Policy-Aware Language Service Composition

Trang Mai Xuan, Yohei Murakami and Toru Ishida

Abstract Many language resources are being shared as web services to process data on the Internet. As dataset size keeps growing, language services are experiencing more big data problems, such as the storage and processing overheads caused by the huge amounts of multilingual texts. Parallel execution and cloud technologies are the keys to making service invocation practical. In the Service-Oriented Architecture approach, service providers typically employ policies to limit parallel execution of the services based on arbitrary decisions. In order to attain optimal performance, users need to adapt to the services policies. A composite service is a combination of several atomic services provided by various providers. To use parallel execution for greater composite service efficiency, the degree of parallelism (DOP) of the composite services need to be optimized by considering the policies of all atomic services. We propose a model that embeds service policies into formulae and permits composite service performance to be calculated. From the calculation results, we can predict the optimal DOP for the composite service that allows the best performance to be attained. Extensive experiments are conducted on real-world translation services. The analysis results show that our proposed model has good prediction accuracy in identifying optimal DOPs for composite services.

Keywords Parallel execution policy · Performance prediction · Degree of parallelism

T. Mai Xuan (✉) · T. Ishida
Department of Social Informatics, Kyoto University,
Kyoto 606-8501, Japan
e-mail: trangmx@ai.soc.i.kyoto-u.ac.jp

T. Ishida
e-mail: ishida@i.kyoto-u.ac.jp

Y. Murakami
Unit of Design, Kyoto University, 91 Chudoji Awata-cho,
Kyoto 600-8815, Japan
e-mail: yohei@i.kyoto-u.ac.jp

Y. Murakami et al. (eds.), *Services Computing for Language Resources*,
Cognitive Technologies, https://doi.org/10.1007/978-981-10-7793-7_4

1 Introduction

The Service-oriented architecture (SOA) is a widely accepted and engaged paradigm for the realization of business processes that incorporate several distributed, loosely coupled partners. It allows users to combine existing services to define composite services that meet users' requirements. With the increasing volume of data on the Internet, the services need to deal with extremely large-scale datasets and thus the emergence of data-intensive services. As in natural language processing area, many language resources (e.g. machine translator, morphological analysis, etc.) have been provided as language services. Some problems in language processing such as translation of a large document often require long processing times. This type of data-intensive service requires parallel execution to improve performance.

Using the cloud environment to host web services offers numerous benefits. With cloud-based infrastructures, service providers are able to provide scalability for their services through parallel execution. However, there are several factors that potentially limit the efficiency of parallel execution such as serial fractions in a task as pointed out in Amdahl's law [1], and parallel overhead [2]. These factors consider the limitation of the program or computing resources from the viewpoint of service providers. Providers can control the limitation to enhance parallel execution efficiency.

In SOA, however, from the perspective of service users, the service providers' policies on parallel execution are an important factor that impacts the parallel execution efficiency. Service users cannot change the policies as they want when invoking the services. Since service providers have different preferences, their policies may also be different. For instance, only if a provider is rich in computing resources may he readily allow large numbers of concurrent requests. Service providers may have different policies that will trigger a degradation in service performance; e.g. the number of concurrent requests exceeds the limit set by the service provider. Therefore, when invoking a service with parallel execution, users should consider the service policy in order to attain the optimal performance improvement.

Composing and optimizing a composite service, or a workflow of atomic services, has gained increasing attention in SOA. Technologies such as data-intensive and many-task computing [3], and scientific workflows [4] have the potential to enable rapid data analysis. Many studies have proposed parallel and pipelined execution technique to speedup workflows [5], as well as adaptive parallel execution approach for workflows in cloud environments regarding the availability of resources [6]. Most of the approaches proposed to date do not consider the policies of the atomic services in optimizing the composite services. Different from current approaches, this chapter focuses on optimizing the Degree of Parallelism (DOP) of a composite service by considering the service policies of all participating providers. To tackle this problem, this chapter proposes a model that embeds parallel execution policies of atomic services into formulae to calculate composite service performance under parallel execution. To this end, we set the following goals:

- *Parallel execution policy model*: We propose a model to capture the policies of atomic services observed when using parallel execution. The model makes it easy to embed service policies into the calculation of composite service performance.
- *Predict optimal parallel execution of composite services*: We define formulae to calculate performance of a composite service with different structures. From the calculation, we predict the optimal DOP for the composite service.
- *Validate the model*: We validate our model by evaluating the accuracy of our model when predicting the optimal DOP of composite services. Extensive experiments are made on the real-world language services.

The remainder of the chapter is organized as follows. Section 2 presents a motivating example. Section 3 describes parallel execution model of atomic and composite services. The prediction model is proposed in Sect. 4. We evaluate our model in Sect. 5. We show some related work in Sect. 6. Finally, Sect. 7 concludes the chapter.

