Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 261 (2010) 5–21

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Stochastic Models for Dependable Services

Katinka Wolter[1,2] Philipp Reinecke[1,3]

Institute of Computer Science
Takustr. 9
Free University Berlin
14195 Berlin, Germany

Abstract

In this paper we investigate the use of stochastic models for analysing service-oriented systems. We propose an iterative hybrid approach using system measurements, testbed observations as well as formal models to derive a quantitative model of service-based systems that allows us to evaluate the effectiveness of the restart method in such systems. In cases where one is fortunate enough as to have access to a real system for measurements the obtained data often is lacking statistical significance or knowledge of the system is not sufficient to explain the data. A testbed may then be preferable as it allows for long experiment series and provides full control of the system's configuration. In order to provide meaningful data the testbed must be equipped with fault-injection using a suitable fault-model and an appropriate load model. We fit phase-type distributions to the data obtained from the testbed in order to represent the observed data in a model that can be used e.g. as a service process in a queueing model of our service-oriented system. The queueing model may be used to analyse different restart policies, buffer size or service disciplines. Results from the model can be fed into the testbed and provide it with better fault and load models thus closing the modelling loop.

Keywords: Fault model, performance model, dependability, adaptivity

1 Introduction

Service-based systems are widely used today. Performance and dependability of service-oriented systems are determined by different factors such as performance and dependability of the single services as well as of the infrastructure connecting them. Since the components of service-oriented systems are typically spread over different locations they can very often only be observed by their behaviour in a network. Neither the status of the service itself nor the status of the system executing it can be determined by the user. Proprietary applications add to the difficulties in accessing and observing services and their performance as well as dependability. The inability to internally monitor and measure service-oriented systems raises the need

[1] This work is supported by the German Research Council (DFG) under grants Wo 898/2-1 and Wo 898/3-1

[2] Email: katinka.wolter@fu-berlin.de

[3] Email: philipp.reinecke@fu-berlin.de

doi:10.1016/j.entcs.2010.01.003

for mathematical models as to, at least, evaluate different system configurations of different fault-tolerance mechanisms in a model.

Quantitative models build abstractions of systems that cannot directly be observed. A useful model can only be defined if some knowledge of the system and its operating environment exists or educated assumptions can be made.

We propose an approach that consists of experiments as well as models in order to derive a solid stochastic model which we will illustrate in an evaluation of the restart method. We use a testbed of a service oriented system being an implementation of our Multi-Level Fault-Injection framework (MLFIT [12]) to obtain system data. Fitting models to our measured data allows us to use an analytical representation of our experimental data in a formal model. But to obtain realistic observations from the testbed suitable models of load, faults and disturbances must be included. As of now we use very simple models that still need improvement. We demonstrate the importance of a good fault model using data obtained from Sandesha, an implementation of Web Services Reliable Messaging (WSRM).

Our approach is suitable for arbitrary models. We demonstrate it using the restart model and a simple queueing model of the restart method. Analogously, one could formulate a stochastic Petri net [8] or a PEPA model [4]. The necessary parameters for the model can be obtained from our testbed, which in turn needs some stochastic models to produce realistic results.

In summary, an iterative modelling approach is necessary in order to carry out some meaningful formal modelling and analysis of service-oriented systems. In this paper we illustrate the iterative approach as applied to the restart method. The restart method enhances performance and dependability of service-oriented systems. The restart method retransmits messages that have not been acknowledged within a given time. This method can be applied in systems whose internal state can neither be monitored nor controlled by the user. The user, or an engine on the client side can attempt to improve the service's quality by reissuing a service requests.

In order to study the restart method, we apply our iterative approach: In the next section we first derive the formal restart model and present some important properties and results from its analysis. We then proceed to refine the model based on practical measurements in our testbed in Section 3. In Section 4 we discuss advantages and disadvantages of measurements in real-world systems and testbeds, before finally, in Section 5, going full-circle on our iterative method by applying insights from measurements to the quantitative modelling approach. Section 6 concludes this paper.

2 The Restart Method

The restart method directly relates to a very simple abstract model [19]: Let the random variable T denote the task completion time or service response time. The task is assumed to complete according to some probability distribution with density function $f(t)$ and distribution function $F(t)$, and it is assumed that each retry terminates the previous attempt. Then, the question guiding our analysis is: In order to minimise various moments of the completion time, what is the best time τ to restart?

Using our simple abstract model, we may answer this question by investigating the following inequality:

$$E[T] < E[T - \tau | T > \tau] \tag{1}$$

for different completion time distributions. E.g., the distribution of a lognormally distributed task completion time and the distribution under restart is illustrated in Figure 1 where each restart comes at a time penalty (cost) of 0.1 time units.

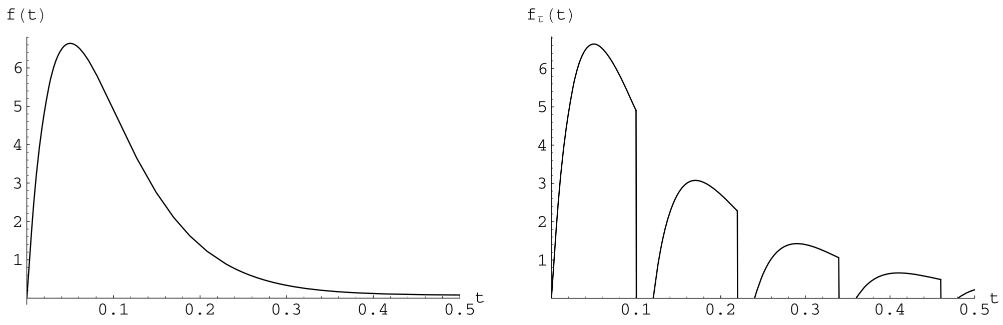


Figure 1. Distribution without and with restart

A number of elegant and interesting results can be derived from (1)[18,19,20]: A restart should not necessarily take place at ‘fixed’ times between successive tries. Only when one wants to minimise the first moment of completion time $\mathrm{E}\,[T]$, the best strategy is to retry at fixed intervals. We choose τ that minimises

$$E[T] = M(\tau) + \frac{(1 - F(\tau)) \cdot \tau}{F(\tau)} \tag{2}$$

where

$$M(\tau) = \int_0^\tau t f(t)\, dt$$

denotes the partial first moment.

