EXPOSING WORKFLOWS TO LOAD BURSTS

Dmytro Dyachuk and Ralph Deters
Department of Computer Science, University of Saskatchewan, Saskatoon, Canada

Keywords: Service-Oriented Architecture, Workflow, Performance, Bursts.

Abstract: Well defined, loosely coupled services are the basic building blocks of the service-orientated design-integration paradigm. Services are computational elements that expose functionality (e.g. legacy applications) in a platform independent manner and can be described, published, discovered, orchestrated and consumed across language, platform and organizational borders. Using service-orientation (SO) it is fairly easy to expose existing applications/resources and to aggregate them into novel services called composite services (CS). This aggregation is achieved by defining a workflow that orchestrates the underlying services in a manner consistent with the desired functionality. Since CS can aggregate atomic and other CS they foster the development of service layers and reuse of already existing functionality. But by defining workflows, existing services are put into novel contexts and exposed to different workloads, which in turn can result in unexpected behaviours. This paper examines the behaviour of sequential workflows that experience short-lived load bursts. Using workflows of varying length, the paper reports on the transformations that loads experience as they are processed by providers.

1 INTRODUCTION

Service-Orientation (SO) (Four Tenets Of Service Orientation) is a design & integration paradigm that is based on the notion of well defined, loosely coupled services. Within SO, services are viewed as computational elements that expose functionality in a platform-independent manner and can be described, published, discovered, orchestrated and consumed across language, platform and organizational borders. While service-orientation (SO) can be achieved using different technologies, Web Services (WS) (Natis, 2003) are the most commonly used, due to the standardization efforts and the available tools/infrastructure (Apache Axis). The Service-Oriented Architecture (SOA) (Chatarji, 2007), first introduced by Gartner in 1996 (Natis, 2003), is a conceptual framework that identifies service-consumers, service-providers and a registry trough which providers publish and consumer discover.

In a service-oriented system, services are offered by service providers that register them with registries (e.g. UDDI). Service consumers (aka clients) discover at runtime service providers by simply queering the registries.

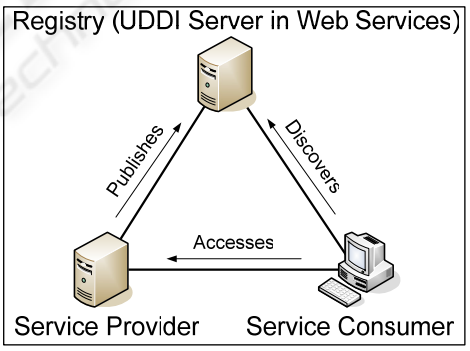

Figure 1: SOA.

Upon discovering a service provider, the consumer obtains from the provider the meta-data of the service in form of an XML document called Web Service Definition Language (WSDL) (Web Service Definition Language) that is then used to establish a binding to the provider (e.g. generation of stubs).

Since services are high-level constructs that hide implementation details, consumers can easily bind to unknown services across platform and language barriers, resulting in a system with very dynamic functional dependencies between its components. Consequently service-orientation supports a loose coupling between consumers and providers, allowing for agile and open systems. One of the

Dyachuk D. and Deters R. (2007).
EXPOSING WORKFLOWS TO LOAD BURSTS.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 218-225
DOI: 10.5220/0002382702180225

most profound implications of the loose-coupling is the ease with which components (e.g. legacy systems) can be connected and aggregated into new services called composite services (CS). Composite Services (CS) aggregate multiple services into one logical unit to accomplish a complex task (e.g. business process). This aggregation is achieved by defining a workflow that orchestrates the underlying services in a manner consistent with the desired functionality. Since CS can aggregate atomic and other CS they foster the development of service layers and reuse of already existing functionality. This gave rise to the idea of service networks in which resources and workflows are shared across organizational boundaries.

Over time many different approaches for modelling business processes have been developed and consequently there is no shortage of languages or concepts for implementing composite services. Below are two different approaches (sequential workflow and state-machine workflow) of modelling a business process that orchestrates three services.

