Accelerating Web Service Workflow Execution via Intelligent Allocation of Services to Servers

Konstantinos Stamkopoulos, University of Ioannina, Greece

Evaggelia Pitoura, University of Ioannina, Greece

Panos Vassiliadis, University of Ioannina, Greece

Apostolos Zarras, University of Ioannina, Greece

ABSTRACT

The appropriate deployment of web service operations at the service provider site plays a critical role in the efficient provision of services to clients. In this paper, the authors assume that a service provider has several servers over which web service operations can be deployed. Given a workflow of web services and the topology of the servers, the most efficient mapping of operations to servers must then be discovered. Efficiency is measured in terms of two cost functions that concern the execution time of the workflow and the fairness of the load distribution among the servers. The authors study different topologies for the workflow structure and the server connectivity and propose a suite of greedy algorithms for each combination.

Keywords: *Composite Web Services, Service Providers, Service Provision, Web Services, Web Services Deployment*

INTRODUCTION

A web service is typically defined in the literature –for example, see Alonso, Casati, Kuno and Machiraju (2004)—as an interface that describes a collection of operations provided through the internet and accessed through standard XML messages. The appropriate deployment of web service operations at a service provider site plays a critical role in the efficient provision of services to clients. To effectively provide solutions to users' tasks, web services are *composed* in *workflows* (see Chen, Zhou, & Zhang, 2006) that combine intermediate service results towards achieving a more complex goal. Such workflows are typically specified in appropriate languages such as BPEL (see Andrews, et al., 2003).

DOI: 10.4018/jdm.2010100104

Motivating Example

Assume an electronic system that assigns rendezvous for patients that need to consult doctors. A workflow that arranges a meeting depending on the availability of a doctor is depicted in Figure 1. Once the meeting has been conducted, the system registers any prescribed medicines and communicates via operations with social security agencies to register the assignment of medicines to patients. The detailed description of these operations is not necessary for the purpose of the paper; still it is important to note that there are *operational services* that receive requests (in the form of XML messages) to which they react (by sending XML messages) and *decision operations* that regulate which operations are to be invoked depending on the state of the workflow.

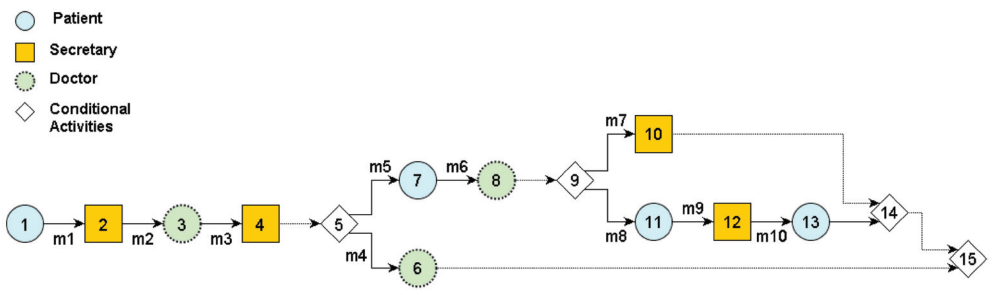
The whole workflow is supported by web service operations, deployed by the ministry of health and social security. The ministry has 5 servers that can host any of the 15 operations of the workflow and *the problem is to decide which of the possible 5^{15} configurations of the deployment of operations to servers (a) provides the fastest closing of each patient case and (b) loads each server in a fair way, so that whenever additional workflows are deployed, or a server fails, a reasonable load scale-up is still possible.*

Background and Problem Statement

In the problem we are dealing with in this paper, *we assume that a service provider has several servers over which web service operations can be deployed. Then, given a workflow and the topology of the servers, the most efficient deployment of the operations must be discovered.* Different topologies refer to the possibility of different networking infrastructure for the servers; this might include particularities relating to the characteristics of the machinery of the data center, its geographical distribution, etc. The workflows of the organization that we need to deploy might be of arbitrary complexity; ranging from simple linear workflows to graphs of large complexity.

Unfortunately, so far, related work has not equipped us with efforts towards the solution of the problem. There are several works in the area of design, composition and security of web services as well as works on the fine tuning of web service workflows. Concerning the latter, there are several works that deal with the regulation of the parameters of a previously obtained server configuration in order to achieve Quality of Service characteristics (see for example, Gillmann, Weikum and Wonner (2002)). However, none of the related research efforts covers the problem of the placement of web services to servers, once the operations and the topology of the servers are given. In other words, we provide the initial step for the administrator or engineer who wants to fine tune the architecture of his system: before fine-tuning for quality of service an initial, high quality allocation of operations to servers must be given; if such an allocation is unsatisfactory, then the approaches of the related work can be used.

Figure 1. Exemplary workflow



Contributions

In our approach, efficiency is measured in terms of two cost functions that concern the execution time of the workflow and the fairness of the load distribution among the servers. The latter means that all servers spend the same amount of time for processing the workflow. This results in a double optimization problem with antagonistic individual measures. We study different topologies for both the workflow and the network of servers and propose algorithms for each case. The contribution of this work lies in (a) the definition of a model which describes the problem, and (b) the proposed algorithms for its solution. Moreover, we have thoroughly experimented and assessed all the proposed algorithms.

This paper is organized as follows: First, we discuss related work. We then start with a formal definition of the problem and introduce algorithms for the deployment of web service operations at the appropriate servers. We also present experimental results, summarize our findings, and discuss issues of future research.

RELATED WORK

Service-Oriented computing has been a very active field of research over the past few years (for general reading around the context of Service Oriented Architectures (SOA) see Erickson & Siau, 2008; Papazoglou & van den Heuvel, 2007; Papazoglou, Traverso, Dustdar, & Leymann, 2007).

The approach proposed in this paper, focuses at the deployment stage of the Web services development lifecycle. Concerning the development of Web services, Yu, Liu, Bouguettaya, and Medjahed (2008) proposed a comparison framework for relevant approaches. The proposed framework consists of a number of key properties involved in facilitating the development of Web services and shall be used hereafter towards orienting the proposed approach with respect to these features and comparing it with other related approaches. Briefly, the key properties identified by Yu et al. (2008) are *interoperability*, *security and privacy*, *quality of Web services* and *management*.

Interoperability, refers to the ability of Web services to collaborate towards achieving a particular goal. As pointed out by Medjahed, Benatallah, Bouguettaya, Ngu, and Elmagarmid, (2003), the basic means for achieving this property are standards and ontologies (e.g. Ding, Fensel, & Klein, 2002), from a specification point-of-view, and mediation, from a technical point of view (e.g. Gravano & Papakonstantinou, 1998). Our approach does not specifically contribute in achieving interoperability. Nevertheless, we rely on the assumption that the composite services that serve as input to the proposed algorithms are able to interoperate as they conform to widely accepted standards such as WSDL, SOAP and BPEL.