More simply, we find that the retry should take place at the time point τ^* where the hazard rate is reciprocal to the inverse of the resulting completion time.

$$\frac{1 - F(\tau^*)}{f(\tau^*)} = \mathrm{E}\,[T_{\tau^*}]$$

The hazard rate

$$h(t) = \frac{f(t)}{1 - F(t)}$$

is known from reliability analysis as the failure rate of system components. In our model the hazard rate describes the *potential* of completion. In case of an increasing hazard rate the potential of task completion increases and one should not restart the task. If the hazard rate decreases so does the likelihood of task completion over time and one should immediately restart. In practise one rarely encounters a strictly

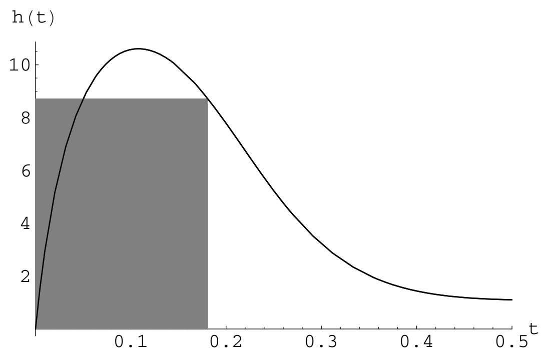


Figure 2. Surface equals rectangle rule

monotonic hazard rate – neither decreasing nor increasing. Mostly, the hazard rate first increases and then decreases. In these situations our approximation should be applied. It allows us to formulate an engineering rule as shown in Figure 2. The optimal restart time τ^* can be found where the integral of the density equals the rectangle determined by that particular point on the density function.

For higher moments of completion time, it is better to initiate restarts at a fast pace at the beginning, and then slow down. For the distribution of the completion time, also interesting results hold. For instance, to maximise the probability of making a deadline, one should do restarts at time points at which the hazard rates are equal. A special case that obeys this criterion is to restart at equi-distant time points, but this is not always the global optimum (it could in fact correspond to a local or global sub-optimum).

To be more specific, two properties are of particular relevance: higher moments can benefit more from restart and if restart is beneficial then more restarts reduce the moments of completion time further, i.e. $E[T_\tau] < E[T] \Leftrightarrow E[T_\tau] < E[T_{\tau^{K+1}}] < E[T_{\tau^K}] < ... < E[T], \quad K >= 1$ which is demonstrated in Figure 3.

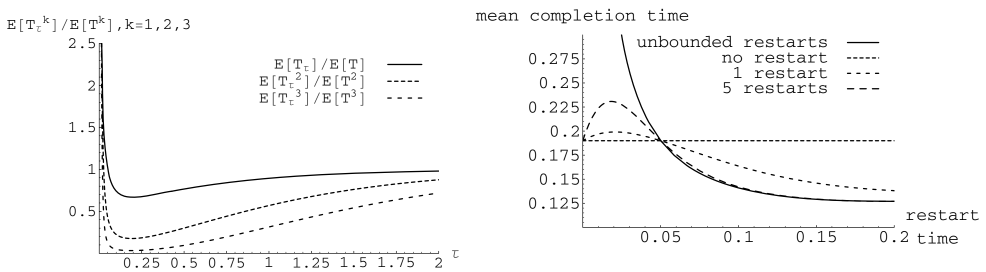


Figure 3. Reduction of moments under restart

Based on these findings, an algorithm that computes the optimal restart timeout can be devised. Since in practical situations we usually do not know the analytical density function of the completion time distribution, the algorithm [21] is based on the histogram of observed completion times.

The restart method gives rise to different modelling issues. First, it constitutes a model itself which is defined through a probability distribution. If the probability distribution of task completion time is known, the distribution under restart as well as optimal restart times for different metrics of interest can be computed as we will show.

Second, the restart method lends itself to mathematical modelling. Different formalisms may be used to model the restart method. We choose a simple queueing model, but other models such as a Petri net or a PEPA model [4] can be formulated as well.

3 System Evaluation

While certainly useful in the analysis of the restart method, the simple abstract model presented in the previous section has a number of shortcomings: First, it assumes that a simple random variable is an accurate model for completion times. Second, the analysis is performed on distributions where the analytical density function is known which often is not the case.

Such problems are common when applying a purely model-based approach. They can be addressed by studying real systems, as we will illustrate in this section. Let us begin with the question whether there is sufficient variability in real-world completion times, and whether completion times can be modelled simply by their distribution, without taking into account possible correlation between subsequent attempts.

In [10,11] we studied completion times for HTTP GET invocations of randomly selected web sites. While this investigation was not aimed specifically at service-oriented systems, the characteristics of HTTP over the transmission control protocol (TCP) are a good starting point, since current SOA (service-oriented architecture) systems are based on SOAP (simple object access protocol) communication, which is commonly encapsulated in HTTP requests.