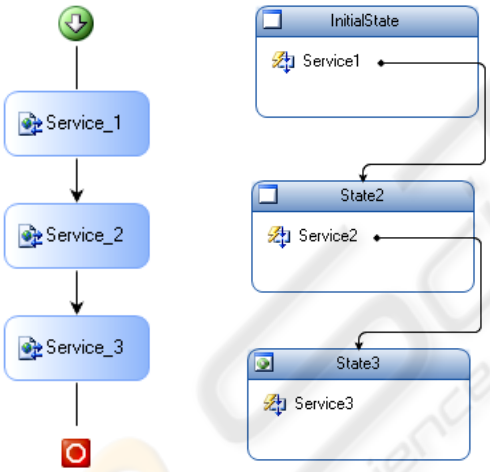

Figure 2: Business Process.

While a *sequential workflow* (left) models the business process from a process-driven viewpoint, a *state-machine workflow* (right) models it as a finite state machine in which state transitions are the result of events.

Sequential workflows are prescriptive and like a script or program define to the order in which the operations (underlying services) are executed. Using constructs such as loops, conditional statements and basic exception handling they assume a scenario in which the workflow *is in control* since the path of execution is determined by workflow internal factors. State-Machine based workflows follow the opposite approach, since they rely on events to trigger state transitions. State-machine workflows are often used for business processes that are event-driven (e.g. involve human feedback) and that should therefore enable many paths of execution. Sequential workflows on the other side, are most often used for implementing machine workflows (e.g. process automation) that are sequential in nature, and have little or no human involvement.

As already mentioned, the aggregation of services enables the definition of new services and thus layers of services arise. But by defining workflows existing services that may expose local resources (e.g. legacy applications) are also subjected to novel and potentially dangerous workloads. This is particularly worrisome for sequential workflows that automate processes and are therefore more likely to experience overloads that can cause ripple effects throughout a network of services.

This paper focuses on the behaviour of sequential workflows that experience short-lived load bursts. Section two presents a brief discussion of general server behaviour. This is followed by the presentation of the experimental setup (section three) and the results of exposing workflows to different loads (section four). The paper concludes with a summary and an outlook on future work.

2 BEHAVIOUR OF SINGLE SERVERS

If a service provider does not share resources with other providers (e.g. no two providers expose the same data base), it can be modelled as a server. If such a provider is capable of handling multiple requests simultaneously it must assign resources for dealing with the incoming requests. Assuming that servers have a finite amount of resources and that each new consumer request will (temporary) reduce the available server resources, it is interesting to examine server behaviours under various loads. Studies (Heiss, 1991) show that servers respond in a common way to loads.

If a server is gradually exposed to an ever-increasing number of service requests it is possible to observe three distinct stages, namely *under-load, saturation* and *over-load.* At the beginning the server experiences a load that is below its capacity (*under-load*). As the number of requests is increased, the throughput (number of completed jobs per time unit) improves. As the rate of incoming requests increases

the server will experience its *saturation* point (a peak load). This saturation marks the point where the server is fully utilized and operating at its full capacity. The saturation point marks also the highest possible throughput. Further increases of the rate at which the requests arrive will now lead to an overload or *thrashing effect.* The service capacity is exceeded and "an increase of the load results in the decrease of throughput" (Heiss, 1991). Typically the main reasons for a server to experience thrashing are:

- *Resource* contention: overload of the physical devices such as CPU, memory, hard drive, etc.
- *Data* contention: a contention caused by locking.

3 WORKFLOWS & LOADS

While the behaviour of a single server towards loads is fairly well researched, there has been little work on the impact of loads on sequential workflows.

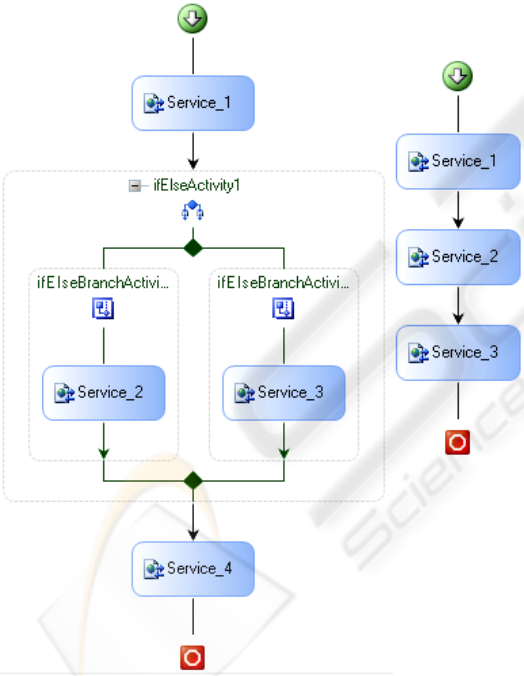

Figure 3: Sequences with and without Branching.

