Analysis of Bursty Workload-aware Self-adaptive Systems

Diego Perez-Palacin
Dpt. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain
diegop@unizar.es

José Merseguer
Dpt. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain
jmerse@unizar.es

Raffaela Mirandola
Dip. di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
mirandola@elet.polimi.it

ABSTRACT

Software is often embedded in dynamic contexts where it is subjected to high variable, non-stable, and usually bursty workloads. A key requirement for a software system is to be able to self-react to workload changes by adapting its behavior dynamically, to ensure both the correct functionalities and the required performance. Research on fitting variable workload traces into formal models has been carried out using Markovian Modulated Poisson Processes (MMPP). These works concentrate on modeling stable workload states, but accurate modeling of transient times still deserves attention since they are critical moments for the self-adaptation. In this work, we build on research in the area of MMPP trace fitting and we propose a Petri net fine-grained model for highly variable workloads that also accounts for transient times. We analyze differences between models of adaptive software that accurately represent workload state changes and models that do not. We evaluate their performance and availability and compare the results.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Modeling Techniques]; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Petri nets*

General Terms

Design, Performance

Keywords

Self-adaptive software, Bursty workload, Markovian models

1. INTRODUCTION

Software is often embedded in dynamic contexts where it is subjected to changes in its execution environment, workload or requirements. Self-adaptive software allows capabilities to detect context changes and to react to them, managing its processing resources autonomously and allocating or releasing them dynamically. Among the multiple sources of change, in this work we deal with changes in the workload.

ICPE'12, April 22-25, 2012, Boston, Massachusetts, USA

The workload, for some kind of systems, is far from being stable but it presents high variability and shows *burstiness*, i.e., irregular spikes of congestion. This is a fact for example in networked and service-based systems, but not only [10]. If the workload model does not account for the existing *burstiness*, then the model analysis can lead to optimistic results; e.g., it declares a fair resource utilizations and probability of congestion, while in the real setting they would not be guaranteed.

Some formal methods that can model workloads considering the *burstiness* in the arrival rate are the Markov Arrival Processes (MAP) and a concrete subtype of them, the Markov Modulated Poisson Process (MMPP) [5]. Research on workload and network traffic fitting using MAPs and MMPPs have been already done and their results show an accurate modeling of the workload variability.

In particular, work on fitting MMPP and MAP parameters from workload traces with *burstiness* is very useful for the analysis of properties, such as performance or availability, of a wide range of systems. However, when we observe self-adaptive systems carefully, we realize that their optimal configurations are different depending on the workload they are receiving. These systems should adapt (e.g., provisioning or release of resources) during the *transient periods*, i.e., when the workload is becoming *bursty* and when the burst of arrivals is finishing. Usually there is no need for a software to change its context during stable periods of workload, it should have been adequately provisioned before, in fact during these transient periods.

Therefore, to properly analyze the performance or availability of self-adaptive systems, we need an accurate model of the workload. This model should include the transient times, even when they correspond to a small percentage of the total time (the rates normal and burst can last for hours while the change between them lasts just some minutes). Otherwise, results from model-based system analysis can be far away from results of the real working system. The reason is that the system starts the adaptation when anticipates the workload is close to be *bursty*. In this way, when the burst of requests arrive, the system is already in its optimal configuration. However, a system model whose workload does not care about transient times is not able to anticipate any workload change, and it will start its adaptations when the bursts of requests are already arriving. This can lead to too pessimistic performance and availability results from the model analysis.

In this work, we propose a model to take into account these transient periods. We exploit the research already done for two-state MMPP fitting and we aggregate to this MMPP a submodel of the transient times between normal arrival rate and bursty rates. Using the aggregation of models, we are able to analyze more accurately the non functional properties (NFP) of the software. To this end we build on the work done in [6, 2] for MMPP and MAP parameter

fitting and we extend the generated models to be able to deal with self-adaptation.

Related Work.

The parameter fitting of Markovian models such as MMPPs and MAPs is a promising research field. For example, works [7, 8, 11, 6, 15, 3, 2] propose MAP and MMPP parameter fitting techiques starting from traffic traces. Some of these fitting works also deal with the modeling of burstiness characteristic and use the index of dispersion as burstiness estimator.

In our work we build on the results obtained in [6, 2] to choose the estimators of the workload trace and fit a two-sate MMPP that models the same characteristics as the workload trace for these estimators. However, to the best of our knowledge, our work is the first one modeling the transient times between workload states and using them when evaluating workload-aware self-adaptive systems.

Paper Organization.

The paper is organized as follows: Section 2 describes MAPs, MMPPs and their parameter fitting from a workload trace. Section 3 explains the meaning of the *transient* time and proposes a model for its representation. In Section 4 we put together the MMPPs model and the new model for the transient time and we present the complete workload model of their aggregation. Section 5 presents an experimental analysis that shows the difference between considering or not the transient time in the workload model by evaluating the performance and availability requirements of a self-adaptive system. Section 6 concludes the paper.

2. MMPP'S AND MAP'S

Accurate characterization of real workload traces is a need to devise a proper workload model. For some kind of systems, e.g. networked ones, such characterization should capture the high variability of the requests as well as the fact that they *burst* in on the system sometimes [10].

MMPPs are suitable to model variability and autocorrelation for event generation. An MMPP is a stochastic process that has been extensively used to model event arrivals processes and network traffic [5, 6, 7], which is able to represent high variability and temporal dependencies in the arrivals. In an MMPP, the arrival rate at each moment is determined by the state of a continuous-time Markov chain (CTMC). So, when the chain is in state *i*, the arrival process is a Poisson process with rate λ_i. An MMPP with N states is defined by a NxN matrix Σ representing the CTMC and a vector Λ of N components representing the arrival rates in each state.

$$\Sigma = \begin{pmatrix} -\sigma_{11} & \sigma_{12} & ... & \sigma_{1N} \\ \sigma_{21} & -\sigma_{22} & ... & \sigma_{2N} \\ & ... & & \\ \sigma_{N1} & \sigma_{N2} & ... & -\sigma_{NN} \end{pmatrix}, \Lambda = (\lambda_1, ..., \lambda_N),$$

where $\forall\, i, j, \quad \sigma_{ij} \geq 0, \ \ \lambda_i \geq 0$ and $\forall\, i, \sum_{j:j\neq i} \sigma_{ij} = \sigma_{ii}$.