2 Motivating Example

The Language Grid is a service oriented infrastructure for sharing and combining language services [7, 8]. Service providers can share their tools (e.g. translator, dictionary, etc.) as atomic web services on this platform. Different providers are permitted to set different policies to maintain the QoS of their provided services. To build complex applications, users can combine different exiting atomic services to define new composite services that meet their requirements. For composite services dealing with large-scale data, using parallel execution can reduce the execution time. However, to raise the efficiency of parallel execution, we need to consider the different policies, especially parallel execution policies, of different atomic services when configuring the composite services.

To illustrate our approach, we show the parallel execution of a composite service in Fig. 1. This simple workflow shows a two-hop translation service consisting of two different translation services: the first service translates a document from Japanese to English, and then the second service translates the result from English to Vietnamese. The abstract atomic services in the composite service can be bound to different concrete translation services provided by different providers such as Google translation service, Bing translator, J-Server, and Baidu translation service. To reduce execution time, the client invokes the composite service with concurrent execution. The input data is split into independent portions, and several portions are processed in parallel. If the client sets the DOP of the composite service to n, each translation service in the composite service is executed concurrently as n processes.

Now, let us consider a scenario where the first and second atomic services are bound to J-Server translation service and Google translation service, respectively. J-Server and Google set different parallel execution policies for their translation services. Google limits the number of concurrent requests sent to its translation service from a registered user to 8. If more than 8 concurrent requests are received, the

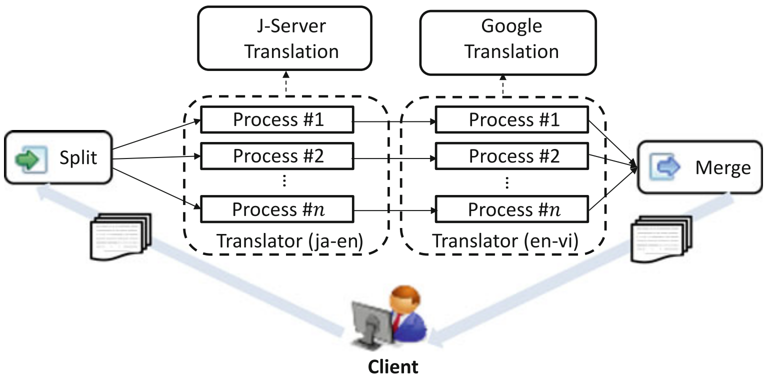


Fig. 1 Parallel execution of a composite service

performance becomes worse. J-Server limits its translation service to 16 concurrent requests per user, if more than 16 requests are sent to the service, the performance does not improve. In this scenario, when configuring parallel execution of the composite service we need to specify a suitable DOP of the composite service. This configuration should match the policies of all atomic services in order to optimize the composite service's performance. Several questions to be asked here are (1) How can we model parallel execution policies of web services? and (2) How the performance of a composite service can be predicted from the atomic services' policies? These issues are addressed in the next sections.

3 Parallel Execution Policy Model

In this section we propose a model to capture parallel execution policies of services based on our observation of executing different atomic services.

3.1 Parallel Execution of a Language Service

Language services are designed to process huge datasets. These services can benefit from the use of parallel execution to improve performance. We use *Data Parallelism* to set the parallel invocation of a language 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 $f(n)$.

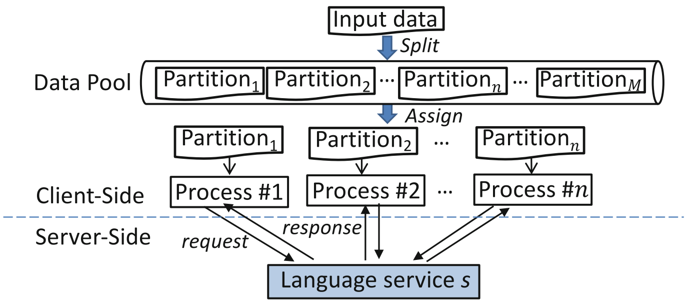


Fig. 2 Parallel execution of a language service

We use *Speedup* as a measure of the reduction in execution time taken to execute a fixed workload when increasing number of concurrent threads. Speedup 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 perform the same task with n concurrent threads.

Several models have been proposed to describe those speedup curves for parallel algorithms and architectures [9]. The most cited model is Amdahl's law [1], which models the effect of the serial fraction of the task (F) to the speedup of the task. F is the fraction of the task's computation (algorithm) that is serial and cannot be parallelized. F ranges from 0 to 1 depending on algorithm of the task.

