2015 IEEE International Conference on Services Computing

Policy-Aware Optimization of Parallel Execution of Composite Services

Mai Xuan Trang
Department of Social Informatics
Kyoto University
Kyoto, Japan
trangmx@ai.soc.i.kyoto-u.ac.jp

Yohei Murakami
Design School
Kyoto University
Kyoto, Japan
yohei.murakami@design.kyoto-u.ac.jp

Toru Ishida
Department of Social Informatics
Kyoto University
Kyoto, Japan
ishida@i.kyoto-u.ac.jp

***Abstract*—Parallel execution and cloud technologies are the keys to speed-up service invocation when processing large-scale data. In SOA, service providers normally employ policies to limit parallel execution of the services based on arbitrary decisions. In order to attain optimal performance improvement, service users need to adapt to parallel execution policies of the services. A composite service is a combination of several atomic services provided by various providers. To use parallel execution for greater composite service efficiency we need to optimize the degree of parallelism (DOP) of the composite services by considering policies of all atomic services. We propose a model that embeds service policies into formulae to calculate composite service performance. From the calculation, we predict the optimal DOP for the composite service. Extensive experiments are conducted on real-world translation services. The results show that our proposed model has good prediction accuracy in identifying the optimal DOPs. Our model correctly predicts the optimal DOP in most cases.**

***Keywords*-Service Composition, Service Policy, Parallel Execution, Big Data.**

I. Introduction

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 large-scale data; this leads to 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.

In this paper we focus on strong scaling, a common parallel execution technique to improve performance of a task that involves large-scale datasets. The data is split into small independent portions that are executed in parallel by multiple task instances. This decreases the overall execution time of the task. Several existing studies on parallel execution in high-performance computing (HPC) have considered trade-offs among several criteria such as execution time, energy consumption, and resource usage to optimize parallel execution efficiency of applications [1], [2]. There are also some existing parallel performance models which introduce several factors that affect the strong scaling efficiency such as serial portions in the task as pointed out in Amdahl's law [3] or parallel overhead [4]. These models consider limitation of the program or computing resources from the point of view of task providers. Providers can control the limitation to gain a better parallel execution efficiency.

In SOA, however, from the perspective of service users, the service providers' policies about parallel execution are an important factor that affects 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 are also different. For instance, if a provider is rich in computing resources, he may readily allow large numbers of concurrent requests. Otherwise, he may limit this to a lower number. Service providers may have different policies to degrade performance of their services if the number of concurrent requests exceeds the limitation. Therefore, when invoking a service with parallel execution, users should adapt to the service policy in order to attain the optimal performance improvement.

Moreover, a composite service, a combination of several atomic services, is an important concept in SOA. If parallel execution is used for greater composite service efficiency, we need to optimize the degree of parallelism (DOP) of the composite service by considering the atomic service policies of all participating providers. To tackle this problem, in this paper, we propose 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:

- *Model parallel execution policies of atomic services*: We need a model to capture the policies of atomic services observed when using parallel execution. The model helps to easily embed service policies into the calculation of composite service performance.
- *Calculate composite service performance under parallel execution*: We define formulae to calculate performance of a composite service consisting of several control structures. From the calculation, we predict the

optimal DOP for composite services.

- *Validate the model*: We need to validate our model by evaluating the accuracy of our model when predicting the optimal DOP of composite services on the real-world translation services.

The remainder of the paper is organized as follows. Section II presents a motivating example. Section III describes parallel execution model of concrete and composite services. The prediction model is proposed in Section IV. We evaluate our model in Section V. We show some related work in Section VI. Finally, Section VII concludes the paper.

II. Motivating Example

Many service-oriented infrastructures have been proposed for sharing and combining web services. A typical example is the Language Grid [5]. Service providers can share their tools (e.g. translator, dictionary, etc.) as atomic web services on these infrastructures. Different providers may employ different parallel execution policies for their provided services. Users can combine different exiting atomic services to define new composite services that meet their requirements. For composite services that contain long-running tasks, using parallel execution can reduce the processing time.

To illustrate our approach, we show the parallel execution of a composite service in Figure 1. This simple workflow is a back-translation service combining two different translation services: Google translation service and Bing 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. Assume that the composite service is configured with $DOP = n$. Each translation service in the composite service is executed concurrently with n processes.