In this work, we consider MMPPs with two states. One of the states will represent the normal arrival rate (and we call it *normal*) and the other will represent the bursty arrival rate (and we call it *bursty*). A graphical representation of this two-state MMPP is given in Figure 1. λ_1, the *normal* arrival rate, and λ_2, the *bursty* arrival rate, are supposed to be much higher than transitions rates σ_{12} *and* σ_{21}.

A two-state MAP, Figure 2, can be seen as a continuous time Markov chain of two states, and the active state defines the arrival rate. In the chain, there can be transitions associated with the arrival of an event (called *completion* transitions, λ_{ij}, darker in the figure) and transitions that are not associated with event arrival (called *background* transitions, σ_{ij}). Moreover, when the chain is in state i, it can also generate arrivals with rate λ_{ii} without changing its state, modeled as a self-transition, λ_{ii}.

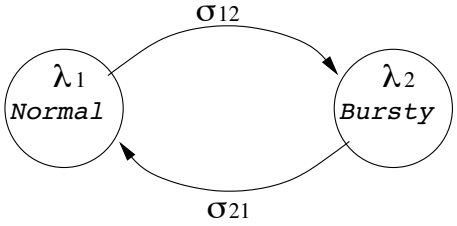

Figure 1: A two states MMPP

Formally, a MAP can be defined by two squared matrices D0 and D1, where $D0_{ij}, i \neq j$ represents the *background* transition rates from state *i* to *j*, $D1_{ij}$ describes *completion* transition rates, and $D0_{ii} = -(\sum_{j:j\neq i} D0_{ij} + \sum_j D1_{ij})$. Thus, $Q = D0 + D1$ is the infinitesimal generator matrix of the chain.

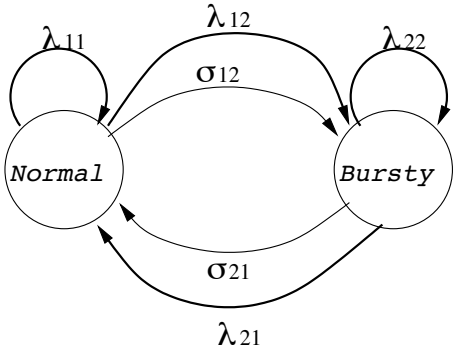

Figure 2: A two states MAP

An MMPP is a MAP that do not admit completion transitions that change the CTMC state, i.e., the elements not in the diagonal of $D1$ must be zero. Then, a two-state MMPP can be seen as a MAP whose matrices D0 and D1 are:

$$D0 = \begin{pmatrix} -(\sigma_{12} + \lambda_1) & \sigma_{12} \\ \sigma_{21} & -(\sigma_{21} + \lambda_2) \end{pmatrix}, D1 = diag(\Lambda)$$

2.1 MMPP fitting from a workload trace

Finding the characterizing values of a trace.

To fit a real workload trace to a two-state MMPP we just need to set its four parameters: $\lambda_1, \lambda_2, \sigma_{12}, \sigma_{21}$. To this end, we will use four characterizing values from the workload trace.

The first value is the *index of dispersion for counts* (IDC) of the trace. The IDC is frequently used as an estimator of the *burstiness* in a trace. The higher IDC value is, the more *burstiness* the trace has. In [6, 7] it is calculated as

$$IDC_t = \frac{var(N_t)}{E(N_t)}$$

where N_t is the number of arrival in an interval of t time units. So, the IDC is the variance in the number of arrivals in t time units divided by the mean number of arrivals in t time units. Since we are interested in the index of dispersion of arrivals in the steady state, we calculate

$$\lim_{t \to +\infty} \quad IDC_t$$

To calculate the IDC we use the algorithm presented in [9, 2]. This algorithm is able to estimate the index of disperson $IDC_{t \to +\infty}$ of a single workload trace.

For the rest of the characterizing values we take advantage of the work in [2], that indeed fits workload traces to MAP caring about the burstiness. Besides the IDC, these values are: the mean inter-arrival time of requests (m), the 50th percentile (i.e, the median) and the 95th percentile. Since in that work the authors are characterizing the burstiness of service times, the burstiness happens for high values of these service times, then making important to know the value for which the 95% of service times are lower. However, we are dealing with inter-arrival times, and the burstiness happens when the values of inter-arrival times are low. So, we prefer to know the value for which the 95% of times the inter-arrival time is higher than. For this reason, we use the 5th percentile instead of their 95th.

Experiment proposed.

As example of workload trace, we have used the monitored arrival times of requests to the FIFA 1998 World Cup site [16]. This has been the most complete example of workload trace we have been able to find. The timestamps are provided with granularity of one second and we have just used the requests that arrived to the Paris server region. Figure 3 shows the count of requests received by this region per minute. Since the workload was very low when the system was started and also the last days after the world cup, we have just concentrated in the middle days. We have used the arrivals of 34.7 consecutive days, then from minute 60,000 until minute 110,000. The arrivals in these 50,000 minutes have been considered in groups of 10 seconds and they are depicted in Figure 4. It is easy to see that the shape of the graph depicts a quite bursty workload. The selection of this time interval is not a restriction just to make the fitting algorithm work better but it exemplifies the kind of workloads we are really interested in. Since we are dealing with systems that are intended to continue working in the long term, we assume that the workload should not start and finish being low (as it happened to the World Cup website), but be always in the normal regime. So, we consider the first and last minutes as the system warm up and cool down, and we consider only the world cup days where the system was most used.