As discussed by Geer (2003), security in the field of Web services management is mainly focused in managing the trade-off between high interoperability and low security risks. Privacy, on the other hand relates to careful reasoning about the data that can be released via Web services (Rezgui, Bouguettaya, & Eltoweissy, 2003). The algorithms proposed by our approach do not embed means for dealing with security and privacy. Despite their importance, these issues are orthogonal to the issue of scheduling the deployment of Web services, which is our main concern. Nevertheless, we assume that the use of the proposed algorithms shall be employed in well controlled environments consisting of secure servers. Moreover, we assume that the design and implementation of the composite services that serve as input to the proposed algorithms shall account for privacy issues.

The field of quality of Web services relates to the problem of selecting a Web service out of a set of available competing services with respect to particular quality characteristics (Vinoski, 2002). Quality management is very important for the interaction of the users with web services. As Burstein et al. (2005) mention: "QoS metrics can affect how services are advertised, can be the topic of negotiation processes, and must be monitored during enactment; thus, when clients' procedures or workflows involve multiple services, the underlying discovery, coordination, and execution systems must be able to monitor QoS measures and control the services accordingly". Various quality characteristics of interest that can be used to characterize Web services can be found in the literature (e.g., Maximilien & Singh, 2004; Conti, Kumar, Das & Shirazi, 2002; Zeng, Benatallah, Ngu, Dumas, Kalagnanam, & Chang, 2004) and a very useful taxonomy of them has been proposed by Yu et al. (2008). The proposed taxonomy distinguishes between runtime quality characteristics such as execution time, availability, reliability, integrity and business quality characteristics such as financial costs, reputation and conformance to standards. In our approach we consider the overall quality of a composite service that should be deployed over a set of available servers. The quality characteristic of interest to us is the composite service execution time. Our approach is complimentary to approaches related to service selection out of a set of competing services. In particular, the proposed algorithms for scheduling the deployment of composite Web services are employed right after the selection of the constituent services that are going to be used in a composite service. The quality characteristics of the constituent services as well as the quality characteristics of the underlying available infrastructure (i.e. servers and network) are the main input parameters of the cost model used by the proposed algorithms.

Our approach is most closely related to the management of Web services. Nevertheless, as Yu et al. (2008) discuss, this issue is also quite broad and encompasses many different dimensions including control management, change management and optimization.

Control management mainly refers to the coordination of Web services towards providing certain dependability guarantees such as the ones achieved through atomicity, isolation and other transactional properties (for a discussion of the related issues and protocols see Papazoglou, 2003; Papazoglou & Kratz, 2007). This issue is orthogonal to our approach that deals with the deployment of composite Web services. We assume that any required coordination logic is already embedded in the Web services workflows that serve as input to the proposed algorithms.

Change management deals with the maintenance and configuration management of Web service workflows. Related approaches deal with changes in Web service workflows triggered either by evolving business requirements or by the evolving quality characteristics of the participating services.

Our work falls in the category of optimization where the general objective is to tune a Web service composition towards achieving a number of desired quality characteristics. Into this context, Cardoso, Sheth, Miller, Arnold, and Kochut (2004) provide a model that characterizes the quality of a composite service based on response time, cost and reliability. The workflows considered include AND and OR decision nodes. The authors discuss METEOR, a system that traces the behavior of the workflow over time and warns users whenever the QoS dangerously reaches the thresholds originally set by the users. Gillmann, Weikum, and Wonner (2002) present tuning techniques for a workflow management system. The goal of the paper is the optimal tuning of the parameters of an environment where composite workflows are to be executed over a network of servers, in a way that quality of service concerning response time, availability and throughput is guaranteed. There is a huge number of configurable parameters, several architectural options for the involved servers (workflow servers, application servers, communication servers), and most importantly, service replicas. A Markov model is used for the determination of the quality of a workflow and a heuristic algorithm for the overall tuning of the system. Zeng et al. (2004) propose a method that, given a desired quality of service for a composite web service, the most appropriate elementary web services are chosen out of a set of candidate services with similar functionality. Appropriateness is decided on the grounds of execution cost, duration, reliability, availability and reputation. Finally, Salellariou and Zhao (2004) propose a method for the reconfiguration of a system whenever the observed quality of service is not satisfactory. A scheduling algorithm involving the starting and ending timepoints for a workflow is employed. The paper proposes the reallocation of tasks through this scheduling algorithm only whenever changes in the monitored system measures are significant. All these works investigate the problem of dynamically tuning workflows to achieve desired quality characteristics; still none of them deals with the deployment stage of the composite services lifecycle, which must take place beforehand. Composite service deployment is taken for granted, with extensions involving service replicas by Gillmann et al. (2002) or communities of similar operations by Zeng et al. (2003). Hence, our approach is complementary to the aforementioned approaches. More specifically, it does not aim at providing specific quality guarantees during the execution of composite services. On the contrary, it aims at providing a starting point to such approaches with tuneable configurations concerning the load characteristics of deployed services.

A final point in the related literature concerns approaches outside the context of service-oriented computing from which we were generally inspired such as replication, workflow management, and load balancing. In particular, Leff, Wolf, and Yu (1993) and Laoutaris, Telelis, Zissimopoulos, and Stavrakakis (2005) deal with the problem of object replication and provide interesting insights on the dimensions of the problem and the gain functions. Constantinescu, Binder, and Faltings (2005) and Srivastava, Widom, Mhnagala, and Motwani (2005) assume the continuous execution of a workflow: the former deals with the deployment of triggers to allow for the efficient execution of the workflow, whereas the second deals with the order of activity execution to achieve the optimal throughput. Concerning load balancing to ensuring quality properties for clients in the context of the web where unpredicted loads can occur, see Cherkasova and Peter Phaal (2002) as well as Cherkasova and Gupta (2004) for interesting facts and scheduling tactics. Moreover, large transaction processing systems distribute transaction processing to a number of servers, in order to increase the availability and efficiency of the overall system. Transaction Processing monitors (TP-monitors) regulate the assignment of requests and load balancing is one of the several criteria they employ. The main techniques used involve simple algorithms (since the employed algorithm must be simple, fast and lightweight) such as round-robin or randomized methods. A more elaborate technique is a workload-aware method, where the TP-monitor tracks the individual load of each server and assigns a new transaction to the server with the smallest load. An excellent source of reference for the topic is Lewis, Bernstein, and Kifer (2001).

PROBLEM FORMULATION

In this section, we formally define the problem under consideration. The objective is to provide algorithms that take as input a workflow of web service operations along with a topology of servers and compute an appropriate mapping of operations to servers. In the rest of our deliberations, we will employ the terminology of WSDL. We will also use the terms composite web service, *orchestration,* and *workflow* of web service operations interchangeably.

Formal Definition of the Problem

Assume a finite set of web service operations $\boldsymbol{O}= \{O_1, O_2, \ldots, O_M\}$ and a finite set of servers $\boldsymbol{S}= \{S_1, S_2, \ldots, S_N\}$. The term "operation" refers to WSDL operations (i.e., modules that may receive an input XML message and produce a result in the form of an output XML message). A transition (o_p, o_n) is a message sent by the web service operation o_p to the operation o_n, i.e., o_p invokes operation o_n through the submission of an XML message. We call operations o_p and o_n *neighboring operations*. A workflow is a directed graph of operations $\boldsymbol{W}(O, E)$, where $E=\{(o_p, o_n) \mid o_p, o_n \in O, \exists$ a transition from o_p to $o_n\}$. Intuitively, a workflow is a graph, with operations being the nodes of the graph and XML messages being modeled as the edges of the graph. A network of servers is an undirected graph $\boldsymbol{N}(S, L)$, where $L=\{(s_i, s_j) \mid s_i, s_j \in S, \exists$ connection between servers s_i and $s_j\}$. The deployment of an operation o to a server s is denoted by $o \rightarrow s$.