Amdahl's Law, however, does not include parameters that account for the impact of a service's policy on the execution speed. SOA allows service providers to make arbitrary decisions in setting policies to control parallel execution of their services. These policies limit the performance speedup achievable with their services. In some cases, if number of concurrent processes exceeds a certain number, service performance may actually be throttled. Figure 3 shows speed enhancements possible with an atomic service with the parallel execution under different conditions. The dashed line shows an ideal case of the service performance speedup ($F = 0$). The dotted and

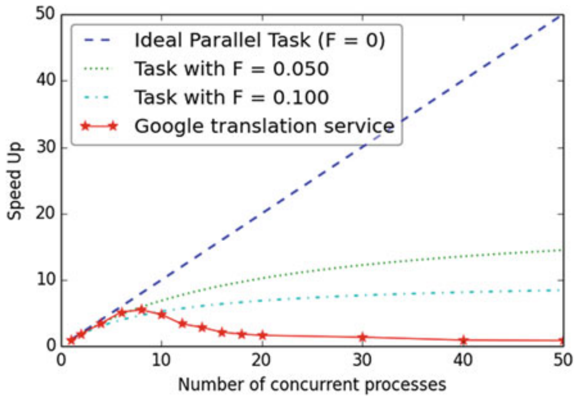


Fig. 3 Parallel execution speedup of a task

dash-dot lines show the performance speedup of the service follow the Amdahl's law with serial faction (F) values of 0.05 and 0.1, respectively. The solid line shows a real case of Google translation service. With this policy, when the number of concurrent processes exceeds 8, service user experiences a drop in the service performance.

3.2 Parallel Execution Policies

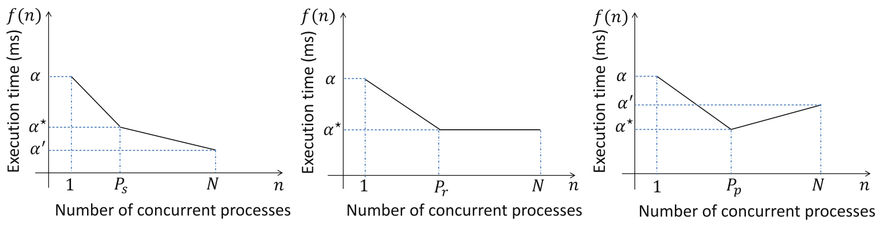
Our assumption is, for each service there exists a certain number of concurrent processes specified by the service provider, beyond which the performance improvement trend is changed. We defined our policy model [10] as following:

Definition (*Parallel Execution Policy*) The parallel execution policy of a web service is defined by the change of the performance improvement trend of the service under parallel execution. This performance improvement trend is determined by the tuple of parameters (α, $\alpha^\star$, α', P), each parameter is defined as follows:

- Suppose that the service processes M data partitions using parallel execution. Execution time of the service depends on number of concurrent processes (n) of the service, denoted by $f(n)$.
- α is execution time of the service when the M partitions are serially executed, i.e., $n = 1$: $f(1) = \alpha$.
- P is the upper bound of concurrent processes specified by the service provider, beyond which the performance improvement trend changes.
- $\alpha^\star$ is time taken by the service to process M partitions with P concurrent processes: $f(P) = \alpha^\star$.
- α' is time taken by the service to process M partitions with N ($P < N \leq M$) concurrent processes: $f(N) = \alpha'$.

Parallel execution policies of language services normally are not explicitly described by the service providers. We tested and observed parallel execution effect of many services registered on the Language Grid [8], and many other external services such as Google translation service, Baidu translation service, Amazon S3 service, etc. We found there were three main patterns in the performance improvement. We described these patterns by using the above parameters and define three types of parallel execution policy as follows: *Slow-down policy*, *restriction policy*, and *penalty policy*.

Slow-down Policy. This policy reduces the performance improvement when the number of concurrent processes exceeds specified number (P_s). This means that execution time of the service decreases as the number concurrent processes increases. When number of concurrent processes exceeds P_s the execution time decreases but at a lower rate. This policy may due to the parallel execution limitation of the services' implementation. The performance pattern, given by this policy, is depicted in Fig. 4a. The execution time of the service to process M partitions can be calculated by the following equation:



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

Fig. 4 Performance patterns of parallel execution policies

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

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

Restriction Policy. With this policy, service providers limit the maximum number of concurrent requests to their service. Service performance shows no improvement when the number of concurrent processes exceeds specified number (P_r). This may due to limitation of the service providers' computing resources. This policy creates the service performance pattern shown in Fig. 4b. Execution time of the service to process M partitions can be calculated by the following equation:

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

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

Penalty Policy. In some cases, due to some commercial strategies or security concerns, the service provider may penalize concurrent requests if the number of them exceeds specified number (P_p), service performance is actually reduced. The performance pattern of this policy is shown in Fig. 4c. The execution time of the service is calculated by the following equation:

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

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

For simplicity, this work assumes each service provider uses a static value of P. Since, the value of P is normally not explicit stated, in order to choose P we invoke the service with different parallel execution configuration using a test data set. We analyze execution time, and determine P.

4 Prediction of Composite Service Performance

In this section we propose a model that can predict composite service performance when using parallel execution. The prediction changes with the workflow structures of the composite services.

4.1 *Parallel Execution of Composite Language Service*

To build an advanced language application, developers combine several atomic language services in a workflow. A service described by a workflow is called *composite service*. In order to improve the composite service performance when processing huge amounts of data, a promising approach is to use parallel execution. In this work, we focus on two parallelism techniques described as following: data parallelism and workflow pipeline execution.

Data parallelism. Similar to Data Parallelism in atomic service, when a composite service processes huge amount of data sets, the data sets are split into independent portions, and several computing tasks of the composite service are instantiated to process several portions in parallel.

