Configuration Decision Making Using Simulation-Generated Data

Michael Smit and Eleni Stroulia*

Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{msmit,stroulia}@cs.ualberta.ca

Abstract. As service-oriented systems grow larger and more complex, so does the challenge of configuring the underlying hardware infrastructure on which their consitituent services are deployed. With more configuration options (virtualized systems, cloud-based systems, *etc.*), the challenge grows more difficult. Configuring service-oriented systems involves balancing a competing set of priorities and choosing trade-offs to achieve a satisfactory state. To address this problem, we present a simulation-based methodology for supporting administrators in making these decisions by providing them with relevant information obtained using inexpensive simulation-generated data. Our services-aware simulation framework enables the generation of lengthy simulation traces of the system's behavior, characterized by a variety of performance metrics, under different configuration and load conditions. One can design a variety of experiments, tailored to answer specific system-configuration questions, such as, "what is the optimal distribution of services across multiple servers" for example. We relate a general methodology for assisting administrators in balancing trade-offs using our framework and we present results establishing benchmarks for the cost and performance improvements we can expect from run-time configuration adaptation for this application.

1 Introduction

Service-oriented architectures, cloud computing, Web 2.0, and 'smart' infrastructures are important technologies and areas of active software research today. Together, they promise to enable novel features of software systems, such as interoperability across organizations, cohesive integrated services that can be deployed and run quickly, and intuitive interactions across people and systems. These technologies are the current state-of-the-art; whether they are further developed or replaced, the functionalities they strive to provide will continue to be sought after. With these functionalities, however, come challenges inherent to complicated systems: there are multiple organizations involved, the behavior of the system is difficult to predict based solely on the behavior of the individual

* The authors acknowledge the generous support of iCORE, NSERC, and IBM.

E.M. Maximilien et al. (Eds): ICSOC 2010 Workshops, LNCS 6568, pp. 15–26, 2011.

components, and technical decisions may be dictated by business decisions made without consideration of technical challenges. With the many options involved (outsourced or self-owned? cloud computing or local infrastructure? green or cheap?), configuring applications built on these technologies is a challenge.

One common decision is the size, scale, and configuration of computing infrastructure. This task is complex even for technical experts. Decisions must be made about trade-offs between many configuration parameters and the overall cost and quality-of-service they imply. Configuration assistance, especially autonomic assistance, aims to support less-technical users in their decision making about systems with which they do not have expertise. Trade-offs of particular interest are those involving performance (quality-of-service) versus cost (financial).

We describe three novel contributions of our service-system configuration tool, and demonstrate each contribution using simulation-generated metrics[1]. First, question answering, where the simulation is run in specific scenarios designed to explore the system behavior under alternative conditions of interest. We demonstrate this method by answering one question about the parallelization of a service and another about the distribution of a service's operations over multiple servers (Section 4.1). Second, we describe our method for exploration of potentially conflicting configuration goals and trade-offs, where we use the data to help an administrator establish service-level agreements (SLAs), appropriate thresholds, accurate costing models, or appropriate configurations to meet a specific SLA or cost model (Section 4.2). Finally, we explore the effect of run-time configuration adaptation on the simulated application, determining the benefit of and targets for a future autonomic system. In particular, we explore two alternatives: one based on response time and server performance metrics, and another based on request statistics and composition knowledge (Section 4.3). The tool developed for this exploration could also be used to train administrators to adapt configurations to changing situations. Finally, we summarize the contributions of this paper in Section 5.

2 Background and Related Work

We consider a *service*, an application that offers a specific function, a "building block" for systems of services. A common software implementation specification is Web Services (WS), which offers a well-defined interface to server-side software and a suite of standards[2]. Services can be *composed* to provide more complex functionalities. *Service Level Agreements (SLAs)* can be used to govern these interactions. An SLA is a contract between two parties, where one promises a certain level of service and the other promises a corresponding payment for it.

[1] Metrics of any type will work as input to our implementation; the focus of this paper is not simulation but rather methods enabled by simulation data.

[2] The terms and language used throughout this document are those used for Web Services; this is done without loss of generality as the methodology is intended to be applicable to distributed applications generally, though the implemented tools may have dependencies based on WS technologies.