Now, let us consider a scenario where Google and Microsoft employ different parallel execution policies for their translation services. Google limits number of concurrent requests sent to its translation service from a registered user to 8. If more than 8 concurrent requests are sent to Google translation service the performance becomes worse. Bing limits its translation service to serve 14 concurrent requests for one user, if more than 14 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 conform to all atomic services' policies in order to attain the optimal performance for the composite service. Several questions to be asked here are

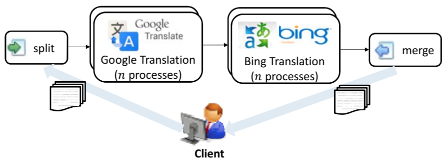


Figure 1: Parallel execution of a composite service

(1) How can we model parallel execution policies of web services? and (2) How can the performance of a composite service be calculated with regards to atomic services' policies? These issues are addressed in the next sections.

III. Parallel Execution Policy Model

A. Parallel Execution of Atomic Services

We use *Data Parallelism* as the parallel execution technique to speed-up service invocation. Suppose that a service is invoked using parallel execution with n concurrent processes. The input data is split into M partitions and n processes of the service are applied over n partitions in parallel. Execution time of the service depends on the number of concurrent processes, denoted by $f(n)$. Increasing n helps to decrease execution time of the service. Ideally, execution time decreases n times if we use n concurrent processes of the service s. However, the performance speed-up is limited by the non-parallel fraction of the task as pointed out in Amdahl's Law [3]. According to this law, all programming tasks include serial parts, which cannot be executed in parallel. Assume that the ratio of computing time for the serial parts to the whole task is F ($0 < F < 1$). The performance speedup of the task when executing n concurrent processes, $S(n)$, is

$$S(n) = \frac{1}{F + (1-F)/n}$$

When n is large, it converges to $1/F$, that is, the effect of parallel execution is limited. For example, if $F = 5\%$, the maximum possible speedup is 20, even if we use an infinite number of concurrent processes. As F goes to zero, $S(n)$ converges to n, which is an ideal situation.

Amdahl's Law, however, does not have parameters to consider the effect of other external factors to the parallel execution efficiency. In SOA, the parallel execution policy is an important factor that affects the performance speedup achievable with parallel execution. In some cases, if number of concurrent processes is larger than a certain number, performance of a service may fall. Figure 2 shows speed enhancements possible of an atomic service when using parallel execution under different conditions.

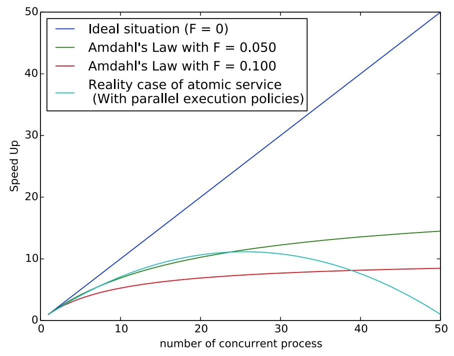
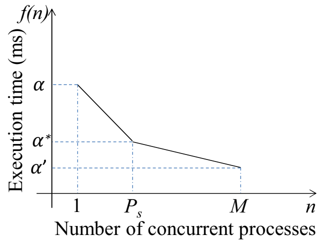
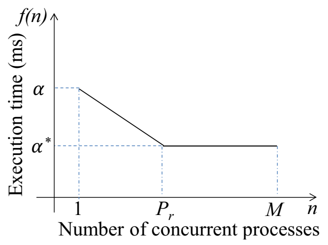


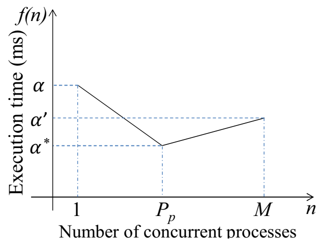
Figure 2: Parallel execution speed-up of an atomic service.



(a) Slow-down Policy



(b) Restriction Policy



(c) Penalty Policy

Figure 3: Performance Patterns for Parallel Execution Policies

B. Parallel Execution Policies of Atomic Service

Our assumption in this paper 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.

