Modeling Parallel Execution Policies of Web Services

Mai Xuan Trang[1(✉)], Yohei Murakami[2], and Toru Ishida[1]

[1] Department of Social Informatics, Kyoto University, Kyoto, Japan
trangmx@ai.soc.i.kyoto-u.ac.jp, ishida@i.kyoto-u.ac.jp
[2] Unit of Design, Kyoto University, Kyoto, Japan
yohei@i.kyoto-u.ac.jp

Abstract. Cloud computing and high performance computing enable service providers to support parallel execution of provided services. Consider a client who invokes a web service to process a large dataset. The input data is split into independent partitions and multiple partitions are sent to the service concurrently. A typical customer would expect the service speedup to be directly proportional to the number of concurrent requests (or the degree of parallelism - DOP). However, we obtained that the achieved speedup is not always directly proportional to the DOP. This may because service providers employ parallel execution policies for their services based on arbitrary decisions. The goal of this paper is to analyse the performance improvement behavior of web services under parallel execution. We introduce a model of parallel execution policy of web services with three policies: Slow-down, Restriction and Penalty policies. We conduct analyses to evaluate our model. Interestingly, the results show that our model have a good accuracy in capturing parallel execution behavior of web services.

Keywords: Parallel execution · Service policy · Performance analysis

1 Introduction

Using the cloud environment to host web services offers numerous benefits. With cloud infrastructures, service providers are able to provide scalability for their services to support parallel execution. However, there are several factors that affect efficiency of parallel execution such as serial fractions in a task as pointed out in Amdahl's law [3,9], and parallel overhead [10]. Service providers may also make arbitrary decisions in selecting policies that control parallel execution of their services. In Service-Oriented Architecture (SOA), service users do not have control over computing resources or services' implementation, and so they need to know performance improvement behaviors of web services in order to configure the optimal parallel invocation to each web service.

In this paper we focus on analysing the effect of data parallelism, a technique often used to improve performance of tasks involving large-scale datasets. The data is split into small independent partitions that are executed in parallel by

Y. Zhang et al. (Eds.): CloudComp 2015, LNICST 167, pp. 244–254, 2016.
DOI: 10.1007/978-3-319-38904-2_25

multiple task instances. This decreases the overall execution time of the task. In previous work [13] we used parallel execution policies of atomic web services to predict the optimal parallelism of composite services. However, the parallel execution policies of a variety of web services was not evaluated. To complement our previous work, this paper performs a series of experiments on real world web services. From the experiment results, we evaluate how well our policy model can capture parallel execution effects of different web services.

The rest of the paper is organised as follows. We begin with a motivating example in Sect. 2. Section 3 introduces parallel invocation of web services and elaborates our proposed policy model. Our testing methodology is described in Sect. 4. We show analysis results and evaluation in Sect. 5. We give some related works in Sect. 6. Finally, Sect. 7 concludes this paper.

2 Motivating Example

Consider a translation application that uses Google translation service to translate a document. In order to reduce the translation time, users configure the application to split the document into M independent partitions, and then send n multiple requests to Google translation service in parallel. Suppose that the method of splitting document is determined (M is fixed). Increasing n is expected to reduce the time taken to translate the whole document. Let *Speed-up* ($S(P)$) of the application be the ratio of the execution time of the application when $n = 1$ to the execution time of the application when $n = P$ ($S(P) = T(1)/T(P)$). A straightforward extrapolation to the higher number of concurrent requests would give the speed-up shown by the dashed line in Fig. 1.

This types of extrapolation is too common and unwarranted in our experience. As we will see, the actual speed-up of the application is more likely to follow the solid line in Fig. 1. The difference between these two predicted curves is significant. This example underscores the importance of obtaining a thorough understanding of the speed-up characteristics of a web service before invoking the services with parallel execution. One way to accomplish this is to assess speed-up patterns by analysing the parallel execution effects of different types of

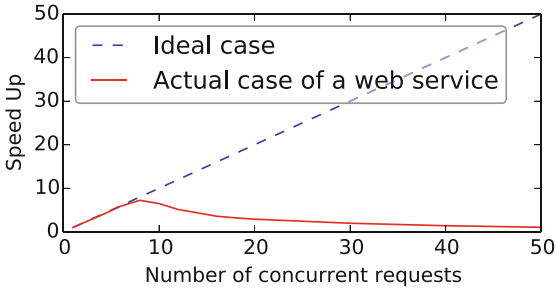


Fig. 1. Different speed-up patterns

web services. Once these patterns are determined we can define a model which can help users to better estimate service performance under parallel execution.