The level of service promised and expected can be described in technical or non-technical terms, however these terms will be typically measurable. SLAs can be general (promising 99% uptime) or negotiated based on needs/capabilities of the involved parties. Although SLAs are the current standard practice, they do not always ensure customer satisfaction. Blomberg [1] conducted a study of interactions between consumers and providers. She identified five problems with how SLAs were used in those interactions, including that SLA metrics do not always reflect the needs of the customer, that SLAs are not proactive, and that not enough information about the performance of the service is available to the client.

Decision making using simulation-generated data has been applied to domains including medicine [2], manufacturing [3], and software agents [4]. It commonly takes the form of answering specific questions (similar to our first contribution) but is less commonly used for configuration exploration and trade-off analysis or for run-time configuration adaptation (our second and third contributions).

Grundy *et al.* used a performance test-bed generator (MaramaMTE) [5] to prototype potential service compositions using simulated compositions, simulated load, and real-life services. Their methodology allows system architects to define a composition, test it, modify as required, and repeat. Their hybrid real-virtual approach assures accuracy but the number of tests that can be performed is limited; administrator intuition and understanding is required. Similarly, Chandrasekaran *et al.*[6] describe a tool for composing web services where known performance information is provided as input to a general simulation environment to simulate the orchestrated composition, producing a statistical estimate of the composed services' performance; however, they do not discuss any validation for their method, whether empirical or analytical.

Brebner *et al.* describe a tool that allows the user to define an SOA environment using building blocks like services, servers, workloads, and metrics based on UML diagrams [7] (instead of automatically as in our approach). Given parameters (*e.g.*, performance data and hardware), an event-based simulation produces performance metrics. This is part of a project looking at SOA performance [8,9].

Miller *et al.* [10] use simulation in combination with a workflow management tool to answer "what-if" questions about workflow adaptations and their effect on quality of service. They do not model the individual nodes in the workflow and they assume QoS information is already available for individual nodes. They describe a hypothetical case study but do not offer a validated implementation.

Almeida and Menasce [11] offer a general method for capacity planning in client-server systems, making explicit the need for models of workloads that the architecture is expected to support and the way its performance may change when aspects of the workload change. This approach is not service-focused and does not use simulation to produce data.

3 Simulation-Generated Dataset

In this section, we describe the generation of the datasets used by our reasoning tools. For a more detailed discussion, the reader should review prior work [12,13].

The simulation framework is a suite of tools that enable (a) the creation of a model of a service-oriented application, and (b) the "virtual" execution of the model to generate and record data about its run-time behavior.

The framework has four main components. The *simulation engine* provides an environment for running simulations (clock management, networks, basic service functionality, composition, *etc.*). *wsdl2sim* takes as input a service interface (specified as a WSDL) and its performance model, and generates a simulation to be run by the engine. The performance model of a service is constructed automatically using load testing[3] under various configurations and fitting a curve to the resulting metrics. The generated simulations are discrete-event and run in simulated time (faster than real time). *JIMMIE* uses an XML-based language to systematically re-configure the simulation and repeatedly run it to test different scenarios. The *dashboard* module facilitates collecting, storing, and visualizing metrics generated by the simulation, as well as interacting with the simulation.

We used this framework to simulate TAPoRware, a text-analysis system that provides a public web services interface. Its focus is on data processing (CPU-bound), with the movement of data being a secondary but important concern. TAPoRware is a single web service with 44 operations, implemented in Ruby. Each operation runs in $O(n)$, bounded by CPU time relative to the size of the input. The tools covered in this paper are listing the words with their counts, generating word clouds, finding the use of a word in context (concordance), and finding two words located near each other in text (co-occurrence).

We evolved the simulated model to add features not in the original application. We added metrics capturing code to generate the dashboard, created a load balancer, added a request generator (Section 3.1), extended the configuration files to allow for JIMMIE integration, and added the ability to manually re-configure at run-time. We treated the operations as if they belonged to separate services.

3.1 Generating Performance Metrics through Simulation

The system under examination is simulated under varying configurations (systematically and repeatably) to predict its behavior and performance in different conditions. The data produced by the simulation is determined by two factors: (a) input (incoming requests and their parameters) and (b) the configuration of the simulated application. The generated data is periodic snapshots of performance data under different combinations of these two factors.