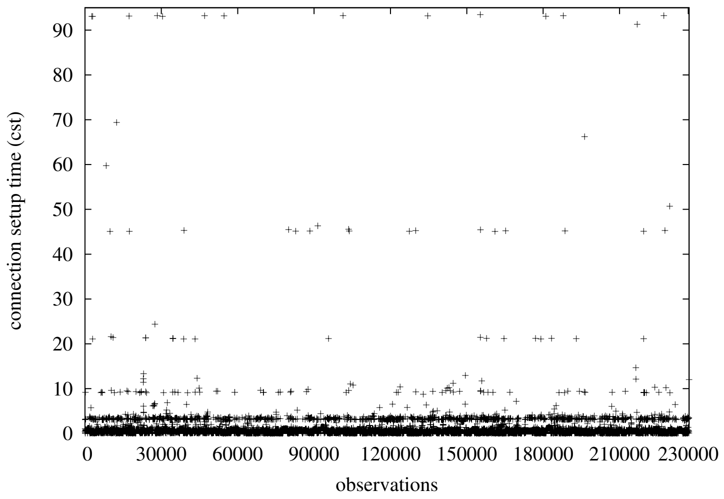


Figure 4. Observation of connection setup times

Figure 4 [10] shows connection setup times observed on over 56 000 randomlychosen URLs. We observe that, while the majority of connection setups finish rather fast, there are several strata of samples at multiples of 3 seconds. These can be attributed to the TCP retransmission timeout (RTO) used to detect packet loss during the connection setup stage [7]. In a service-oriented system, such delays result in drastically increased message-transmission times. Restart may help to avoid these delays.

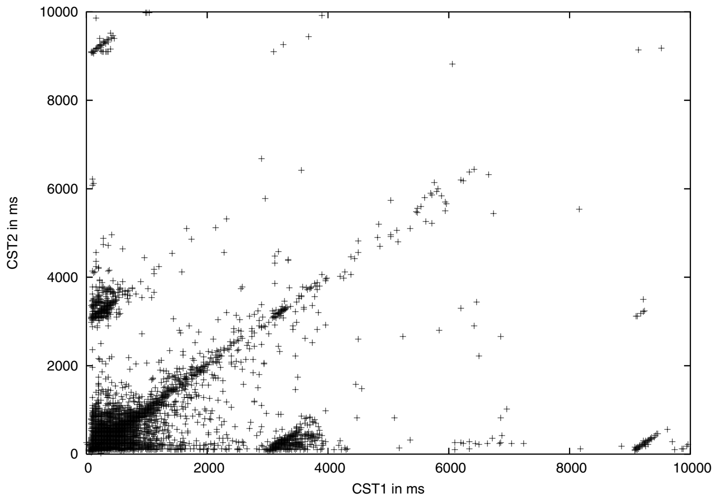


Figure 5. Scatter plot of CST1 versus CST2

In order to investigate correlation between attempts, we performed another series of experiments where we downloaded the same object twice in succession. Figure 5 shows a scatter plot of the first and second connection setup time. If connection setup times were highly correlated, we should observe a straight diagonal. The observation that for the values of the TCP RTO timeout (3s, 6s, etc.) there are distinct off-diagonal clusters implies that extreme delays are often not correlated. In particular, the clusters on the CST1 axis indicate that a large connection setup time on the first attempt may be followed by a very small connection setup time on a retry.

3.1 Testbed of a Service-Oriented System

Experimental evaluation of service-oriented systems is especially difficult. These systems are usually distributed over various physical locations and very complex. Therefore measurements are blurred by various undesired and unspecified effects. Typically, neither the internal state of the system components nor the communication paths are known to a degree that would enable complete explanation of observed data.

To address these problems, in [12] we proposed the Multi-Level Fault-Injection Testbed (MLFIT) framework. The framework is aimed at providing a testbed for service-oriented systems where faults can be injected at various levels, based on realistic models of faults and disturbances. This allows controlled experimentation with a service-oriented system, while at the same time keeping experiment complexity manageable.

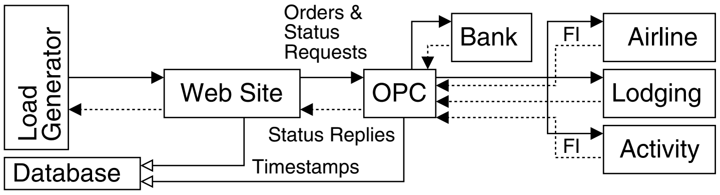


Figure 6. Architecture of the SOA testbed

Currently, MLFIT is implemented using SUN's SOA reference application Java Adventure Builder [15]. We are interested in quantitative properties such as dependability, performance and adaptivity. These properties are to be evaluated from a user's perspective. The user accesses a complex distributed system but observes the system behaviour only through its web interface and the response times seen there.

We want to apply the restart method and are, therefore, interested only in timing behaviour. We issue requests to the Adventure Builder system and monitor response times to determine the empirical distribution function of our restart model.

The user will not see any system details, and for setting up the restart model we would not want to dive into the system either. Since the testbed is no real, widely distributed system, some characteristics of such systems must be simulated.

The Adventure Builder consists of a web site as an entry point for the user. This web site is connected to an order processing center service which dispatches requests to the bank and the airline, the lodging service and an activity service in parallel. The order processing center collects the other services' replies and delivers them to the user.

We limit ourselves purely to timing disturbances as these are of interest for our model. To imitate the effects of a large network between the services and between the services and the user we use fault-injection at selected points as indicated by *FI* in Figure 6. Requests are generated by a load generator and measurements are stored in a data base. The testbed uses two stochastic models, one for load generation and one for fault-injection.

Both models are currently fairly simple: the fault-injection applies 3% IP packet loss and the load model consists of 10, 25, and 50 users, respectively, each performing 100 bookings.