Depending on the used constructs (e.g. loops, conditional statements, parallel execution etc.) a sequential workflow can execute operations in a variety of ways.
However, for the analysis of the basic behaviours of workflows it seems sufficient to limit the discussion to the sequence pattern since the more advanced execution paths can be treated as extensions and/or variations of the sequence pattern.
Our previous studies with currently used Web Services platforms (e.g. AXIS 1.x, AXIS 2.x and the Windows Workflow Foundation) (Apache Axis, Eclipse, Visual Studio Home) showed that the behaviour of providers can be simulated (XJ Techologies) (with sufficient accuracy) by use of fairly simple models. Instead of modelling the different resources of providers (e.g. processor, memory, network etc.) and trying to emulate page faults, context switches or locking of resources, it is sufficient to combine all resources into one type that is distributed equally over all current requests (Processor Sharing is the standard scheduler for providers).

3.1 The Impact of a Single Short Burst

In the first experiment workflows of various lengths were exposed to a load of 100 requests. The providers in the workflow are to process jobs that require 100 % of the resources for 1.2 seconds (load = 120 %). The impact of the 100 requests on the first provider can be seen by the residence times of the requests (time it takes to process each request).

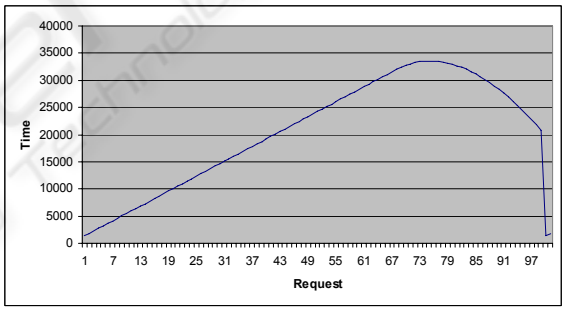

Figure 4: Residence Times of Requests on 1st Provider.

Since each request requires 120 % of resources per second, the provider encounters immediately a light overload. This is seen by the residence times (shown in milliseconds) that begin to rise until no longer new requests arrive. The rise in residence times is due to the overloading of the server, requests come in faster than they are completed. However since the burst consists of only 100 requests, there is a point when no new requests are received and the completion of requests frees up resources, enabling an ever faster processing of the remaining requests. It is noteworthy that the decline is steeper than the rise.

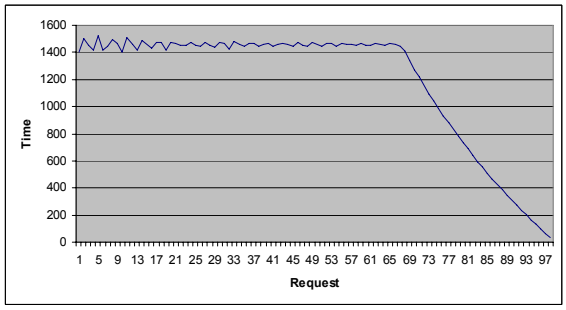

Figure 5: Interdeparture times for 1st Provider.

The corresponding job interdeparture times (measured in milliseconds) of the first provider are shown in fig. 5. There is a brief warm-up period in which the interdeparture times fluctuate. This is followed by a phase in which the interdeparture times gradually stabilizes followed by a rapid decent. This transition from stabilized interdeparture time to rapid decent marks the point when no longer new jobs arrive and the existing ones enjoy more and more resources which allow them to depart increasingly faster.