Definition (Parallel Execution Policy). *A 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 these parameters (α, $\alpha^\star$, α', P), each parameter is defined as following:*

- *Suppose that the service processes M data partitions using parallel execution. Execution time of the service depends on number of concurrent processes 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 needed to process M partitions with N ($P < N \leq M$) concurrent processes: $f(N) = \alpha'$.*

According to the values of these parameters we recognize three types of parallel execution policy:

Slow-down Policy. This policy throttles the performance increase when number of concurrent processes exceeds specified number (P_s). The performance pattern yielded by this policy is depicted in Figure 3a. Equation 1 gives the execution time of the service:

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

- When $1 \leq n < P_s$, service performance steadily increases with the number of concurrent processes.
- When $P_s \leq n \leq N$, service performance continues to improve but at a slower rate ($\frac{\alpha-\alpha^\star}{P_s-1} > \frac{\alpha^\star-\alpha'}{N-P_s}$).

Restriction Policy. Service providers limit the maximum number of concurrent requests that their services can serve. Service performance saturates at specified number (P_r). Figure 3b shows performance pattern of this policy. Equation 2 gives the execution time of the service:

$$f(n) = \begin{cases} \alpha - \frac{\alpha - \alpha^\star}{P_r - 1}(n-1), & \text{if } 1 \leq n < P_r \\ \alpha^\star, & \text{if } P_r \leq n \leq N \end{cases} \quad (2)$$

- When $1 \leq n < P_r$, service performance steadily increases with the number of concurrent processes.
- When $P_r \leq n \leq M$, execution time of the service remains the same ($\alpha' = \alpha^\star$).

Penalty Policy. With this policy, service providers specify a certain number of concurrent requests that yields good service performance. If the number of concurrent requests send to the services exceeds specified number (P_p), service performance is reduced. The performance pattern of this policy is shown in Figure 3c. Equation 3 calculates the execution time of the service:

$$f(n) = \begin{cases} \alpha - \frac{\alpha - \alpha^\star}{P_p - 1}(n-1), & \text{if } 1 \leq n < P_p \\ \alpha^\star + \frac{\alpha' - \alpha^\star}{N - P_p}(n - P_p), & \text{if } P_p \leq n \leq N \end{cases} \quad (3)$$

- When $1 \leq n < P_p$, service performance steadily increase with the number of concurrent processes.
- When $P_p \leq n \leq N$, service performance falls ($\alpha' > \alpha^\star$).

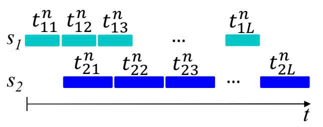
IV. Prediction Model

A. Parallel Execution of Composite Services

Two parallelism techniques, that are used to speed-up invocation of a composite service, are data parallelism and workflow pipeline execution.

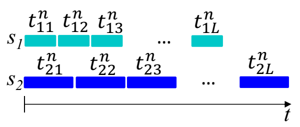
Data parallelism. When considering data-intensive service applications which process large amounts of data. Benefiting from the large number of resources available in a grid, a composite service can be instantiated as several computing tasks running on different hardware resources and processing different input data in parallel. Using this technique, the input data sets are split into independent portions, and several processes of the composite service are instantiated to process several portions in parallel.

Pipeline execution. 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.



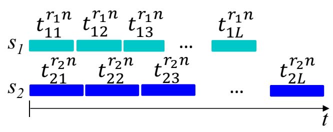
t_{11}^n t_{12}^n t_{13}^n ... t_{1L}^n
s_1
t_{21}^n t_{22}^n t_{23}^n ... t_{2L}^n
s_2
t

(a) Sequential Structure



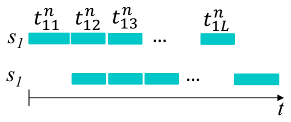
t_{11}^n t_{12}^n t_{13}^n ... t_{1L}^n
s_1
t_{21}^n t_{22}^n t_{23}^n ... t_{2L}^n
s_2
t

(b) Parallel Structure



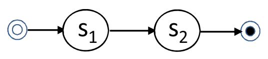
$t_{11}^{r_1n}$ $t_{12}^{r_1n}$ $t_{13}^{r_1n}$ ... $t_{1L}^{r_1n}$
s_1
$t_{21}^{r_2n}$ $t_{22}^{r_2n}$ $t_{23}^{r_2n}$ ... $t_{2L}^{r_2n}$
s_2
t