3 Parallel Invocation of a Web Service

We use *Data Parallelism* to perform parallel invocation to a web service as follows. Assume that a client wants to process a large dataset. At the client-side, the input data is split in to M partitions and n threads of the client are created to send n partitions to the service in parallel as shown in Fig. 2. At server-side the service needs to serve n requests in parallel. Execution time required for processing the input data depends on the number of concurrent requests, denoted by $f(n)$.

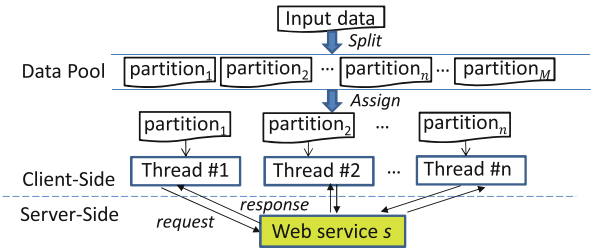


Fig. 2. Parallel invocations of a web service

3.1 Performance Speed-Up

We use *Speed-up* as a measure of the reduction in execution time taken to execute a fixed workload when increasing number of concurrent threads. Speed-up is calculated by the following equation: $S(n) = f(1)/f(n)$, where $f(1)$ is the execution time required to perform the work with a single thread and $f(n)$ is the time required to performance the same task with n concurrent threads.

Different web services may cause different speed-up behaviors. Such behaviors was examined in [1] and three categories were drawn:

- *Linear*–the speed-up ratio is equal to the number of concurrent processes, n, i.e., $S(n) = n$.
- *Sub-linear*–the speed-up ratio with n concurrent processes is lass than n, i.e., $S(n) < n$
- *Super-linear*–the speed-up ratio with n concurrent processes is greater than n, i.e., $S(n) > n$

Several models have been proposed to describe those speed-up behavior categories for parallel algorithms and architectures [9]. A well known and most cited model is Amdahl's law [3], which models the effect of the serial fraction of the task to the speed-up of the task as shown in Fig. 3. Different ratios of serial parts (F) yield different speed-up behaviors.

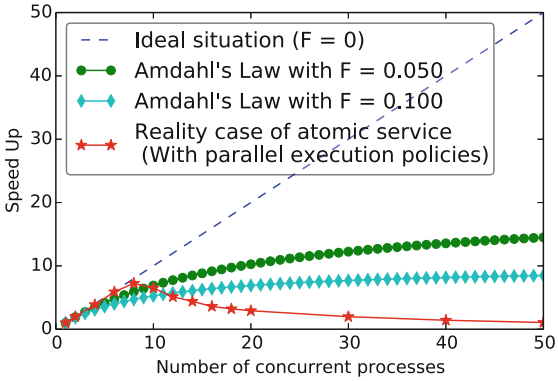


Fig. 3. Speed-up behaviors represented by Amdahl's Law model [13]

Most of existing models assume that the performance speed-up is determined chiefly by task limitations or computing resource limitations. The effect of service provider's arbitrary decision about how to implement parallel execution (parallel execution policies) on the performance speed-up (the solid line in Fig. 3) was not considered.

3.2 Parallel Execution Policy Model

In our previous work [13] we have proposed a model to capture parallel execution policies of web services. The model is defined by a tuple of parameters (α, $\alpha^\star$, α', P), with three policies as follows:

Slow-Down Policy. Performance improvement is throttled when the number of concurrent requests exceeds specified number (P_s) as showed in Fig. 4a. The execution time of the service is given by the following equation:

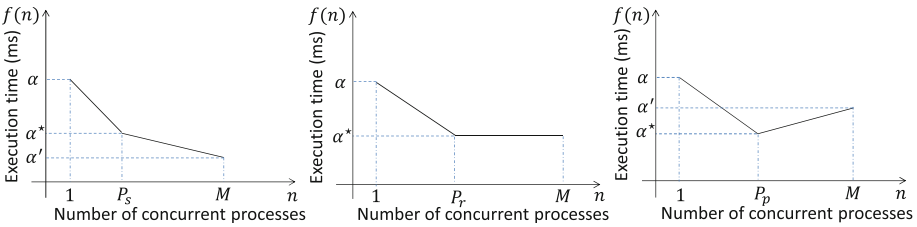
$$f(n) = \begin{cases} \alpha - \frac{\alpha - \alpha^\star}{P_s - 1}(n-1), & \text{if } 1 \le n < P_s \\ \alpha^\star - \frac{\alpha^\star - \alpha'}{M - P_s}(n - P_s), & \text{if } P_s \le n \le M \end{cases}$$

$$\text{with: } \alpha > \alpha^\star > \alpha', \text{ and } \frac{\alpha - \alpha^\star}{P_s - 1} > \frac{\alpha^\star - \alpha'}{M - P_s}$$