Figure 7 shows the histograms of the response times for a scenario with 10 users and samples of the airline with and without fault-injection. Please note the different scaling in the plots. This data is approximated using a phase-type distribution [9] and used to parameterise the queueing model in Section 5.

A continuous phase-type distribution (PH) is the time to absorption in a continuous-time Markov chain [9]. It is commonly represented as a tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ of the initial probability vector $\boldsymbol{\alpha}$ and the sub-generator matrix $\mathbf{Q}$.

Figure 8 shows two typical phase-type models which we have both used to ap-

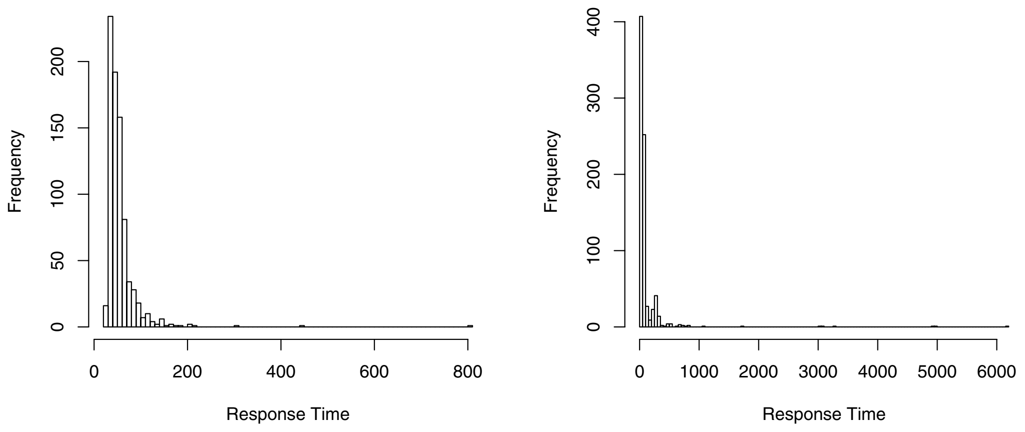


Figure 7. Histograms of data sampled for the airline with 10 users without fault-injection (left) and with fault-injection (right)

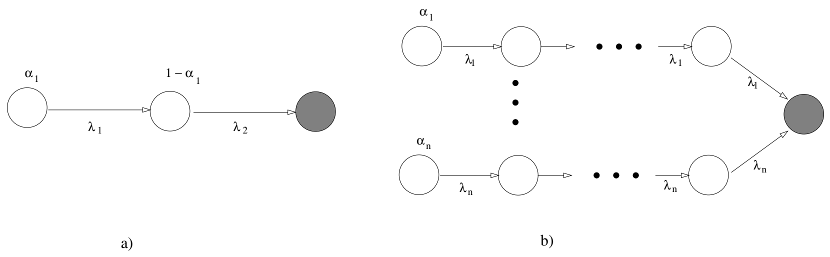


Figure 8. Acyclic phase-type distribution model (ACPH(2)) (a) and Hyper-Erlang distribution (b)

proximated our data. Figure 8 a) shows a general acyclic phase-type distribution of order 2 (ACPH(2)). Figure 8 b) shows a Hyper-Erlang model with n Erlang branches of possibly different length.

In order to fit our data using a phase-type distribution we have to determine the order of the distribution, and the fitting method. A previous evaluation of fitting phase-type distributions to transmission times in a WSRM scenario [13], as discussed in the next section, has shown that an ACPH(2) as shown in Figure 8) obtained using moment-matching [17] are sufficient to capture the relevant characteristics. We prefer this model as it can conveniently be used in our queueing model.

The parameters of the fitted ACPH(2) model are listed in Table 1.

	α_1	λ_1	λ_2	$E[T]$	c^2
`nla10`	0.98199	0.035568	0.035557	55.73	0.51
`lla10`	0.03317	0.000660	0.013733	123.075	10.24

Table 1
ACPH(2) Parameters ($\alpha_2 = 1 - \alpha_1$).

The first data set (`nla10`) consists of observations from the testbed without fault-injection. In this situation response times in the testbed exhibit little variation as the testbed is hosted on several machines in the same lab connected by a dedicated network. This can be seen in the low squared coefficient of variation $c^2 = 0.51$. With 3% IP packet loss (data set `l1a10`) the expected response time is much longer and the duration of response times varies much more as can be seen in the larger squared coefficient of variation of the second fitted model.

In Section 5 we use both fitted models as service time distributions in a queue including restart. One may use the phase-type models for other purposes such as emulating a service's behaviour in a testbed or to include a delay with this distribution in a Petri net or a PEPA model.

4 Restart in WSRM

For a closer investigation we have implemented the restart algorithm in web services reliable messaging (WSRM). We have used the Sandesha WSRM implementation [1]. A more advanced fault-injection is used to mimic effects of an unreliable network. Faults are generated according to a two state Gilbert-Elliot model as shown in Figure 9. This model is commonly used to study packet loss in network models [3,2]. We considered three different loss levels,

- $S_1 = 0.05s$ lossy, $120s$ loss free
- $S_2 = 1s$ lossy, $30s$ loss free
- $S_3 = 1s$ lossy, $8s$ loss free

where the time durations denote the mean time of exponentially distributed length.