The operations of $\boldsymbol{O}$ can be distinguished into decision and operational ones. This follows the classification proposed by Leymann and Roller (2004), where a workflow comprises a control flow and a data flow subgraph. The operational nodes are the ones performing specific tasks for the workflow, whereas the decision nodes control the flow of execution. Following the fundamental distinction of Leymann and Roller (2004) for control nodes to *forks* (control nodes with multiple outputs, acting as routers for the execution flow) and *joins* (control nodes acting as rendezvous points that synchronize multiple parallel execution flows), we consider three types of decision operations/nodes, namely *AND, OR,* and *XOR*, as forks. We also assume three complementary types, denoted */AND*, */OR* and */XOR* respectively, to allow the definition of *well-formed workflows*. A workflow is well-formed if for every decision node *a*, there exists a complement node */a*, and all paths stemming from *a* also pass from */a*. Intuitively, decision nodes and their compliments act as parentheses. The reasons for this requirement are hidden in the semantics of the graph. Assuming a decision node, the semantics are as follows: (a) *AND* nodes involve the execution of all their outgoing paths with a rendezvous barrier at */AND*, (b) *OR* nodes do the same, but it suffices that one of the paths successfully reaches */OR* and (c) *XOR* nodes involve a probabilistically weighted pick of a path to be executed. In BPEL, AND nodes may correspond to plain flow activities, OR nodes may correspond to flow activities with conditional attributes and XOR nodes may correspond to switch or pick activities.

Assume a cost model $\boldsymbol{Cost}(\boldsymbol{W})$ that computes the cost of successfully completing the workflow $\boldsymbol{W}$. More details on the alternative costs that can be used are provided in the sequel. In the broadest possible variant of the problem, we can also assume a set of user constraints $\boldsymbol{C}$, concerning for example an upper bound on the completion time of a workflow or on the distribution of load among the servers.

The desideratum is a mapping of the operations $\boldsymbol{O}$ *of a workflow* $\boldsymbol{W}$ *to the set of servers* $\boldsymbol{S}$*, such that the operational cost is minimized (and the constraints* $\boldsymbol{C}$ *are met).* Formally, this optimum assignment of operations to servers is modeled as a finite set $\boldsymbol{M}= \{r_1, r_2, \ldots, r_M \mid \forall\ i=1,2,\ldots,M: r_i$ a rule of the form $o \rightarrow s$, $o \in O$ and $s \in S\}$ with the minimal $\boldsymbol{Cost}(\boldsymbol{W})$ that respects $\boldsymbol{C}$. Obviously, more than one mapping can be derived; we are interested in the one with the lowest possible cost. Depending on the algorithm employed this can be the overall optimal value (e.g., in the case of an exhaustive algorithm), or a local optimum (e.g., in the case of a greedy algorithm).

PROPOSED ALGORITHMS

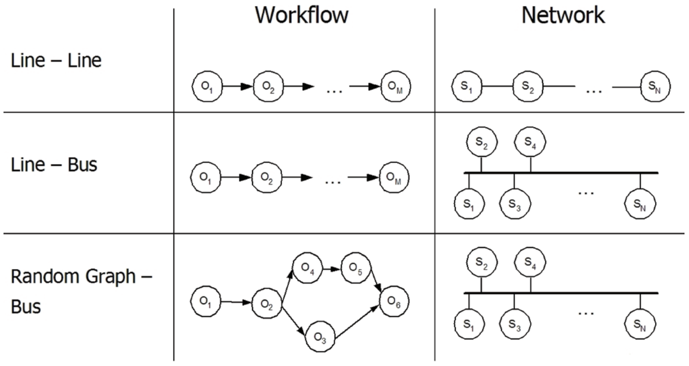
In this section, we present our proposed algorithms for determining an appropriate deployment of web service operations to servers.

We have experimented with different types of workflow and server topologies. We have considered random graph topologies as well as the special, simple case of linear workflows. The latter, being the most simple case, has served both the purpose of providing initial foresights for our experimental configuration and as an intuitive aid in the explanation of more complex cases. The network of servers forms either a linear topology (mainly for initial experimental reasons) or a bus topology. In Figure 2, we depict the combinations that were eventually considered as valid cases. In all our deliberations, we assume *N* servers and *M* operations.

Exhaustive Algorithm, Metrics and Notation

The exhaustive algorithm considers all possible mappings and outputs the one having the minimum cost. Due to the exponential search space of the exhaustive algorithm (for *N* servers

Figure 2. Examined configurations



and *M* operations, we have N^M configurations), we proceed with a set of heuristic solutions.

Regarding cost, we focus mainly on two cost metrics: *execution time* of the workflow and *load distribution*. Concerning the execution time of the workflow, the obvious desideratum is its minimization. Concerning the fairness of the distribution of load to servers, we want to guide our algorithms to fair solutions where the amount of work (i.e., the sum of computational cycles due to the assigned operations) is proportional to the computational power of each server. Details on the two metrics are given in Table 1. Unless otherwise stated, in the sequel, we will assume an equally weighted sum of the execution time and load distribution as our cost model. To use the same units, we assess fairness in the form of a time penalty that measures the deviation of the load of each server from the average load (which is the average time needed for a server to complete its workload). In a fair situation, all servers dedicate to the workflow the same amount of time. This is particularly important since an unbalanced network of servers has to deal with (a) possible bottlenecks due to some overloaded server in peak time and (b) difficulties in managing any other tasks, such as operation migration in cases of failures.

Clearly, the two metrics are antagonistic to each other. Take the case of a linear workflow (where each operation waits its preceding one to complete before it starts) where all operations are assigned to the most powerful server. Then, although the completion time is optimized (since no server communication costs are involved), the fairness of load distribution is destroyed. Inverse situations can also be encountered.