Restriction Policy. Service performance statures when number of concurrent requests reaches to a specified number (P_r) as shown in Fig. 4b. The execution time of the service is given by the following equation:

$$f(n) = \begin{cases} \alpha - \frac{\alpha - \alpha^\star}{P_r - 1}(n-1), & \text{if } 1 \le n < P_r \\ \alpha^\star, & \text{if } P_r \le n \le M \end{cases}$$

$$\text{with: } \alpha^\star < \alpha, \text{ and } \alpha' \cong \alpha^\star$$



(a) Slow-down policy (b) Restriction policy (c) Penalty policy

Fig. 4. Performance improvement patterns of parallel execution policies

Penalty Policy. Service performance is reduced when number of concurrent requests exceeds a specified number (P_p) as shown in Fig. 4c. The execution time of the service is calculated by the following equation:

$$f(n) = \begin{cases} \alpha - \frac{\alpha - \alpha^\star}{P_p - 1}(n-1), & \text{if } 1 \le n < P_p \\ \alpha^\star + \frac{\alpha' - \alpha^\star}{M - P_p}(n - P_p), & \text{if } P_p \le n \le M \end{cases}$$
$$\text{with: } \alpha > \alpha^\star, \text{ and } \alpha' > \alpha^\star$$

4 Testing Methodology

We implement a testing system to evaluate our proposed model. A client is created to invoke web services with parallel execution. One challenge is to collect different web services provided by different providers for analysis. One of the most reliable sources we used is the Language Grid [5]. The Language Grid (LG) provides an infrastructure for sharing and combining language services. Different groups or providers can join and share language services on the Language Grid Platform[1]. Currently, more than 140 organizations have joined the Language Grid to share over 170 language services. We also assessed web services from outside the LG, such as from ProgrammableWeb[2].

Experiment Implementation. We implement a client using multi-threading technique to invoke web services. First, the input data is split into independent partitions. Then, n threads of the client are initialized to process n partitions in parallel. Therefore n requests are sent to the service concurrently. We also use pooling technique to stream data partitions to the client whenever a thread is available. We use the integration of the Language Grid and UIMA[3] [12] to realize our test system. First, we create a Document Splitter to split input document into independent partitions and store partitions to a queue. We create

[1] Web services on the LG: http://langrid.org/service_manager/language-services.
[2] ProgrammableWeb: http://www.programmableweb.com/.
[3] Apache UIMA: http://uima.apache.org/.

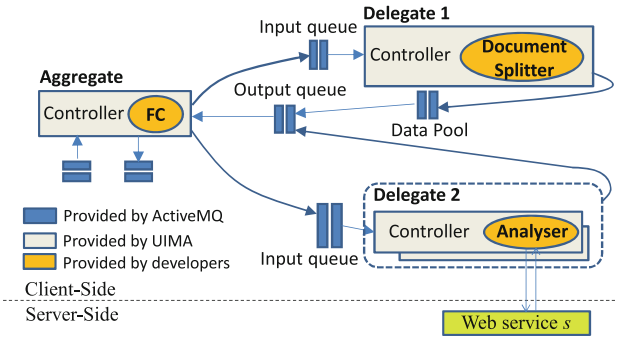
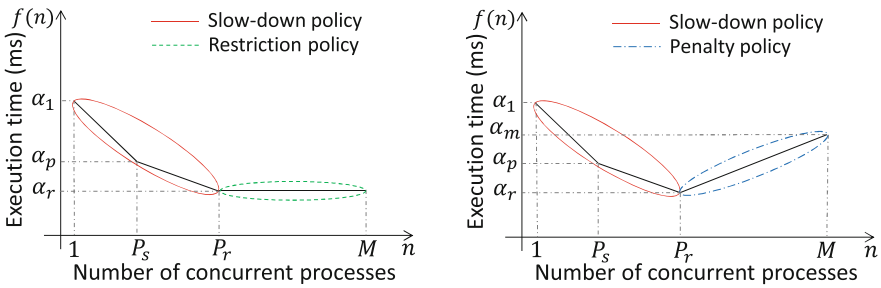


Fig. 5. Implementation concept of the test system

a client which invoke a web service to process data partitions from the queue. We implement a Follow Controller (FC) to connect the Document Splitter and the Client, and control the queues and number of threads of the Client. Figure 5 shows implementation concept of our experiment in the UIMA framework. With this implementation, all n threads of the client are running at all the time. This means that the service has to serve n concurrent requests at all the time.