Incoming requests vary based on several parameters: the arrival rate of requests, the type of each request (*i.e.*, the operation being invoked, as each has a different performance profile), and the size of each request. The simulation framework offers the ability to write custom request generators; we authored four: a) stress testing; b) reading from log files (either from the real-world application or one of the other generators); c) stochastic, based on probability distributions for request arrival rate (Poisson), type of request (weighted random based on real logs), and size of request (parameterized double gaussian and exponential distributions based on curve-fitting size distributions from the real log files); d)

[3] soapUI (`http://www.soapUI.org`) was used to generate SOAP messages.

stochastic, where the distribution parameters can be modified at run-time. Each of these request generators records repeatable traces of generated traffic.

The configuration of the simulated application is described with an XML document "understood" by the simulation framework. The specification document identifies the available servers, the software services of the system and their distribution on these servers, the appropriate "stub" classes that should be loaded to emulate them, and the request generation strategy and parameters. Note that the JIMMIE component of the simulation framework can systematically produce a multitude of such configuration specifications in order to run automatically hundreds of simulated experiments.

Given such a specification document, our framework can be invoked to simulate the subject system and record the performance metrics. These metrics are captured periodically, persisted to database storage, and (optionally) visualized at run-time using the dashboard. Metrics captured include queue length for each server, CPU utilization for each server and globally, response time for each request and rolling averages, the total time required to process the sample set of requests, and other related metrics based on those commonly used in SLAs. Metrics are stored every 5 seconds; simulation-wide averages are stored at the end of the simulation period.

4 Reasoning Using Simulation-Generated Data

This section describes three methodologies and tools we have developed to reason about specific questions around the system configuration, and tested using simulation-generated data[4]. First, answering questions about the system's behavior under specific configuration decisions. Second, exploring general trade-off analysis with conflicting goals, such as cost and performance. Finally, testing how simulation data can be used at run-time to identify system reconfigurations.

4.1 Answering Questions About Configurations

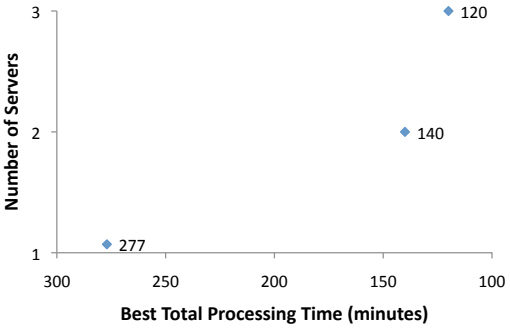
The first use of simulation-generated data is to answer specific questions around the system's behavior in different configuration and load conditions. In this approach, the configurations are systematically modified to test a series of configuration options, and the results are compared to answer the question of interest, usually around making a trade-off. To illustrate this scenario, consider two questions based on a real-world TAPoRware issue, namely the benefits of concurrency and what distribution of services on what number of servers is best suited to meet expected load.

First, *for a single-server installation of TAPoRware, does allowing the List Words operation to process multiple responses simultaneously improve the overall response time?* JIMMIE was used to generate configuration files specifying 1000 requests of the List Words type, size 1000KB, but with requests arriving with

[4] These tools are intended to be general; other performance metrics from other applications could be used in lieu of the sample simulated application used here.

Svc.	Req.	Response Time
1	1	29.81 ± 0.48
2	1	29.83 ± 0.53
1	2	58.88 ± 4.1
2	2	58.76 ± 1.0
1	5	142.92 ± 22.3
2	5	145.77 ± 32.3

(a) Response time for 50 1MB requests with varying service and request concurrency.



(b) The best total time for 1, 2, and 3 servers with varying operation distributions.

Fig. 1. Question answering results

various levels of concurrency: 1, 2, or 5. A single-server single-operation server was simulated and configured to process 1 or 2 requests concurrently (with any outstanding requests waiting in a queue). The experiment outcomes are shown in Figure 1a. The results show that serving 2 concurrent requests instead of one does not improve total processing time or response time by a significant or reliable amount. This is not surprising given the performance model; a single request is sufficient to occupy the only available processor. The question could be extended to ask what gains result from adding a second processor to a concurrent service.

The second question considers the *preferable distribution of 6 services over 2 or 3 servers*. The goal is to identify the configuration that camn potentially produce the best response times (performance) versus lowest cost trade-off.