The most interesting aspect of the displayed departure rates is that a short overload (burst) with of a *constant* arrival rate (1 job/sec), leads to a departure rate that is *no longer constant*. Figure 5 shows that ca. 70 % of the requests have a departure rate of 1/1.4 job/sec and ca. 25 % a rate above 1 job/sec second (speedup). The impact of this transformation can be seen when looking at the interdeparture times of the second provider.

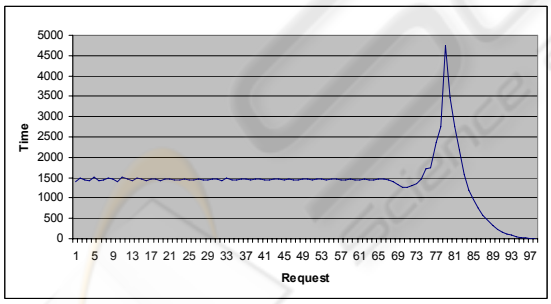

Figure 6: Interdeparture times for 2nd Provider.

The departure rate of the second provider seems at first glance very different from that of the first one. To explain the behaviour of the second provider we combined both graphs. As can be seen in figure 7, the departure rates for the second provider match those of the first provider for nearly 70 % of the requests. The departure rate of the 2nd provider differs from the 1st provider only in the last 30 %. The spike starts exactly when the jobs of the 1st provider begin to arrive at 1.2 job/sec As soon as they arrive faster than 1.2 job/sec, the 2nd provider begins to experience a gradual overload that leads to slowdown in the departure rates (peak emerges). An interesting aspect in the departure rates of the 2nd provider is that the rates decrease and increase at a faster rate than that of the 1st provider. This effect can also be seen when more providers are observed (fig. 8). With every additional provider a new spike emerges.

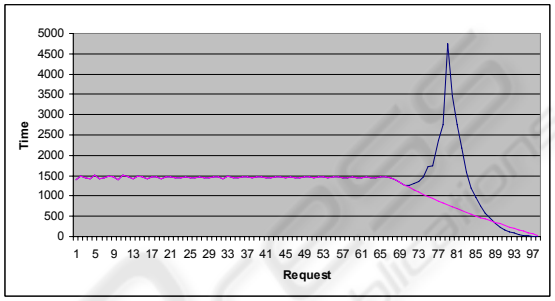

Figure 7: Interdeparture times of Providers 1 & 2.

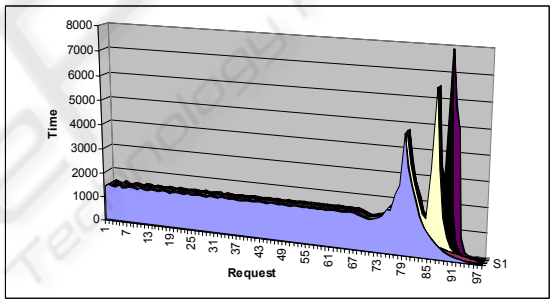

Figure 8: Interdeparture times of the first five Providers.

3.2 Multiple Short Bursts

To simulate short bursts the requests arrive now in 10 groups of 100 requests (arrival rate 1 second/request). Each group of 100 requests is separated by a period of 100 seconds (no requests). Again the providers in the workflow are exposed to jobs that require 100 % of the resources for 1.2 seconds (load = 120 %).

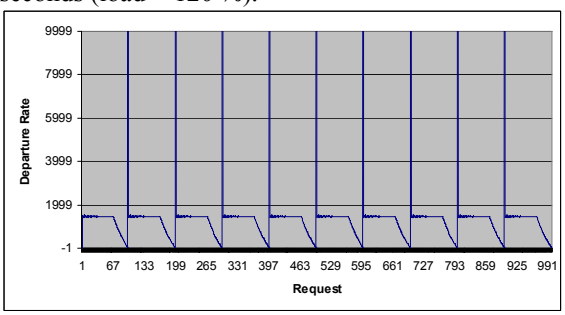

Figure 9: Interdeparture times of 1st Provider (enlarged).

Figures 9 & 10 display the interdeparture times of the 10 bursts from the first server. As can bee seen in figure 9 (zoomed) the familiar burst pattern (fig. 5) appears.