We have experimented with the exhaustive algorithm in small configurations to identify the properties that characterize the solutions that are close to the optimal one. These properties can be summarized as follows:

1. *Analogy between load and computational power of a server.* This clearly affects the fairness of load distribution.
2. *Minimization of the size of messages exchanged between servers.* The desideratum here is to distribute the operations to servers in such a way that neighboring operations are preferably assigned to the same server. By doing so, the fraction of messages sent over each communication line is expected to be reduced. At the same time, there is an antagonistic concern of not overloading anyone server too much; in fact, the desideratum is to preserve the aforementioned analogy between load and computational power of a server. Similarly to the minimization of the size of messages among servers, the *minimization of the*

Table 1. Notation and cost formulae

Symbol	Description
C(op)	The cycles necessary for operation *op* to complete
P(s)	Computational power of server *s* (Hz)
Server(op)	The server where operation *op* is deployed
$T_{prop}(s_i, s_j)$	Propagation time of the link between servers s_i and s_j.
$Path(s_i, s_j)$	The path followed by a message from s_i to server s_j.
$T_{trans}(op_i, op_j)$	Transmittance time needed for the communication of operations op_i and op_j. $T_{trans}(op_i, op_j) = \sum_a \frac{\mathrm{MsgSize(opi,opj)}}{\mathrm{Line\_Speed}(s_a, s_b)}$, $(s_a,s_b) \in \mathrm{Path(Server}(op_i), \mathrm{Server}(op_j))$
$T_{proc}(op)$	Processing time of a deployed operation *op*. $T_{proc}(op) = \frac{\mathrm{C(op)}}{\mathrm{P(Server\ (op))}}$
$MsgSize(op_i, op_j)$	Message size sent from operation op_i to operation op_j, assuming $(op_i, op_j) \in E$.
$Line\_Speed(s_i, s_j)$	Line speed (bps) between servers s_i and s_j.
Load(s)	Total load of server *s*, as the sum of the processing time of operations deployed to it. $Load(s) = \sum_j \mathrm{T_{proc}(O_j)}$
$T_{comm}(op_i, op_j)$	Assuming $(op_i, op_j) \in E$, the communication time between operations op_i and op_j, $T_{comm}(op_i, op_j) = \sum_a \mathrm{T_{prop}}(s_a, s_b) + \mathrm{T_{trans}}(op_i, op_j)$, $(s_a,s_b) \in \mathrm{Path(Server}(op_i), \mathrm{Server}(op_j))$
Time_Penalty	A translation of "fairness" to the time that a server needs to conclude its work, as opposed to the avg. such time among all servers $Time\_Penalty = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\lvert \mathrm{Load}(s_i) - \mathrm{Load}(s_j) \rvert}{(1/2) \times N \times (N-1)}$
$T_{execute}$	Execution time of workflow *W*. $T_{execute} = \sum_{j=1}^{M} \mathrm{T_{proc}(O_j)} + \mathrm{T_{comm}^{(total)}}$

number of messages exchanged between servers is also desirable.

Algorithms for a Line-Line Configuration

The case where both the workflow and the server topology are lines is the simplest possible one. Still, it is briefly mentioned here because of the simple observations and heuristics that can be applied to it.

The *Line-Line* algorithm receives a workflow of web service operations ***W(O, E)***, and a server configuration ***N(S, L)*** as its input. The algorithm operates in two discrete phases. In the first phase, the algorithm tries to produce a load distribution as fair as possible, while attempting to minimize the number of exchanged messages. In the second phase, the algorithm tries to move operations to neighboring servers to avoid sending large messages over low capacity links. For *N* servers and *M* operations, the complexity of the first phase is O(*M*) and the complexity of the second one is O(*N*). The algorithm is depicted in Figure 4.

First, the algorithm computes the ideal load per server (Line 7). The ideal load is computed as the fraction of the total workload that corresponds to the server given the percentage of the computational power the server can contribute to the overall computational power. Then, it starts assigning the operations of ***W*** to the servers of ***N*** starting from the first operation/server on the left. When a server comes as close

Figure 3. Critical bridge

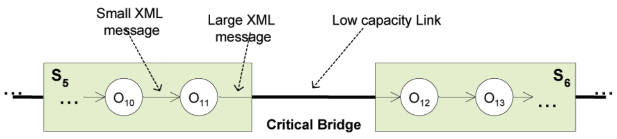


Figure 4. Algorithm Line-Line

Input: A workflow **W**(O, E) (in line topology), where O = (O_1, O_2, …, O_M) a set of operations, E = {(O_i, O_{i+1}) | ∀ i = 1,…, M-1} the set of transitions among operations and **N**(S, L) a server network (in line topology), where S = {S_1, S_2, …, S_N} a set of servers and L = {(S_i, S_{i+1}) | ∀ i = 1, …, N-1} the network connections among servers (M>N).
Output: A mapping **M** of O to S.

```
Begin
  1   Operations_List = ($O_1$, $O_2$, …, $O_M$)
  2   Servers_List = ($S_1$, $S_2$, …, $S_N$)
  3   Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$
  4   Sum_Capacity = $\sum_{j=1}^{N} P(S_j)$
  5   M = ∅
  6   s = Servers_List.pop
  7   Ideal_Cycles = Sum_Cycles × P(s) / Sum_Capacity
  8   Current_Cycles = 0
  9   while Operations_List is not empty do
 10       o = Operation_List.pop
 11       if SizeOf(Operation_List) ≥ SizeOf(Servers_List) then
 12           if Current_Cycles + C(o) < Ideal_Cycles + 0.2×Ideal_Cycles or
 13             Current_Cycles = 0  or s is $S_N$ then
 14               Current_Cycles = Current_Cycles + C(o)
 15           else
 16               s = Servers_List.pop
 17               Ideal_Cycles = Sum_Cycles × P(s) / Sum_Capacity
 18               Current_Cycles = C(o)
 19           end if
 20           M = M ∪ {o→s}
 21       else
 22           s = Servers_List.pop
 23           M = M ∪ {o→s}
 24           while Operation_List is not empty do
 25               o = Operation_List.pop
 26               s = Servers_List.pop
 27               M = M ∪ {o→s}
 28           end while
 29       end if
 30   end while
 31   Fix_Bad_Bridges (W, N, M);
 32   return M
End
```

The ideal load per server is computed

Phase A: original mapping of operations to servers. ***While*** *unassigned operations exist and operations are more than the servers, pick next*

If *server far from ideal load, assign operation*

else*, pick next*

If less operations than servers, simple assignment

Phase B: if critical bridges exist, reallocate operations as possible to its ideal load (test in lines 11-12), the algorithm considers the next server. The algorithm makes also some provision for the case when the operations are less than the number of servers (Lines 20-28). The first phase ends, when all operations have been allocated. The second phase of the *Line-Line* algorithm is based on the idea of a *critical bridge*, which is a link between two servers of the network with (a) a small capacity and a large message load (in bytes), plus (b) a small-sized message concerning an adjacent operation. Figure 3 depicts such a case. Whenever a critical bridge is detected, the algorithm deploys the receiver of the large message to the server of the sender of the message (or vice-versa). This is achieved through the call of function *Fix Bad Bridges* (Line 30) which is detailed in Figure 5.

The algorithm *Line-Line* (Figure 4) comes with variants. The first variation simply avoids the second phase of the algorithm. A second variation considers the assignment of operations to servers both from left-to-right and from right-to-left and maintains the better of the two. The combination of these variants produces four alternatives for the computation of the best configuration with the obvious complexities.

Algorithms for a Line – Bus Configuration

In this subsection, we move to a more realistic case, where all servers are connected to each other through a network bus. The workflow is still a simple line. We can produce several greedy variants of a simple algorithm, which are subsequently listed.

Fair Load. The simplest of all the involved variants is tuned to obtain the best possible load distribution. *Fair Load* (Figure 7) starts by computing the ideal number of cycles that should be assigned to a server based on its capacity. Then, it sorts servers by their capacity and operations by their execution cost. The algorithm processes the sorted list of operations, each time, assigning the next heaviest operation to the most appropriate server. The most appropriate server is the server that needs the most cycles to complete its ideal number of cycles, at the time of the assignment. *Fair Load* is a variant of the worst-fit algorithm for the bin packing problem.