(c) Conditional Structure

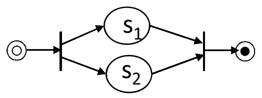


(d) Loop Structure

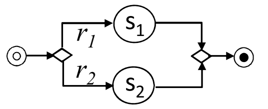
Figure 4: Processing time-line of different structures



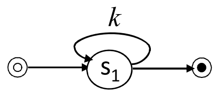
(a) The sequential structure



(b) The parallel structure



(c) The conditional structure



(d) The loop structure

Figure 5: Four Types of Composite Structures

A pooling technique is used such that when processing M partitions, n out of M partitions 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 Figure 4a, where $L = \lceil M/n \rceil$ is number of time-steps needed to send M partitions, each time n partitions are sent in parallel. t_{ij}^n is the time that n concurrent processes of service s_i take to finish processing n partitions.

B. Performance Prediction of Different Workflow Structures

We propose equations to predict performance of composite services in different structures. There are four basic composite structure: *Sequential*, *Parallel*, *Conditional* and *Loop*, see Figure 5 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 [6][7], and direct aggregation method with multiple QoSs for the service composition [8][9]. We adapt the formulae proposed in [6] to calculate execution time of composite services under parallel execution.

1) Sequential Structure: Consider a *Sequential* combination of two services s_1 and s_2 as shown in Figure 5a. s_1 and s_2 may have different performance policies when using parallel execution as specified in Section III-B.

Suppose that the composite service processes M data partitions using parallel execution as described above. DOP of the composite service is set to n, so that each atomic service is executed with n concurrent processes. Let (α_1, $\alpha_1^\star$, α_1', P_1) and (α_2, $\alpha_2^\star$, α_2', P_2) are the parallel execution policies of s_1 and s_2. Execution time of s_1 and s_2 to process M partitions, $f_1(n)$ and $f_2(n)$, can be predicted by using our policy model. Assume that $f_1(n) < f_2(n)$, processing time-line of the composite service is depicted in Figure 4a. We can easily see that execution time of the composite service ($f_c(n)$) is calculated as follows:

$$f_c(n) = f_2(n) + t_{11}^n \tag{4}$$

t_{11}^n is the time taken by service s_1 to finish processing first n partitions in parallel. Suppose that the execution time of s_1 to process each partition is approximate equal. We have:

$$t_{11}^n \approxeq \frac{f_1(n)}{\lceil M/n \rceil} \tag{5}$$

From Equation (4) and (5) we have:

$$f_c(n) \approxeq f_2(n) + \frac{f_1(n)}{\lceil M/n \rceil} \tag{6}$$

To generalize, consider a composite service consisting of a *Sequential* structure of k services ($s_1, s_2, ..., s_k$). Execution time of each service when processing M data partitions using parallel execution with n concurrent instances are $f_1(n)$, $f_2(n)$, ..., $f_k(n)$ respectively. Execution time of the composite service ($f_c(n)$) can be predicted as follows:

$$f_c(n) \approxeq \max_{i=1}^{k} f_i(n) + \frac{\sum\limits_{i=1}^{k} f_i(n) - \max\limits_{i=1}^{k} f_i(n)}{\lceil M/n \rceil} \tag{7}$$

2) Parallel Structure: Consider a *parallel* combination of two service s_1 and s_2 as shown in Figure 5b. In this case, the two services process data in parallel. The original dataset is split into M partitions, n of which are sent to the composite service concurrently. DOP of the composite service is set to n, so n processes of each atomic service are executed in parallel. Suppose that (α_1, $\alpha_1^\star$, α_1', P_1) and (α_2, $\alpha_2^\star$, α_2', P_2) are parallel execution policies of s_1 and s_2. Execution time of s_1 and s_2, $f_1(n)$ and $f_2(n)$, can be calculated by using the policy model. The processing time-line of the composite service is illustrated in Figure 4b. From this processing time-line, the execution time of the composite service ($f_c(n)$) is predicted as the maximum value of $f_1(n)$ and $f_2(n)$:

$$f_c(n) \approxeq \max(f_1(n), f_2(n)) \tag{8}$$

To generalize, consider a parallel combination of k services (s_1, s_2, ..., s_k). Suppose that parallel execution policy of service s_i is (α_i, $\alpha_i^\star$, α_i', P_i), execution time of service s_i with n concurrent processes ($f_i(n)$) can be calculated by using our policy model. The execution time of the composite service is predicted with the following generalized equation:

$$f_c(n) \approxeq \max_{i=1}^{k} f_i(n) \tag{9}$$

3) Conditional Structure: Figure 5c shows a *Conditional* combination of two service s_1 and s_2. Again, the input data is separated into M partitions. We configure the pool so that n partitions can be sent to the composite service concurrently. 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. This means that $r_1 n$ partitions are processed by s_1 in parallel, others $r_2 n$ partitions are processed by s_2 concurrently. We configure the parallel execution so that s_1 and s_2 are processed with $r_1 n$ and $r_2 n$ concurrent instances respectively. In total, s_1 processes $r_1 M$ partitions, and s_2 processes $r_2 M$ partitions. Execution time of s_1 and s_2, $f_1(r_1 n)$ and $f_2(r_2 n)$, are calculated by using the policy model.

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 Figure 4c; the number of time-steps, $L = \lceil M/n \rceil$. The execution time of the conditional structure $f_c(n)$ is predicted as follows:

$$f_c(n) \approxeq \begin{cases} f_1(1) + f_2(1), & \text{if } n = 1 \\ \max(f_1(r_1 n), f_2(r_2 n)), & \text{if } n > 1 \end{cases} \tag{10}$$

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

- $\sum\limits_{i=1}^{k} r_i = 1$.
- Execution time of service s_i when processing $r_i M$ partitions with $r_i n$ concurrent instances can be calculated with $f_i(r_i n)$ by applying the policy model.

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

$$f_c(n) \approxeq \begin{cases} \sum\limits_{1}^{k} f_i(1), & \text{if } n = 1 \\ \max\limits_{i=1}^{k} f_i(r_i n), & \text{if } n > 1 \end{cases} \tag{11}$$

4) Loop Structure: Consider the *Loop* structure of a service s_1 as shown in Figure 5d, where the iteration number of 2. In this case, this *Loop* structure can be converted to a *Sequential* structure of two services s_1. The composite service processes M partitions. n processes of service s_1 are executed to process n partitions in parallel. (α_1, $\alpha_1^\star$, α_1', P_1) is parallel execution policy of s_1, execution time of the service is $f_1(n)$ can be predicted by using the policy model. The processing time-line of the composite service is illustrated in Figure 4d; 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 $\lceil M/n \rceil$ 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's ΔT is duration from second time step to $\lceil M/n \rceil$ time step, ΔT is calculated as follows:

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

In 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 as bellow:

$$\begin{aligned} f_c(n) &\approxeq 2\frac{f_1(n)}{\lceil M/n \rceil} + \Delta T \\ &\approxeq 2\frac{f_1(n)}{\lceil M/n \rceil} + (\lceil M/n \rceil - 1)\frac{\max(f_1(n), f_1(2n))}{\lceil M/n \rceil} \end{aligned} \tag{12}$$

To generalize, consider a *Loop* structure where s_1 is executed k time. The structure can be converted to a sequence structure of k service s_1. Processing of the composite service follows pipeline-processing time-line. At time-step i ($1 \leq i < k$), in concurrent requests are sent to s. From time-step k to $\lceil M/n \rceil$, kn concurrent requests are sent to service s_1. The execution time of the composite service can be calculated by following equation:

$$f_c(n) \approxeq \frac{2\sum\limits_{j=1}^{k-1} \max\limits_{i=1}^{j} f(in)}{\lceil M/n \rceil} + (\lceil M/n \rceil - k + 1)\frac{\max\limits_{i=1}^{k} f(in)}{\lceil M/n \rceil}$$

C. Performance Prediction of Complex Composite Services

A composite service may contain 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. After the reduction, it is easily to calculate the execution time of the composite service by using the equation for the sequential structure. We adopt the reduction methodology proposed in [6] to calculated execution time of a complex workflow under parallel execution. Figure 6 shows an example of the reduction of a complex case. Where performance of s_{12}, s_{34} and s_{5loop} are calculated by applying the equations for parallel, conditional and loop structures respectively. Finally, the performance of the composite service is calculated by applying sequential combination for s_{12}, s_{34}, and s_{5loop}.