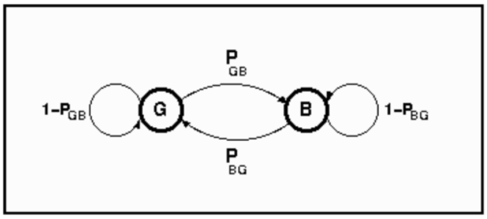


Figure 9. Gilbert-Elliot loss model

In the restart experiments we have used three different oracles to determine the restart interval:

- Fixed Intervals (4s)
- Jacobson/Karn
- QEST Algorithm

Fixed Intervals uses static intervals of length 4 seconds. The Jacobson/Karn oracle uses the algorithm commonly found in TCP implementations [6], while the QEST algorithm uses a timeout computation based on our restart model.

We compare these algorithms based on different metrics. First we use the moments metric mentioned earlier combined with a fairness metric. The first moment

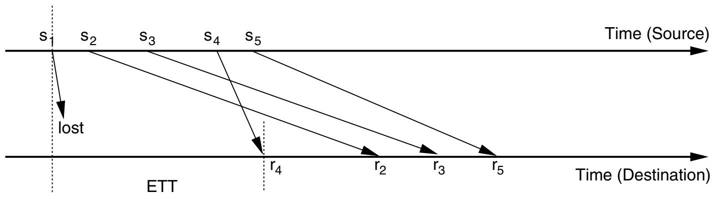


Figure 10. Definition of metrics

of transmission time (ETT) is estimated by the time between sending the first copy of a packet and its first receipt at the destination as shown in Figure 10. The fairness metric URC (unnecessary resource consumption) counts the number of transmissions that were unnecessary in retrospect as they were not needed to guarantee transmission of the packet.

We have approximated this data using different phase-type distributions. We considered three classes of acyclic phase-type distributions (ACPH): Two-state ACPHs (ACPH(2)) computed using moment-matching [17], Hyper-Erlang (HErD) distributions with 15 Erlang branches (cf. Figure 8) fitted using the G-FIT tool [16], and full acyclic phase-type distributions of order 30 that were matched to the data with the PhFit tool [5].

Note that, for analytical purposes, PH distributions of low order are preferable. On the other hand, higher-order distributions are capable of capturing characteristics of the data more accurately.

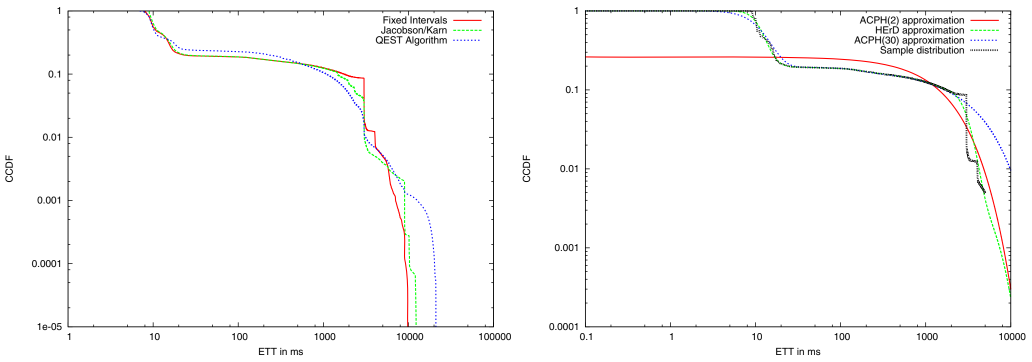


Figure 11. CCDF for fault model S_3 (left) and fitted phase-type distributions (right)

We plot in the two following figures traces using fault model 3 and several fitted distributions for fault models 1 and 3. We find that the data is seriously influenced by the fault model. In consequence, the best fit is obtained by different distributions.

The figure of the traces shows steps at the values of the TCP retransmission timeouts (3s, 6s, 9s, ...). In [13] we have used the fitted model as the service-time distribution in an M/G/1 queue.

4.1 Another Metric: Adaptivity

Until now we have validated the restart algorithm and the different oracles using the expected transmission time and the fairness metric URC. A dynamic metric to

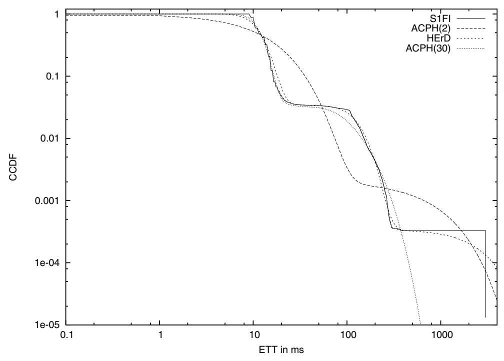


Figure 12. CCDF for fault model S_1 and fitted phase-type distributions

evaluate the effectiveness of the restart method is system adaptivity. Our metric of adaptivity is based on a payoff metric p_i indicating the usefulness of a sequence of trials $i = 1, \ldots, N$. Each trial comes with a benefit that relates p_{i-1} and p_i. A positive decision increases the benefit $\Delta_i = \frac{p_i + p_{i-1}}{2}$, while a neutral decision conserves the previous benefit, $\Delta_i = p_i$ and a negative decision has zero benefit, i.e. $\Delta_i = 0$. The maximum accumulated benefit is limited by the number of decisions $N - 1$. System adaptivity is expressed as the distance to the optimum benefit, i.e.

$$\text{Adaptivity} = \sum_{i=2}^{N} \Delta_i/(N-1).$$

This metric of adaptivity takes on values between 0 and 1 where the perfect adaptive system has adaptivity 1.

Adaptivity of a system or an algorithm is defined by means of the payoff metric. The payoff metric can be based on ETT and URC as

$$P = \frac{1}{1 + \alpha ETT + (1-\alpha)URC},$$

where $0 \leq \alpha \leq 1$ denotes a weighting factor expressing the relative importance of timeliness vs. fairness.