Figure 5. Function fix bad bridges

Function Fix Bad Bridges
Input: The workflow **W**, the network **N** of algorithm Line-Line, and a starting mapping **M**.
Output: A possibly updated mapping **M**. The function checks whether critical bridges exist and moves a deployed operation to the next/previous server (left or right depending on where the expensive message is)

```
Begin
  1   Sort the speeds of the lines of N at ongoing order in list $L_1$.
  2   Sort the sizes of messages that are sending between servers (from last operation at server i to first
      operation at server i+1) at ongoing order in list $L_2$.
  3   for i=1 to N-1 do
  4       if Is_Critical_Bridge(S, M, i, $L_1$, $L_2$, shift_direction) = True then
  5           if shift_direction = right then
  6               M = M + {LastOperationAt($S_i$)→$S_{i+1}$} – {LastOperationAt($S_i$)→$S_i$}
  7           else
  8               M = M + {FirstOperationAt($S_{i+1}$)→$S_i$} – {FirstOperationAt($S_{i+1}$)→$S_{i+1}$}
  9           end if
  10      end if
  11  end for
End
```

Figure 6. Function is critical bridge

Function Is Critical Bridge

Input: The workflow **W**, the network **N** of algorithm Line-Line, a mapping **M**, an integer "i" representing the bridge to be checked and the sorted lists L_1 and L_2 with the network connections and the message sizes for each bridge, respectively.

Output: the function returns True if the bridge among S_i και S_{i+1} is critical; otherwise it returns False. The shift_direction variable stores the direction of the move that must take place, in the former case.

```
Begin
    if Line_Speed($S_i$, $S_{i+1}$) $\leq$ Top20 of $L_1$ and
    MsgSize(LastOperationAt($S_i$), FirstOperationAt($S_{i+1}$)) $\geq$ Bottom20 of $L_2$ then
        if MsgSize(PenultOperationAt($S_i$), LastOperationAt($S_i$)) $\leq$ Top20 of $L_2$ then
            shift_direction = right
            return True
        end if
        if MsgSize(FirstOperationAt($S_{i+1}$), SecondOperationAt($S_{i+1}$)) $\leq$ Top20 of $L_2$ then
            shift_direction = left
            return True
        end if
    end if
    return False
End
```

Fair Load – Tie Resolver for Cycles. *Fair Load* does not take execution time into consideration. A simple extension involves resolving any ties that may come up during the selection process among operations with the same number of cycles. The algorithm *Fair Load – Tie Resolver for Cycles*, (or, *FLTR*$_1$ for brevity), depicted in Figure 8, operates as *Fair Load* with respect to its basic principle. The difference lies in the fact that whenever we need one among a number of operations with the same cost, we no longer pick one at random. Instead, we employ a gain function, *Gain_Of_Operation_At_Server* that returns the communication savings (i.e., how many bytes will not be put on the bus), if the next operation is deployed to a certain server (Figure 9). The best such assignment among all candidate operations and servers is picked. The algorithm uses two lists, *Servers_List* και *Operations_List*, with pointers to the respective sets. The algorithm also needs to initialize the mapping ***M*** to a random configuration, or else, the first calls of function *Gain_Of_Operation_At_Server* would not return any gain at all.

Fair Load – Tie Resolver for Cycles and Servers. The algorithm *Fair Load – Tie Resolver for Cycles* can be extended to also handle ties among servers. The algorithm *Fair Load – Tie Resolver for Cycles and Servers*, (or, *FLTR*$_2$ for brevity), depicted in Figure 10, simply customizes appropriately the previous gain function to also consider the case in which there is a tie among the servers to be chosen next, with respect to their distance from their ideal load.

Summarizing, both *Tie Resolver* algorithms handle practically the same configurations with *Fair Load*, with the only difference that special attention is paid to situations where ties occur, with the overall goal to reduce the communication cost. However, it is still possible to send large messages over the network. The following extension tries to alleviate this problem.

Fair Load–Merge Messages' Ends. Algorithm *Fair Load–Merge Messages' Ends* (or, *FLMME* for brevity), depicted in Figure 11, extends *FLTR*$_2$ by adding an extra test during the deployment decision. If the assignment of an operation to a server

Figure 7. Algorithm fair load

```
Input: A workflow W(O, E) (in line topology), where O = (O_1, O_2, …, O_M) a set of operations, E = {(O_i, O_{i+1}) | ∀ i =
1,…, M-1} the set of transitions among operations and N(S, L) a server network (in bus topology), where S = {S_1, S_2,
…, S_N} a set of servers and L = {(S_i, S_{i+1}) | ∀ i = 1, …, N-1} the network connections among servers (M>N).
Output: A mapping M of O to S.
Begin
  1.  Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$
  2.  Sum_Capacity = $\sum_{i=1}^{N} P(S_i)$
  3.  Ideal_Cycles(S_i) = Sum_Cycles × $\frac{P(S_i)}{Sum\_Capacity}$, ∀ i = 1, …, N
  4.  Servers_List = (s_1, s_2, …, s_N) = (S_1, S_2, …, S_N)
  5.  Operations_List = (o_1, o_2, …, o_M) = (O_1, O_2, …, O_M)
  6.  Sort Servers_List so that ∀ i = 1, …, N-1  Ideal_Cycles(s_i) ≥ Ideal_Cycles(s_{i+1})
  7.  Sort Operations_List so that ∀ i = 1, …, M-1  C(o_i) ≥ C(o_{i+1})
  8.  M = ∅
  9.  for i=1 to M do
 10.          M = M ∪ {o_i→s_1}
 11.          Ideal_Cycles(s_1) – = C(o_i)
 12.          Move s_1 in Servers_List so that ∀ i = 1, …, N-1:
 13.                  Ideal_Cycles(i) ≥ Ideal_Cycles(i+1)
 14.          Continue with new Servers_List = (s_1, s_2, …, s_N)
 15.  end for
 16.  return M
End
```

The ideal load per server is computed

Sort servers per capacity, operations per cost and init mappings to ∅

***For** all (sorted) operations j, assign each operation to the strongest available server & resort servers*

results in a large message, the assignment is cancelled and the operation is assigned to the sender of the message, thus alleviating the need to send the message (see function *There Is Constraint* in Figure 12). A message is considered large whenever the time needed to transfer it is larger than the execution time of the costliest group of operations over the server with the most available cycles at the time the decision is made.

***Heavy Operations – Large Messages**.* Algorithm *Heavy Operations–Large Messages* (Figure 13) operates like *Fair Load*, with the fundamental difference that operations are not treated separately, but as groups. Two operations are clustered in the same group if they exchange a large message. As in the previous case, a message is considered large whenever the time needed to transfer it is larger than the execution time of the costliest group of operations over the server with the most available cycles at the time the decision is made. Recall that, in the bus topology, the communication cost between every pair of servers is considered the same. Activities that have been grouped together are always assigned to the same server.