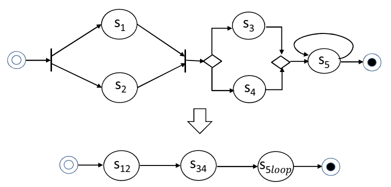


Figure 6: Reduction of a complex workflow

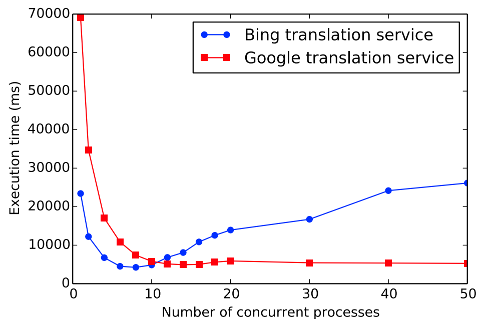


Figure 7: Policies of real translation services

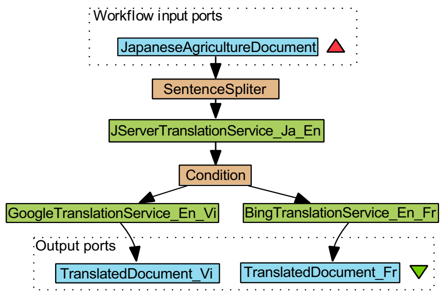

Workflow input ports
JapaneseAgricultureDocument
SentenceSpliter
JServerTranslationService_Ja_En
Condition
GoogleTranslationService_En_Vi
BingTranslationService_En_Fr
Output ports
TranslatedDocument_Vi
TranslatedDocument_Fr

Figure 8: A complex two-hop translation service

V. Experiments

We conducted experiments to evaluate our prediction model. Specifically, we attempted to answer the following questions: (1) How do the parallel execution policies categorize real world services?, and (2) How accurate is our prediction model, compared to the actual result?

A. Parallel Execution Policies of Atomic Services

Experiment Setup. We focused on analyzing real world translation services. We used the proposed integration engine of the Language Grid and UIMA [10] to configure and invoke services with parallel execution. In our experiments, a document with 200 sentences is translated from Japanese to English. The results demonstrate the impact of the policies of different services as follows.

Slow-down Policy and ***Restriction Policy***. Several services (e.g. Bing translation service and Baidu translation service) combine these two policies. Blue line in Figure 7 shows a slow-down and restriction pattern of Bing translation service with $P_{sbing} = 4$ and $P_{rbing} = 14$. When less than 4 concurrent processes are submitted to Bing, the performance of Bing translation service increases with the number of concurrent processes. With more than 4 concurrent processes the performance continues to improve but with a slower rate. If the number of concurrent processes is larger than 14, the performance of Bing translation service saturates.

Slow-down Policy and ***Penalty Policy***. We invoked and analyzed performance of Google translation service. Parallel requests are sent from one account, and Google translation service limits number of concurrent requests from one user account. Red line in Figure 7 shows a slow-down and penalty policies of Google translation service with $P_{sgoogle} = 4$ and $P_{pgoogle} = 8$. When there are fewer than 8 concurrent processes, the performance of Google increases with the number of concurrent processes, but the speed of the improvement decreases when the number of concurrent processes exceeds 4. If number of concurrent processes exceeds 8, performance of Google translation service becomes worse.

B. Performance Prediction of Composite Services

We consider here a realistic case when a Japanese agriculture expert wants to translate a Japanese document that contains two parts, one is information about rice and the other is information about fertilizer. The former is intended to transfer information to Vietnamese farmers, while the latter is for French fertilizer suppliers. We assume that there are no direct translation services from Japanese to Vietnamese and French. The Japanese expert does not want to translate the whole document into Vietnamese or French due to high cost of the translation. In order to do this task we create a composite service shown in Figure 8. This composite service is combination of three translation services with two structures, i.e. *Sequential* structure and *Conditional* structure. 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 and Bing employ slow-down and restriction policies with $P_{sjserver} = 4$, $P_{rjserver} = 10$ and $P_{sbing} = 4$, $P_{rbing} = 14$, Google employs slow-down and penalty policies with $P_{sgoogle} = 4$ and $P_{pgoogle} = 8$. Figure 9 shows a performance prediction when the composite service translates a document of 200 sentences (100 sentences about rice, 100 sentences about fertilizer). The green line shows the execution time predicted by our model, while the red line is the actual execution time. Our model predicts that the composite service attains best performance when number of concurrent processes is 28 which matches the real result.