Fitting MMPP parameters.

The characterizing values of the trace are the following. The number of requests that we have dealt with is 140,998,569. The mean inter-arrival time of requests is 0.021276 seconds (i.e, close to 47 requests per second), calculated as the number of received requests divided by $3 \cdot 10^6$ (the amount of seconds in 50,000 minutes). The median (percentile 50th) of the inter-arrival times is 0.0159744408 and the 5th percentile is 0.00367 (this is, the inter-arrival time of the 95% of requests was higher than this value). The IDC is 686,200, we admitted a tolerance of $1 \cdot 10^{-7}$ for its calculation using the algorithm in [2]. The amount of time that the

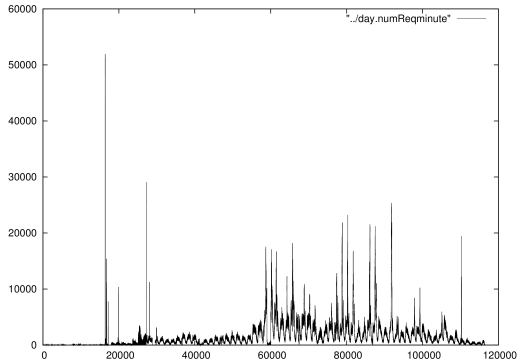

Figure 3: Requests per minute received in Paris region

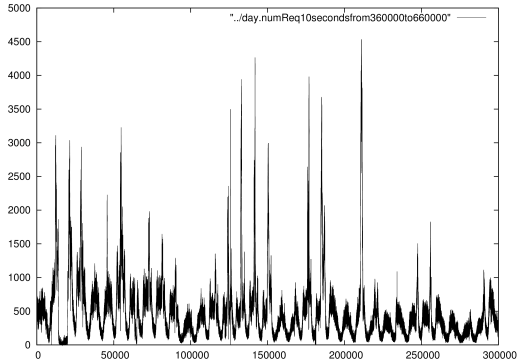

Figure 4: Requests every 10 seconds

algorithm considered approximate to infinite and for which the algorithm stopped was 45,140 seconds.

From these characterizing values, we fitted the MMPP. To fit the mean, 50th and 5th percentiles we have used the same equations as [2]. To fit the ICD, we have used the equation in [6, 7] that concretely deal with two-state MMPP parameters[1].

The results are:

$$\sigma_{11} = \sigma_{12} = 0.0000001314169$$

$$\sigma_{22} = \sigma_{21} = 0.0000273058047$$

$$\lambda_1 = 45.5395329586$$

$$\lambda_2 = 350.195877$$

As expected, we can see that the mean sojourn time in each state, $\sigma_{12}^{-1}, \sigma_{21}^{-1}$, is orders of magnitude higher than the mean requests inter-arrival times, $\lambda_1^{-1}, \lambda_2^{-1}$.

[1]This equation is $IDC_{t \to +\infty} = 1 + \frac{2\sigma_{12}\sigma_{21}(\lambda_1 - \lambda_2)^2}{(\sigma_{12}+\sigma_{21})^2(\lambda_1\sigma_{21}+\lambda_2\sigma_{12})}$

2.2 GSPN workload model

An accurate workload model with burstiness, as the one proposed by the MMPP, is a necessary and very useful tool for the eventual analysis of systems that execute under such conditions.

GSPNs [1] are broadly used to model the behavior and workload of systems and also as analyzable models to predict properties of software systems. GSPNs have already been used to analyze some properties of self-adaptive software systems, such as performance and energy [13, 12, 14]. Since our workload model should represent the injection of requests in the system in the same language as the behavioral system model, we pursue the proposed MMPP workload model but in terms of GSPN.

Since both GSPNs and MMPPs represent markovian processes, we can get a GSPN with the same behavior as the MMPP in a quite straightforward manner. This GSPN, the one in Fig. 5 representing the two state MMPP in Fig. 1, has as many places as states the MMPP, in this case P_1 and P_2 (for *normal* and *bursty*, respectively). Another place, $P_{arrivals}$, will mean the injection of requests in the system, i.e. injection of tokens in the GSPN that represents the behavior of the self-adaptive system. The time transitions T_{12} and T_{21} represent the MMPP change of state, then their firing rates are σ_{12} and σ_{21} obviously. The last two transitions, $T_{arrival1}$ and $T_{arrival2}$, represent the arrival rates in the MMPP, therefore their firing rates are λ_1 and λ_2 and they feed the $P_{arrivals}$ place.

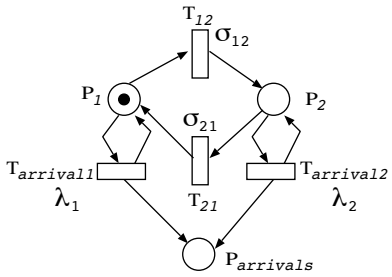

Figure 5: GSPN for the two states MMPP

3. MODELING TRANSIENT TIME BETWEEN STATES

3.1 Problem statement

As declared in the Introduction, a self-adaptive system needs some time to perform corrective actions (e.g., provisioning or release of resources) to fit into the new execution context. In systems whose adaptations depend on the workload variations, such adaptations should happen when the system changes from *normal* to *bursty* or vice versa, i.e., the system adapts to the environment during the transient times between states.

When looking at the real workload trace in Figure 6 we observe that such transient time, although fast, is not immediate, it lasts for around 41.6 minutes, starting around 850 and ending around 1100 ($\frac{1100-850}{6} = 41.6$). The figure shows a period of 250 minutes which corresponds to the zoom in the range from 209,500 to 211,000 in Figure 4. The transient time is assumed to be fast w.r.t. the mean sojourn time in each stable state that last for many hours. Our workload model should reflect the transient time accurately since in this period the self-adaptive system:

- perceives that the workload is leaving the normal state and the burst of arrivals are near to arrive, and
- performs its adaptations to change its configuration to a new one able to withstand the burst of requests.