Workflow pipeline execution. When a single workflow is operated in parallel on many data partitions, *Workflow pipeline execution* denotes that the processing of several independent partitions by several instances of an atomic service are independent. This parallelism enables pipeline processing of a workflow. That is, when n concurrent requests are sent to a composite service, multiple instances of each atomic service are created and processed in parallel. A pooling technique is used such that when processing M data sets, n out of M data sets are streamed to the composite service in parallel without waiting for responses. The execution of the composite service is done in pipeline manner. Consider an example of a sequential composition of two services. This example yields the pipeline processing time-line shown in Fig. 5, where $L = \lceil M/n \rceil$ is number of time-steps needed to send M data sets. At the beginning of time period, n data set are sent in parallel. t_{ij}^n is the time that n concurrent processes of service s_i take to finish processing n data sets at time step j.

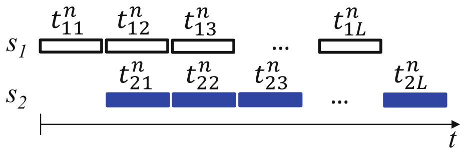
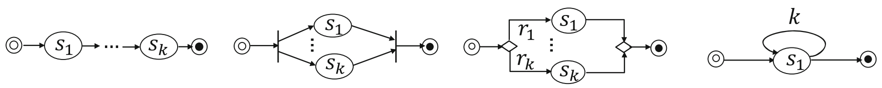


Fig. 5 Pipeline processing time-line of a composite service



(a) Sequential **(b)** Parallel **(c)** Conditional **(d)** Loop

Fig. 6 Four types of composite structures [11]

4.2 Prediction Model

A composite service can be seen as a set of atomic services that cooperate to execute a process that defines the interaction workflow. There are four basic composite structure: *Sequential*, *Parallel*, *Conditional* and *Loop*, see Fig. 6, where circles represent atomic services and arrows represent the transfer of data between services. QoS of a composite service is aggregate QoS of all atomic services. Existing QoS calculation methods can be classified into two categories: Reduction method with single QoS for service composition [11], and direct aggregation method with multiple QoSs for the service composition [12]. We adapt the formulae proposed in [11] to calculate execution time of composite services in different structures under parallel execution.

4.2.1 Prediction of Sequential Structure

Consider a *Sequential* combination of two services s_1 and s_2. These two services have different parallel execution policies as specified in Sect. 3. $f_1(n)$ and $f_2(n)$ are predicted execution time of s_1 and s_2 when processing M partitions, respectively. Assume that $f_1(n) > f_2(n)$, processing time-line of the composite service is depicted in Fig. 7a. From this time-line, we can easily see that execution time of the composite service ($f_c(n)$) is calculated as follows:

$$f_c(n) = f_1(n) + t_{2L}^n \tag{1}$$

t_{2L}^n is the time taken by service s_2 to finish processing last n partitions in parallel. Given that the execution time of s_2 to process each partition is approximate equal:

$$t_{2L}^n \approxeq f_2(n)/\lceil M/n \rceil \tag{2}$$

From Eqs. (1) and (2) we have:

$$f_c(n) \approxeq f_1(n) + f_2(n)/\lceil M/n \rceil \tag{3}$$

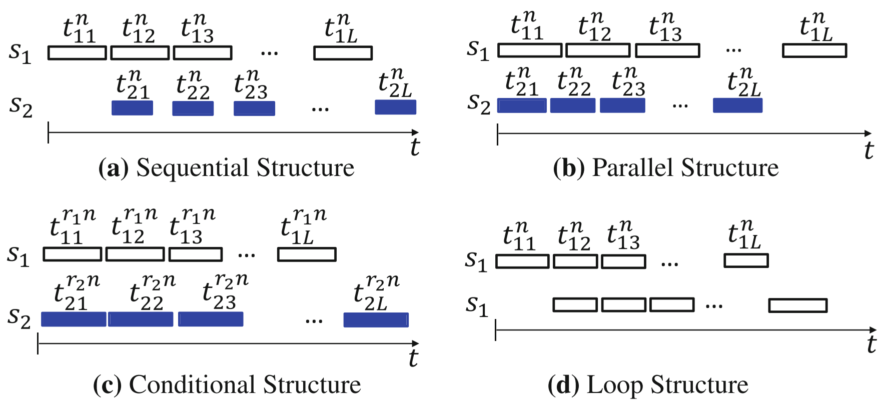


Fig. 7 Processing time-line of different structures

In general, execution time of a composite service that is a sequential combination of k services $(s_1, s_2, \ldots, s_k)$ can be predicted as follows:

$$f_c(n) \approxeq \max_{i=1}^{k} f_i(n) + \left(\sum_{i=1}^{k} f_i(n) - \max_{i=1}^{k} f_i(n)\right) / \lceil M/n \rceil \tag{4}$$

where $f_1(n)$, $f_2(n)$, ..., $f_k(n)$ are execution time of each services using parallel execution with with n concurrent instances.