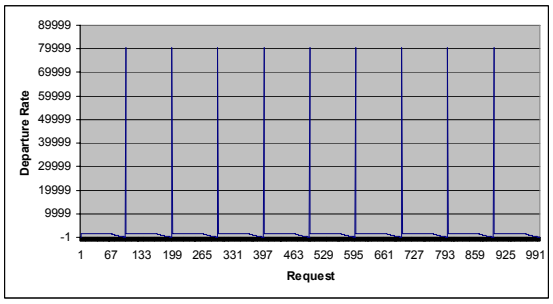

Figure 10: Interdeparture times of 1st Provider.

The departure rates of the second provider show a similar picture. Once again one notices that each pulse produces the already known burst pattern of a second provider (fig. 6). In addition one can also notice that the gap between the burst once again has been reduced.

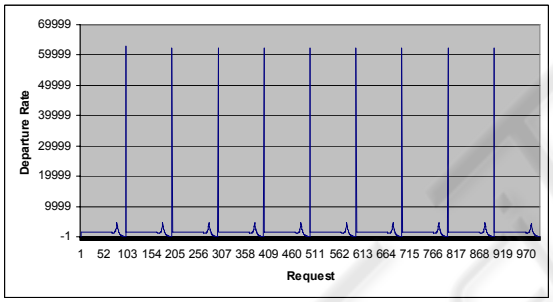

Figure 11: Interdeparture Times of 2nd Provider.

This gradual reduction of the original gap continues as more providers are introduced.

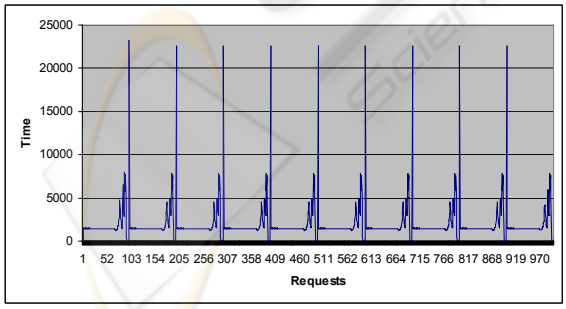

Figure 12: Interdeparture Times of 6th Provider.

When the interdeparture times of each provider are presented in one graph (concatenated) an interesting picture emerges. As shown in figure 13, the high spikes that mark the gap between the original bursts gradually disappear. This means that as more providers are chained in a sequence pattern the more the bursts merge into a completely new pattern.

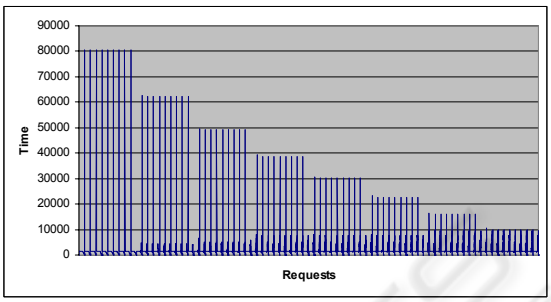

Figure 13: Concatenated interdeparture times of the first 8 Providers.

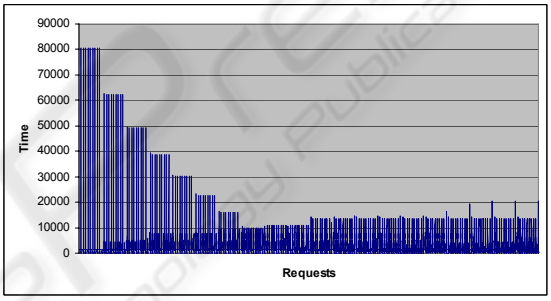

Figure 14: Concatenated interdeparture times of the first 20 Providers.

Figure 14 shows the results of exposing a sequential workflow consisting of 20 providers to the 10 bursts. The spikes that mark the original gaps between two bursts gradually shrink until they totally disappear (8th provider). Starting at the 9th provider the phenomenon of emerging spikes (as seen in section 3.1) begins to be the only reason for spikes. As a result of these experiments we conclude that a sufficiently long sequential workflow can transform an input pattern with bursts into a completely different output pattern.