In a two-state MMPP the transient time is not modeled as we can see in Figure 7. This figure represents a workload trace generated by the fitted MMPP in Section 2 and we observe that the change from *normal* state (arrival rate around 455 requests each 10 seconds) to *bursty* state (around 3500 requests during 10 seconds) is abrupt, no transient time is perceived.

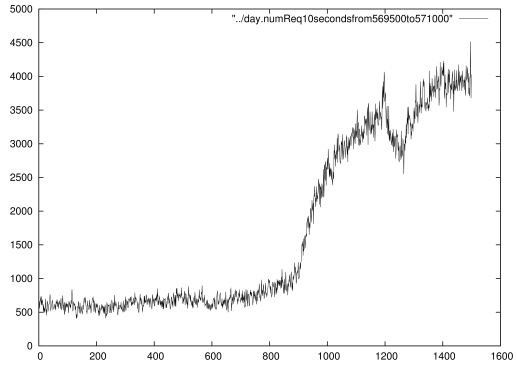

Figure 6: Real workload trace: focus on the increment

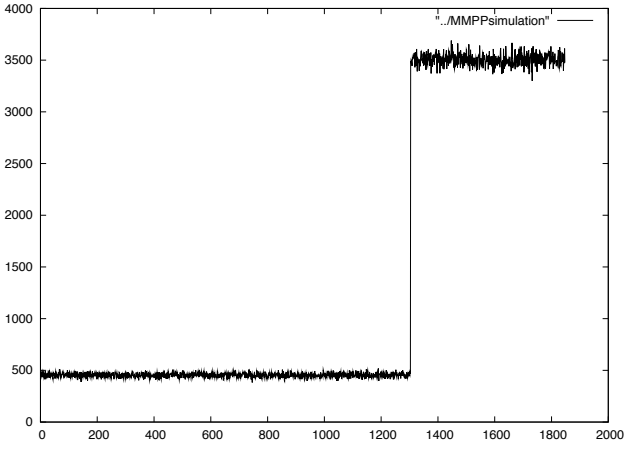

Figure 7: Workload trace modeled by the MMPP

3.2 Setting parameters of workload model

We pursue a GSPN to model the transient times in the real workload trace, i.e., the increments and decrements in the arrival rates of the requests. The zones of increment or decrement can be characterized by three parameters:

- the well-known λ_1 and λ_2,

- the amplitude of the zone, we call it mt_{inc} or mt_{dec}, they are measured in seconds, and they represent the mean amount of time that the workload is increasing from *normal* state to *bursty* state or decreasing from *bursty* to *normal* respectively,

- and additionally, from these parameters we can also calculate the acceleration of the curve in the zone, mr_{inc} or mr_{dec}, in $requests \cdot seconds^{-2}$.

In the following we describe how these parameters can be obtained from a real workload trace. Algorithm 1 shows the case of the calculation of the mean amount of time that the workload is increasing.

First (line 1 in Algorithm 1), we apply the technique presented in Section 2.1 to get λ_1 and λ_2.

Second (line 2), we go all over the counts[2] in the workload trace. Let us call $count_j$ the number of requests received by the count in position j. We find each j such that $count_{j-1} < \lambda_1 \leq count_j$, we call it $candidate_j$. That is, the candidates are the counts where the arrival rate has changed from being under the mean for the *normal* state to be over the mean.

Third (lines 3..9), for the first $candidate_j$ we find the first k, $k > j$, such that $(\lambda_1 > count_k) \vee (count_k > \lambda_2)$.

- If the first condition holds, we can discard $candidate_j$ since it means that the workload is not incrementing, but it had just exceeded the mean for a while and it has returned under the mean again (this is the usual behavior when the workload is in a stable state).

- If the second condition holds, we keep $candidate_j$ since we could have found a period of increment in the workload from *normal* to *bursty* arrival rates, this period is $[j, k]$.

Fourth (lines 10..17), for each $[j, k]$ period we have to discover whether it can be considered as a real workload increment or not. We assume that a real increment happens when the counts between $[j, k]$ increase constantly in a *coarse-grained* view of the workload.

As *coarse-grained* we mean that we zoom-out the counts in order to mask the short-term variability. To create the *coarse-grained* view, we reduce the $k - j$ monitored counts to N values, where each $N_n, 0 \leq n < N < (k - j)$, counts the number of arrivals in a period of length L. L is a choice and represents how much coarse will be the study. A too low L will not avoid the short-term variability (and then we will not realize that the workload is truly increasing), and a too big L will recognize as periods of constant increment some that should not be. Once L value is chosen, N is calculated as the largest value for which $N \cdot L \leq (k - j)$, i.e, $(k - j) < (N + 1) \cdot L$. If $N \cdot L \neq (k - j)$, we obviate in the study the last values $k - j - N \cdot L$ of the interval, that is, the interval to work changes from $[j, k]$ to $[j, j + N \cdot L]$. Now, we sum up in N_n the values of each group of L counts. Therefore, $N_n = \sum_{i=0}^{L-1}(count_{n \cdot L+j+i})$.

To finish the fourth step, we decide that $[j, k]$ is a real constant increment. Ideally, a constant increment happens if the number of counts N_n are increasing values, i.e, if $\forall\, n \in \{1..N-1\}$, $N_{n-1} < N_n$. However, we found that in every increment interval in the trace, there is at least one unexpected count of arrivals that is very different from its neighbor counts (too less or too much) that are also visible in the coarse-grained view. This unexpected count prevents satisfying the *for all* in the previous condition. To solve it, we add a percentage of tolerance $tol \in \mathbb{R}$, $0 \leq tol \leq 1$. Then, the amount of counts N_n, $n \in \{1..N-1\}$ that must satisfy the condition $N_{n-1} < N_n$ is reduced from $N - 1$ (i.e, all counts) to $(1 - tol)(N - 1)$.