4.2.2 Prediction of Parallel Structure

Consider a composite service with service s_1 and s_2 in the parallel structure. In this case the two services process data in parallel. The original dataset is separated into M partitions, n of which are sent to the composite service concurrently. Suppose $f_1(n)$ and $f_2(n)$ are execution time of s_1 and s_2 predicted using the policy model. The processing time-line of the composite service is illustrated in Fig. 7b. From this processing time-line, we can easily predict the execution time of the composite service $(f_c(n))$ is the maximum value of $f_1(n)$ and $f_2(n)$:

$$f_c(n) \approxeq \max(f_1(n), f_2(n)) \tag{5}$$

To generalize, consider a parallel combination of k services $(s_1, s_2, \ldots, s_k)$. Using policy model, $f_i(n)$ is predicted execution time of service s_i with n concurrent processes. The execution time of the composite service is predicted with the following generalized equation:

$$f_c(n) \approxeq \max_{i=1}^{k} f_i(n) \tag{6}$$

4.2.3 Prediction of Conditional Structure

Consider a *Conditional* combination of two service s_1 and s_2. The *Conditional* structure states that a portion of n partitions will be sent to service s_1, the remainder is sent to service s_2. Suppose that the ratios are r_1 and r_2 ($r_1 + r_2 = 1$). This means that $r_1 n$ partitions are processed by s_1 in parallel, while the remaining $r_2 n$ partitions are processed by s_2 concurrently. In total, $r_1 M$ partitions are processed by s_1, and $r_2 M$ partitions are processed by s_2. Using the policy model we can calculate execution time of s_1 and s_2 are $f_1(r_1 n)$ and $f_2(r_2 n)$ respectively.

When $n = 1$, partitions are sent to the composite service one at a time, one partition is processed by s_1 or s_2 at a time. The execution time the composite service to process M partitions is execution time of s_1 to process $r_1 M$ partitions adds execution time of s_2 to process $r_2 M$ partitions. When $n > 1$, since there are $r_1 n$ concurrent processes of s_1 and $r_2 n$ concurrent processes of s_2, the execution of the conditional structure is similar with parallel structure where s_1 processes rM partitions and s_2 processes $r_2 M$ partitions concurrently. In this case, processing time-line of the conditional structure is as illustrated in Fig. 7c; the number of time-steps, $L = \lceil M/n \rceil$. The execution time of the conditional structure $f_c(n)$ is predicted as follows:

$$f_c(n) \approxeq \begin{cases} f_1(1) + f_2(1), & \text{if } n = 1 \\ \max(f_1(r_1 n), f_2(r_2 n)), & \text{if } n > 1 \end{cases} \tag{7}$$

To generalize, consider a *Conditional* structure of k services ($s_1, s_2, \ldots, s_k$). Suppose that $r_1, r_2, \ldots, r_k$ are the ratios of requests sent to each service. We have:

- $\sum_{i=1}^{k} r_i = 1$
- $r_i M$ partitions are processed by s_i
- With n partitions sent to the composite service in parallel, $r_i n$ instances of service s_i will be initiated and processed concurrently.
- Execution time of service s_i when processing $r_i M$ partitions with $r_i n$ concurrent instances can be calculated with $f_i(r_i n)$, where the parallel execution policy of s_i is $(\alpha_i, \alpha_i^\star, \alpha_i', P_i)$.

The execution time of the composite service ($f_c(n)$) is predicted by the following equation:

$$f_c(n) \approxeq \begin{cases} \sum_{1}^{k} f_i(1), & \text{if } n = 1 \\ \max_{i=1}^{k} f_i(r_i n), & \text{if } n > 1 \end{cases} \tag{8}$$

4.2.4 Prediction of Loop Structure

Consider a loop of service s_1 with the iteration number of 2. This loop can be converted to a *Sequential* structure of two services s_1. Execution time of service s_1 is $f_1(n)$ can be predicted by using the policy model. The processing time-line of the composite service is illustrated in Fig. 7d; the number of time-steps is $L = \lceil M/n \rceil$. At the first time step, n requests are sent to s_1 in parallel, t_{11}^{n} is the time taken by n instances of s_1 to process the first n partitions. From the second time step to $(\lceil M/n \rceil - 1)$ time step, $2n$ requests are sent to service s_1 concurrently, so the time to process n partitions is maximum value of $f_1(n)$ and $f_1(2n)$. Let ΔT is duration from second time step to $(\lceil M/n \rceil - 1)$ time step, ΔT is calculated as follows:

$$\Delta T \approxeq (\lceil M/n \rceil - 1) \frac{\max(f_1(n), f_1(2n))}{\lceil M/n \rceil}$$

At the last time step, n last partitions are processed in parallel by n concurrent instances of s_1. The execution time of the composite service $(f_c(n))$ is predicted by the equation below:

$$\begin{aligned} f_c(n) &\approxeq 2\frac{f_1(n)}{\lceil M/n \rceil} + \Delta T \\ &\approxeq 2\frac{f_1(n)}{\lceil M/n \rceil} + (\lceil M/n \rceil - 1)\frac{\max(f_1(n), f_1(2n))}{\lceil M/n \rceil} \end{aligned} \tag{9}$$

