSPN model requires to be set up with parameters that connect it to the behavior of the system that has been modeled. We show in the following how such parameters can be derived.

4.1.1 Buffering Parameters

We first define a marking[2] for R_B, i.e., the number of resources available for buffering. This is defined according to the real buffer size, measured in the DBMS. Remark that each DBMS defines a particular amount of memory to be used for database operations and this can be tuned. The parameters we have to collect from the DBMS are:

- *Memory Pages* ($\mathcal{M}^P$): number of blocks of memory allocated for database operations;
- *Memory Page Size* ($\mathcal{M}_s^P$): amount of bytes assigned to each $\mathcal{M}^P$.

Remark that the greater the number of memory pages, the faster is the transfer from disk to memory, but the greater is rate of I/O communication, which is usually time expensive. On the other hand, the larger the memory page, the slower the transfer to memory.

As from

$$\mathrm{Av}^{\mathcal{M}} = \mathcal{M}^P \cdot \mathcal{M}_s^P$$

we have the amount of memory available to store messages from/to the database system, then the marking of R_B is such that

$$\#R_B = \mathrm{Av}^{\mathcal{M}}.$$

Once R_B is marked, we model its resources consumption by assigning weights to the arcs IR_B and OR_B. To define those values, we have to collect the mean size (bytes) of:

- Ω^{In}: messages received in the database system;
- Ω^{Out}: messages produced by the system as answer.

Thus, $IR_B = \Omega^{\mathrm{In}}$ and $OR_B = \Omega^{\mathrm{Out}}$. Ω^{In} and Ω^{Out} can be derived from samples of database transactions.

After assigning $\#R_B$, IR_B and OR_B to the GSPN, it becomes already possible to estimate the database *Buffering Response Time* (B^{RT}), taking into account the concept of *Mean Response Time* (M^{RT}). In Petri net, M^{RT} results from the *expectation* (ξ) of marking in a given place X ($\xi(X)$), with respect to: (i) the rate (λ) of requests; or (ii) the delay (d) between requests, i.e.,

$$(i) \ M^{RT} = \frac{\xi(X)}{\lambda} \text{ or, equivalently,} (ii) \ M^{RT} = \xi(X) \cdot d.$$

Tools like *TimeNET* syntactically implement these formulas respectively by

$$(i) \ M^{RT} = \xi(X)/\lambda \quad \text{and} \quad (ii) \ M^{RT} = E\{\#X\} \cdot d.$$

So, B^{RT} can be estimated as follows:

$$B^{RT} = \frac{\xi(B_I) + \xi(B_O)}{\lambda}.$$

Note that λ simply results from $1/d_\lambda$, where d_λ is the delay of the timed transition T_λ. In practice, B^{RT} represents the average of time spent by transactions before and after processing.

[2]"#" denotes the marking of a place p, for $\#p \in \mathbb{N}$.

4.1.2 Processing Parameters

This phase starts when t_I fires tokens towards the place P_I, finishing when the transition t_O releases them. There are basically four processing parameters to be derived: the delay d for the transition T_d, the marking for R_P and the weights for the arcs IR_P and OR_P, which connect the model from/to the repository R_P of processing resources.

Marking R_P requires to measure the system in order to collect the major number of operations simultaneously supported by the DBMS, without queueing requests. This can be done by gradually increasing the workload of requests until the point where the system starts to queue. This specific point can be detected by a sudden increase in the response time, when the processing resources are at all consumed.

Thus, $\#R_P$ receives the value of the workload applied before observing evidences of queue, and 1 is assigned to the weight of the arcs IR_P and OR_P.

Processing Response Time (P^{RT}):

$$P^{RT} = \frac{\xi(P_I) + \xi(P_O)}{\lambda},$$

where, $\xi(P_I)$ and $\xi(P_O)$ are respectively the expectation of marking in P_I and P_O.

4.1.3 Database Mean Response Time

From B^{RT} and P^{RT}, one can estimate the *overall database* M^{RT} by:

$$\Sigma^{RT} = \frac{\xi(Exp)}{\lambda} \quad \text{or, equivalently,} \quad \Sigma^{RT} = B^{RT} + P^{RT}.$$

5 MODEL ASSESSMENT

Consider the process shown in Fig. 2.

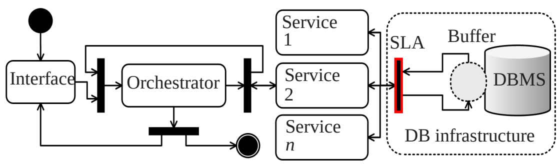

Figure 2: Evaluated Process.