Fifth, if it has been decided that the interval represents a constant increment in the coarse-grained view, we get a parameter to characterize the interval $[j, k]$: the amplitude $t_{inc} = k - j$. Besides, we can derive more parameters from the interval such as the acceleration r_{inc} in the request arrival, calculated from λ_1, λ_2 and t_{inc} as $r_{inc} = \frac{\lambda_2 - \lambda_1}{t_{inc}}$.

Sixth, we repeat the third, fourth and fifth steps for all $candidate_j$.

Finally, using the discovered t_{inc} and derived r_{inc} in each iteration, we get the values of mt_{inc} and mr_{inc} as the mean of them (lines 18..23).

Algorithm 1 Parameter estimation

Require: Workload trace with the *count* of arrivals
Ensure: mt_{inc}
1: (λ_1,λ_2) ← MMPPfitting($count$);
2: $candidates$ ← findCandidates($count$, λ_1);
3: $intervals \leftarrow \emptyset$;
4: **for all** $candidate \in candidates$ **do**
5: k ← getFirstCrossingValue($count$,$candidate$,λ_1,λ_2);
6: **if** $count_k$>λ_2 **then**
7: $intervals$ ← addInterval($intervals$, [$candidate$, k]);
8: **end if**
9: **end for**
10: L ← chooseL(); tol ← chooseTol();
11: **for all** $interval \in intervals$ **do**
12: N ← calculateN(L, $interval$);
13: $subtrace$ ← makeCoarse($trace$, $interval$, N);
14: **if not** isContinuousIncrement($subtrace$, tol) **then**
15: $intervals$ ← discardInterval($intervals$, $interval$);
16: **end if**
17: **end for**
18: $numberOfIntervals \leftarrow 0$; $incrTime \leftarrow 0$;
19: **for all** $interval \in intervals$ **do**
20: $incrTime \leftarrow incrTime + interval$.amplitude;
21: $numberOfIntervals \leftarrow numberOfIntervals$+1;
22: **end for**
23: $mt_{inc} \leftarrow \frac{incrTime}{numberOfIntervals}$;
24: **return** mt_{inc}

We perform the same steps to discover the periods of time where the workload is decreasing. Using these periods, we will obtain mt_{dec} and mr_{dec}.

Note that mr_{inc} and mr_{dec} are real positive values ($\mathbb{R}^+$). So, we are assuming a constant acceleration and deceleration in the workload during transient time. On the one hand, this linear increment in the arrival rate is more accurate than the previously assumed immediate increment. Besides, the linearity in the increment/decrement corresponds to the long-term view of the increment, since we are still modeling the variability in the short-term. On the other hand, we are approximating to be linear any workload increment/decrement between states. Other possible representations of the workload increment/decrement are possible, but the identification of the kind of increment in the coarse-grained view is more complicated since we would also come into the field of curve fitting.

3.3 GSPN model for the transient time

The GSPNs in Figure 8 (a) and (b) model the transient time from *normal* to *burst* (increment in the arrival rate of requests) and from *burst* to *normal* (decrement in the arrival rate of requests), respectively.

[2]Remember that a count means the number of requests received in 10 seconds.

GSPN model: from normal to burst.

A key point is that, during the transient period, the arrival of requests are modeled as tokens created in place $P_{arrivals}$ at a variable rate. This rate will be $\lambda_1 + \lambda_{inc} \cdot \#P_{inc2}$ since transitions $T_{arrival1\prime}$ and $T_{arrivalInc}$ provide the tokens[3]. The former transition generates the workload of the *normal* state, λ_1, while the latter transition generates the increment of requests[4].

A token in P_{inc1} means that the system enters in the transient state so leaving the *normal* one. Then, every σ_{inc} units of time a new token is set in P_{inc2} to precisely generate the increment of requests. When the number of tokens in P_{inc2} is w_1, it means that the transient time has completed and the system enters in the *bursty* state P_2 by firing transition t_{inc2}.

Although not yet observed in the figure, transition t_{12} will fire when the normal arrival rate of request in the system has finished, hence to start this transient period.

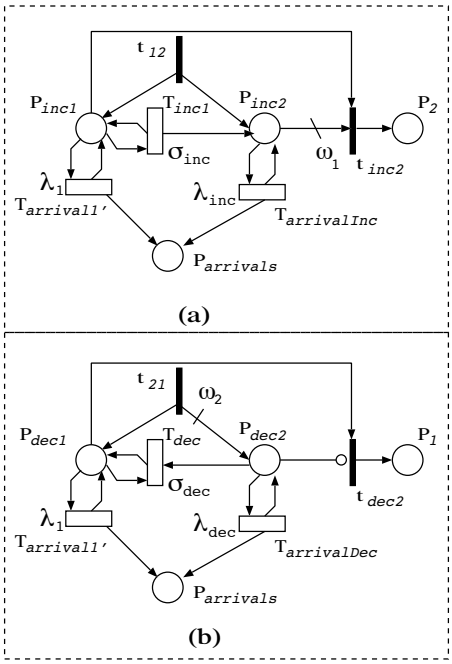

Figure 8: GSPNs: (a)increasing and (b)decreasing arrival rates of requests

Fitting GSPN parameters.

We use the four parameters computed in the previous subsection, $\lambda_1, \lambda_2, mr_{inc}$ and mt_{inc}, to set the parameters of the GSPN, $w_1, \lambda_{inc}, \sigma_{inc}$.

