Evaluating the Scalability of a Web Service-Based Distributed e-Learning and Course Management System

Stephen Gilmore and Mirco Tribastone

LFCS, University of Edinburgh
{stg, mtribast}@inf.ed.ac.uk

Abstract. A growing concern of Web service providers is *scalability*. An implementation of a Web service may be able at present to support its user base, but how can a provider judge what will happen if that user base grows? We present a modelling approach based on process algebra which allows service providers to investigate how models of Web service execution scale with increasing client population sizes. The method has the benefit of allowing a simple model of the service to be scaled to realistic population sizes without the modeller needing to aggregate or re-model the system.

1 Introduction

Web Services are gaining more and more popularity as an approach to distributed computing. This flourishing is in part due to the use of well-known standard protocols for message exchange such as HTTP [1], XML [2], SOAP [3], and WSDL (Web Service Description Language) [4], as well as a large number of frameworks to improve developer's productivity (e.g. Apache AXIS [5], and Java WSDP [6]). Web Services are also being supported by businesses (e.g., [7]) to provide programmer-friendly interfaces for their services.

This paper addresses scalability performance aspects of e-Learning oriented Web Services. We present a scenario in which Web Service technology is used to implement a Distributed Course Management System (DCMS). One of the most severe problems a DCMS has to deal with is the performance degradation occurring when many users are requesting the service simultaneously. Let us imagine a DCMS is available for collecting final course projects of a class. Teaching staff usually put a deadline on those activities, and students are likely to get their projects ready very close to the due date. The DCMS has to cope with a flash crowd-like effect, as server resources (i.e. memory, CPU and bandwidth) have to be shared among a large number of users, thus paving the way for performance penalties experienced by users. In order to assess scalability properties of the system, we first develop a simple analytical model of the request/response message exchange pattern in SOAP, the Web Service communication protocol. This model constitutes the basis for the DCMS model where concurrent clients performing SOAP requests are taken into account.

M. Bravetti, M. Nuñes, and G. Zavattaro (Eds.): WS-FM 2006, LNCS 4184, pp. 214–226, 2006.

2 Related Work

Performance issues of traditional Web servers have been extensively investigated (for example, [8,9]). Research on performance evaluation of Web Services has been primarily focused on comparing SOAP implementations [10,11] or investigating performance of SOAP approaches to scientific computing [12,13].

In comparison to this, less effort has been invested in modelling the newer technology of Web Services. In [14] a profile-driver model for cluster-based web services is presented. Application profiles are obtained by mapping workload characteristics to resource (i.e. CPU, disk, memory) utilisation using linear fitting. Results shows that remote invocation overhead is important for the accuracy of the model; however, in our work we disregard method call overhead as in our scenario we focus on the most constrained system activity occurring at rate which turns out to be several order of magnitudes slower than SOAP processing. [15] propose an analytical model for a multi-tier web service based on a network of queues, where each queue represents an application tier. Although this work shares some common ideas (chain multi-tier approach), we propose a much simpler model where server overload, tier replication and multiple session classes are not taken into account. A general account of Web Service scalability is found in [16].

Structure of This Paper: The remainder of this paper is structured as follows. In Section 3 we briefly introduce PEPA, the stochastic process algebra employed for our modelling. In Section 4 we develop the model of request/response message exchange for Web Services. In Section 5 we discuss methodology and numerical results of parameter estimation. In Section 6 the DCMS case study is shown and a preliminary model is given. Stiffness problems in model evaluation are solved by means of the simplified model which is presented in Section 7. Numerical results are presented in Section 8. Section 9 concludes the paper.

3 Overview of Performance Evaluation Process Algebra

In Performance Evaluation Process Algebra (PEPA) [17], a system is viewed as a set of *components* which carry out *activities* either individually or in cooperation with other components. Activities which are private to the component in which they occur are represented by the distinguished action type, τ. Each activity is characterized by an *action type* and a duration which is exponentially distributed. This is written as a pair such as (α, r) where α is the action type and r is the *activity rate*. This parameter may be any positive real number, or may be unspecified. We use the distinguished symbol $\top$ to indicate that the rate is not specified by this component. This component is said to be *passive* with respect to this action type and the rate of the shared activity is defined by another component.

3.1 Combinators of the Language

PEPA provides a set of combinators which allow expressions to be built which define the behaviour of components via the activities that they engage in. These combinators are presented below.

