Partitioning BPEL program for decentralized execution based on Swarm Intelligence

Takwa Mohsni¹ and Zaki Brahmi²

¹ Faculty of Law, Management and Economics, Jendouba University, Tunisia.
E-mail: mohsni.takwa@gmail.com

² Higher Institute of Computer Sciences and Communication Techniques, Sousse University, Tunisia.
E-mail: zakibrahmi@gmail.com

Abstract. Web services have become the leading technology in business processes management. Business Process Execution Language (BPEL), the standard technology to compose services, assumes a single "orchestrator" that controls the execution flow and coordinates the interactions with selected services. Once its specification has been developed, the composite service may be orchestrated either in a centralized or in a decentralized fashion. Decentralized orchestration offers performance improvements in terms of increased throughput and scalability and lower response time. Decentralized orchestration can be done by partitioning a BPEL program into a set of BPEL sub-programs; each of them is committed to perform a part of the program. All the sub-programs BPEL must be coordinated to perform the overall program.

In this paper, we propose a novel approach to solve the composite web services partitioning problem using swarm intelligence basing on its capacity to build a simple artificial societies of insects who can collectively provide a complex task.

Keywords: Orchestration web services, BPEL, Swarm Intelligence

1 Introduction

Business Process Execution Language BPEL, the current industry standard frequently used to express Web Services (WS) orchestrations, defines how multiple service interactions between partners can be coordinated internally in order to achieve a business goal (orchestration) [1][2]. It is predominantly deployed in centralized servers, which implies that all interactions and intermediate data must go through one server. Therefore, the problems in relation to centralized management that have been encountered in the non-service environment, including poor performance, impaired reliability, limited scalability, and restricted flexibility [3]. Moreover, due to the distributed and dynamic natures of the web services environment, these problems are even aggravated. Thus, becoming one of the major obstacles for wide deployment of the web services technology, especially for applications where transfer of large amount of intermediate data is needed.

To address these problems, previous works, to overcome the single server bottlenecked, presented a decentralized execution mode for BPEL programs [4][5][6]. These approaches are based on different style of decentralization such as BPEL partitioning and message passing style. The most use technique is BPEL program partitioning, which consist of dividing BPEL program into a set of sub-programs. These sub-programs are executed on different servers and coordinate to execute the over program. Such decentralized execution of a composite web service is effective for improving response times and throughput as it can increase parallelism when executing "glue" code and can reduce message overheads [3].

In this paper, we propose a novel approach based on swarm intelligent system as a solution for BPEL partitioning problem. Fundamentally, we consider the problem of partitioning BPEL program as swarm intelligence problem. Therefore, we main motivation is that the swarm system can be made by the set of web services that have the properties of swarm system; parallel, associative, self-organization and interacting locally with each other and their environment to enable an interesting global behavior emerge. As well, swarm intelligence systems are used to solve many problems such as an optimization problem [7][8], data analysis [9], robotics systems [10] and image processing [11].

The remainder of the paper is organized as follows: Section 2 gives related previous work on service composition by partitioning of BPEL program for decentralized execution. In section 3, we formulate the research problem. After that, we present our new swarm algorithm for partitioning BPEL program in section 4 and evaluation results in section 5. Finally, we conclude this research in section 6.
2. Related Work

The BPEL program partitioning problem was first introduced by [4]. The idea is to represent a BPEL program as a Program Dependence Graph (PDG) and distinct two set of activities: mobile and fixed. Authors proposed two heuristic partitioning algorithms named, respectively, Merge-by-Def-Use (MDU) and Pooling-and-Greedy Merge (PGM). The aim of the MDU partitioning algorithm is to determine the best partitions at which each portable task must be executed in order to optimize the throughput of the decentralized program. However, due to large computation time to MDU algorithm, the authors are choose to apply the PGM heuristic to solve this problem - (1) - the greedy-merge is a refinement to minimize the data on the network and - (2) - the pooling heuristic tries to minimize the total number of messages.

Another approach is proposed in [5] based on genetic algorithm, is an improvement of work presented in [14]. In order to use the genetic algorithm, this runs on 4 steps; Genetic encoding - Initial population generation - Genetic operators and Fitness Function. The genetic encoding is used to represent this initial population by a genome. The genome is encoded by an array of integers of length equal to the number of mobile nodes in the BPEL program. Each gene represents a mobile node \( P_i \) and has value ranging from \( 0 \) to \( m \), where \( m \) is the number of fixed node in BPEL program. Authors are used two genetic operators (crossover operator and one mutation operator) and a fitness function to explore the search space. A local optimizer is applied to each feasible individual in the population, to explore the neighboring solutions of the individual for possible improvement.