First, note that the modeling of transient times increases the state space for the analysis. Fortunately, we can decide the amount of increment we allow. For the transient time that models the change from *normal* to *bursty*, the state space grows linearly with parameter $w_1 \geq 1$, and we can freely decide its value. The rationale is that w_1 corresponds with the amount of token variability in P_{inc2}. Then, observe that P_{inc2} can have tangible markings in the interval $[0, w_1 - 1]$, while markings where $\#P_{inc2} = w_1$ are vanishing and do not affect for the state space analysis. This allowed variability entails that we model $w_1 - 1$ increments in the workload between λ_1 and λ_2. So, each token will increase the arrival rate in $inc = \frac{\lambda_2 - \lambda_1}{w_1}$ units. This inc is the value of λ_{inc}. Finally, we calculate how fast are created the tokens in P_{inc2}, this is, we calculate σ_{inc} of T_{inc1}. This transition fires $w_1 - 1$ times for each transient period, and it should fire in mt_{inc} time units. So its firing rate $\sigma_{inc} = \frac{w_1 - 1}{mt_{inc}}$.

Now it can be easily seen that we preserve the short term variability in the workload increment since the arrival rate is still based on stochastic processes exponentially distributed.

GSPN model: from burst to normal.

The differences between this model, in Figure 8 (b), and the previous one are:

- t_{21} starts this transient state by setting w_2 tokens in P_{dec2}. Again, parameter w_2 represents the amount of complexity we can afford to model the decrementing period in the workload. The size of the state space will be w_2+1 times the state space of the workload model without decrementing period.

- Tokens in P_{dec2} decrease at rate σ_{dec}, when P_{dec2} is empty the system enters in the *normal* state P_2. The σ_{dec} firing rate is $\frac{w_2}{mt_{dec}}$.

- The transient state generates requests at rate

$$\lambda_1 + \lambda_{dec} \cdot \#P_{dec2}.$$

  Where λ_{dec} is $\frac{\lambda_2 - \lambda_1}{w_1}$.

4. COMPREHENSIVE WORKLOAD MODEL

So far we have proposed GSPN models separately, one model for the two characteristic states, in Figure 5, and two for the transient times, the increment of the workload in Figure 8(a) and the decrement in Figure 8(b). Our challenge now is to merge these three GSPN models to get a single one that cares for burstiness and transient times as required by self-adaptive systems.

Before merging the GSPNs we need to slightly modify the net of Figure 5: we remove the arc from T_{12} to P_2 and the arc from T_{21} to P_1. The rationale behind this modification is that we want to avoid the immediate change between *normal*, P_1, and *burst*, P_2, and vice versa.

The resulting GSPN is the one in Figure 9. We have used the composition operator for GSPNs formally defined in [4]. The essence of the operator is easy to understand, it overlaps the transitions (places) with the same name. For example, the place $P_{arrivals}$ appears in the three nets, however in the resulting net it appears only once, having as input arcs all the input arcs of the three original places.

The expert reader can argue that the GSPN in Figure 9 can be equivalent to a M-state MMPP where $M = 1 + w_1 + w_2$. In that case, the parameters of that M-state MMPP with the same characteristics as our workload model would be:

[3] $\#P_i$ is the number of tokens in place P_i.

[4] It is worth noting that we are considering *infinite server* semantic for all transitions

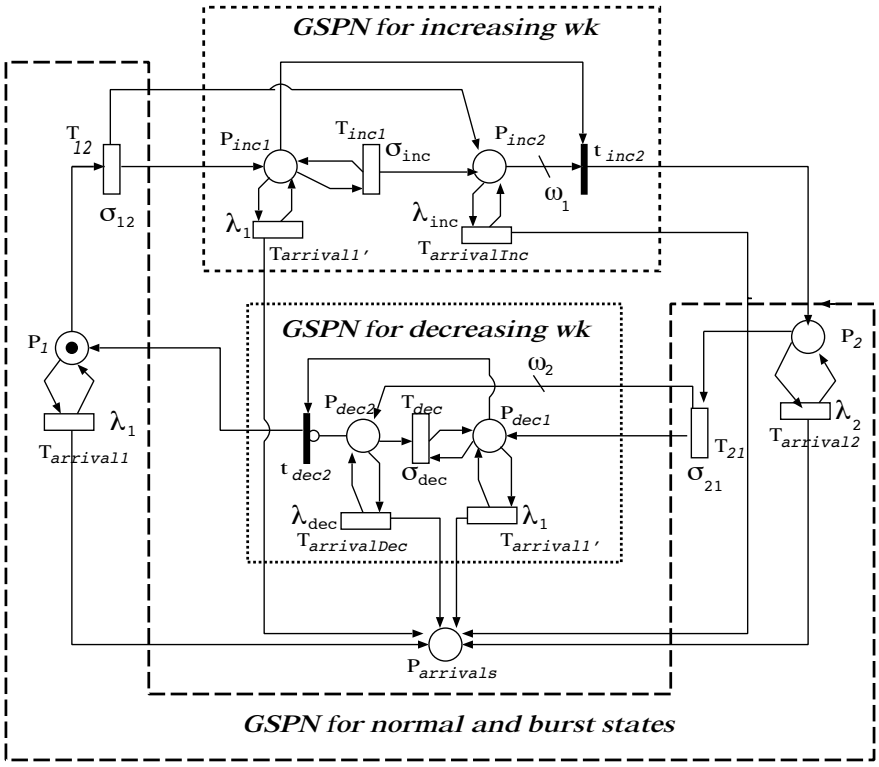

Figure 9: Complete workload model

$$\Sigma = \begin{pmatrix} -\sigma_{11} & \sigma_{12} & 0 & ... & & & 0 \\ 0 & -\sigma_{inc} & \sigma_{inc} & 0 & ... & & 0 \\ & ... & & ... & & ... & \\ 0... & 0 & -\sigma_{22} & \sigma_{22} & 0 & ... & 0 \\ 0... & & 0 & -\sigma_{dec} & \sigma_{dec} & 0 & ... & 0 \\ & ... & & ... & & ... & \\ 0... & & & & 0 & -\sigma_{dec} & \sigma_{dec} \\ \sigma_{dec} & 0... & & & & 0 & -\sigma_{dec} \end{pmatrix}$$

$$\Lambda = (\ \lambda_1, \lambda_1+\lambda_{inc}, ..., \lambda_1+(w_1-1)\lambda_{inc}, \lambda_2, \lambda_1+w_2\lambda_{dec}, ..., \lambda_1+\lambda_{dec})$$

Then, a question arise: could that M-state MMPP be directly obtained from the workload trace using the technique presented in Section 2 for two-state MMPP?