The process starts when remote users invoke an orchestration service, via a web browser. Requests are organized according to the process workflow, and prepared to access remote services, which may access other services or interact with databases (dashed circle). Between a service and its consumer, a SLA regulates the QoS that is to be offered. Usually, this SLA is empirically constructed and, as a consequence, it is not rare to observe services delaying longer than the minimum necessary to match their contracts, which can entail legal penalties for providers, bad reputation for services, money loses for customers, and so on. Our goal here is to anticipate the behavior of the database service when it is variably accessed.

5.1 Database Construction

For the experiments that follow, we consider a partial structure of a relational database system, composed by the following structures:

- `PRODUCT` (<u>ProdID</u>, ProdDesc, ProdColor)
- `CLIENT` (<u>CliID</u>, CliName, CliAddress)
- `INVOICE` (<u>InvID</u>, InvDate, InvValue, ShipmentDate, DeadlineDate, FKClient#)
  - FKClient references CLIENT
- `MOVINVOICE` (Quant, Discount, UnitValue, Label, Status, FKInvoice#, FKProduct#)
  - FKInvoice references INVOICE,
  - FKProduct references PRODUCT

In order to access the database, we implement the following operations in *Relational Algebra*[3].

$$\begin{aligned} C &\leftarrow \Pi^*(Client) \\ I &\leftarrow \Pi^*(Invoice) \\ M &\leftarrow \Pi^*(MovInvoice) \\ P &\leftarrow \Pi^*(Product) \end{aligned}$$

Define query 1:

$$\begin{aligned} \Pi^*(\ & \sigma\,(I.ShipmentDate && <= '10/08/2015' \\ & \wedge I.DeadlineDate && <= '11/05/2015')) \\ & && (C \bowtie I \bowtie M \bowtie P) \end{aligned}$$

Define query 2:

$$\begin{aligned} \Delta &\leftarrow \Pi^*\,(\sigma\,(M.UnitValue >= 5.000,00 \\ &\quad \wedge \quad M.FKProduct = 23))\,(M) \end{aligned}$$

$$\begin{aligned} M &\leftarrow \Pi\ Quant, Discount, UnitValue, Status, \\ &\quad Label \leftarrow 'Profitable'\ (\Delta) \end{aligned}$$

Define query 3:

$$\begin{aligned} \Lambda &\leftarrow \Pi\ I.InvID, I.InvValue, P.ProdID, 'Delayed', 'Sold' \\ &\quad (\sigma \quad (I.DeadlineDate <= 'CurrentDate' \\ &\quad \wedge \quad I.ShipmentDate >= '10/10/2015' \\ &\quad \wedge \quad I.InvValue >= 100.000,00)) \\ &\quad\quad (I \bowtie M \bowtie P) \\ M &\leftarrow M \cup \{Quant, Discount, \Lambda\}. \end{aligned}$$

[3]Notation * refers to all attributes from a relation.

Query 1 returns the clients and their respective invoices, admitting that: (i) the products had already been shipped; (ii) the deadline for payment will be in at most a month. Query 2 updates the status of a financial transaction (Relation `MOVINVOICE`), labeling it as *profitable* if a given price matches. Finally, Query 3 inserts into a relation results brought from another relation, in a nested instruction.

For simulation, we have considered the respective query versions in *Structured Query Language* (SQL). Optimization and relevance have not been considered when implementing these queries, as we are actually more interested on their timed behavior.

5.2 Database Measurements

Now, we feed our GSPN model. We use an *Apache* tool called *JMeter* (Apache, 2014) to build a test plan that repeatedly executes each query. Then, we gradually increase the workload of requests to observe the point when queues start to appear. That is the point when input parameters are collected. Table 2 presents the inputs to our GSPN model.

Table 2: GSPN Input Parameters.

GSPN Input	Query 1	Query 2	Query 3
Buffering parameters			
$\#R_B$	$\mathcal{M}^P * \mathcal{M}_s^P = 1000 \cdot 4096$		
$IR_B \wedge OR_B$	1435	12	142
Processing parameters			
$\#R_P$	2	3	1
$IR_P \wedge OR_P$	1	1	1
d	286	215	122

Buffering parameters assign values for $\#R_B$, IR_B and OR_B. The marking of R_B is defined according to the DBMS configurations for $\mathcal{M}^P$ and $\mathcal{M}_s^P$. The impact of each operation when allocating resources from R_B, is modeled by the conservative weight of the arcs IR_B and OR_B. By definition, IR_B and OR_B are the measured input and output message sizes, Ω^{In} and Ω^{Out}, respectively.