Initially, each operation constitutes a group by itself. The algorithm employs three lists, one for the available cycles of each server, one for the size of each message and one for the cycles of each group. In the beginning of each step, these lists are sorted. In each step, the algorithm decides whether (a) to assign the most expensive group of operations to the server with the most available cycles, or (b) to avoid the exchange of a large message over the network. The decision is taken on the basis of the existence of a large message on the top of the list of the messages. If such a message exists, then option (b)

Figure 8. Algorithm fair load – tie resolver for cycles

```
Input: a workflow of web service operations W(O, E), with O = (O_1, O_2,…, O_M) and a server configuration N(S, L),
  with S = {S_1, S_2, …, S_N} and L all the combinations of server pairs with the same network costs (bus)
Output: a mapping M of O to S
Begin
    1.  $Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$ , $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$
    2.  $Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}$ , i = 1, …, N
    3.  $Servers\_List = (s_1, s_2, …, s_N) = (S_1, S_2, …, S_N)$
    4.  $Operations\_List = (o_1, o_2, …, o_M) = (O_1, O_2, …, O_M)$
    5.  Sort Servers_List so that $\forall$ i = 1, …, N-1  $Ideal\_Cycles(s_i) \geq Ideal\_Cycles(s_{i+1})$
    6.  Sort Operations_List so that $\forall$ i = 1, …, M-1  $C(o_i) \geq C(o_{i+1})$
    7.  Initialize M to a random Mapping
    8.  while Operations_List is not empty do
    9.          $gain_1$ = Gain_Of_Operation_At_Server($o_1$, $s_1$, M)
    10.         i=2
    11.         while $C(o_1) = C(o_i)$ and $i \leq M$ do{
    12.                 $gain_2$= Gain_Of_Operation_At_Server($o_i$, $s_1$, M)
    13.                 if $gain_2 > gain_1$ {
    14.                 swap($o_1$, $o_i$)
    15.                 $gain_1 = gain_2$ }
    16.                 i++
    17.         } //end inner while
    18.         $M = M - \{o_1 \rightarrow Server(o_1)\}$
    19.         $M = M \cup \{o_1 \rightarrow s_1\}$
    20.         Delete $o_1$ from Operations_List
    21.         $Ideal\_Cycles(s_1) -= C(o_1)$
    22.         Move $s_1$ in Servers_List so that $\forall$ i = 1, …, N-1:
    23.                 $Ideal\_Cycles(i) \geq Ideal\_Cycles(i+1)$
    24.         Continue with new $Servers\_List = (s_1, s_2,…, s_N)$
    25. } //end outer while
    26. return M
End
```

The ideal load per server is computed

Sort servers per capacity, operations per cost and init mappings randomly

***While** there are operations o_i equally costly with current (o_1) that give less cost if placed at current server (s_1), assign o_i at s_1*

When done with tie-breaking, same as before: move to next operation, resort servers, move to next server

Figure 9. Function gain of operation at server

```
Function Gain_Of _Operation_At_Server     //message size that we avoid putting in the network
Begin
    1.  gain = 0
    2.  if $O_i \in (O_2, O_3, …, O_M)$ and $\{O_{i-1} \rightarrow S_j\} \in M$ then
    3.          gain += MsgSize($O_{i-1}$, $O_i$)
    4.  if $O_i \in (O_1, O_2,…, O_{M-1})$ and $\{O_{i+1} \rightarrow S_j\} \in M$ then
    5.          gain += MsgSize($O_i$, $O_{i+1}$)
    6.  return gain
End
```

is followed. In this case, either (b1) both message ends are placed at the same server, or (b2) the two groups are merged. Option (b1) is followed, if one of the two operations that communicate through the large message is already placed at a server. Otherwise, the groups to which the communicating operations belong are merged. Note that messages must be removed from the list whenever both their ends are placed at the same server.

The complexities of the algorithms are *O(M×logM+N×logN+MN)* for *Fair Load*, and *O(M×(M×logM+N×logN+MN))* for the rest of the algorithms. In the algorithm *Heavy Operations–Large Messages*, instead of the

Figure 10. Algorithm fair load – tie resolver for cycles and servers

```
Input: A workflow W(O, E) (in line topology), where O = (O_1, O_2, …, O_M) a set of operations, E = {(O_i, O_{i+1}) | ∀ i = 1,…, M-1} the set of transitions among operations and N(S, L) a server network (in bus topology), where S = {S_1, S_2, …, S_N} a set of servers and L = {(S_i, S_{i+1}) | ∀ i = 1, …, N-1} the network connections among servers (M>N).
Output: A mapping M of O to S.
Begin
  1.  Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$
  2.  Sum_Capacity = $\sum_{i=1}^{N} P(S_i)$
  3.  Ideal_Cycles(S_i) = Sum_Cycles × $\frac{P(S_i)}{Sum\_Capacity}$, ∀ i = 1, …, N
  4.  Servers_List = (s_1, s_2, …, s_N) = (S_1, S_2, …, S_N)
  5.  Operations_List = (o_1, o_2, …, o_M) = (O_1, O_2, …, O_M)
  6.  Not_Assigned_Operations = M
  7.  Initialize M to a random Mapping
  8.  while Not_Assigned_Operations ≠ 0 do
  9.          Sort Servers_List so that ∀ i = 1, …, N-1 Ideal_Cycles(s_i) ≥ Ideal_Cycles(s_{i+1})
  10.         Sort Operations_List so that ∀ i = 1, …, Not_Assigned_Operations-1 C(o_i) ≥ C(o_{i+1})
  11.         best_gain = 0
  12.         for each operation o_i in Operations_List where C(o_i) = C(o_1) do
  13.                 for each server s_j in Servers_List where Ideal_Cycles(s_j)=Ideal_Cycles(s_1) do
  14.                         gain = Gain_Of_Put_Operation_At_Server(o_i, s_j, M)
  15.                         if gain > best_gain then
  16.                                 best_gain = gain
  17.                                 bestO = o_i
  18.                                 bestS = s_j
  19.                         end if
  20.                 end for
  21.         end for
  22.         M = M – {bestO→Server(bestO)}
  23.         M = M ∪ {bestO→bestS}
  24.         Not_Assigned_Operations – –
  25.         Ideal_Cycles(bestS) – = C(bestO)
  26.         Delete bestO from Operation_List
  27. end while
  28. return M
End
```

The ideal load per server is computed

Sort servers per capacity, operations per cost and init mappings randomly

***For all** unassigned operations (current= o_1) Sort servers per capacity left, operations as before. If there are combinations of servers and operations with a **tie** to the current pair of o1, s1, then find the best pair (bestO, bestS) and assign them the mapping for this round*

When done with tie-breaking, move to next operation & housekeeingp

last operand of the sum (i.e., *MN*) the cost to be added is *1*.

3.4 Algorithms for a Random Graph – Bus Configuration

In this third family of algorithms, we consider the case where the servers are still connected through a bus but the workflow is a random graph. All algorithms are practically the same with the category *Line-Bus*, with simple modifications that take the structure of the workflow into account. The algorithms must take into consideration that an operation can receive more than one message and that decision nodes possibly imply the execution of a subset of the workflow. Remember that decision nodes (*AND, OR, XOR*) start subgraphs of the workflow that are executed in parallel for *AND, OR* nodes. For *XOR* nodes, one of the alternatives is chosen. At the same time, the */AND, /OR, /XOR* nodes act as rendezvous points (and thus, they need to wait for at least two messages in the case of */AND*, */OR* flows).