Prefix: $(\alpha, r).P$: Prefix is the basic mechanism by which the behaviours of components are constructed. This combinator implies that after the component has carried out activity (α, r), it behaves as component P.

In this paper we will make use of *functional rates* [18] which allow the rate at which an activity is performed to depend on the current state of the model. (In Petri nets terms, a "marking-dependent" rate.)

Choice: $P_1 + P_2$: This combinator represents a competition between components. The system may behave either as component P_1 or as P_2. All current activities of the two components are enabled. The first activity to complete distinguishes one of these components and the other is then discarded.

Cooperation: $P_1 \underset{L}{\bowtie} P_2$: This describes the synchronization of components P_1 and P_2 over the activities in the cooperation set L. The components may proceed independently with activities whose types do not belong to this set. A particular case of the cooperation is when $L = \emptyset$. In this case, components proceed with all activities independently. The notation $P_1 \parallel P_2$ is used as a shorthand for $P_1 \underset{\emptyset}{\bowtie} P_2$. In a cooperation, the rate of a shared activity is defined as the rate of the slowest component.

Hiding: P/L This component behaves like P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \overset{def}{=} P$ Constants are components whose meaning is given by a defining equation: $A \overset{def}{=} P$ gives the constant A the behaviour of the component P. This is how we assign names to components (behaviours). An explicit recursion operator is not provided but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

One process constant is pre-defined. The deadlocked process named *Stop* enables no activities [19].

3.2 Formal Semantics of the Language

Process algebras are concise formally-defined modelling languages for the precise description of concurrent, communicating systems. The PEPA process algebra benefits from formal semantic descriptions of different characters which are appropriate for different uses. The structured operational semantics presented in [17] maps the PEPA language to a Continuous-Time Markov Chain (CTMC) representation. A denotational semantics for the language maps PEPA models to elements of metric spaces [20]. A continuous-space semantics maps PEPA models to a system of ordinary differential equations (ODEs) [21], admitting different solution procedures. We use both the CTMC and ODE semantics in the present paper.

3.3 Analysis Tools for PEPA

The reason to have a formally-defined high-level language for performance modelling is that it is possible to implement software tools which evaluate models according to the formal semantics of the language. In the present study we used the PRISM probabilistic model-checker [22], which accepts PEPA as one of its input languages, to perform transient analysis of the CTMC. We used the PEPA Workbench [23] to compile the PEPA model to a differential equation form which we could solve using a fifth-order Runge Kutta numerical integrator.

Because we are modelling in a high-level language it is possible to apply these very different numerical evaluation procedures to compute performance results from the same model. This is a freedom which we would not have if we had coded a Markov chain or differential equation-based representation of the model directly in a numerical computing platform such as Matlab.

4 Model of a Request/Response SOAP Exchange

Web Services use SOAP as the underlying protocol for inter-process communication. Being based on XML, it requires more resources than traditional binary-based RPC protocols such as, e.g. CORBA or RMI. Moreover, sending binary data over XML-based protocol is a critical performance issue.

Several approaches have been presented so far to allow efficient transmission of binary data over SOAP. SOAP with Attachments (SwA) [24] uses the MIME mechanism [25] to send MultiPart/Related messages. DIME [26] is a specification from Microsoft which encapsulates SOAP messages and attachments into binary records. DIME is no longer supported and has been replaced by MTOM [27], a WC3 Recommendation which enables optimised MIME serialisation of SOAP messages.

4.1 Message Life Cycle

We describe a fair approximation of a SOAP message life cycle, as we used to model the system. Although SOAP also supports asynchronous (one-way) messages, we focus on the Request/Response exchange pattern. Moreover, let us suppose that the client may transmit a binary file with the request. We assume the attachment is being sent according to the SwA specification, though our model is consistent with other mechanisms as well.

The client is the originator of the request. We may describe it as a process which evolves through the following series of activities:

1. *Message creation.* This involves XML formatting activities.
2. *File attachment.* This phase depends on the mechanism employed (e.g. SwA, DIME, MTOM, Base64 Encoding) and the file size.

3. *Message sending.* Key factors are message size and network bandwidth.
4. *Response awaiting.* Performance issues are related to the server throughput and network available bandwidth.
5. *Response processing.* HTTP and XML parsing are taken into account.