5 Experiments

This section describes the results of testing the performance impact of parallel execution for web services provided by different providers. We observe that performance improvement patterns of different web services follow different parallel execution policies as defined in Sect. 3.2. Interestingly, from the analysis we observe that, a web service may employ a combination of parallel execution policies as shown in Fig. 6.



(a) Slow-down and restriction

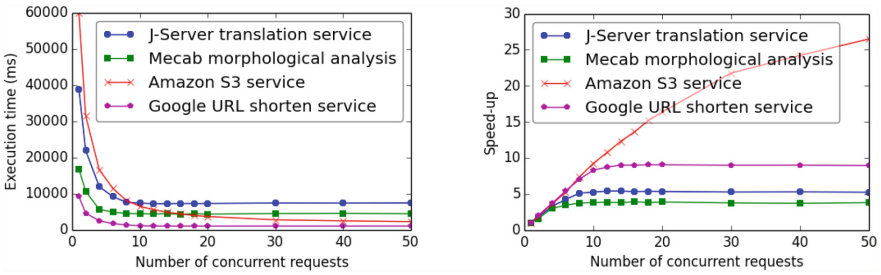
(b) Slow-down and penalty

Fig. 6. Combination of policies

5.1 Combination of Slow-Down Policy and Restriction Policy

Figure 7 depicts different performance improvement behaviors of several web services. The results demonstrate that these services employ both the slow-down policy and the restriction policy as follows:

- Performance improvement of J-Server translation service follows slow-down and restriction policies with $P_s = 4$ and $P_r = 16$.
- Performance improvement of Mecab morphological analysis service follows slow-down and restriction policies with $P_s = 4$ and $P_r = 14$.
- Performance improvement of Google URL shorten service follows slow-down and restriction policies with $P_s = 2$ and $P_r = 12$
- Performance improvement of Amazon S3 service follows slow-down policy with $P_s = 14$. We have not observed restriction behavior of Amazon S3 when n increases until 50.



(a) Performance improvement (b) Speed-up

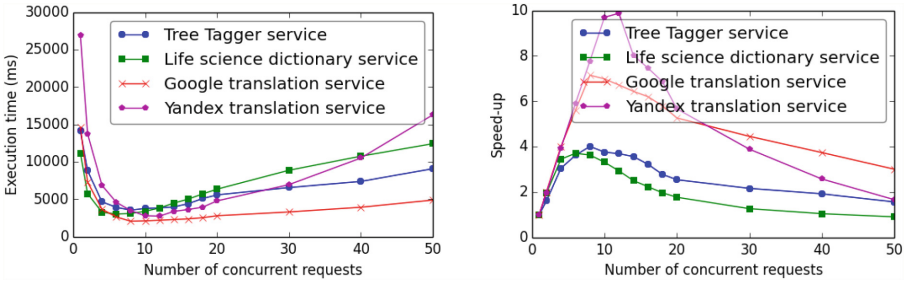
Fig. 7. Web services with slow-down and restriction policy

5.2 Combination of Slow-Down Policy and Penalty Policy

Figure 8 depicts performance improvement behaviors of several other web services. The results demonstrate that the performance improvement of these services is combination of slow-down policy and penalty policy as follows:

- Performance improvement of TreeTagger service follows slow-down and penalty policies with $P_s = 4$ and $P_r = 8$.
- For Life science dictionary service, when number of concurrent requests larger than 6 some requests are blocked and error message are returned (the failed requests are resubmitted until corrected responses are returned). Eventually, the execution time of the service is increased. The performance improvement follows slow-down and penalty policies with $P_s = 6$ and $P_p = 8$.