The extension to the algorithms is quite simple: all the algorithms of this family (with the exception of algorithm *Fair Load* that remains exactly the same) assign an execution probability to each operation (and thus, each message) due to the existence of XOR decision

Figure 11. Algorithm fair load – merge messages' ends

```
Input: A workflow W(O, E) (in line topology), where $O = (O_1, O_2, …, O_M)$ a set of operations, $E = \{(O_i, O_{i+1}) \mid \forall i = 1,…, M-1\}$ the set of transitions among operations and N(S, L) a server network (in bus topology), where $S = \{S_1, S_2, …, S_N\}$ a set of servers and $L = \{(S_i, S_{i+1}) \mid \forall i = 1, …, N-1\}$ the network connections among servers (M>N).
Output: A mapping M of O to S.
Begin
1.  $Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$
2.  $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$
3.  $Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}, \forall i = 1, …, N$
4.  $Servers\_List = (s_1, s_2, …, s_N) = (S_1, S_2, …, S_N)$
5.  $Operations\_List = (o_1, o_2, …, o_M) = (O_1, O_2, …, O_M)$
6.  $Messages\_List = (m_1, m_2, …, m_{M-1}) = ( (O_1, O_2), (O_2, O_3), …, (O_{M-1}, O_M) )$
7.  Sort Messages_List so that $\forall i = 1, …, M-2$ $MsgSize(m_i) \geq MsgSize(m_{i+1})$
8.  Not_Assigned_Operations = M
9.  Initialize M to a random Mapping
10. while Not_Assigned_Operations $\neq 0$ do
11.     Sort Servers_List so that $\forall i = 1, …, N-1$ $Ideal\_Cycles(s_i) \geq Ideal\_Cycles(s_{i+1})$
12.     Sort Operations_List so that $\forall i = 1, …, Not\_Assigned\_Operations-1$ $C(o_i) \geq C(o_{i+1})$
13.     best_gain = 0;
14.     for each operation $o_i$ in Operations_List where $C(o_i) = C(o_1)$ do
15.         for each server $s_j$ in Servers_List where $Ideal\_Cycles(s_j)=Ideal\_Cycles(s_1)$ do
16.             gain = Gain_Of_Put_Operation_At_Server($o_i$, $s_j$, M)
17.             if gain > best_gain then
18.                 best_gain = gain
19.                 bestO = $o_i$
20.                 bestS = $s_j$
21.             end if
22.         end for
23.     end for
24.     M = M – {bestO→Server(bestO)}
25.     if There_Is_Constraint (bestO, bestS, M, $MsgSize(m_{(M-1)\times 0.1})$, constraints_flag) then
26.         if constraints_flag = left_message then
27.             M = M ∪ {bestO→ServerOf(LeftOperationOf(bestO))}
28.             Ideal_Cycles(ServerOf(LeftOperationOf(bestO)) – = C(bestO)
29.         else
30.             M = M ∪ {bestO→ServerOf(RightOperationOf(bestO))}
31.             Ideal_Cycles(ServerOf(RightOperationOf(bestO)) – = C(bestO)
32.         end if
33.     else
34.         M = M ∪ {bestO→bestS}
35.         Ideal_Cycles(bestS) – = C(bestO)
36.     end if
37.     Not_Assigned_Operations – –
38.     Delete bestO from Operation_List
39. end while
40. return M
End
```

The ideal load per server is computed

Sort servers per capacity, operations per cost and init mappings randomly

***While** there are operations o_i equally costly with current (o_1) that give less cost if placed at current server (s_1), assign o_i at s_1 and keep the winner pair (operation at server) at (bestO, bests)*

***If** this assignment produces large msg's in the network, move the operation left or right (according to the result of ThereISConstraint)*
***Else** just do the assignment*
In any case, finally, do housekeeping

nodes. The determination of this probability is based on monitoring initial executions of the workflow or simple prediction mechanisms. Thus, the execution cost is a practically a weighted cost, amortized for a large number of workflow executions (as opposed to a single execution as in the case of linear workflows).

EXPERIMENTS

In this section, we present experimental results for the assessment of the proposed algorithms. First, we present an experiment for the validation of the cost model. Then, we discuss experiments for each of the three workflow-network configurations explored in the previous sections,

Figure 12. Function there is constraint

Function There Is Constraint

Input: An operation $O_i \in O$, a server $S_j \in S$, a threshold big_message_size that determines whether a message is large or not and a mapping $\mathbf{M} \subseteq O \times S$.

Output: True if there is a constraint in assigning O_i to S_j; otherwise False. In the former case, a flag constraints_flag takes one of the values left_message or right_message to signify which of the two messages (O_{i-1}, O_i) and (O_i, O_{i+1}) triggers the constraint. If both messages trigger a constraint violation, the one furthest from the threshold value is highlighted.

```
Begin
    1.  if $O_i = O_1$ then
    2.      if MsgSize($O_1$, $O_2$) $\geq$ big_message_size then
    3.          constraints_flag = right_message
    4.          return True
    5.      end if
    6.  else if $O_i = O_M$ then
    7.      if MsgSize($O_{M-1}$, $O_M$) $\geq$ big_message_size then
    8.          constraints_flag = left_message
    9.          return True
    10.     end if
    11. else if $O_i \in (O_2, O_3, \ldots, O_{M-1})$ then
    12.     if MsgSize($O_{i-1}$, $O_i$) $\geq$ big_message_size then
    13.         constraints_flag = left_message
    14.     end if
    15.     if MsgSize($O_i$, $O_{i+1}$) $\geq$ big_message_size then
    16.         if constraints_flag = left_message and MsgSize($O_{i-1}$,$O_i$) $\geq$ MsgSize($O_i$,$O_{i+1}$)
    17.             constraints_flag = left_message
    18.             return True
    19.         else
    20.             constraints_flag = right_message
    21.             return True
    22.         end if
    23.     end if
    24. end if
    25. return False
End
```

and finally, we summarize our experimental findings.

Cost Model Validation

To validate our cost model we performed an experiment that relied on the reference example discussed previously. More specifically, we implemented the workflow schema of Figure1 using a widely-used infrastructure that supports the development of web services. Then, we executed this implementation in a configuration of three servers S1, S2, S3, and compared the mean execution time of the workflow (calculated over 100 executions of the workflow) with the execution time that is estimated based on the proposed cost model.

To implement the web service operations that constitute the examined workflow we used AXIS v1.1. The web service operations that concern the patient, the secretary and the doctor were deployed, respectively, on a P-IV 3.4 GHz, a P-IV 1.6 GHz and a P-IV 2.13 GHz server. The application server that we used in all three servers was Apache Tomcat 4.1. The three servers were connected through a typical 100 Mbps Ethernet.