The answer is yes. Nevertheless, there are some restrictive challenges to obtain the characterizing values of a M-state MMPP. These are: the algorithm to fit parameters of a M-state MMPP is much more time consuming and the estimation of its parameters are much more prone to inaccuracies. Moreover the current techniques to fit MMPP parameters do not directly deal with our problem (gaining accuracy in the transient times models).

5. EXPERIMENTAL ANALYSIS

In this section, we illustrate the results obtained in our experimentation. To this end we have considered a very simple system with different workload models: first MMPPs and second our GSPN model, that includes the transient time between workload states. A third experiment is used as a benchmark for comparing the accuracy of the obtained results, it is a system simulation having a real workload trace, the one in Figure 4.

The system we use in this experimentation is a very simple software made of only one activity that requires on average 3ms of processing time [5]. There is a single processor executing a maximum of ten concurrent requests, queueing and serving them following a FIFO policy. Requests above ten are rejected.

We assume that requirements to architect the system are:

- R1- availability: *at least 99% of requests must be served, and*
- R2- performance: *the mean response time should be lower than 1 second*.

Note that the response time is not a critical requirement, since the maximum length of queue of requests to be served is nine, and they are served in a mean of 3ms. On the contrary, requirement R1 is the critical one.

When we analyzed the system considering a workload model without burstiness (i.e., taking into account the mean inter-arrival time derived from the real trace), the requirements were satisfied.

On the contrary, when taking into account the arrival in bursts, the analysis of the system showed that R1 cannot be guaranteed.

[5]To be able to compare approaches without including more variables that can distort results, we assume that the mentioned processing time is exponentially distributed with mean 3ms

A possible solution passes through the addition of a second processing resource. Now, having two processing resources, the system is able to satisfy both R1 and R2 also during bursty periods. However, the second processing resource has been added just to allow the requirements satisfaction during the periods of burstiness, which represents the worst-case scenario for the system. So, during the normal arrival rate periods, there is a wasting of resources.

We can use the model proposed in Section 3 to take into account the workload variability. To this end, we consider a system enhanced with a monitoring component. The monitor is a passive observer that measures the system workload. Then, the monitor notifies to a separate component, which acts as a controller, when the workload is changing and when to add a second processing resource. In the same way, it also decides to switch off one of the processing resources when the workload decreases. So, the system deployment is no longer static but it is dynamically adaptable.

We have set the following parameters for the self-adaptive system:

- The maximum arrival rate of requests that can be served by only one processing resource is the 80% of its maximum capacity. In other words, the controller decides to add a new processing resource when the workload goes above $\frac{1}{3ms} \cdot 0.8 \approx 266$ requests per second.
- The maximum arrival rate of requests that can be served using both processing resources is 40 requests per second. When the workload rate is under this value, the second processing resource is shut down.
- Booting and shutting down times of the processing resources is one minute.

In the following, we explain the set-up of each experiment and the obtained results. After, we compare and discuss results.

MMPP workload model.

As MMPP workload we used the one already calculated in Section 2. We composed the MMPP model, in GSPN terms, with the GSPN that models the behavior of the described system. We analyzed the resulting GSPN and obtained the following results:

The percentage of requests rejected is 1.43%, so the availability is 98.56%; and the mean response time is 5.3ms.

Then, R2 is satisfied while R1 cannot be guaranteed.

MMPP with transient times workload model.

Using the MMPP parameters already calculated in Section 2 we applied the process described in Section 3 to identify in the trace in Figure 4 periods of coarse-grained-constantly-increasing workload.

The parameters L and tol have been set to 5 minutes and to 0.2, respectively. Then, the workload parameters $mt_{inc}, mr_{inc}, mt_{dec}$ and mr_{dec} are:

$$mt_{inc} = 5192s \quad mr_{inc} = 0.58 requests \cdot s^{-2}$$

$$mt_{dec} = 3770s \quad mr_{dec} = -0.8081 requests \cdot s^{-2}$$

Following the procedure described in Section 3 we defined the structure of the GSPN models for the transient times. We then used the previous results as parameters of these GSPN models.

To complete the model definition, we decided the amount of affordable increment in the state space as $w_1 = 10, w_2 = 9$. Using these values, the remaining GSPNs parameters $w_1, \lambda_{inc}, \sigma_{inc}, w_2, \lambda_{dec}$ and σ_{dec} have been derived.

The GSPN modeling the workload has been obtained as described in Section 4 by composing the MMPP part with the GSPN derived for the transient times. Next, we composed the GSPN workload model with the GSPN that represents the behavior of the system. We analyzed this GSPN and we obtained the following results:

The percentage of requests rejected is 0.56%, so the availability is 99.44%; and the mean response time is 5ms.

Hence, R1 and R2 are satisfied.

Real workload trace execution.

For validation purpose, we have implemented a simulator of the system described in the example. We run the simulator and we injected the requests following the real workload trace.

We have obtained the following results:

The percentage of requests rejected is 0.05%, so the availability is 99.95%; and the mean response time is 3.7ms.

With the simulation and the real workload both requirements are satisfied.

	MMPP	**MMPP with transient times**	**Real trace**
Availability	98.56%	99.44%	99.95%
Performance	5.3ms	5ms	3.7ms

Table 1: Evaluation results with different input workload

5.1 Results discussion

Looking at Table 1 we can observe that the results obtained with both the MMPP model and MMPP with explicit transient time workload model are pessimistic with respect to the real ones. Indeed, the analysis of the models produced results showing lower availability and higher average response time with respect to the results obtained by the system simulation using the real workload trace. However, the results obtained with the MMPP including the transient time model are better than the ones obtained with the simple MMPP and closer to the system simulation results.