The server performs the following activities:

1. *Request processing.* This involves both HTTP and XML parsing.
2. *Attachment processing.* This depends on how many processing resources are needed by the server in order to deal with the attachment.
3. *Response creation.* This phase includes server's method invocation and XML response message formatting.
4. *Response sending.* This is dependent on the available network bandwidth.

Setup of the Model. We consider the model in the optimistic scenario where hardware and software failures are assumed to occur sufficiently infrequently that we will not represent them. Further, the server is sufficiently well-provisioned that we may also neglect the possibility failures caused by out-of-memory errors or overrunning the thread limit on the JVM hosting the Web container. We will return to review these optimistic assumptions after we compute performance results from our model.

4.2 PEPA Model of the System

It is straightforward to obtain a PEPA representation from the system description presented in Section 4.1. Figure 1 shows the model of a request/ response message exchange. The system here is made up of only two components that perform a single exchange by synchronising on all of their common activities.

$$
\begin{aligned}
\mathit{ClientA} &\stackrel{\mathit{def}}{=} (\mathit{create}, \alpha).\mathit{ClientB} \\
\mathit{ClientB} &\stackrel{\mathit{def}}{=} (\mathit{attach}, \beta).\mathit{ClientC} \\
\mathit{ClientC} &\stackrel{\mathit{def}}{=} (\mathit{queue}, \lambda).\mathit{ClientD} \\
\mathit{ClientD} &\stackrel{\mathit{def}}{=} (\mathit{request}, \top).\mathit{ClientE} \\
\mathit{ClientE} &\stackrel{\mathit{def}}{=} (\mathit{response}, \top).\mathit{ClientF} \\
\mathit{ClientF} &\stackrel{\mathit{def}}{=} (\mathit{processResponse}, \gamma).\mathit{Stop}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{ServerA} &\stackrel{\mathit{def}}{=} (\mathit{queue}, \top).\mathit{ServerB} \\
\mathit{ServerB} &\stackrel{\mathit{def}}{=} (\mathit{request}, \mu).\mathit{ServerC} \\
\mathit{ServerC} &\stackrel{\mathit{def}}{=} (\mathit{save}, \theta).\mathit{ServerD} \\
\mathit{ServerD} &\stackrel{\mathit{def}}{=} (\mathit{processRequest}, \eta).\mathit{ServerE} \\
\mathit{ServerE} &\stackrel{\mathit{def}}{=} (\mathit{response}, \phi).\mathit{Stop}
\end{aligned}
$$

$$
\mathit{ClientA} \underset{\{\mathit{queue}, \mathit{request}, \mathit{response}\}}{\bowtie} \mathit{ServerA}
$$

Fig. 1. PEPA model of request/response message exchange

5 Parameter Estimation

5.1 Experimental Design

We conducted experiments to estimate the appropriate numerical values for the parameters used in our model. We implemented a simple Web Service in which SwA was enabled to allow it to save a binary file attached by the client. The implementation of the server interface as well as the method for processing attachments are timed methods, in order to let us gather measurement data on their invocation.

The client makes a designer-tunable number of service calls, the attachment file size being passed as application argument. The designer may also set an inter-message idle period; however, our results were not affected by changes in this parameter.

5.2 Test Environment

We performed our tests with both client and server running on the same host, although our Web Services was implemented to be remotely accessible. We used a desktop with the following configuration: Intel Dual Xeon 3.2 GHz processor and 2 GB RAM running Microsoft Windows XP 64bit Edition. Our Web Service framework uses Sun Java Application Server Platform Edition 8.2, Java 2 Platform Standard Edition 5.0, Java WSDP 1.6 and JavaBeans Application Framework 1.0.2. Class binding, automatic WSDL file generation and application deployment were supported by NetBeans IDE 5.0.

We used 200 ms inter-message idle period and 1000 service invocations for each experiment; file size was 20 MB. Table 1 shows experimental results we obtained in our tests.

Table 1. Experimental results

Activity Name	Mean (ms)
create	0.592
attach	0.040
processResponse	0.154
save	81.100
processRequest	0.775

6 Distributed Course Management System Model

In order to assess the scalability issues of a Web Service-based distributed application we consider the following scenario. A Web Service is implemented for distributed e-Learning and Course Management System. We restrict our analysis to a case where one single course is being managed. We assume that no other services simultaneously run on the server; thus, the server download capacity c_s as well as server upload capacity μ_s are fully available for the Web Service.