– Performance improvement of Google translation service follows slow-down and penalty policies with $P_s = 4$ and $P_p = 8$.
– Performance improvement of Yandex translation service follows slow-down and penalty policies with $P_s = 10$ and $P_p = 12$



(a) Performance improvement (b) Speed-up

Fig. 8. Web services with slow-down and penalty policy

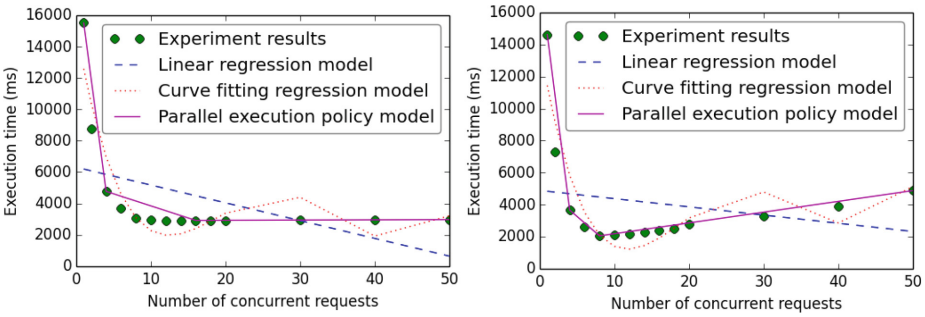
In our analysis, we have analysed more than 50 web services, about two-thirds of them are registered in the Language Grid, the others are collected from outside the Language Grid. The experiment results show that, performance improvement of most of the collected web services can be categorized into the two categories listed above.

5.3 Evaluation

We evaluate our parallel execution policy model by using regression analysis. Our model is compared with two regression models: a linear fitting model and a curve fitting model with a quartic regression (curve fitting function: $y = ax^4 + bx^3 + cx^2 + dx + e$). Figure 9 shows comparison of our policy model and regression models of two different services: J-Server translation service and Google translation service.

We use standard error (S), and R-squared (R^2) to compare the models. S gives some idea of how much the model's prediction differs from the actual results. R^2 provides an index of the closeness of the actual results to the prediction. S and R^2 are calculated by the following equations:

$$S = \sqrt{\frac{\sum_{1}^{n}(Actual_i - Prediction_i)^2}{n-p}}, \text{ and } R^2 = 1 - \frac{\sum_{1}^{n}(Actual_i - Prediction_i)^2}{\sum_{1}^{n}(Actual_i - mean(Actual))^2}$$



(a) J-Server translation service (b) Google translation servicesss

Fig. 9. Evaluating policy model of different services

where n is number of observations, p is the number of regression parameters ($p = 2$ in the case of linear regression and our model, $p = 5$ in the case of the quartic regression model).

The F-test is used to calculate P-value for evaluating statistical significance of our policy model. Table 1 shows comparison of the policy model with the two regression models for different web services. The results show that, in all cases, the policy model has the lower standard error and higher R-Squared than either the linear regression model or the quartic regression model. The P-value of the policy model is significantly low (much less than 0.05). We repeated the evaluation with other web services, the results were similar. We conclude that our policy model has much better accuracy in capturing performance improvement behaviors of web services than the conventional regression models. The policy model is also highly statistically significant and can faithfully estimate the parallel execution effects of web services.

Table 1. Comparison of the proposed model with regression models

	S (milliseconds)			R-squared (%)			P-value
	Linear model	Quartic model	Policy model	Linear model	Quartic model	Policy model	Policy model
J-Server tran.	3287.94	1583.47	1049.75	21.3	86.3	92.0	1.23e-09
Google tran.	3415.02	1680.13	1075.34	4.9	82.73	90.6	2.55e-09
Mecab	3310.78	1634.90	764.73	19.9	85.3	95.7	3.71e-11
Amazon S3	13734.94	6264.98	4795.82	31.2	89.3	91.6	1.57e-09
Google URL	2080.93	1014.05	698.97	23.2	86.3	91.3	4.06e-09
Tree tagger	3078.94	1297.93	659.91	1.2	86.8	95.5	5.82e-11
LSD	2521.01	1267.16	885.72	4.5	89.5	93.2	1.56e-09

6 Related Work

Several papers have addressed the performance of web services. An optimization model for optimal resource allocation across a set of web service class running on the same physical server in virtual environment was proposed in [2]. Bonneta et al. [4] presented a service scripting language–S, together with its compiler and runtime system to efficiently exploit today's multi-core parallel architectures to scale the number of concurrent requests.