We can more elaborately define the payoff through the savings metric, SAV, which is defined as follows. The amount of time saved by restarting instead of waiting for the first transmission to finish provides the third performance metric which we consider here. This time is measured as the difference between the time required for the first transmission, $r_{i1} - s_{i1}$, and the Effective Completion Time (ETT), i.e. the time that was actually required to transmit the message, possibly including restarts:

$$SAV_i = (r_{i1} - s_{i1}) - ETT_i.$$

Note that a failed first transmission attempt leads to SAV $= \infty$.

In the definition of the savings-based payoff function P^2 a threshold value SAV^* is used. Restart is considered useful if the reduction in completion time (SAV_i) is

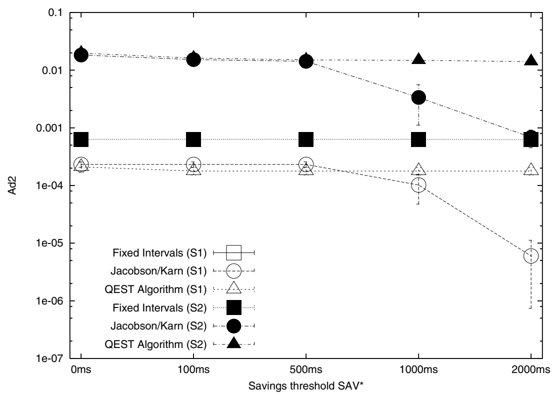


Figure 13. Adaptivity

larger than the threshold SAV^* (e.g. SAV^* may be 100 ms):

$$P^2_{SAV^*}(m_i) = \begin{cases} 1 \; : \; SAV_i > SAV^* \\ 0 \; : \; \text{else} \end{cases} \quad . \tag{3}$$

This payoff function is also bounded to $[0, 1]$ and has its optimum at 1.

Figure 13 shows the adaptivity of the different restart oracles as measured in terms of the savings metric. Note that the QEST oracle is successful at reducing completion times even at high SAV^* thresholds.

5 The Quantitative Models

We can compute optimal restart timeouts for our two models by minimising (2). Figure 14 shows the expected response time under restart over the restart interval τ. Minimisation of (2) gives us $\tau_1 = 83.95$ and $\tau_2 = 58.72$ for the `nla10` and `l1a10` models, respectively. According to the prediction from the analytical model, these timeouts should result in mean response times $E[T_1] = 52.89$ and $E[T_2] = 64.56$. Interestingly, the lower response time is achieved using the larger restart interval.

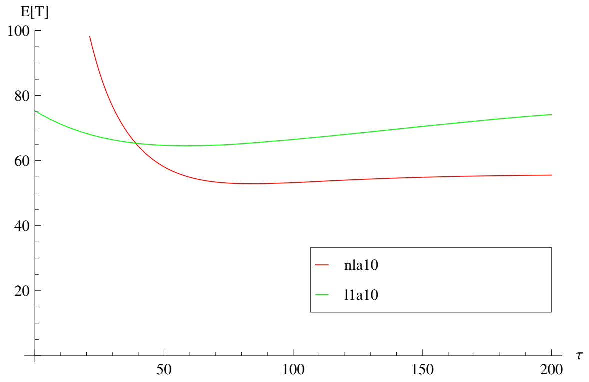


Figure 14. $E[T]$ under restart for the `nla10` and `l1a10` models.

However, it should be noted that the analytical model may be too simplistic, in that it does not take into account the effect of restart on other users of the same server. That is, restarting a job to reduce response times may result in increased load, which in turn may increase response times. In order to study the restart method in a distributed environment, we set up a simple queueing model with multiple input streams, a single server and a restart algorithm implementing different restart strategies. The model is shown in Figure 15.

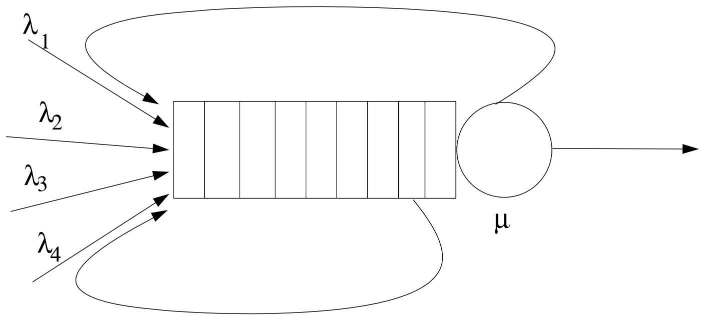


Figure 15. Single server queue to model restart

The model is parameterised with the above service-time distributions obtained in the SOA testbed. Using simulation, we evaluate performance of the Fixed Intervals, Jacobson/Karn and QEST restart strategies. Analysis of the model shows that restart helps to decrease completion times and avoid overload.

Jobs arrive at rate $\lambda = \sum_i \lambda_i$ to the queue. Each job draws a randomly distributed service time. While waiting in line, the timeout value for the job is decreased. The timeout may expire before the job enters service. Then the job remains at its position in the queue using a newly drawn random service time. The time already waited is added to the job's response time. If the new service time again leads to a response time that exceeds the timeout, new attempts of drawing a short service time are made repeatedly.

Job response times in this model consist of the service time and the waiting time, which is determined by the service times of the jobs ahead of the considered job in the queue. Without restart the model represents an M/G/1 queue, for which analytical solutions are available. The comparison between simulation results without restart and the analytical solution in Table 2 shows that the simulation captures the behaviour of the queueing system well.

Please note that the response times in the queue cannot be compared with the response time of the analytical restart model which does not include any waiting time.

The model allows us to compute the queue length and the response time as measures of congestion. Simulated for a given mission time, we can furthermore study the number of completed jobs. The model does not allow for a specification of customers with individual queues and timeouts.