3. Problem Statement

The BPEL program partitioning problem is to partition a BPEL program into a set of decentralized sub programs. This partitioning offers a better performance of the overall composite program. A description of the BPEL program partitioning problem was firstly introduced by [4] as described below: (i) The BPEL program is represented by a set of activities, which can be classified into fixed activities and mobile activities. Fixed activity can be "receive", "reply" or "invoke" activity for the reason that must executed at the particular server. Other activities are mobile activities, including activities "send", "assign", "if", and so on. The mobile activities can be executed at any server. (ii) The BPEL activities present certain links between them. These links are described by two types of dependencies; (1) Data dependencies ;(2) Control dependencies. The decentralized BPEL sub-process generated by the partitioning must preserve these dependencies. (iii) Each Fixed activity must be assigned to separate sub-process. Thus, a sub-process has exactly one fixed activity and zero or more mobile activity. By definition, a fixed activity cannot be merged with another fixed activity and mobile activities cannot form a partition without a fixed activity.

Formally, we can define the PPBPEL problem of partitioning BPEL process as follows: PPBPEL = \(< E, \emptyset, D, C, \text{SPBPEL, } \beta > \) where:

- \( E \) a set of \( f \) fixed activities: \( E = \{e_1, \ldots, e_f\} \)
- \( \emptyset \) a set of \( p \) mobile activities: \( \emptyset = \{01, \ldots, 0p\} \)
- \( D \) a set of Data dependencies between fixed activities and mobile activities: \( D = \{ < di, dj > | di \text{ uses the data generated by } dj, \text{ where } di, dj \in E \cap \emptyset \} \)
- \( C \) a set of Control dependencies between two activities BPEL: \( C = \{ < ci, cj, k > | \text{there exists a control dependency of type } k \text{ from } ci \text{ to } cj, \text{ where } ci, cj \in E \cup \emptyset \text{ and } k \in \{\text{true, false}\} \} \)
- \( \text{SPBPEL} \) a set of \( m \) sub-process \((E \cup \emptyset)\), generated by \( \beta \) function: \( \text{SPBPEL} = \{ \text{SP1, SP2, \ldots, SPm} \} \), where:

The set of sub-process \( \text{SPBPEL} \) must preserve the following requirements:

1. \( \text{SP}_i \cap \text{SP}_j = \emptyset, \text{if } i \neq j \) where \( 1 \leq i, j \leq f, \)
2. \( \text{SP}_1 \cup \text{SP}_2 \cup \ldots \cup \text{SP}_m = E \cup \emptyset, \) and
3. \( \text{SP}_i \subseteq \{ e_1 \text{ or } e_2 \text{ or } \ldots \text{ or } e_f \} \)

- \( \beta \) the method that implements the objective function \( \text{ObjF(\text{SPBPEL})} \) to give the throughput of the given BPEL program under partitioning (equation1)

\[
\text{ObjF} (\text{SPBPEL} ) = \min \{ T (S_1), \ldots, T (S_n) \}
\](1)
With; \( n \) is the number of communicating servers, each of which implements a portion of the overall service and \( T(S_i) \) is a function used to return the throughput delivered by each individual server \( S_i \) (equation 2), is calculated as follows:

\[
T (S_i) = \frac{\text{Capacity } (S_i)}{\text{Cost } (S_i)}
\]

Where; Capacity \( (S_i) \) is the raw compute capacity of server \( S_i \) and Cost \( (S_i) \) is the total amount of work that needs to be performed on server \( S_i \) for a single request.

4. Swarm Intelligence-based approach

A swarm is a large number of homogenous simple agents interacting locally among themselves and their environment, with no central to allow a global interesting behavior to emerge [12].

4.1. Motivation: Process recruitment in Swarms

In some species of ants, the transportation of large prey in a cooperative way involves two or more ants that cannot do the transportation alone. The main purpose of the cooperative transportation is to maximize the trade-off between the gained energy (food) and the energy spent to take the prey to the nest. Further, this process spends up the transportation [13]. The group involved in the cooperative transportation is formed by a process called recruitment. When a single scout ant discovers a prey, it firstly attempts to seize and transport it individually. After unsuccessful attempts, the recruitment process starts. Inspired by the recruitment process among ants colonies, we assume that a set of mobile activities \( \theta = \{ \theta_1, \theta_2, \ldots, \theta_p \} \) to be recruiter (performed) by a set of Fixed activities \( E = \{e_1, e_2, \ldots, e_f\} \) The question remains: How the set of mobile activities were dived between the set of fixed activities responding the local information of each activity?