Some studies discuss about performance effect of deploying web services on the cloud. Ristov et al. [7] shown that migrating web service on the cloud reduces their performance compared to using the same hardware resources. Although the cloud can scale its resources, it does not guarantee that the performance will scale the same as the scaling factor. Virtualization is another layer that also produces performance discrepancy.

Other studies have introduced several parameters that impact web service performance such as the computation that the web service is performing [8], the CPU power and cores, the message size and the introduced security [11], and even the resource orchestration in the cloud virtual environment [6]. Most existing works do not consider the effect of services policies on performance improvement from the view of service users as we focus in this paper.

7 Conclusion

This paper analysed performance improvement behaviors of different web services under parallel execution. We provided analyses and evaluations of our parallel execution policy model which includes three types of policies: Slow-down policy, Restriction policy, and Penalty policy. By conducting a series experiments on more than 50 web services, we have experimentally confirmed our model well captures the effects of parallel execution policy. Our model has been proved to be superior to regression models in capturing the parallel execution effects of web services. The evaluation results also showed that our parallel execution policy model can well illustrate performance improvement behaviors of web services under parallel execution.

Our model introduced a new factor, which is service' policy, that affect parallel execution efficiency of the service. The model is useful for service users in understanding the parallel execution policies of web services. This will enable users to alter their parallel invocation of a web service to the service policy in order to attain the optimal speed-up. However, the three types of parallel execution policies may not correctly cover all types of web service policies. To make our model more rigorous, we will continue our analysis with larger number of web services and more parameters for parallel execution such as number of concurrent requests per second or the time when users invoke a web service.

Acknowledgments. This research was partly supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) from Japan Society for Promotion of Science (JSPS).

References

1. Alba, E.: Parallel evolutionary algorithms can achieve super-linear performance. Inf. Process. Lett. **82**(1), 7–13 (2002)
2. Almeida, J., Almeida, V., Ardagna, D., Cunha, Í., Francalanci, C., Trubian, M.: Joint admission control and resource allocation in virtualized servers. J. Parallel Distrib. Comput. **70**(4), 344–362 (2010)
3. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the Spring Joint Computer Conference, pp. 483–485. ACM (1967)
4. Bonetta, D., Peternier, A., Pautasso, C., Binder, W.: S: a scripting language for high-performance restful web services. ACM SIGPLAN Not. **47**(8), 97–106 (2012)
5. Ishida, T. (ed.): The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability. Cognitive Technologies. Springer, Heidelberg (2011)
6. Gusev, M., Ristov, S., Velkoski, G., Simjanoska, M.: Optimal resource allocation to host web services in cloud. In: Proceedings of the Sixth IEEE International Conference on Cloud Computing (CLOUD), pp. 948–949. IEEE (2013)
7. Ristov, S., Velkoski, G., Gusev, M., Kjiroski, K.: Compute and memory intensive web service performance in the cloud. In: Markovski, S., Gushev, M. (eds.) ICT Innovations 2012. AISC, vol. 207, pp. 215–224. Springer, Heidelberg (2013)
8. Ristov, S., Gusev, M., Velkoski, G.: Modeling the speedup for scalable web services. In: Bogdanova, A.M., Gjorgjevikj, D. (eds.) ICT Innovations 2014. AISC, vol. 311, pp. 177–186. Springer, Heidelberg (2015)
9. Sun, X.-H., Chen, Y.: Reevaluating amdahls law in the multicore era. J. Parallel Distrib. Comput. **70**(2), 183–188 (2010)
10. Tallent, N.R., Mellor-Crummey, J.M.: Effective performance measurement and analysis of multithreaded applications. In: ACM Sigplan Notices, vol. 44, no. 4, pp. 229–240. ACM (2009)
11. Tentov, A., et al.: Performance impact correlation of message size vs. concurrent users implementing web service security on linux platform. In: Kocarev, L. (ed.) ICT Innovations 2011. AISC, vol. 150, pp. 367–377. Springer, Heidelberg (2012)
12. Trang, M.X., Murakami, Y., Lin, D., Ishida, T.: Integration of workflow and pipeline for language service composition. In: Proceedings of the 9th edn. of the Language Resources and Evaluation Conference (LREC), pp. 3829–3836 (2014)
13. Trang, M.X., Murakami, Y., Ishida, T.: Policy-aware optimization of parallel execution of composite service. In: Proceedings of the 12th IEEE International Conference on Services Computing (SCC), pp. 106–113. IEEE (2015)