We use the two ACPH(2) models from Section 3.1 and [14] to specify the service-time distributions of our queueing models. We set the load $\rho = 0.95$ and choose the arrival rate λ accordingly.

We analyse the effect of the different restart strategies on the mean response time. The simulation results are listed in Table 2. We observe that restart reduces

	`nla10`	`l1a10`
Analytical Results for the M—G—1 Queue		
	854.7	13262.1
Model		
None	826.82 ± 7.57	13507.7 ± 226.5
FI	746.70 ± 6.65	527.8 ± 11.7
JK	444.52 ± 4.53	165.5 ± 3.1
QEST	410.35 ± 7.20	192.5 ± 12.02

Table 2
Mean response time with 95% confidence interval

completion times in both scenarios. The improvement is particularly striking with the high-variance service-time distribution (`l1a10`), where the mean response time was reduced from more than 13500 time units to as little as 165.5 time units by the Jacobson/Karn strategy. With the low-variance service-time distribution (`nla10`), the effect is much less pronounced. We also observe that with both distributions the adaptive Jacobson/Karn and QEST strategies perform much better than the static Fixed Intervals strategy.

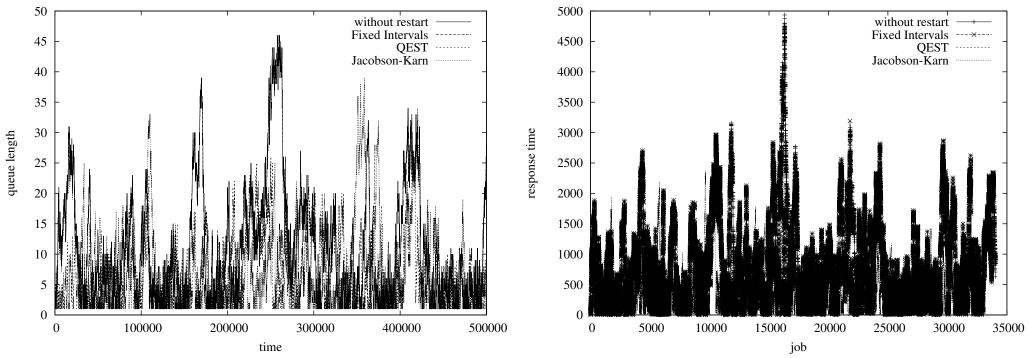
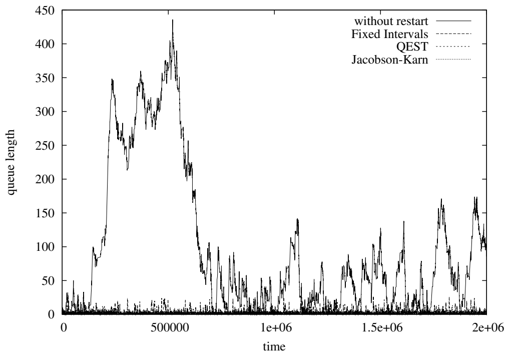


Figure 16. Queue length and response time for nla10 service time distribution

Figures 16 and 17 show the evolution of the queue length and the response time of the individual jobs for both service time distributions. Without restart the queue length and the service time can grow extremely large for the `l1a10` service time distribution while restart avoidds such extremes irrespective of the timeout computation algorithm.

Table 3 illustrates that in both simulations all strategies complete roughly the same number of jobs.

The differences between the different strategies can be explained by looking at the development of the timeouts in Figure 18. For the `nla10` scenario we observe that the QEST timeout first rises and then stays constant throughout the rest of the experiment. In contrast, the Jacobson/Karn timeout fluctuates, often dropping to values below 100, which explains that Jacobson/Karn is much more likely to



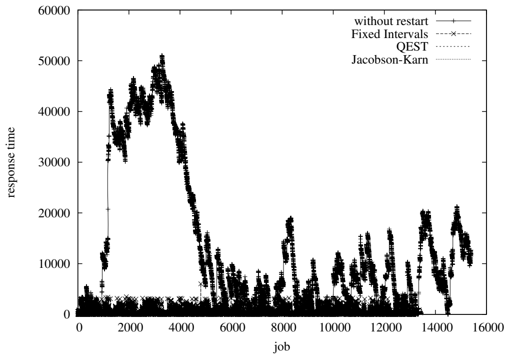
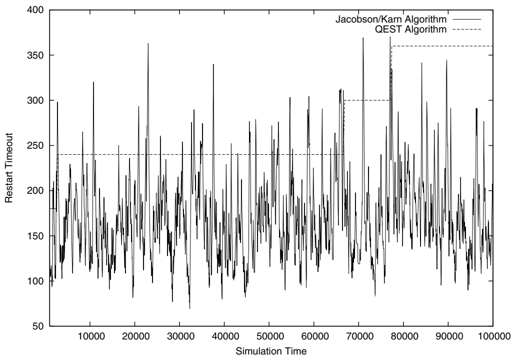


Figure 17. Queue length and response time for l1a10 service time distribution

	nla10	l1a10
none	34049 (-)	15394 (-)
FI	33891 (3)	13157(98)
JK	32232 (246)	13527(603)
QEST	33010 (57) (8695 (15))	12642 (533)

Table 3
Number of completed jobs and number of restarts (in brackets).

restart than QEST. With the `l1a10` service-time distribution we again see that Jacobson/Karn's timeout fluctuates more than the timeout of the QEST algorithm. Here, the QEST timeout is around 400, while Jacobson/Karn's timeout varies between less than 100 and more than 1400.