The clients' (i.e. students) arrival process is assumed to be well-described by a Poisson distribution with rate λ. The system allows a maximum number of students (course size) N. We assume that all students have the same values for download capacity c_c and upload capacity μ_c. Like the server, we also suppose that no other process but the Web Service client-side application consumes network resources.

When multiple clients are involved, the server has to share its bandwidth among them. A model of the behaviour of the network is therefore necessary. We address this issue by developing a simple model for characterising service performance of the system. In this model we assume an ideal network in which no loss occurs and network nominal *capacity* means *available bandwidth.* We also suppose that transmissions are established on top of TCP connections where fairness against concurrent requests is perfect.

Given the above assumptions, if we denote i $(i > 0)$ as the number of uploading clients at any point in time, the uploading rate of each connection *request* is:

$$\mathit{request} = \min\left\{\frac{c_s}{i}, \mu_c\right\} \tag{1}$$

Similarly, if j is the number of downloading clients (i.e., clients who are receiving the response message), the downloading rate of each connection *response* is:

$$\mathit{response} = \min\left\{\frac{\mu_s}{j}, c_c\right\} \tag{2}$$

6.1 PEPA Model of the System

We present the model of the DCMS by taking into account the behaviour of server bandwidth when multiple connections are allowed. Local activities are unaffected by concurrent requests. Thus, the model of the client is the same as in Fig. 1. As for the server, we need to distinguish each of the possible number of clients which upload to him simultaneously. Let $Server_i$ be the process description of the server downloading from i concurrent clients. The model of the server as well as the description of the system are described in Fig. 2.

Model analysis has been carried out by setting local activity rates as they were obtained in our experimental tests (cfr. Tab. 1). Table 2 shows the complete parameter set. It is worthwhile to observe that network parameters represent bandwidths normalised by the message size being sent. For instance, $c_s = 0.001$ means that the server is able to get the entire message completed in 1000 s; this value resembles a realistic situation where a server equipped with a 10 Mbps connection has to download a file about 1 GB long. We also would like to point out that server upload capacity is much faster than its download capacity because of the size of the message being transmitted: here we have assumed 1 KB long SOAP response messages in our parameter set. The value of λ is to consider flash crowd-like effect, such that triggered for instance by simultaneous service requests when a deadline is due.

As our model considers client components which perform only one request, transient analysis has to be carried out for evaluating the performance of the

$$
\begin{aligned}
\mathit{Server_0} &\overset{def}{=} (\mathit{queue}, \top).\mathit{Server_1} \\
\mathit{Server_i} &\overset{def}{=} (\mathit{queue}, \top).\mathit{Server_{i+1}} + (\mathit{request}, \min\{\tfrac{c_s}{i}, \mu_c\}).(\mathit{save}, \theta). \\
&\quad (\mathit{processRequest}, \eta).(\mathit{response}, \min\{\tfrac{\mu_s}{i}, c_c\}).\mathit{Server}_{i-1} \\
&\quad (0 < i < N) \\
\mathit{Server_N} &\overset{def}{=} (\mathit{request}, \min\{\tfrac{c_s}{N}, \mu_c\}).(\mathit{save}, \theta).(\mathit{processRequest}, \eta). \\
&\quad (\mathit{response}, \min\{\tfrac{\mu_s}{N}, c_c\}).\mathit{Server}_{N-1}
\end{aligned}
$$

$$
\left(\underbrace{\mathit{ClientA} \parallel \mathit{ClientA} \parallel \ldots \parallel \mathit{ClientA}}_{N} \right) \underset{\mathit{queue}, \mathit{request}, \mathit{response}}{\bowtie} \mathit{Server}_0
$$

Fig. 2. Model of the server in DCMS

system. In the following we describe some preliminary studies which have been conducted in order to assess scalability issues of the model.

Table 2. Parameter set for model analysis

Parameter	Meaning	Rate (s^{-1})
α	*create*	1689.20
β	*attach*	25000.00
γ	*processResponse*	6493.50
θ	*save*	12.33
η	*processRequest*	1290.32
λ	*queue*	20.00
N	Population size	100
c_s	Server download bandwidth	0.001
μ_s	Server upload bandwidth	$c_s/3$
c_c	Client download bandwidth	$(c_s/10) \cdot 10^6$
μ_c	Client upload bandwidth	$c_c/30$

We mapped the PEPA model to CTMC representation. We found the underlying Markov chain does not scale with the number of model components. We calculated different state space sizes by varying N, as shown in Tab.3. We argue that the CTMC representation highlights lack of scalability which makes performance analysis intractable even for unrealistic values of N.