Before using our simulation-based process, one of the authors of this paper used the same information available to this tool and designed his own configuration, which he expected to be the best cost-perfromance compromise. We then used JIMMIE to generate all possible distributions of 6 services on 2 and 3 servers[5]. Traffic was generated based on logs from the existing service. The simulation was run faster than real time, so each configuration was simulated in between 2 and 4 minutes. One server hosting all 6 services is used as a baseline.

The best total times for 1, 2, and 3 servers are shown in Figure 1b. The fastest two-server configurations took 140 minutes to process all of the incoming requests, *i.e.*, twice the cost, half the time. The fastest three-server configurations completed shortly after the traffic generator stopped generating requests (120 minutes), 57% less time. This represents 3 times the cost for a 2.3 performance improvement factor: a more expensive improvement for the expected load. Service distribution is important: 75% of the three-server configurations performed worse than the best two-server configuration.

[5] The idea of using different distributions instead of a complete mirror of the service allows faster operations to be assigned to one server and slower operations to another; a fast operation stuck behind a slow operation in a queue will experience a more dramatic increase in its response time.

The manually-designed configuration took 137 minutes to run, 14% longer than the best three-server configurations and 2% faster than two servers. This confirmed our intuition that configurations produced by experts based on their understanding of the software, expected load, and environment may not necessarily meet the properties desired of the system.

4.2 Trade-Off Analysis and SLA Decision Support

Configuring a software system involves meeting a set of goals, which might be well-specified or only intuitively understood, and might be complementary or might conflict. Our trade-off analysis methodology supports decisions involving the balancing of conflicting goals, whereby one might relax goals in one area in order to achieve other goals, perceived as more important. Without loss of generality, we focus our discussion on cost and performance metrics, as they are common and more readily quantifiable. One of the shortcomings of SLAs identified by Blomberg [1] was that the parties involved in creating SLAs lacked the information and intuition needed to make appropriate decisions about quality of service levels; our approach attempts to provide both information and intuition when planning.

In the previous section, a specific trade-off question was answered. This is effective if the system manager knows which few configuration options should be considered as alternatives. When the right set of options is not known, and the goal is a general exploration of the trade-offs, a large number of possible configurations can be simulated to generate a knowledge base that powers a decision support tool. Using this knowledge base, we visualize various conflicting or correlated metrics to understand the impact that one goal will have on another. The system manager can select potentially viable configurations using a visual tool. Based on their selections, we develop a "fitness score" for each configuration.

Visual Selection Tool. The configuration exploration simulations are visualized as a series of weighted scatter plots in a two-dimensional space, where the two axes correspond to two metrics, offering a head-to-head comparison. For example, Figure 2 shows a sample plot of correlated values, the average response time (x) versus the total time required to complete the processing. The configurations are clustered based on their values for the two metrics. Each point on the plot corresponds to a cluster, and is sized based on the number of configurations that belong in this cluster. The user is shown a series of these plots and is asked to draw a rectangle around the configurations that meet the requirements of the system under configuration. Each successive plot would have clusters colored different shades based on how many configurations in that cluster have been previously selected. Once the user has made all of their selections, he is presented with a list of candidate configurations based on how often those configurations were selected on individual plots. To select a final configuration, the user can watch a recording of the simulation for each candidate to observe metrics in action, choose based on a simple metric (such as the cheapest, the most energy efficient...), or trust a more complex fitness score (as explained in the next section). Hovering over a point produces a "tool-tip" listing the configurations involved, important metrics, and key configuration parameters.

