Towards Concurrent SLA-based Management in a Composite Service Data Centre

J. Smith and P. Watson
Towards Concurrent SLA-based Management in a Composite Service Data Centre

J. Smith and P. Watson

Abstract

The ongoing aim behind the work described here is to investigate support for a data centre type environment where an application can be implemented as a composition of components, or workflow. For instance, an application might reuse established components. As is typical in data centre operation, hosting of an application is governed by a Service Level Agreement (SLA) between provider and application owner. Such an SLA describes levels of service and corresponding charges (paid to the provider) and refunds (paid by the provider). The infrastructure can of course vary mapping of hosted applications to machine resources in seeking to meet SLAs efficiently. However, it is also possible to dynamically vary the mapping of composite application to components; for instance to switch between alternate implementations of some particular component. The suggestion is that the availability of alternative components, or even compositions, will arise naturally in a shared repository, and that the presence of the extra degree of control can make the application hosting more resilient. While previous work by the authors has demonstrated basic mechanisms towards a composite service data centre, the current work begins to combine such mechanisms towards the control of both resource and component mapping in the context of concurrent workloads. The paper describes experimental work using a prototype implementation of an adaptive workflow engine.
Abstract

The ongoing aim behind the work described here is to investigate support for a data centre type environment where an application can be implemented as a composition of components, or workflow. For instance, an application might reuse established components. As is typical in data centre operation, hosting of an application is governed by a Service Level Agreement (SLA) between provider and application owner. Such an SLA describes levels of service and corresponding charges (paid to the provider) and refunds (paid by the provider). The infrastructure can of course vary mapping of hosted applications to machine resources in seeking to meet SLAs efficiently. However, it is also possible to dynamically vary the mapping of composite application to components; for instance to switch between alternate implementations of some particular component. The suggestion is that the availability of alternative components, or even compositions, will arise naturally in a shared repository, and that the presence of the extra degree of control can make the application hosting more resilient. While previous work by the authors has demonstrated basic mechanisms towards a composite service data centre, the current work begins to combine such mechanisms towards the control of both resource and component mapping in the context of concurrent workloads. The paper describes experimental work using a prototype implementation of an adaptive workflow engine.

About the author

Paul Watson is Professor of Computer Science and Director of the North East Regional e-Science Centre. He graduated in 1983 with a BSc (I) in Computer Engineering from Manchester University, followed by a PhD in 1986. In the 80s, as a Lecturer at Manchester University, he was a designer of the Alvey Flagship and Esprit EDS systems. From 1990-5 he worked for ICL as a system designer of the Goldrush MegaServer parallel database server, which was released as a product in 1994. In August 1995 he moved to Newcastle University, where he has been an investigator on research projects worth over £13M. His research interests are in scalable information management. This includes parallel database servers, data-intensive e-science and grid computing. In total, he has over thirty refereed publications, and three patents. Professor Watson is a Chartered Engineer, a Fellow of the British Computer Society, and a member of the UK Computing Research Committee.

Suggested keywords

COMPOSITE SERVICE,
WORKFLOW,
SERVICE ORIENTED,
DATA CENTRE,
SLA
Towards Concurrent SLA-based Management in a Composite Service Data Centre

Jim Smith and Paul Watson
School of Computing Science, Newcastle University, UK
Jim.Smith@ncl.ac.uk, Paul.Watson@ncl.ac.uk

Abstract

The ongoing aim behind the work described here is to investigate support for a data centre type environment where an application can be implemented as a composition of components, or workflow. For instance, an application might reuse established components. As is typical in data centre operation, hosting of an application is governed by a Service Level Agreement (SLA) between provider and application owner. Such an SLA describes levels of service and corresponding charges (paid to the provider) and refunds (paid by the provider). The infrastructure can of course vary mapping of hosted applications to machine resources in seeking to meet SLAs efficiently. However, it is also possible to dynamically vary the mapping of composite application to components; for instance to switch between alternate implementations of some particular component. The suggestion is that the availability of alternative components, or even compositions, will arise naturally in a shared repository, and that the presence of the extra degree of control can make the application hosting more resilient. While previous work by the authors has demonstrated basic mechanisms towards a composite service data centre, the current work begins to combine such mechanisms towards the control of both resource and component mapping in the context of concurrent workloads. The paper describes experimental work using a prototype implementation of an adaptive workflow engine.