In general case where a service s is looped k time. The execution time of the composite service can be calculated by following general equation:

$$f_c(n) \approxeq \frac{2\sum_{j=1}^{k-1} \max_{i=1}^{j} f(in)}{\lceil M/n \rceil} + (\lceil M/n \rceil - k + 1)\frac{\max_{i=1}^{k} f(in)}{\lceil M/n \rceil} \tag{10}$$

In the case a composite service that contains different control structures. To calculate execution time of this type of composite service, we first reduce the complex workflow to a simple one which contains only sequential structure. We use the reduction methodology proposed in [11] for converting the complex service to a sequential combination of components, each component is a composite service with a control structure. Using equation for each structure defined above we calculate execution time of each component. Finally, applying equation of sequential structure we can calculate execution of the whole complex composite service.

5 Evaluation

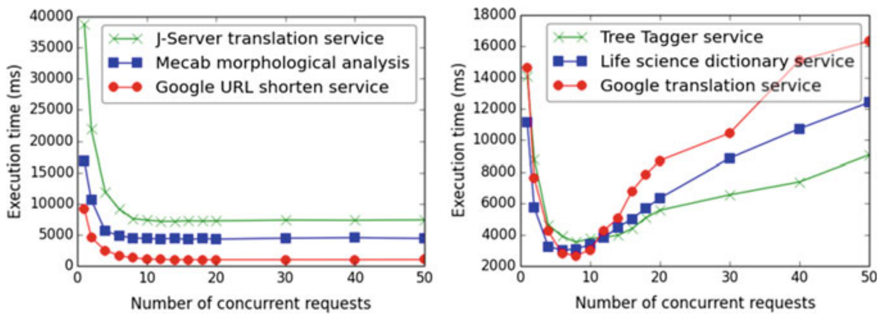
In this section we describe the results of testing the performance impact of parallel execution of web services offered by different providers. We also evaluate the accuracy of our prediction model by comparing its output to actual results.

5.1 Evaluation of the Parallel Execution Policy Model

We focus on analysing real world translation services. We use the integration engine of the Language Grid and UIMA [13] to configure and invoke services with parallel execution. In our experiments, a document with 500 paragraphs is translated from Japanese to English. The document is separated into one paragraph partitions, and several partitions are translated in parallel. The results demonstrate there are two group of parallel execution policies: Combination of *Slow-down Policy* and *Restriction Policy*, and combination of *Slow-down Policy* and *Penalty Policy* as shown in Fig. 8. For example, Mecab morphological analysis service employs slow-down and restriction policies with $P_{smecab} = 4$ and $P_{rmecab} = 14$, whereas the Tree Tagger service employs slow-down and penalty policies with $P_{stree} = 4$ and $P_{ptree} = 8$.

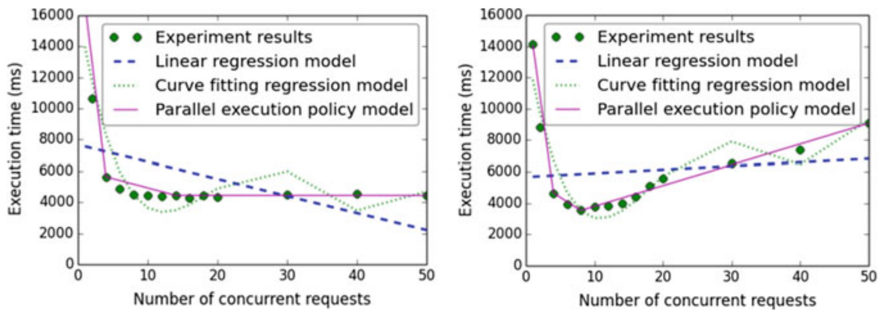
We evaluate our parallel execution policy model by using regression analysis. Our model is compared with two regression models: a linear fitting model and a curve fitting model with a quartic regression (curve fitting function: $y = ax^4 + bx^3 + cx^2 + dx + e$). Figure 9 shows comparison of our policy 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. We also calculate P-value for evaluating statistical significance of our policy model. Table 1 shows comparison of the policy model with other two regression models. 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). This indicates that our policy model is highly statistically significant and can faithfully estimate the parallel execution effects of web services.