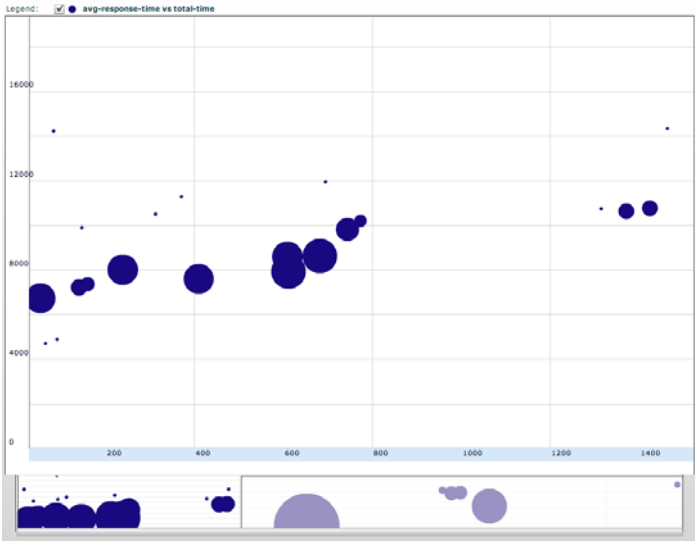


Fig. 2. A visualization of configurations' Average Response Time versus their overall time to process

Fitness Score. Based on the user's selected simulations, the visual tool computes a fitness score. Each configuration, in each head-to-head comparison, is given a comparison score h_{ij} based on its values for the compared metrics i and j:

$$h_{ij} = weight_{ij}[i] * value_{ij}[i] + weight_{ij}[j] * value_{ij}[j]$$

The *weight* is determined by two factors. First, how many configurations were selected out of the total set: if the user accepted a metric's value for most configurations, that metric is relatively unimportant in distinguishing between good and bad configurations. Second, how much of the total range of that metric is selected. If the user was more selective, the metric is more important. For example, consider a comparison of average response time (x) and cost (y) that plots 100 configurations, where response time ranges from 1 to 101 seconds and cost from 200 to 500 dollars. The user draws a selection box from (1,200) to (10,400); there are 34 configurations with response time between 1 and 10, and 56 configurations with cost between 200 and 400. The first factor is $1 - \frac{34}{100} = .66$ for response time and $1 - \frac{56}{100} = .44$ for cost. The second factor for response time is $1 - \frac{(10-1)}{101-1} = .91$ and for cost is $1 - \frac{400-200}{500-200} = .33$. Thus $weight_{ij}[r.t.] = 1.57)$ and $weight_{ij}[cost] = .77$. Note that response time and cost may have different weights when compared to other metrics.

The $value_{ij}[k]$ is normalized based on the relative position of the configuration's value for metric k: at the "good" end of the selection is 2, at the "bad" end is 1, everything in between on a linear scale. Values outside the selection have value 0. Returning to our example, a configuration with response time of 1 (best in the selection) and cost of 500 (outside the selection) would have values 2 and

0, respectively. The fitness score for that configuration for this pair of metrics would be $h_{ij} = 1.57 \times 2 + .77 \times 0 = 3.14$ out of a possible 8 points.

Demonstration. We used the configuration traces from Section 4.1, namely a performance versus cost trade-off for approximately 800 candidate configurations. Using the visual selection tool, we selected ranges in head-to-head comparisons based on the perceived desires of the user community (fast response, consistently, at low cost). Comparisons deemed unimportant were ignored (value = 0). There were 6 metrics, and $\binom{6}{2} = 15$ potential head-to-head comparisons; only 6 comparisons were considered meaningful. Using the visual selection tool, 153 candidate configurations were identified, each having been selected 4 times.

This reduced the candidate configurations by 81%. There remained variation within the remaining configurations; the fitness score allows us to rank the configurations using preferences expressed in the visual tool. We computed the fitness score for each of the six comparisons. Of the 153 candidate configurations identified by the visual tool, 9 were missing from the top 153 of the new ranked list, in all cases due to being very close to the boundaries of the selection box (*i.e.* marginal candidates). This identified 49 top configurations (all within .1 of 41.2, with the remaining configurations under 40, on a theoretical maximum score of 48). This is a 94% reduction. In the future, we plan to empirically validate whether this selection process is actually satisfactory to the user.

4.3 Run-Time Configuration Changes

Configuring software prior to deployment involves identifying a configuration to meet projected future needs. These may not be as predicted or may change over time, and configurations may not perform as expected. Changes will be necessary, and can be made manually or autonomically. Manual changes require well-trained experts who are able to make appropriate configuration changes in response to changed circumstances. We propose and implement a simulation-based tool to train administrators in this task. Providing metric visualization and capturing tools allows for interactive and visual training. Combining modifiable request generation with run-time configuration adaptation through a GUI in simulation allows for training in varying circumstances. It can also be used to train an autonomic system capable of changing system configurations in real-time.