1 Introduction

In common with other component based systems, service oriented computing aims to facilitate reuse of established components, thereby both saving on development effort and pushing application design to a higher level of abstraction where errors are less likely. Thus, the services offered by an organization can be represented as machine readable interfaces (WSDL). These are stored in registries with associated metadata describing functional and non-functional properties, where they can be found so as to support use of the underlying service by client applications. Thus it is common for complex applications to be implemented as composite services (aka workflows) linking services at widely distributed locations. In a distributed setting, it is to be expected that one execution of such a complex application may differ from a previous one, e.g. through using different services or service instances; even that a composition may be dynamically altered during execution, e.g. in the event of a failure. To this end, the client may specify a set of requirements, which may be encapsulated in a service level agreement (SLA). For instance, [15] demonstrates composition which is dynamic in respect of non-functional attributes such as duration, price and availability. However, it is also possible to consider composition in respect of functional requirements too, e.g. [10]. Such approaches tend to be concerned with adapting an individual workflow.

In recent years there has been a notable tendency for organizations to out-source some or all of their IT applications, and several companies have developed large data centres to host the client applications. Thus, work such as [3] demonstrates that multiple applications can be hosted dynamically and securely, but a need arises for the improvement of management algorithms if utility computing is to achieve efficiency [2]. Recent work, e.g. [13], has shown good scalability in terms of mapping monolithic applications. Also, while the inherently multi-dimensional nature of SLAs is well appreciated [9], it appears to be typical for experiments in data centre control to emphasize user response time. The notion of a shared repository in which applications may be developed, potentially exploiting reuse, and hosted, can be seen appearing in offerings such as Amazon Web Services [1], and CARMEN [5].

Variation of of component replication is used in support of meeting SLAs for composite service invocations in [11]. In general there can be bounds on useful replication, e.g. due to database access [4]. In the context of multi-tier web server applications, [14] addresses the problem of taking account of bounds on parallelism in those tiers in or-
der to use data centre resources efficiently. Such concerns are significant too in the composite service data centre of this work, but this paper is more closely related to a scenario described in [6] where web server applications are dynamically switched between levels of quality (full graphics and text only) in order to meet response time requirements. Specifically, the experiment described here demonstrates that dynamic swapping of component implementations can be used to meet multi-dimensional SLA requirements.

2 Adaptive and Concurrent Management of Composite Services

A prototype system has been implemented by wrapping an established 'basic' workflow engine [7] in "black-box" fashion, as shown in Figure 1. At deployment, a workflow is modified by a Translator to add extra identification parameters to each component invocation. Each component invocation from the deployed composite service is intercepted by a Broker, which uses these identification parameters and statistics forwarded by probes, P1 and P2, to determine, in accordance with a chosen policy, how to process that request. The Broker dynamically partitions the pool of available machines into a number of subpools, according to the number of workloads, and moves machines between subpools in accordance with target sizes which are determined on the basis of the probe measurements. The Broker also constructs an invocation map for each deployed workflow, determining between alternate component services again on the basis of the measurements reported by the probes. Further details of the prototype may be found in [12]. In the current implementation, a policy is implemented directly as Java code within a controller class within the Broker.

In this experiment, the policy seeks (approximately) to maximize revenue for the provider. The first mechanism employed is a slight adaptation of the measured loads heuristic of [8] in selection of subpool sizes. This heuristic follows from the well known queueing result of Little which equates offered load in a queueing system to the product of arrival rate and response time. Both workflow arrival rate and workflow response time are measured for each workload in probe P1. For a given interval, the heuristic partitions the pool of machines in proportion with the product of the offered loads and workload revenue coefficients; the former measured during the previous interval. The slight adaptation to the previously described heuristic is in the definition of the cost coefficient. In the experiment described here, the coefficient is computed as the square of the average of possible refund values divided by the average of possible charge values, rather than simply refund divided by the charge. The idea behind squaring the refund value is to ensure that few resources are allocated to a less profitable workload. The use of averaging reflects the presence of alternate levels of charge or refund, depending for instance on result quality. A second mechanism is employed to control the response time where necessary in the scenario where a lower quality answer on-time is preferable to a higher quality answer late. Specifically this mechanism switches between maximizing a named attribute, e.g. result quality, and minimizing response time for each component service (measured in probe P2) with alternative implementations within a composite service, depending on whether the measured average response for the composite service is (respectively) less than or greater than a defined bound. Effectively, one non-functional attribute is traded-off with another. The longer term idea is to decide between admission control, i.e. rejection of requests and quality downgrade dynamically, and in the case of the latter to decide dynamically which attributes to downgrade. The experiment described here demonstrates a particular example scenario.