(a) Slow-down and restriction policies **(b)** Slow-down and restriction policies

Fig. 8 Parallel execution policies of atomic services



(a) Mecab morphological service

(b) Tree Tagger service

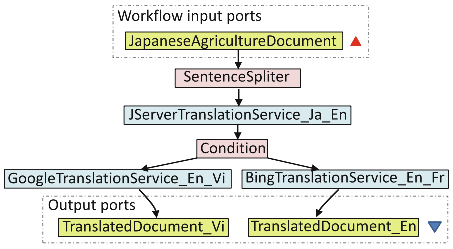
Fig. 9 Regression analysis

Table 1 Comparison of the proposed model with regression models

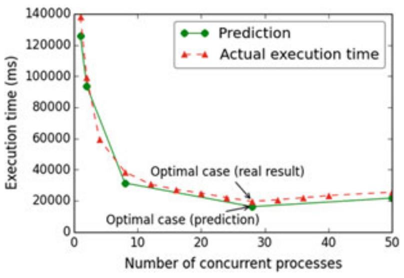
	S (milliseconds)			R-Squared (%)			P-value
	Linear model	Quartic model	Policy Model	Linear model	Quartic model	Policy model	Policy model
J-Server Tran.	3287.94	1583.47	1049.75	21.3	86.3	92.0	1.23e-09
Mecab	3310.78	1634.90	764.73	19.9	85.3	95.7	3.71e-11
Google URL	2080.93	1014.05	698.97	23.2	86.3	91.3	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
Google Tran.	3415.02	1680.13	1075.34	4.9	82.73	90.6	2.55e-09

5.2 *Evaluation of the Prediction Model*

We analyse a realistic composite service which is a two-hop translation combining t-wo translation service as shown in Fig. 10a. This composite service is combination of three translation services with two structures, i.e. *Sequential* structure and *Condition-al* structure. We use this composite service to translate a mixed document containing information about rice and fertilizer. First, the document is translated into English using J-Server translation service. Then, that part of translated document, containing information about rice, is translated into Vietnamese by Google translation service. The other part, containing information about fertilizers, is translated into French by Bing translation service. J-server, Bing translation service, and Google translation service have different parallel execution policies as specified in previous section. Figure 10b shows a performance prediction when the composite service translates a


Workflow input ports
JapaneseAgricultureDocument
SentenceSpliter
JServerTranslationService_Ja_En
Condition
GoogleTranslationService_En_Vi
BingTranslationService_En_Fr
Output ports
TranslatedDocument_Vi
TranslatedDocument_En

(a) Two-hop translation service



(b) Evaluation result

Fig. 10 Evaluating a composite service

document of 500 paragraphs. We can see that, in this case our model correctly predict the best DOP (28) where the composite service attains the best performance.

In order to evaluate the accuracy of the proposed model we invoked the above composite service with 15 different agriculture documents with different sizes ranging from 100 paragraphs to 1500 paragraphs. We use several measures as follows: Mean Prediction Error (*MPE*) to evaluate prediction bias, Mean Absolute Error (*MAD*) to show absolute size of prediction errors, and Tracking Signal (*TS*) to check whether there is some bias or not.

Table 2 shows evaluation of the model in two aspects:

- Predicting the optimal number of concurrent processes: $MPE = 0$ and $MAD = 1.07$ mean that the model yields good predictions; the average absolute error is 1.07 units.
- Prediction of the optimal execution time: $TS = 15$ indicates that the model is not so accurate in predicting the optimal execution time.

The proposed prediction model is not so accurate and always under-predict the optimal execution time. One reason for this is that our current model omits some parallel overhead such as time for creating and terminating threads. The accuracy of the model would be improved by adding the parallel overhead time to calculate execution time of an atomic service under parallel execution. We will consider this issue in our future works.

6 Related Work

Scientific workflows have emerged as an useful instrument to comprehensively design and share the best practices and create reproducible scientific experiments. Many Scientific Workflow Management Systems (SWMSs) have been developed, such as Taverna [14], or Kepler [15] to enable graphical design, execution and monitoring of local or distributed scientific workflows. In the era of big data, workflow optimization has become an important issue. One of the common optimization targets

Table 2 Evaluation result

Input data	Optimal degree of parallelism (DOP)			Optimal execution time (millisecond)		
	Prediction	Actual result		Prediction	Actual result	
100 paragraphs	28	24	MPE = 0 MAD = 1.07 TS = 0	2596	3531	MPE = 1204.33 MAD = 1204.33 TS = 15
200 paragraphs	24	24		5373	6550	
300 paragraphs	24	24		8390	9360	
400 paragraphs	24	24		11386	12670	
500 paragraphs	24	28		13984	15287	
600 paragraphs	28	28		17580	18574	
700 paragraphs	24	24		23177	24696	
800 paragraphs	24	24		25273	26344	
900 paragraphs	24	28		30170	31146	
1000 paragraphs	28	28		35567	36627	
1100 paragraphs	24	24		42164	43238	
1200 paragraphs	28	28		43960	45789	
1300 paragraphs	24	24		46757	48043	
1400 paragraphs	28	24		54229	55631	
1500 paragraphs	28	28		57951	59136	

is to improve the scientific workflow runtime performance. As the typical scientific workflow is executed in e-Science infrastructures, several different approaches exist to intelligently schedule workflows or tasks on the Grid and Cloud [16]. There also some recent works on scheduling workflows based on resource/task allocation [17]. These solutions are usually implemented for a specific SWMS and aim at the acceleration of workflow through scheduling.

There are also some existing studies proposing different factors to compute QoS in order to optimize service composition. In [18], to deal with context-aware QoS, where the QoS of a service may vary in different context, authors proposed a dynamic service selection approach based on context-aware QoS. A method that involved human to analyze QoS for service composition was also proposed in [19].