Here we examine the potential benefit of (substantial and expensive) adding autonomic managers to the software by conducting the following experiment. A colleague unassociated with this project generated 160 minutes of variable

Table 1. Cost and performance metrics (and improvements) for 5 configurations

Config	Response Time	Total Time	Cost Increase	Perf. Increase
3-server	1467.3	14272.5	0.0%	0.0%
4-server	698.8	11887	11.0%	52.4%
5-server	316.5	10433.5	21.8%	78.4%
6-server	151	9826.5	37.7%	89.7%
Variable	375	11211.5	3.1%	74.4%

Table 2. Cost and performance metrics (and improvements) comparing manual configuration, manual configuration with composition, and static configurations

Config	Response Time	Total Time	Cost Increase	Perf. Increase
3-server	1353.8	12555	0.0%	0.0%
4-server	611.2	10801	14.7%	54.9%
5-server	199	9775.5	29.8%	85.3%
6-server	52.9	9662.5	53.9%	96.1%
Manual	149.9	9847	-3.5%	88.9%
w/ Composition	91.38	9763	11.0%	93.3%

request traffic, which we logged and used as a test set. Five application configurations were tested: three, four, five, and six servers, and one where an expert user added and removed 1-6 servers at run-time. The distribution of operations was not changed. The expert user had no prior knowledge of the contents of the recorded requests; his decisions were based on watching performance metrics: the load on each server, the queue at each server, and the overall response time. For each configuration, we used average response time and total time as performance measures. We also calculated configuration cost, based on total active CPU time (loosely, a cloud-computing scenario[6]). A server contributed to the cost if it was configured to process requests, whether it actually received requests or not.

The full results are in Table 1; we set a fixed three-server configuration as the baseline and made comparisons relative to that. Fixed five- and six-server configurations performed better than the variable configuration, but were more expensive. Compared to a 3-server configuration, the manually varied configuration was 3% more expensive but performed 75% better. Autonomic changes could allow for faster, more fine-tuned changes; the current method was limited by human response times to a simulation running 100 times faster than in real life. On the other hand, autonomic changes might not be as "intelligent" as manual changes. Nonetheless, based on these results we believe an autonomic manager could improve performance while reducing costs.

Composition-Aware Adaptation. We also evaluated the potential of using composition information to better inform run-time configuration adaptation. We simulated a scenario based on a real-world usage of TAPoR: pre-indexed texts which can be analyzed, in addition to user-submitted texts. A web service request for a list of available indexes precedes some interactions with the service. Some (but not all) interactions with the service follow pre-composed "recipes", where a specific sequence of operations is called in rapid sequence. We modeled the operations as individual services to allow the recipes to represent composed services.

To test this approach, we modified a random 70% of the requests in the data set used above, adding composition. We augmented the user display with a request-type arrival rate metric. The user was able to see the distribution of

[6] The estimated costs depend on the price model assumed; in this work, we assume a hardware-leasing model. Given alternative cost-calculation models, the estimated costs would be different. Our approach is independent of any particular cost models.

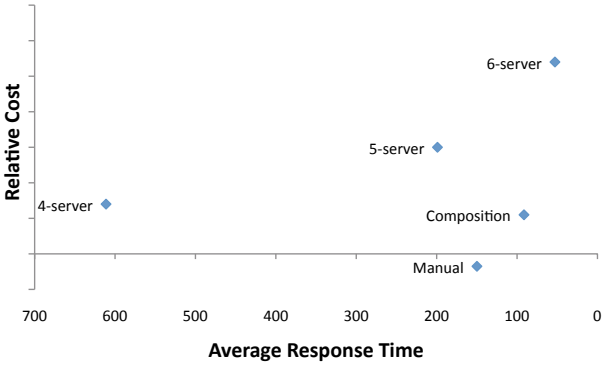


Fig. 3. Relative cost and performance for manual configuration, manual configuration with composition, and static configurations

arriving requests and predict future capacity needs. For example, a request for the list of available indices takes little time and will not impact typical performance metrics, but is a reliable predictor of future requests. This experiment configuration does not perfectly simulate the real-world scenario but we expect it to provide a fairly accurate indication as to the viability of using composition-related knowledge for this application. For this experiment, the administrator used two modification strategies. The first was based on using performance metrics as above, regardless of composition. The second used request metrics: the number and type of requests sent, and the number of requests that have not received a response (as tracked by the traffic generator).