3.3 The Impact of a Constant Load between Short Bursts

To simulate short bursts on top of a constant medium load, 1000 requests are used. Again 10 groups of 100 requests are formed. However this time the groups have alternating arrival rates shown in fig 15. The 1st, 3rd, 5th, 7th and 9th group have an interarrival time of 2 seconds and the 2nd 4th 6th 8th and 10th an interarrival time of 1 second. Again the providers in the workflow are exposed to jobs that

require 100 % of the resources for 1.2 seconds (load = 120 %).

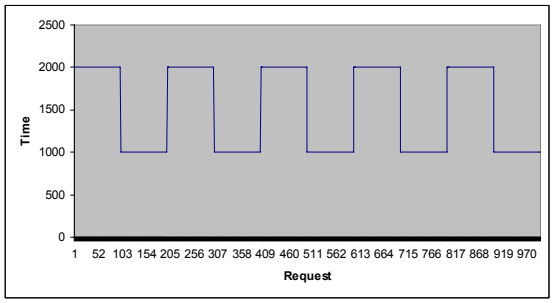

Figure 15: Alternating interarrival times (arrival rates).

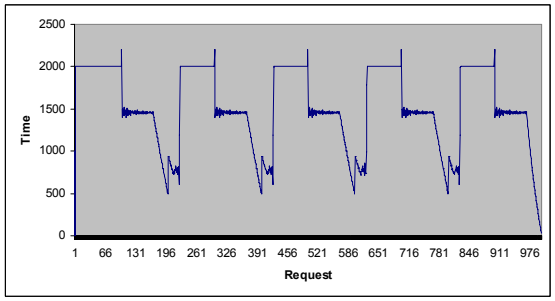

Figure 16: Interdeparture times of 1st Provider.

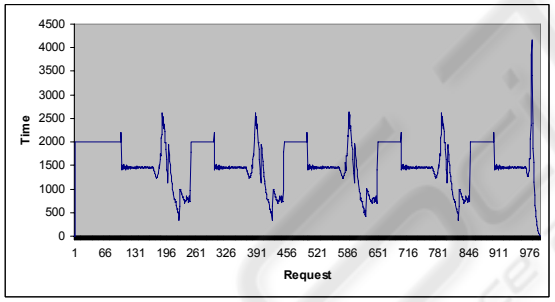

Figure 17: Interdeparture times of 2nd Provider.

Figure 16 shows the interdeparture times of the first provider. As soon as the arrival rate switches from 2 to 1 second the provider encounters an overload.

As a result of this overload the already familiar pattern of the provider emerges with a brief warm-up period in which the departure rates fluctuate. This is followed by a phase in which the interdeparture time gradually stabilizes followed by a rapid decent.

The departure rates of the second provider are also very similar to those of the previous runs. Only when longer sequences are observed significant differences become visible [fig. 17].

As seen in figure 18 & 19 larger sequences of providers transform faster into more chaotic sequences.

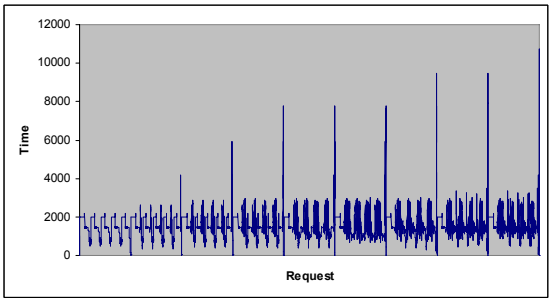

Figure 18: Concatenated departure rates of the first 8 Providers.

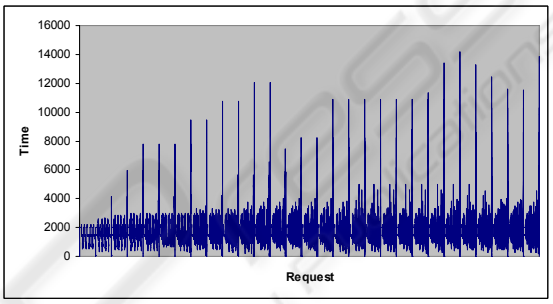

Figure 19: Concatenated interdeparture times of the first 20 Providers.

These results indicate that short lived bursts have a significant large impact on workflows that already experience a constant medium load.