Processing parameters assign values for $\#R_P$, IR_P, OR_P and d. The marking of R_P models how many instances of a given transaction is supported by the database server. Then, IR_P and OR_P model the impact of each transaction on $\#R_P$, and d represents the mean time required to simultaneously process $\#R_P$ (with no queue formation).

Remark that d represents the probability function that bridges the modeled behavior to the structure that stochastically represents this behavior. Therefore, the value to be assigned to d is obtained by measuring the M^{RT} of samples running in the real system. The number of samples to be considered has to be statistically relevant, usually evidencing a tendency for a stationary behavior. Remark also that every different query to be evaluated may lead to a different value for d and, therefore, has to be individually measured.

5.3 Contract Compositions

Now we exemplify our approach in the context of three challenging questions that are usually faced by engineers when composing SLA contracts. Then, we simulate the model to answer them.

5.3.1 Response Time

Consider the following service contract:

Contract 1: Let $W = \{w_1, w_2, \cdots, w_n\}$ be a set of workloads (requests per second - req/s) possibly arriving at a given DBMS. Which contract for mean response time (M^{RT}) could be guaranteed for w_i, $i = 1, \cdots, n$? As workload variation is quite common over a database structure, whenever w_i changes it becomes more and more difficult to predict the M^{RT} of a transaction, as the system gets to behave nondeterministically, buffering and releasing requests, consuming parallel resources, etc. This makes the rate of performance degradation and recovery unpredictable a priori. However, independently of this variable environment, a service provider is required to deliver his services with M^{RT} no less than the promised standard. Then, it is valuable to know, for each w_i, how many req/s the application supports before exceeding its contract.

We use our model to find out this information. After feeding the model with the statistical data in Table 2, we simulate it for each $w_i \in W$, applied over each proposed query. For the sake of clarity, we cluster our evaluations in three classes of workloads: w_{Light}, w_{Mid} and w_{Heavy}, meaning respectively 1, 5 and 10 req/s. *TimeNet* tool (Zimmermann, 2014) has been used to perform the simulations, considering a confidence level of 95% and a relative error of 10%. In order to check the accuracy of our estimations, we compare the estimated M^{RT} to the M^{RT} measured from the real database system, using the same workload levels. The results are presented in Table 3.

Table 3: Performance evaluation.

Query	*Source*	MRT under w_i			
		w_{Light}	w_{Mid}	w_{Heavy}	$\equiv$
1	System	260	623	1895	
Select	Model	329	405	1989	81%
2	System	278	640	1475	
Update	Model	218	482	1661	73%
3	System	177	815	1995	
Insert	Model	210	646	2193	92%

The accuracies of our estimations are respectively on the order of 81%, 73% and 92%, reaching 82% in a general case, which certainly is reasonable from a stochastic point of view.

For query 1, for example, we have estimated a M^{RT} of 329 *ms*, when simulating with w_{Light}, while the measured M^{RT} has been of 260 *ms*. When increasing the workload to w_{mid}, it has been estimated a M^{RT} of 405 *ms* against the measured 623 *ms*. With w_{Heavy}, we estimate that a transaction takes 1989 *ms* to answer, while the real transaction has taken 1895 *ms*.

As it can be seen, when we increase w_i, the system becomes less deterministic due to presence of queues. Nevertheless, the estimated M^{RT} keeps tracking the real system behavior.

Using our estimations, one can construct realistic contract clauses for services. Two examples are introduced next.

- Suppose, for example, that a service is required to be delivered in at most 700 *ms*. In this case, the model suggests that keeping the system under this contract requires to admit at most a w_{Mid} workload of requests.
- Now, suppose that we know the mean rate of requests arriving in the system. Consider that w_{Heavy} is expected. In this case, the construction of a contract for the M^{RT} would be quite easy. For example, for query 1 it could be defined a M^{RT} contract of 2000 *ms*; for query 2 the M^{RT} contract would be 1700 *ms*; and, for query 3 the M^{RT} contract would be 2200 *ms*.

5.3.2 Contracts with Acceptable Violations

Now, consider the following service contract:

Contract 2: For a given workload level w_i, which agreement for the M^{RT} could be guaranteed, in a way to admit at most 10% *of contract violation* ?