The full results are in Table 2; the trade-offs are shown in Figure 3. When using composition knowledge, we achieved a 93% reduction in response time for an 11% cost increase. Without composition, we achieved a slightly worse reduction in response time (89%) but with a 3.5% cost *decrease* compared to the base. Based on this, admittedly simple, experiment, we found no evidence that including composition in our decision-making improved the run-time adaptation.

5 Conclusion

The work discussed in this paper simulated a real-world application using a services-aware simulation framework, which we used to generate performance data in a variety of configurations. We used this simulation-generated data to present three contributions to configuration planning. First, a question-answering methodology and tool, which we demonstrated by answering two questions. Second, a trade-off analysis decision support tool to help non-technical users derive increased value from their services by improving their understanding of the software service system. We described how this visual tool can be used by users to identify their preferences, which we use to assign a fitness score to future simulation results. Third, we describe a potential training/evaluation tool, but more importantly we use this tool to assess the viability of an autonomic manager for this particular application, showing that we can gain substantial performance

increases for slightly increased cost by manually provisioning just-enough resources. We also suggest that for this service system, knowledge of how services are composed is not likely to improve an autonomic manager.

This work is based on simulated data, and though the simulation when compared to real-world metrics in a set of configurations was statistically identical, the simulation cannot be guaranteed to be accurate. It will also not account for pathological cases. In the near future, we will be using simulation-generated data to inform an autonomic management system capable of re-configuring a system at run-time based on correlating observations of the current environment with past experience. Additional efforts focus on using simulation-generated data to reason more generally about value and perceived quality in trade-off analysis.

References

1. Blomberg, J.: Negotiating meaning of shared information in service system encounters. European Management Journal 26(4), 213–222 (2008)
2. Groothuis, S., Godefridus, van Merode, G., Hasman, A.: Simulation as decision tool for capacity planning. Computer Methods and Programs in Biomedicine 66(2), 139–151 (2001)
3. Uribe, A.M., Cochran, J.K., Shunk, D.L.: Two-stage simulation optimization for agile manufacturing capacity planning. International Journal of Production Research 41(6), 1181–1197 (2003)
4. Kuehne, R., Wille, C., Dumke, R.: Software agents using simulation for decision-making. SIGSOFT Softw. Eng. Notes 30(1), 5 (2005)
5. Grundy, J., Hosking, J., Li, L., Liu, N.: Performance engineering of service compositions. In: Proceedings of IW-SOSWE 2006, pp. 26–32. ACM, New York (2006)
6. Chandrasekaran, S., Miller, J., Silver, G., Arpinar, B., Sheth, A.: Performance analysis and simulation of composite web services. Electronic Markets 13(2), 120–132 (2003)
7. Brebner, P.C.: Performance modeling for service oriented architectures. In: Proceedings of ICSE 2008, pp. 953–954. ACM, New York (2008)
8. Brebner, P., O'Brien, L., Gray, J.: Performance modeling for e-government service oriented architectures (SOAs). In: Ashley Aitken, S.R. (ed.) ASWEC, Australia, pp. 130–138. ACS (March 2008)
9. O'Brien, L., Brebner, P., Gray, J.: Business transformation to SOA: aspects of the migration and performance and QoS issues. In: Proceedings of SDSOA 2008, pp. 35–40. ACM, New York (2008)
10. Miller, J.A., Cardoso, J., Silver, G.: Using simulation to facilitate effective workflow adaptation. In: Annual Simulation Symposium, p. 0177 (2002)
11. Menasce, D., Almeida, V.: Capacity Planning for Web Services: metrics, models, and methods. Prentice Hall PTR, Upper Saddle River (2001)
12. Smit, M., Nisbet, A., Stroulia, E., Edgar, A., Iszlai, G., Litoiu, M.: Capacity planning for service-oriented architectures. In: Proceedings of CASCON 2008, pp. 144–156. ACM, New York (2008)
13. Smit, M., Nisbet, A., Stroulia, E., Iszlai, G., Edgar, A.: Toward a simulation-generated knowledge base of service performance. In: Proceedings of MW4SOC 2009, Newport Beach, CA, USA (2009)