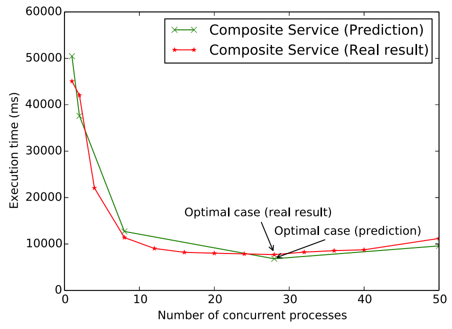


Figure 9: Performance prediction of the two-hop translation service

Table I: Evaluating the prediction accuracy

Input data	Optimal Degree of Parallelism (DOP)		
	Prediction	Actual result	
Document 1	24	24	MPE = 0 MAD = 1.07 TS = 0
Document 2	28	24	
Document 3	28	28	
Document 4	28	28	
Document 5	24	24	
Document 6	24	28	
Document 7	24	24	
Document 8	28	28	
Document 9	28	24	
Document 10	28	28	
Document 11	24	28	
Document 12	24	24	
Document 13	24	24	
Document 14	28	28	
Document 15	28	28	

With the optimal DOP, execution time of the composite service decreases by nearly 85%.

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 50 sentences to 200 sentences. We used the following measures to evaluate accuracy of the proposed model in predicting optimal DOP and optimal execution time.

- First we calculate the difference between the actual result and prediction value, this indicates errors of the prediction: Error (e) = Actual result - Prediction result

For n time periods where we have actual result and prediction values, we calculate:

- Mean Prediction Error (MPE): $MPE = \sum_{i=1}^{n} (e_i)/n$
- Mean Absolute Deviation (MAD): $MAD = \sum_{i=1}^{n} |e_i|/n$
- Tracking signal (TS): $TS = \sum_{i=1}^{n} e_i/MAD$

While MAD is a measure that indicates the absolute size of the errors, the MPE measure indicates the prediction model bias. The ideal value of MPE is 0, when $MPE < 0$ the prediction model tends to over-predict, when $MPE > 0$ the model tends to under-predict. The tracking signal (TS) checks whether there is some bias or not. Theoretically, if there is no bias, this sum should remain close to zero. The division by the MAD aims at measuring the distance from the mean in terms of MAD. A prediction model has good prediction accuracy if the value of TS close to zero. A control limit of TS for a good prediction model is typical in (-4, 4). Table I shows evaluation of the model when predicting DOP:

- $MPE = 0$ and $MAD = 1.07$. This means that the model yields good predictions; the average absolute error is 1.07 units.
- TS = 0. This means that in overall there is no bias of the prediction. We can assume that our proposed model well predicts the optimal DOP.

VI. Related Work

To improve performance of data-intensive services, many studies have been conducted. Many argue that the current technology of Web service is not suitable for data-intensive environment, and thus should be extended. Some studies introduce SOA-aware data flow to optimize the management of data-intensive services in cloud [11], [12]. Some propose the approaches to the data transmission among multiple data-intensive services [13], [14]. There are also some studies on modeling, verification and discovery of data-intensive Web service [15], [16]. However, all the studies mentioned above do not concern the parallel technology in service composition. Neither do they consider how the parallel execution policy of each atomic service affects the efficiency of the execution of the composite service.

Scientific workflows have emerged as a 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 [17], Kepler [18], Triana [19] or WINGS/Pegasus [20] to enable graphical design, execution and monitoring of scientific workflows. In the era of big data, workflow optimization has become an important issue. One of the common optimization targets is to improve the scientific workflow runtime performance [21]. 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 [22]. These solutions are usually implemented for a specific SWMS and aim at the acceleration of workflow through scheduling.

There are also some existing studies working on optimizing parallel execution for workflow with consideration about trade-offs between execution time and cost of service used. In [23], authors proposed resource allocation heuristics for scheduling parallel application on Grid environments to optimize the combined cost and time of the application and manage the trade-off between cost and execution time. A model, that optimizes QoS of composite service based on services in parallel, is proposed in [24]. This model considers the balance of throughput and cost in order to decide the DOP of the composite service where the whole process allows users to obtain the optimal benefit.