Actually, in this simple example we can see that the expected rejection probability of requests from model analysis with MMPP is $\frac{1.43}{0.05} = 28.6$ times higher than the calculated by simulating the real trace. Adding the transient times to the MMPP model, we have reduced this error to be $\frac{0.56}{0.05} = 11.2$ times higher; so, we have brought the result a 60% closer to the real one. Besides, the conclusion from the analysis of the model with MMPP workload would be that the proposed adaptive solution for the system does not satisfy availability requirement. This decision would be wrong because the actual system satisfies it.

Regarding the mean response time, adding the transient times to the MMPP model, we have just reduced the error of the results from being 1.43 times the real ones to be 1.35 times.

Note that, although all the experiments regarding requirement R1 seem to produce very similar results, this is not the case since availability is used to be measured as the "number of nines". In other words, if we compare a system with 99% of availability and another one with 99.9%, the latter is not just 0.9% more available than the first one but it is ten times more available. In our experiments, the availability obtained with the MMPP workload model without transient times resulted 28.6 times lower than the availability of the system with the real workload. Adding the transient time between states to the workload model we have been able to reduce the error of around the 60%, of course this is still not enough to guarantee results very close to the real ones.

6. CONCLUSION

Modern techniques to model high variable workloads and burstiness are based on Markovian models such as Markov Arrival Processes and Markov-Modulated Poisson Processes. They offer a powerful theory to model workload. Moreover, since they are based on Markovian processes, they can be easily included into the rest of the system model if it is also Markovian, such as the broadly used Markovian queueing networks or stochastic Petri nets. In this work we have identified a gap in the workload modeling for self-adaptive systems when using the MMPPs that makes inaccurate the analysis results. This gap refers to the modeling of the transient time between workload states.

This transient time is not modeled in MMPPs, because they focus on modeling stable workload states. Although these transient times may not be important for static systems, they are crucial when analyzing worklod-aware self-adaptive systems. To solve this challenge we have exploited previous results on MMPP fitting and we have proposed a model based on Petri Net taking into account the arrivals during the transient time between states. The obtained model has then been integrated in a Petri net describing the MMPP, so allowing a more complete representation of the workload.

A first experimentation comparing the results obtained with the proposed model and the classical MMPP models tested against a real trace workload, showed an increment in the analysis accuracy when the transient time are taken into account.

Besides, form our experimentation it is evident that, although we have reduced the errors in the analysis results, there is still a gap between model analysis results and real simulation ones.

At present, we are working on the implementation of our methodology on a real testbed, to assess its effectiveness through a more comprehensive set of real experiments. Another direction that deserves further investigation is the representation of the workload transient times when there are more than two stable states. Since the addition of the transient time models increases the state space of the model to analyze, if the MMPP that models the stable states has more than two states, it may not be possible to create the incrementing and decrementing transient time model between any two states. Contrarily, we should search which state transitions deserve attention to model their transient times and which ones do not deserve it.

Acknowledgments

This work has been partially supported by the European Community's Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498), by CICYT DPI2010-20413, by Fundación Aragón I+D and by the IDEAS-ERC Project 227977-SMScom.

7. REFERENCES

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley Series in Parallel Computing - Chichester, 1995.

[2] G. Casale, N. Mi, L. Cherkasova, and E. Smirni. Dealing with burstiness in multi-tier applications: Models and their parameterization. *IEEE Trans. Software Eng. To appear*, 2012.

[3] G. Casale, E. Z. Zhang, and E. Smirni. Kpc-toolbox: Best recipes for automatic trace fitting using markovian arrival processes. *Perform. Eval.*, 67:873–896, September 2010.

[4] S. Donatelli and G. Franceschinis. PSR Methodology: integrating hardware and software models. In *ICATPN*, volume 1091 of *LNCS*, pages 133–152, 1996.

[5] W. Fischer and K. Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. *Perform. Eval.*, 18:149–171, September 1993.

[6] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *Selected Areas in Communications, IEEE Journal on*, 9(2):203 –211, feb 1991.

[7] H. Heffes and D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *Selected Areas in Communications, IEEE Journal on*, 4(6):856 – 868, sep 1986.

[8] A. Horváth and M. Telek. Markovian modeling of real data traffic: Heuristic phase type and map fitting of heavy tailed and fractal like samples. In M. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, volume 2459 of *Lecture Notes in Computer Science*, pages 267–282. Springer Berlin / Heidelberg, 2002. 10.1007/3-540-45798-4_17.

[9] N. Mi, G. Casale, L. Cherkasova, and E. Smirni. Burstiness in multi-tier applications: symptoms, causes, and new models. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 265–286, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[10] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance impacts of autocorrelated flows in multi-tiered systems. *Perform. Eval.*, 64:1082–1101, October 2007.

[11] H. Okamura and T. Dohi. Faster maximum likelihood estimation algorithms for markovian arrival processes. In *Quantitative Evaluation of Systems, 2009. QEST '09. Sixth International Conference on the*, pages 73 –82, sept. 2009.

[12] D. Perez-Palacin and J. Merseguer. Performance sensitive self-adaptive service-oriented software using hidden markov models. In *Proceedings of WOSP/SIPEW '11*, pages 201–206, 2011.

[13] D. Perez-Palacin, J. Merseguer, and S. Bernardi. Performance aware open-world software in a 3-layer architecture. In *WOSP/SIPEW '10*, pages 49–56, New York, NY, USA, 2010. ACM.

[14] D. Perez-Palacin, R. Mirandola, and J. Merseguer. Enhancing a qos-based self-adaptive framework with energy management capabilities. In *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS*, QoSA-ISARCS '11, pages 165–170, New York, NY, USA, 2011. ACM.

[15] T. Rydén. An em algorithm for estimation in markov-modulated poisson processes. *Comput. Stat. Data Anal.*, 21:431–447, April 1996.

[16] World Cup 1998 Access logs. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`. 1998.