It is well known that the ODE-based representation of the model offers better scalability, as the size of the space vector does not change for N varying. However, we encountered stiffness problems when running time-series analysis, as the expected time to obtain model results was high (i.e., 10^8 s) even if the state vector size would suggest easy computability. We conjecture these problems are due to the differences of several order of magnitude between some activity rates (e.g., μ_c against β). We actually modified the parameter set by imposing unreal-

Table 3. CTMC State space sizes for N varying

N	State space size
1	9
2	72
3	540
4	3888
5	27216
6	186624

istic values (all close to 1) which made the running much faster, as we obtained results in 10^{-2} s.

7 A Simplified Model

The scalability problems discussed above lead us to a simplified model where all activities which occur at fast rate have been disregarded. The model is shown in Fig. 3.

$$
\begin{aligned}
\mathit{ClientIdle} &\stackrel{\mathit{def}}{=} (\mathit{queue}, \lambda).\mathit{ClientUploading} \\
\mathit{ClientUploading} &\stackrel{\mathit{def}}{=} (\mathit{request}, \top).\mathit{Stop} \\
\mathit{Server}_0 &\stackrel{\mathit{def}}{=} (\mathit{queue}, \top).\mathit{Server}_1 \\
\mathit{Server}_i &\stackrel{\mathit{def}}{=} (\mathit{queue}, \top).\mathit{Server}_{i+1} + (\mathit{request}, \min\{\tfrac{c_s}{i}, \mu_c\}).\mathit{Server}_{i-1} \\
&\quad (0 < i < N) \\
\mathit{Server}_N &\stackrel{\mathit{def}}{=} (\mathit{request}, \min\{\tfrac{c_s}{N}, \mu_c\}).\mathit{Server}_{N-1}
\end{aligned}
$$

$$
\left(\underbrace{\mathit{ClientIdle} \parallel \mathit{ClientIdle} \parallel \ldots \parallel \mathit{ClientIdle}}_{N} \right) \underset{\{\mathit{queue},\mathit{request},\mathit{response}\}}{\bowtie} \mathit{Server}_0
$$

Fig. 3. Simplified PEPA model of the DCMS

When mapping the PEPA model to CTMC representation, we found that the model is still not scalable as the space state size is 3^N. In the continuous-space representation the rates are separated by fewer orders of magnitude and performance results could be evaluated at low computational cost. In particular, we required only 0.03 seconds of compute time to obtain a 10^6 seconds time series analysis.

8 Numerical Results

We obtained numerical results using the parameter set as follows. We considered a maximum number of users $N = 100$, requesting service according to a flash