To the best of our knowledge, there is no existing works that consider service providers' decision on parallel execution of atomic services in service composition. In this regard, we conclude that our contribution is novel.

7 Conclusion

This chapter proposed a prediction model that considers the policies of atomic service providers in predicting the performance of a composite service under parallel execution. To the best of our knowledge, this is the first attempt to incorporate service providers' policies on parallel computing into service composition. We found that the parallel execution policies of atomic services fell into three different categories: Slow-down policy, Restriction policy, and Penalty policy. Based on these policies, our prediction model can calculate the execution times of composite services when using parallel execution and estimate the optimal degree of parallelism for the composite services. Our model is helpful in building a mechanism that can control the parallel execution of workflows; a Workflow Management System that uses this mechanism can execute a workflow with optimal DOP.

We conducted experiments on real-world translation services to evaluate the accuracy of our model. The analysis results show that our model offers good prediction accuracy with regard to identifying the optimal degree of parallelism for composite services. Our model is, however, not so accurate in predicting the execution time. Our future work includes improving the model to increase prediction accuracy and extending the model for other QoS metrics such as cost and reputation.

In designing the proposed model, we assumed that parallel execution policy of one service is static. That is, the limit set by the service provider on the number of concurrent process does not change regardless of input data size. However, in cloud environment, it seems highly likely that service providers will dynamically change their policy for requests of different sizes. In our future work, we will enhance our model to consider dynamic service policies.

Acknowledgements This research was partly supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) and a Grant-in-Aid for Young Scientists (A) (17H04706, 2017-2020) from Japan Society for the Promotion of Science (JSPS).

References

1. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), pp. 483–485. ACM, New York, NY, USA (1967)
2. Tallent, N.R., Mellor-Crummey, J.M.: Effective performance measurement and analysis of multithreaded applications. SIGPLAN Not. **44**(4), 229–240 (2009)
3. Raicu, I., Foster, I., Zhao, Y., Szalay, A., Little, P., Moretti, C.M., Chaudhary, A., Thain, D.: Towards data intensive many-task computing. In: Data Intensive Distributed Computing: Challenges and Solutions for Largescale Information Management, vol. 13, no. 3, pp. 28–73 (2012)
4. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M.: Workflows for e-Science: scientific workflows for grids. Springer Publishing Company, Incorporated (2014)
5. Pautasso, C., Alonso, G.: Parallel computing patterns for grid workflows. In: 2006 Workshop on Workflows in Support of Large-Scale Science, pp. 1–10 (2006)
6. de Oliveira, D., Ogasawara, E., Ocaa, K., Baio, F., Mattoso, M.: An adaptive parallel execution strategy for cloud-based scientific workflows. Concurrency Comput. Pract. Experience **24**(13), 1531–1550 (2012)
7. Ishida, T. (ed.): The Language Grid: Service-Oriented Collective Intelligence for Language Resource Interoperability. Springer Science & Business Media (2011)
8. Murakami, Y., Lin, D., Ishida, T.: Service-Oriented Architecture for Interoperability of Multilanguage Services, pp. 313–328. Springer, Berlin (2014)
9. Sun, X.H., Chen, Y.: Reevaluating Amdahl' s law in the multicore era. J. Parallel Distrib. Comput. **70**(2), 183–188 (2010)
10. Trang, M., Murakami, Y., Ishida, T.: Policy-aware optimization of parallel execution of composite services. IEEE Trans. Serv. Comput. **PP**(99), 109–113 (2017)
11. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semant. Sci. Serv. Agents World Wide Web **1**(3), 281–308 (2004)
12. Yu, Q., Bouguettaya, A.: Framework for web service query algebra and optimization. ACM Trans. Web **2**(1), 6:1–6:35 (2008)
13. Xuan, T.M., Murakami, Y., Lin, D., Ishida, T.: Integration of workflow and pipeline for language service composition. In: Chair, N.C.C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., Piperidis, S. (eds.) Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), pp. 3829–3836. European Language Resources Association (ELRA), Reykjavik, Iceland (2014)
14. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17), 3045–3054 (2004)
15. Ludscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system. Concurrency Computat. Pract. Experience **18**(10), 1039–1065 (2006)
16. Yu, J., Buyya, R., Ramamohanarao, K.: Workflow Scheduling Algorithms for Grid Computing, pp. 173–214. Springer, Berlin (2008)
17. Szabo, C., Sheng, Q.Z., Kroeger, T., Zhang, Y., Yu, J.: Science in the cloud: Allocation and execution of data-intensive scientific workflows. J. Grid Comput. **12**(2), 245–264 (2014)
18. Lin, D., Shi, C., Ishida, T.: Dynamic service selection based on context-aware QoS. In: 2012 IEEE Ninth International Conference on Services Computing, pp. 641–648 (2012)
19. Lin, D., Ishida, T., Murakami, Y., Tanaka, M.: Qos analysis for service composition by human and web services. IEICE Trans. Inf. Syst. **97**(4), 762–769 (2014)