According to the theoretical model of response threshold described by [13], we assume that each fixed activity \( e_i \in E \) has a single Tendency value to perform a mobile activity \( \theta_j \in \theta \). The Tendency reflects the capability of fixed activity \( e_i \in E \) to perform a mobile activity \( \theta_j \in \theta \). The fixed activities produce a stimulus for mobile activities, because of their internal response thresholds; have different tendencies to recruit the mobile activities.

Calculating Tendency: The tendency of fixed activity \( e_i \) to recruit a mobile activity \( \theta_j \) is determined by the stimulus and the internal response thresholds as shown in equation (3):

\[
T_{\theta_j} (S_{ei}) = \frac{S_{ei}^2}{S_{ei}^2 + \tau_{\theta_j}^2}
\]

In our proposed approach, the stimulus \( S_{ei} \) is determined by the capacity of server \( S_i \) that contains the fixed activity \( e_i \). The stimulus decreases proportionally to the server capacity that's reduced each time when the server performs a mobile activity. Thus server with large capacity for a mobile activity \( \theta_j \) has higher tendency to perform this mobile activity. The capacity of a server \( S_i \) is determined by its resources. Each server \( S_i \) has a limited number of resources denoted by \( S_i,.res \). The process of recruitment must respect all server resources limitation that's measured by the sum of the resource of the local fixed activity \( e_i \in S_i \) denoted by \( res(S_i,.e) \) and the resource needed to perform \( \theta_j \) denoted by \( res(S_i, \theta_j) \) (Equation 4):

\[
\forall i \sum_{\theta_j \in \theta} res(S_i, \theta_j) + res(S_i,.e) \leq S_i,.res
\]

The \( S_{ei}(t) \) local stimulus associated with fixed activity \( e_i \) is calculated at each time \( t \) as (equation 5):

\[
s_{ei} (t) = S_{ei}.res \cdot t - \sum_{\theta_j \in \theta} res (S_i^t, \theta_j) + res (S_i^t,.e_i^t)
\]

Therefore, each mobile activity \( \theta_j \in \theta \) has an internal response threshold \( \tau_{\theta_j} \), an internal variable, to respond to the stimulus \( S_{ei}(t) \) sent by the fixed activity \( e_i \) at time \( t \). The internal response threshold \( \tau_{\theta_j} \) decreases proportionally to the fixed activity stimulus. Thus, fixed activities with large stimulus have higher tendency to perform the current mobile activity. It is given by equation (6):

\[
\tau_{\theta_j} (t) = 1 - S_{\alpha} (t)
\]
Dependency Constraints: The BPEL program activities are characterized by the existence of control dependencies and data dependencies among them, as noted in section 3. An effective partitioning plan must guarantee (respect) this dependency constraints. Responding to this problem, the algorithm satisfy the constraint that each mobile activity \( \theta_j \in \theta \) must be recruited by a fixed activity \( e_i \in E \) possessing data dependency \( D(e_i, \theta_j) \) or control dependency \( C(e_i, \theta_j) \) with it, where it is considered as the most capable to perform the current mobile activity. The fixed activities that don't have any dependency with the mobile activity \( \theta_j \), its Tendency to perform the current mobile activity must be reduced to zero.

Optimal Tendency: The goal of partitioning BPEL process is to find optimal partitions that maximize the BPEL process reward. However, a partitioning BPEL program algorithm must guarantee that each mobile activity belongs to the partition maximizing its performance. In other words, the mobile activity will create a partition with the fixed activity that has the maximum Tendency value in the List \( L.Tendencies \). This list contains the Tendencies values of the set of fixed activities \( E \) as shown in Table1, where the length of Table is proportional to the number of fixed activities and each case of the table contains the Tendency value of current fixed activity, corresponds to the case, to perform the mobile activity \( \theta_j \).

\[
\begin{array}{c|c|c|c|c}
T(e_1,\theta_j) & T(e_2,\theta_j) & \ldots & T(e_f,\theta_j) \\
\end{array}
\]

Table 1: List \( L.Tendencies \) of Tendencies values

4.2. Algorithm Description

Our algorithm based Swarm intelligence is an approximate algorithm that uses to solve Partitioning BPEL Program Problem. The aim of algorithm is to allow mobile activity to decide individually with which fixed activity to be merges, in a simple and efficient way, minimizing communication costs and maximizing the throughput.