Now, instead of purely estimating the M^{RT}, we derive a refined version of it, admitting a certain percentage of contract violation. This may be a common clause to be defined by lawyers, but this is a quite complex decision for engineers. We show next how to estimate contract 2 by combining our performance and availability models.

For each workload level w_i, $i = Light, Mid, Heavy$, we gradually increase the M^{RT} assigned to the transition T_{SLA} of our availability model. Intuitively, by increasing the acceptable M^{RT} we decrease the failure rate. Table 4 presents the estimations *w* have obtained for query 1. A similar proceeding can be naturally adopted for the others.

In the second row, we present a range of possible SLA for the M^{RT}. Then we individually assign each

Table 4: Failure evaluation for query 1.

w_i	Suggested SLA for the M^{RT} (*ms*)						
	100	200	300	400	500	600	700
	Estimated failure rate (%)						
w_H	67	45	33	22	17	12	**10**
w_M	52	32	25	18	14	**9**	**7**
w_L	43	24	21	14	**10**	**8**	**6**

M^{RT} to the delay X_{SLA} of our availability structure. Afterwards, we simulate the model, variating w_i for each configuration, collecting the percentage of failure as an answer.

For example, by using the workload w_{Light}, we have estimated (Table 3) a M^{RT} of 329 *ms*. Nevertheless, one can observe in Table 4 that 500 *ms* is the minimum M^{RT} that ensures a failure rate of at most 10%. For w_{Mid}, equivalent condition is reached using a M^{RT} of 600 *ms*, while w_{Heavy} requires at least 700 *ms* to satisfy the contract 2.

5.3.3 Contracts with Acceptable Unavailability

Now, consider the following service contract:

Contract 3: Given a prefixed agreement for the M^{RT}, which is the highest workload supported by the system such that the contract is not violated more than 10%?

Contract 3 inversely approaches the problem with respect to contracts 1 and 2. It supposes that the service will be delivered in at most M^{RT}, and the aim is to discover which workload could break this rule. Moreover, it considers to accept a failure rate of at most 10%.

Once again we use query 1 to illustrate the contract 3. We firstly show the contract options for M^{RT}. As query 1 takes 286 *ms* to answer under minimum (Table 2), then we start our simulations by considering a M^{RT} of 300 *ms*. Afterwards, we increase this parameter for eight more scenarios and the results are shown in Table 5.

Table 5: Workload evaluation for query 1.

SLA	Estimated Workload	Failure Rate
M^{RT} (*ms*)	(*Req/sec*)	($\leq$ 10%)
300	**0,91**	10,00%
400	**1,11**	9,98%
500	**1,43**	8,68%
600	**1,92**	9,99%
700	**2,12**	9,99%
800	**4,76**	9,86%
900	**10,53**	9,89%
1000	**11,76**	8,97%

Consider, for example, that a service has to be delivered in at most 700 *ms*. In this case, we inform to the service supplier that his system can support, at most $2,12$ *req/sec* and, under this workload, the rate of failure would be stochastically less than 10%.

6 FINAL COMMENTS

In this paper, it has been presented a model to analyze resources allocation in databases infrastructures. The model allows to orchestrate and estimate the performance of a range scenarios, upon different workload profiles. Estimations can then be used as a tool to construct dataafdbase service contracts, besides to be useful for load balancing and scaling in database infrastructures, specially in service-oriented environments.

The approach is illustrated by an example where the performance of database operations is estimated. A comparison against measurements collected from the real database system is conducted to validate the results. The general accuracy of the estimations has been on the order of 80%.

In spite of encouraging results, some challenges remain in the database contracts composition. For example, it is still difficult to identify, among all database requests, those delaying longer than acceptable, which could be helpful to identify advanced classes of contracts. Moreover, we intend to adapt our approach to the optimizer-level, where concurrency could be taken into account. Cache effect analysis is another topic that compose our prospects of future research.

REFERENCES

Adams, E. J. (1985). Workload models for DBMS performance evaluation. In *Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, CSC '85, pages 185–195, New York, NY, USA. ACM.

Apache (2014). *jMeter 2.3.2*. http://jmeter.apache.org/.

Bruneo, D., Distefano, S., Longo, F., and Scarpa, M. (2010). Qos assessment of ws-bpel processes through non-markovian stochastic petri nets. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1 –12.

Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. (2001). Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles*, Alberta, Canada.

Desrochers, A. A. (1994). Applications of Petri nets in manufacturing systems: Modeling, control and performance analysis. IEEE Press.