To the best of our knowledge, no existing works take parallel execution policies of task providers into account when identifying and configuring workflows with optimal DOP to attain the best performance as we do in this paper.

VII. Conclusion

In SOA, service users do not have control on computing resources, to optimize parallel execution of a composite service, users need to configure the composite service with a suitable degree of parallelism that conforms to service policies. This paper proposed a prediction model that considers the policies of atomic services to optimize parallel execution of the composite service. We model parallel execution policies of atomic services with three different categories: Slow-down policy, Restriction policy, and Penalty policy. This policy model is then embedded into the calculation of composite service performance. Based on this calculation, we estimate the optimal DOP for the composite service where it attains the best performance. Our model can deal with different composite structures as well as complex cases, where a composite service contains combination of structures, or an atomic service employs combination policies. We conducted experiments on real-world translation services to evaluate accuracy of our model. The results show that our model has good prediction accuracy with regards to identifying the optimal DOP for composite services.

In designing the proposed model, we assumed that parallel execution policy of a service is static. It does not change even the input data size is changed. However, in cloud environments, it seems highly likely that providers will dynamically change their policies in response to requests with different sizes. In our future work, we will enhance our model to consider dynamic service policies.

Acknowledgment

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] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[2] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos, "Prediction-based power-performance adaptation of multithreaded scientific codes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1396–1410, 2008.

[3] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the spring joint computer conference*. ACM, 1967.

[4] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson, *Effects of communication latency, overhead, and bandwidth in a cluster architecture*. ACM, 1997, vol. 25, no. 2.

[5] T. Ishida, *The Language Grid*. Springer, 2011.

[6] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281–308, 2004.

[7] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," in *Proceedings of the 8th IEEE International Enterprise distributed object computing conference (EDOC)*, 2004.

[8] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[9] Q. Yu and A. Bouguettaya, "Framework for web service query algebra and optimization," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 1, p. 6, 2008.

[10] M. X. Trang, Y. Murakami, D. Lin, and T. Ishida, "Integration of workflow and pipeline for language service composition," in *LREC*, 2014.

[11] D. Habich, W. Lehner, S. Richly, and U. Assmann, "Using cloud technologies to optimize data-intensive service applications," in *Proceedings of The 3rd International Conference on Cloud Computing (CLOUD)*. IEEE, 2010, pp. 19–26.

[12] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin, "Qos-driven service selection for multi-tenant saas," in *Proceedings of the 5th International Conference on Cloud computing (CLOUD)*. IEEE, 2012, pp. 566–573.

[13] S. Koulouzis, R. Cushing, K. A. Karasavvas, A. Belloum, and M. Bubak, "Enabling web services to consume and produce large datasets," *Internet Computing, IEEE*, vol. 16, no. 1, pp. 52–60, 2012.

[14] Q. Zagarese, G. Canfora, E. Zimeo, and F. Baude, "Enabling advanced loading strategies for data intensive web services," in *Proceedings of The 19th International Conference on Web Services (ICWS)*. IEEE, 2012, pp. 480–487.

[15] P. A. Bernstein, N. Dani, B. Khessib, R. Manne, and D. Shutt, "Data management issues in supporting large-scale web services." *IEEE Data Eng. Bull.*, vol. 29, no. 4, pp. 3–9, 2006.

[16] Y. Li and C. Lin, "Qos-aware service composition for workfow-based data-intensive applications," in *IEEE International Conference on Web Services (ICWS)*, 2011.

[17] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[18] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[19] I. Taylor, I. Wang, M. Shields, and S. Majithia, "Distributed computing with triana on the grid," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 9, pp. 1197–1214, 2005.

[20] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman, "Wings: Intelligent workflow-based design of computational experiments," *IEEE Intelligent Systems*, pp. 62–72, 2010.

[21] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[22] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for scheduling in distributed computing environments*. Springer, 2008, pp. 173–214.

[23] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade-off management for scheduling parallel applications on utility grids," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1344–1355, 2010.

[24] D. Yu, C. Li, and Y. Yin, "Optimizing web service composition for data-intensive applications," *International Journal of Database Theory and Application*, vol. 7, no. 2, 2014.