3.4 The Impact of Admission Control

It is well-known that overload situations lead to a decline in service throughput. Heiss and Wagner (Heiss, 1991) proposed to use an adaptive load control as a means of admission control. This prevents overloads, by first determining the maximum number of parallel requests (e.g. maximum number of simultaneous consumers) and then buffering/queuing new requests once the saturation point has been reached. Adding admission control to an already existing service (e.g. CS) can be achieved by using a proxy that shields/hides the original provider and thus enables the introduction of a transparent admission control and scheduling (Cherkasova 1998, Elnikety 2004, Harchol-BalterSchroeder, 2003).

To test the impact of an admission control each of the providers is shielded by a proxy that controls the maximum number of parallel request for each service provider (proxy introduces no overhead).

Figure 20 shows the result of limiting the concurrently processed requests to 1 for each provider when the alternating load of section 3.3 (2

and 1 second arrivals) is used. As expected, setting limit of concurrently processed requests to 1 for each provider ensures that *no transformations* of the original arrival rates occur.

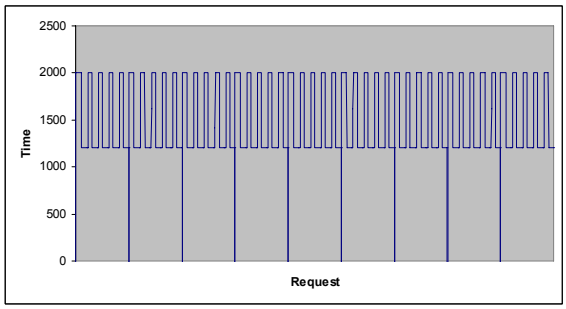

Figure 20: Interdeparture times of first 9 Providers (max 1).

Changing the limit to 10 concurrent requests already leads to transformations (fig. 21).

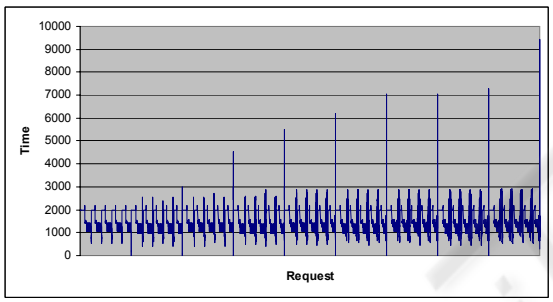

Figure 21: Interdeparture times of first 9 Providers (max 10).

When examining the first provider, one can also see the emergence of an alternating pattern. Since the provider allows only 10 concurrent requests at any moment in time the leaving of completed and the entering of new requests results in a reoccurring pattern of available resources.

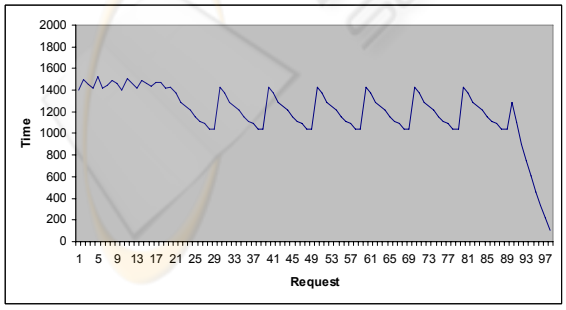

Figure 22: Interdeparture time of 1st Providers (max 10).

By concatenating the output of the first 20 providers one can see how the interdeparture time becomes increasingly chaotic.

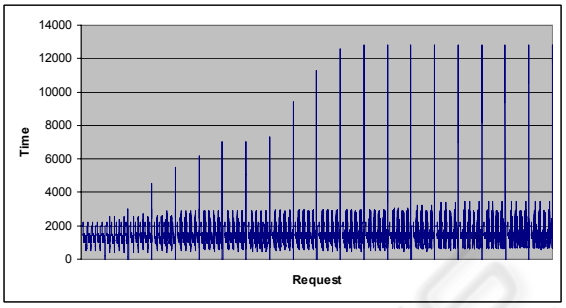

Figure 23: Interdeparture times of 20 providers (max 10).