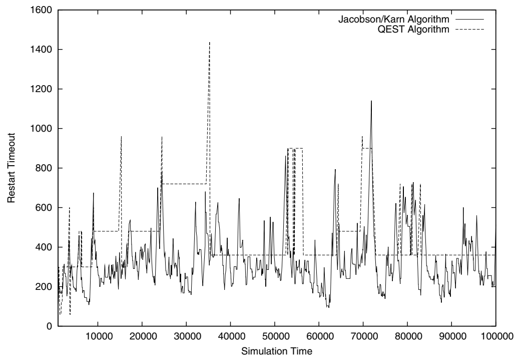


Figure 18. Timeout over time for `nla10` (left) and for `l1a10` (right)

We conclude from the simulation study that the restart method improves response times even in highly loaded systems. On the other hand, the added complexities of the competitive scenario result in timeout values and mean response times that are significantly larger than predicted by the simple analytical model. We see that the timeout value adjusts in scenarios with packet loss where longer response times are to be expected. This avoids unnecessary flooding of the network and increased congestion. The adaptive computation of the timeout value guarantees that the restart frequency is adjusted to network and system conditions.

6 Conclusions

In this paper we have proposed an iterative and mixed modelling and experimentation approach for evaluating service-oriented systems.

We have seen that formulation of a stochastic model of a complex distributed system requires information that can be obtained only by conducting experiments, even if the model is fairly simple. Experiments using real systems often lead to unexplained effects and are extremely time consuming. A testbed may be a suitable compromise allowing us to completely control the system while still being realistic in its dynamics. Therefore we use a testbed to obtain a realistic response time distribution. We have approximated the observed response times using phase-type distributions. This gives us a small Markovian model which captures the information of an empirical data set and can be included in a larger model. We have used the phase-type distribution first as probability distribution of the restart model and, second, as service time distribution in a queueing model.

The analytic formulation of the restart model provided us with fast results while for the queueing model with restart we had to resort to simulation. We will in the future enhance our testbed by more elaborate fault and load models and will study its timing behaviour and representations thereof using small phase-type distributions.

References

[1] Apache Software Foundation. Apache Sandesha. http://ws.apache.org/sandesha/.

[2] Oliver Hohlfeld, Rüdiger Geib, and Gerhard Haß linger. Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments. In *Proc. of the 16th International Workshop on Quality of Service (IWQoS)*, pages 239–248, June 2008.

[3] Gerhard Haßlinger and Oliver Hohlfeld. The gilbert-elliott model for packet loss in real time services on the internet. In *MMB*, pages 269–286, 2008.

[4] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[5] András Horváth and Miklós Telek. PhFit: A General Phase-Type Fitting Tool. In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 82–91, London, UK, 2002. Springer-Verlag.

[6] Van Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18(4):314–329, 1988.

[7] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice.* Addison Wesley, 2001.

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

[9] Marcel F. Neuts. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach.* The Johns Hopkins University Press, 1981.

[10] P. Reinecke, A. van Moorsel, and K. Wolter. A Measurement Study of the Interplay between Application Level Restart and Transport Protocol. In *Proc. International Service Availability Symposium (ISAS)*, number 3335 in Lecture Notes in Computer Science, Munich, Germany, May 2004. Springer.

[11] P. Reinecke, A. P. A. van Moorsel, and K. Wolter. The Fast and the Fair: A Fault-Injection-Driven Comparison of Restart Oracles for Reliable Web Services. In *Proc.3rd International Conference on the Quantitative Evaluation of SysTems (QEST) 2006*, Riverside, CA, USA, September 2006. IEEE.

[12] P. Reinecke and K. Wolter. Adaptivity Metric and Performance for Restart Strategies in Web Services Reliable Messaging. In *Proc. Seventh Int. Workshop on Software Performance (WOSP)*, Princeton, NJ, USA, June 2008. to appear.

[13] P. Reinecke and K. Wolter. Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging. In *Proc. SPEC International Performance Engineering Workshop (SIPEW)*, volume 5119 of *LNCS*, Darmstadt, Germany, June 2008. Springer-Verlag. to appear.

[14] Philipp Reinecke, Sebastian Wittkowski, and Katinka Wolter. Response time measurements using the sun java adventure builder. In *Proc.1st International Workshop on the Quality of Service-Oriented Software Systems (QUASOSS)*, Amsterdam, The Netherlands, August 26 2009.

[15] Sun Microsystems. Java adventure builder reference application. https://adventurebuilder.dev.java.net/, 2006. Last seen April 28th, 2009.

[16] Axel Thümmler, Peter Buchholz, and Miklos Telek. A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Trans. Dependable Secur. Comput.*, 3(3):245–258, 2006.

[17] M. Telek and A. Heindl. Matching Moments for Acyclic Discrete and Continous Phase-Type Distributions of Second Order. *International Journal of Simulation Systems, Science & Technology*, 3(3–4):47–57, December 2002.

[18] A. P. A. van Moorsel and K. Wolter. "optimal restart times for moments of completion time". *IEE Proceedings Software*, 151(5), 2004.

[19] Aad P. A. van Moorsel and Katinka Wolter. Analysis of restart mechanisms in software systems. *IEEE Transactions on Software Engineering*, 32(8):547–558, August 2006.

[20] K. Wolter. Self-Management of Systems through Automatic Restart. In Ozalp Babaoglu, Márk Jelasity, Alberto Montresor, Christof Fetzer, Stefano Leonardi, Aad van Moorsel, and Maarten van Steen, editors, *Self-Star Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[21] Katinka Wolter. *Stochastic Model for Fault Tolerance - Restart, Rejuvenation and Checkpointing.* Springer Verlag, 2010.