**Algorithm1** : Algorithm-based Swarm Intelligence for mobile activity \( \theta_j \)

**Input:** FA = \( \emptyset \) //set of Fixed Activities
MA = \( \emptyset \) //set of Mobile Activities
SPBPEL = \( \emptyset \) //Sub-Processes BPEL
Business Process Execution Language Process (BPEL Process)

**Output:** SPBPEL = \{SP1, SP2, \ldots, SPp\}, // p Sub-Process BPEL

Begin
1: (FA,MA)←Parsing(BPEL) //Generate FA and MA
2: for each fixed activity \( e_i \in E \) do
3: if (Data-DP(\( e_i,\theta_j \)) Or Control-DP(\( e_i, \theta_j \))) then
4: Tendency (\( e_i, \theta_j \)) ← \( \frac{S_2}{S_2 + \tau_{\theta_j}} \)
5: else Tendency (\( e_i, \theta_j \)) ← 0
6: end-if
7: (L.Tendencies).addElement(Tendency (\( e_i, \theta_j \)))
8:end-for
9: T.max←MaxTendencyValue(L.Tendencies)
10: picks fixed activity \( e_i \) which has \( T.max \)
11: SPBPEL.add(SPi(\( e_i,\theta_j \)))
End

Algorithm given above (Algorithm 1) present the details of our approach as described follows:
1. Identify the set of fixed activities \( E \) and the set of mobile activities \( \theta \).
2. When the mobile activity \( \theta_j \) received a stimulus for a fixed activity \( e_i \in E \), because its internal response, The Tendency(\( e_i, \theta_j \)) must be compute:
   a. If \( \theta_j \) has data dependency or control dependency with \( e_i \), The Tendency (\( e_i, \theta_j \)) is calculated using the equation (3)
   b. If \( \theta_j \) not has any dependency with \( e_i \), The Tendency (\( e_i, \theta_j \)) reduced to zero:

\[
T_{\tau_{\theta_j}(e_i)} = 0
\]

3. When the Tendencies values are calculated, the list \( L.Tendencies \) is created that contains the set of Tendencies values as indicated in section 4.1.
4. Seek the maximum Tendency value \( T.max \) of the List of Tendencies values as shown in equation (7) and merges with the fixed activity which has \( T.max \).
5. The steps 2, 3 and 4 are repeated for each mobile activity $\theta_j \in \theta$.

The procedures and functions used in this algorithm are:

- **Parsing:** Takes as input a BPEL process and return the set of fixed activities $FA$ and the set of mobile activities $MA$, which make up the BPEL process.
- **Data-DP:** Takes as input two BPEL activities and return a boolean value True, if there is data dependency between the two activities; False otherwise.
- **Control-DP:** Takes as input two BPEL activities and return a boolean value: True, if there is control dependency between the two activities; False otherwise.
- **MaxTendencyValue:** is the function that returns the maximum Tendency value in the List of Tendencies values $LT$.

5. Experimental results

To evaluate the results generated by our algorithm (SI), we compared it to the results generated by the genetic algorithm (GA)\cite{5}. We implemented the two algorithms with Java and executed on a desktop computer with Dual-core 2.30 GHz and 2 GB RAM.

The computation time and optimality of the algorithms depend on the size and the complexity of the BPEL process problem. The size of the problem is dependent on two parameters, the number of fixed activities in the BPEL process and the number of mobile activities in the BPEL process. The complexity of the problem largely depends on the levels of control dependencies constraints in the BPEL process. Thus, we generated two sets of test problems.

![Figure 1](image1.png)

Figure 1: Comparisons of the computation time of our algorithm-based Swarm Intelligence (SI) versus Genetic Algorithm (GA) for resultant partitioning of different BPEL program sizes.

The first set of test problems included 6 test problems with different number of fixed and mobile activities. The results are shown in Fig(1). The Time to compute the GA is very large compared to the algorithm-based Swarm Intelligence (SI). We can see that the computation time of the GA increases largely as the problem size increases. In counterpart, the computation time of our algorithm-based SI increases slowly as the problem size increases. This is because the computation time of the GA depends not only on the number of activities like our algorithm, but also on the corresponding partitioning topology of the solutions produced during evolving steps.

![Figure 2](image2.png)

Figure 2: Comparisons of the computation time (in milliseconds) of our algorithm-based Swarm Intelligence (SI) versus Genetic Algorithm (GA) for resultant partitioning of different complexities levels.
The second set of test problems included 4 test problems with different levels of control dependencies constraints. The compute time of two algorithms, increases as the level of complexities increases. The time to compute the GA is also large compared the time to compute our algorithm.

6. Conclusion

In this paper, we introduced our research efforts to address web services orchestration issues in decentralized and dynamic fashion. The approach introduced there- Swarm BPEL- deals the problem of partitioning BPEL program for decentralized execution based on the swarm intelligence. This algorithm solves the Partitioning BPEL process an approximated and distributed fashion.

References