Dewitt, D. J. and Gray, J. (1992). Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35:85–98.

Elhardt, K. and Bayer, R. (1984). A database cache for high performance and fast restart in database systems. *ACM Transactions on Database Systems*, 9:503–525.

Josuttis, N. (2008). *SOA in Practice*. O'Reilly, 1 edition.

Kartson, D., Balbo, G., Donatelli, S., Franceschinis, G., and Conte, G. (1995). *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., 1st edition.

Kim, S., Son, S., and Stankovic, J. (2002). Performance evaluation on a real-time database. In *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, pages 253–265.

Krompass, S., Scholz, A., Albutiu, M.-C., Kuno, H. A., Wiener, J. L., Dayal, U., and Kemper, A. (2008). Quality of service-enabled management of database workloads. *IEEE Data(base) Engineering Bulletin*, 31:20–27.

Lin, C. and Kavi, K. (2013). A QoS-aware BPEL framework for service selection and composition using QoS properties. *Int. Journal On Advances in Software*, 6:56–68.

Lumb, C. R., Merchant, A., and Alvarez, G. A. (2003). Façade: Virtual storage devices with performance guarantees. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 131–144, Berkeley, CA, USA. USENIX Association.

Marsan, M. A., Balbo, G., and Conte, G. (1984). A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. In *ACM Transactions on Computer Systems*, volume 2, pages 1–11.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE, v.77*, pages 541–580.

Nicola, M. and Jarke, M. (2000). Performance modeling of distributed and replicated databases. *IEEE Trans. on Knowl. and Data Eng.*, 12(4):645–672.

Osman, R. and Knottenbelt, W. J. (2012). Database system performance evaluation models: A survey. *Performance Evaluation*, 69(10):471 – 493.

Raibulet, C. and Massarelli, M. (2008). Managing non-functional aspects in SOA through SLA. In *Int. Conference on Database and Expert Systems Application*, Turin, Italy.

Ranganathan, P., Gharachorloo, K., Adve, S. V., and Barroso, L. A. (1998). Performance of database workloads on shared-memory systems with out-of-order processors. *Operating Systems Review*, 32:307–318.

Reisig, W. and Rozenberg, G. (1998). Informal introduction to petri nets. In *Lectures on Petri Nets I: Basic Models*, pages 1–11, London, UK. Springer-Verlag.

Reiss, F. R. and Kanungo, T. (2005). Satisfying database service level agreements while minimizing cost through storage QoS. In *Proceedings of the IEEE International Conference on Services Computing*, volume 2, pages 13–21, Washington, USA.

Rud, D., Schmietendorf, A., and Dumke, R. (2007). Performance annotated business processes in service-oriented architectures. *International Journal of Simulation: Systems, Science & Technology*, 8(3):61–71.

Schroeder, B., Harchol-Balter, M., Iyengar, A., and Nahum, E. (2006). Achieving class-based QoS for transactional workloads. In *Proceedings of the 22nd International Conference on Data Engineering*, Washington, DC, USA. IEEE Computer Society.

Sturm, R., Morris, W., and Jander, M. (2000). *Foundations of Service Level Management*. Sams Publishing.

Teixeira, M. and Chaves, P. S. (2011). Planning database service level agreements through stochastic petri nets. In *Brazilian Symposium on Databases*, Florianopolis, Brazil.

Teixeira, M., Lima, R., Oliveira, C., and Maciel, P. (2011). Planning service agreements in SOA-based systems through stochastic models. In *ACM Symposium On Applied Computing*, TaiChung, Taiwan.

Teixeira, M., Ribeiro, R., Oliveira, C., and Massa, R. (2015). A quality-driven approach for resources planning in service-oriented architectures. *Expert Systems with Applications*, 42(12):5366 – 5379.

Tok, W. H. and Bressan, S. (2006). DBNet: A service-oriented database architecture. *International Workshop on Database and Expert Systems Applications*, pages 727–731.

Tomov, N., Dempster, E., Williams, M. H., Burger, A., Taylor, H., King, P. J. B., and Broughton, P. (2004). Analytical response time estimation in parallel relational database systems. *Parallel Comput.*, 30:249–283.

Zhou, S., Tomov, N., Williams, M. H., Burger, A., and Taylor, H. (1997). Cache modeling in a performance evaluator for parallel database systems. In *MASCOTS*, pages 46–50.

Zimmermann, A. (2014). *TimeNET 4.0*. Technische Universität Ilmenau, URL: http://www.tu-ilmenau.de/TimeNET.