3 Experiment

In the experiment described here, as in [12], the two example workflows shown in 2 are used. Both OneCall and TwoCallAnd invoke a service Calc, but in the latter case, Calc is only invoked if the prior call to Test returns true. Both Test and Calc are implemented as configurable CPU-heavy loads. Each has a single operation, test and calc respectively, which is invoked in these experiments. As in the earlier work, two implementations of the more expensive service are defined with differing properties, as shown in Table 1. The execution cost of the test operation in this work is equal to that of the cheaper implementation of calc. The actual execution times of single invocations of the two calc implementations in this experiment are 4.7 and 47 seconds.

There are many possible definitions for an SLA which could reflect the various possible outcomes of a request. For
Figure 2. Example workflows.

(a) OneCall

(b) TwoCallAnd

Figure 3. Example SLA agreement.

Figure 4. Recorded pool sizes.

In the experiment, there are three concurrent workflows; workload A comprises requests against OneCall and the other two, B and C, make requests against TwoCallAnd. TwoCallAnd is configured such that test returns true in 50% of requests; in just those requests is calc invoked. The workflow requests in all three workloads follow Poisson sequences. In the case of workload A which submits 500 requests against OneCall, the Poisson rate is increased from 0.067 to 0.155 at time 2000 seconds. In the case of workload B which submits 1000 requests against TwoCallAnd (recall that 500 of these invoke the expensive operation), the Poisson rate is decreased at time 2000 seconds from 0.38 to 0.1125. In the case of workload C, the 250 requests against TwoCallAnd are submitted at a constant Poisson rate of 0.0565.

The SLA for each of the workloads has the form shown in Figure 3. However, while A and B have $c = 0.5$, C has $c = 0.05$. Intuitively then, the system should allocate most resources to A and B and potentially switch between implementations of Calc in order to control the average response time, particularly in the case of workload C which should be squeezed in terms of resources.

4 results

The experiment is conducted in a Linux environment. The pool comprises 20 2.8GHz machines in a cluster. The workload generators and adaptive workflow engine are run on a separate 3GHz machine. Figure 4 shows how the actual pool sizes vary during the workload. As expected, most of the machines are allocated to the two higher value workloads. The significant transfer of machines between these two workloads corresponds to the change, at time 2000, in request submission rates described earlier.

Figure 5 shows the measured durations of individual requests in each of the three workloads. In the case of workload A, there is a period of settling at startup, during which 6 requests are mapped to the cheaper alternative of Calc. Following this, the response times for requests in workload A are mostly constant, and close to the expected value. Those in workload B, are mostly split evenly between about 5 and about 50 seconds, reflecting the ratio of requests which make an invocation of calc. In practice, all of the calc invocations in workload B, and all apart from the 6 requests mentioned above in workload A are mapped
to the more expensive implementation of Calc, reflecting the higher value of these workloads in revenue terms. By contrast, the response times in workload C oscillate about the chosen switching value, which was set to 90 seconds in this experiment. In practice, out of the 144 invocations of calc in workload C, 87 were mapped to Calc000 and 57 to Calc001. For the lower value workload, which is squeezed for resources, the service mapping is dynamically mapped between cheaper and more expensive implementations, in order to trade quality so as to keep the average response time close to that specified in the SLA.

5 Conclusion

An experiment in the management of concurrent workloads is presented, where the workloads invoke composite services which are mapped dynamically onto a shared pool of machine resources and whose component invocations are mapped dynamically between alternate implementations. It is anticipated that dynamic composition of this nature might become a feature of shared repositories which are starting to appear. The results presented here are limited in several ways, but demonstrate that it can be possible to employ dynamic service invocation mapping in support of concurrent SLA based management of composite services. Specifically, a lower value workload can be implemented at degraded quality. A somewhat simplistic approach is employed here which maps all invocations in a workflow for response time or quality. In general, this offers an alternative to rejecting either the workload or particular requests, and an obvious direction for ongoing work is to evaluate such alternatives. As hinted earlier of course, trading between non-functional properties can also be possible in the context of monolithic applications.

References