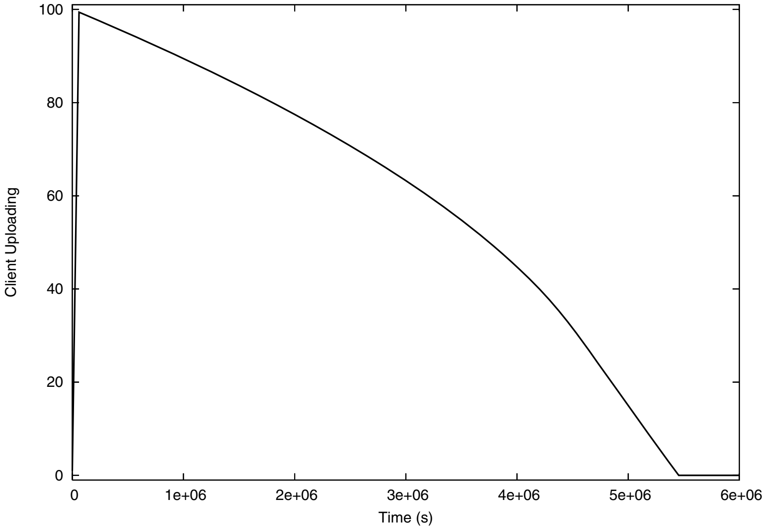

Fig. 4. Evolution of the number of clients uploading

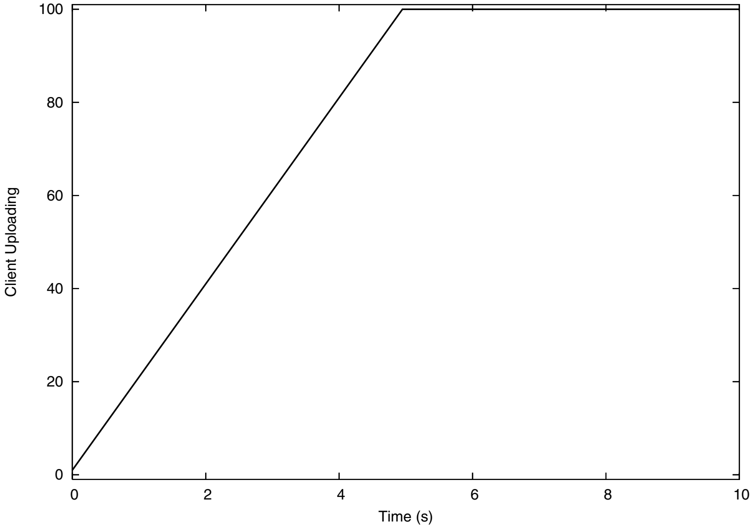

Fig. 5. Flash crowd effect in DCMS

crowd-like effect at rate $\lambda = 20$. Server download capacity c_s was set to 0.001, and client upload capacity $\mu_c = c_s/30$.

Figure 4 shows a time series plot of the number of client uploading to the server. The initial burstiness of requests is shown in Figure 5.

Figure 6 plots service durations for different server bandwidths (i.e., $c_s = 0.01, 0.02,$ and 0.1). Finally, Figure 7 shows service durations for different values of N, when $c_s = 0.1$ and $\mu_c = c_s/30$.

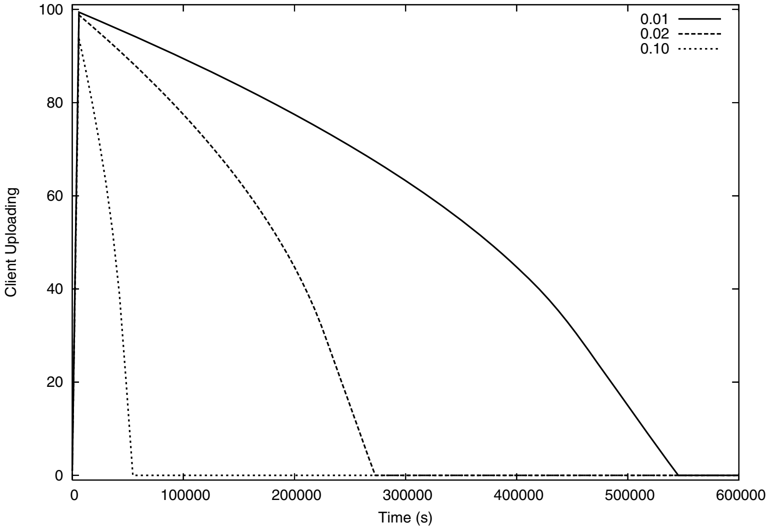

Fig. 6. Time series for different server bandwidths

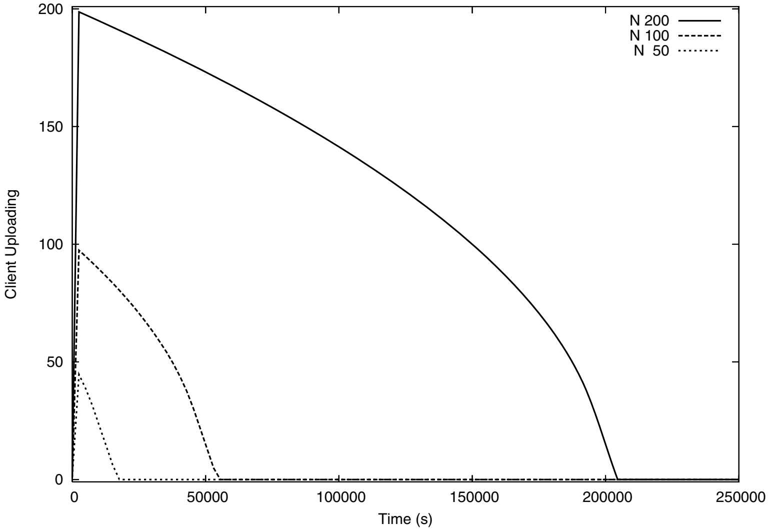

Fig. 7. Time series for different number of users

Commentary on the Results: We note that the system requires a significant amount of time to get every client request completed. Earlier we outlined a series of assumptions about the model setup which included the optimistic assumptions of absence of failure of various kinds, and did not include the possibility of users aborting long-running file uploads only to restart them again later. Since unsuccessful file transfers (of whatever kind) will only tend to delay things more we can safely interpret the results presented above as saying that even in this very optimistic setting the system is impractical for use.