Since the load scenario of section 3.3 consisted of a constant 60 % load that increased to 120 % in the bursts it is interesting to compare it to the lighter load of section 3.2 (no constant load only 120 % in bursts).

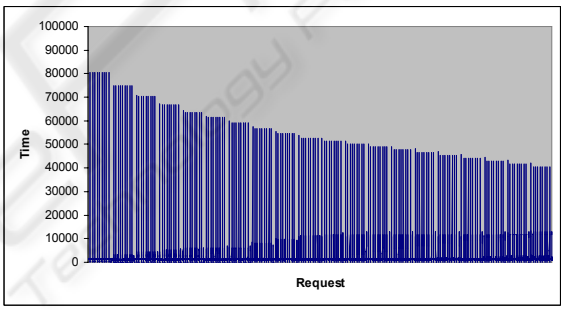

Figure 24: Interdeparture time of 20 providers (max 10).

As can be seen in figure 24, the gradual decline of the gap is still visible in this lighter workload scenario. However by shielding the proxy there are less distortions of the original workload.

It is however important to note that the positive impact of the admission control is also a result of the constant loads each request imposes. As soon as an exponential distribution of arrivals and loads is introduced it becomes difficult to determine what the maximum amount of concurrent request should be for each given moment in time.

4 CONCLUSIONS

This paper focuses on the behaviour of sequential workloads that experience short periods of overloads (bursts). Two basic scenarios were used, short bursts that were separated by a pause of requests and bursts that appeared between a medium (60 %) load.

The results of the experiments showed:

1) Workflows that encounter even short-lived overloads distort the original arrival rates.
2) The more providers a workflow contains, the more chaotic the resulting departure rate becomes.
3) The impact a burst has depends on the load surrounding it. Bursts that are embedded in a medium load have a bigger impact than those embedded in a light or no load.
4) Adding an admission control reduces the distortion introduced by the workflow.

These results show that there is a need for a load and configuration management of sequential workflows that can encounter bursts. Only if tools for monitoring/tracking/routing requests and loads are available will it become feasible to ensure dependable services in a complex service network.

5 FUTURE WORK

Based on the presented findings our future work will focus on:

1) Further investigation into workflow behaviour. The investigated workflows are still very basic and don't reveal the impact conditional statements or loops will have.
2) The development of mechanisms to monitor and manipulate the loads workflows experience.
3) The study of seamless replication as a mechanism to soften the impact of sudden bursts.
4) Investigating if and how workflows can be annotated (semantic marking) to enable better management.
5) Provenance of workflow behaviours.

REFERENCES

Eclipse, http://www.eclipse.org/.

Apache Axis, http://ws.apache.org/axis/.

Four Tenets Of Service Orientation, http://msdn.microsoft.com/msdnmag/issues/04/01/Indigo/default.aspx.

Visual Studio Home, http://msdn.microsoft.com/vstudio/. Web Service Definition Language (WSDL), http://www.w3.org/TR/wsdl.

Chatarji, J., Introduction to Service Oriented Architecture (SOA), http://w-ww.devshed.com/c/a/Web-Services/Introduction-to-Service-Oriented-Architecture-SOA.

Cherkasova, L. (1998). Scheduling Strategy to Improve Response Time for Web Applications, *HPCN Europe 1998: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking,* 1998, Springer-Verlag, 305-314.

Elnikety, S., Nahum, E., Tracey, J. And Zwaenepoel, W. (2004). A method for transparent admission control and request scheduling in e-commerce web sites, *WWW '04: Proceedings of the 13th international conference on World Wide Web,* 2004, ACM Press, 276-286.

Harchol-Balter, M., Schroeder, B., Bansal, N. And Agrawal, M. (2003). Size-based scheduling to improve web performance. *ACM Trans.Comput.Syst.,* **21**(2), 207-233.

Heiss, H. And Wagner, R. (1991). Adaptive Load Control in Transaction Processing Systems, *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases,* 1991, Morgan Kaufmann Publishers Inc, 47-54.

Natis, Y.V. (12 April 2003). *Service Oriented Architecture.* Gartner.

XJ TECHOLOGIES, Anylogic 5.5, http://www.xjtek.com/2007.