9 Conclusions

This paper has assessed the scalability of a Web service which supports secure distributed file upload using the Web service attachments API. The issue of scalability extends basic evaluation of performance: a service may have acceptable performance at present, but the question is how this performance will be likely to change as greater numbers of service subscribers are added.

Models of distributed systems which are based on a discrete-state interleaving semantics are limited by the well-known state-space explosion problem: the size of the system as a whole is bounded by the product of the state space size of the individual components which it contains. Markovian models (whether obtained from process algebras, Petri nets or another modelling formalism) are victims of this problem. By mapping to a continuous-state differential equation representation the PEPA language allows modellers to assess scalability. The state-space is never constructed, making it possible to have a scalable analysis process. We move directly from the model with parameters fitted from measurement data to time series plots showing the changes in the number of each kind of component over time. The solution to a system of differential equations is definitive, as the solution of a Markov chain is, thus there is no repetition cost as found in other modelling approaches used to assess scalability (such as simulation). The numerical procedures used have low computational cost.

Acknowledgements. This work was supported by the EU IST-3-016004-IP-09 project SENSORIA.

References

1. W3C. HTTP protocol specification. http://www.w3.org/Protocols/rfc2616/rfc2616.html.
2. W3C. XML protocol specification. http://www.w3.org/TR/2004/REC-xml-20040204/.
3. W3C. SOAP protocol specification. http://www.w3.org/TR/soap/.
4. W3C. WSDL protocol specification. http://www.w3.org/TR/wsdl.
5. Apache AXIS. http://ws.apache.org/axis/.
6. Java Web Services Development Pack. http://java.sun.com/webservices/jwsdp/index.jsp.
7. Google web APIs. http://www.google.com/apis/.
8. M. Arlitt and L. Williamson C.˙ Internet web servers: Workload characterization and performance implications. In *IEEE/ACM Transaction on Networking*, October 1997.
9. L. Slothouber. A model of web server performance. In *Proceedings 5th International World Wide Web Conference*, 1996.

10. D. Davis and M. Parashar. Latency performance of SOAP implementations. In *Proceedings 2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002.
11. M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, and M. Lewis. Toward characterizing the performance of SOAP toolkits. In *Proceedings 5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, November 2004.
12. R. van Engelen. Pushing the SOAP envelope with web services for scientific computing. In *Proceedings International Conference on Web Services (ICWS'03)*, pages 346–354, 2003.
13. K. Chiu and M. Govindaraju. Investigating the limits of SOAP performance for scientific computing. In *Proceedings 11th IEEE International Symposium on High-Performance Distributed Computing*, 2002.
14. C. Stewart and K. Shen. Performance modeling and system management for multi-component online services.
15. B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its application. In *Proceedings of the ACM SIGMETRICS*, June 2005.
16. Robert van Engelen. Are web services scale free? `http://www.cs.fsu.edu/~engelen/powerlaw.html`, June 2005.
17. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
18. J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 120–135, Aachen, Germany, September 2001. Springer-Verlag.
19. N. Thomas and J. Bradley. Terminating processes in PEPA. In K. Djemame and M. Kara, editors, *Proceedings of the Seventeenth UK Performance Engineering Workshop*, pages 143–154, University of Leeds, July 2001.
20. M. Kwiatkowska and G. Norman. Metric denotational semantics for PEPA. In M. Ribaudo, editor, *Proceedings of the Fourth Annual Workshop on Process Algebra and Performance Modelling*, pages 120–138. Dipartimento di Informatica, Università di Torino, CLUT, July 1996.
21. J. Hillston. Fluid flow approximation of PEPA models. In *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, pages 33–43, Torino, Italy, September 2005. IEEE Computer Society Press.
22. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proceedings 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, 2006. To appear.
23. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.
24. W3C. SOAP with Attachments. `http://www.w3.org/TR/SOAP-attachments.`
25. MIME Multipart/Related Content-type RFC. `http://www.ietf.org/rfc/rfc2387.txt.`
26. DIME protocol specification. `msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt.`
27. W3C. MTOM. `http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/.`