Concerning the cost model, we measured the cycles required for each web service operation by executing the workflow on a single server (i.e., the P-IV 1.6 GHz). In this context, we measured the time required by each operation to perform its computations, along with the time required for preparing its invocations to the web service operations with which it communicates (e.g., for the web service operation 3, we measured the time required for its internal computation and the time required for preparing

Figure 13. Algorithm heavy operations - large messages

Input: A workflow **W**(O, E) (in line topology), where $O = (O_1, O_2, \ldots, O_M)$ a set of operations, $E = \{(O_i, O_{i+1}) \mid \forall i = 1,\ldots, M-1\}$ the set of transitions among operations and **N**(S, L) a server network (in bus topology), where $S = \{S_1, S_2, \ldots, S_N\}$ a set of servers and $L = \{(S_i, S_{i+1}) \mid \forall i = 1, \ldots, N-1\}$ the network connections among servers (M>N).
Output: A mapping **M** of O to S.

```
Begin
1.   Sum_Cycles = $\sum_{i=1}^{M} C(O_i)$
2.   Sum_Capacity = $\sum_{i=1}^{N} P(S_i)$
3.   Ideal_Cycles(S_i) = Sum_Cycles × $\frac{P(S_i)}{Sum\_Capacity}$, $\forall i = 1, \ldots, N$
4.   Servers_List = $(s_1, s_2, \ldots, s_N) = (S_1, S_2, \ldots, S_N)$
5.   Group_Of_Oper_List = $(g_1, g_2, \ldots, g_M) = (O_1, O_2, \ldots, O_M)$
6.   Messages_List = $(m_1, m_2, \ldots, m_{M-1}) = ( (O_1, O_2), (O_2, O_3), \ldots, (O_{M-1}, O_M) )$
7.   Not_Assigned_Operations = M
8.   while Not_Assigned_Operations $\neq 0$ do
9.       Sort Servers_List so that $\forall$ i=1…N-1 Ideal_Cycles($s_i$) $\geq$ Ideal_Cycles($s_{i+1}$)
10.      Sort Group_Of_Oper_List so that $\forall$ i=1…number of groups-1 C($g_i$) $\geq$ C($g_{i+1}$)
11.      Sort Messages_List so that
12.          $\forall$ i = 1, …, size of Messages_List-1 MsgSize($m_i$)$\geq$MsgSize($m_{i+1}$)
13.      if Tproc($g_1$) at server $s_1$ > Time of sending $m_1$ via bus then
14.          for each $o_i \in g_1$ do
15.              M = M $\cup$ {$o_i \rightarrow s_1$}
16.              Not_Assigned_Operations – –
17.              Ideal_Cycles($s_1$) – = C($o_i$)
18.          end for
19.          Delete $g_1$ from Group_Of_Oper_List
20.      else
21.          if source($m_1$) is not assigned and target($m_1$) is assigned then
22.              M = M $\cup$ {source($m_1$)$\rightarrow$ServerOf (target($m_1$))}
23.              Not_Assigned_Operations – –
24.              Ideal_Cycles(ServerOf (target($m_1$))) – = C(source($m_1$))
25.              Delete source($m_1$) from its group in Group_Of_Oper_List
26.          else if source($m_1$) is assigned and target($m_1$) is not assigned then
27.              M = M $\cup$ {target($m_1$)$\rightarrow$ServerOf (source($m_1$))}
28.              Not_Assigned_Operations – –
29.              Ideal_Cycles(ServerOf (source($m_1$))) – = C(target($m_1$))
30.              Delete target($m_1$) from its group in Group_Of_Oper_List
31.          else // both source($m_1$) and target($m_1$) are not assigned
32.              $g_{new}$ = Merge (group(source($m_1$)), group(target($m_1$)) )
33.              Insert $g_{new}$ in Group_Of_Oper_List
34.          end if
35.      end if
36.      for i=1 to size of Messages_List do
37.          if source($m_i$) and target($m_i$) are assigned
38.              Delete $m_i$ from Messages_Lists
39.  end while
40.  return M
End
```

The ideal load per server is computed

Initially, each operation is a separate group

***While** there are operations unassigned: sort servers by remaining capacity, groups by cost and messages by size **if** cost of assigning costliest operation at server is larger than sending larger msg, assign all the group of this operation to the server*

***else** pick largest msg m1 and check the ends of its edge; **if** one of the two ends is unassigned, assigned the other in the same server; **else** both ends groups' are merged*

Side-effect of a groups assignment is that some msg will not travel in the network, so it must be removed from the msg list

the invocation of web service operation 4). The sum of the aforementioned execution times was transformed into computational cycles based on the server's computational power. It should be further noted that in the calculation of the estimated execution time of the examined workflow we assumed that the decision operations (operations 5 and 9 of the reference example) do not require any computational cycles. The parameters used in our cost model are given in more detail in Table 2.

The mean execution time that was experimentally measured for the examined workflow was 187 msec. The estimated execution time that was calculated with respect to the parameters of Table 2 was 192 msec, i.e., the cost model overestimated the execution time with the acceptably small error of 2%.

Table 2. Cost Model Parameters for our motivating example

Symbol	**Description**									
C(op) **- The cycles necessary for operation *op* to complete**										
C(1)	C(2)	C(3)	C(4)	C(6)	C(7)	C(8)	C(10)	C(11)	C(12)	C(13)
24992000	39632000	39632000	54272000	24992000	39632000	54272000	24992000	39632000	24992000	24992000
P(s) **- Computational power of server *s* (Hz)**										
P(S1)				**P(S2)**				**P(S3)**		
$3.4 * 10^6$				$1.6 * 10^6$				$2.13 * 10^6$		
Server(op) **- The server where operation *op* is deployed**										
Server(1)	Server(2)	Server(3)	Server(4)	Server(6)	Server(7)	Server(8)	Server(10)	Server(11)	Server(12)	Server(13)
S1	S2	S3	S2	S3	S1	S3	S2	S1	S2	S1
Msgize(op_i, op_j) **- Message size sent from operation op_i to operation op_j, assuming $(op_i,op_j) \in E$ (Mbits)**			***Line speed (Mbps) between servers s_i and s_j***				**$T_{comm}(op_i, op_j)$ - Communication time needed for the communication of operations op_i and op_j (msec)**			
0.00666			100				0.0666			

Experimental Methodology

We have varied several parameters of the workflow-network configurations used. We use the results of Head et al. (2005) and Ng, Chen, and Greenfield (2004) to determine appropriate values for our experiments. In Ng et al. (2004), three types of SOAP messages are used: simple messages of 873 bytes (0.00666 Mbits), medium messages of 7581 bytes (0.057838 Mbits), and complex messages of 21392 bytes (0.163208 Mbits). We assume 4, 10, and 20 ms as the time needed for the execution of a web service operation (this includes the serialization, network time, deserialization and server execution time). Assuming a value of 37% for the parsing of a message, this results in 2.5, 6.3 and 12.7 M cycles for simple, medium and complex messages, respectively (over a 1.6 MHz CPU). Based on the previous in our experiments we assume simple, medium and heavy web service operations, requiring respectively, 10, 20 and 30 M cycles.

In each experiment we considered a specific workflow of web service operations deployed over a specific network of servers. The operations' cost, the associated messages, the computational power of the servers and the network characteristics were randomly selected based on the probabilities given in Table 3.

In all the graphical representations, the horizontal axis of each diagram depicts the execution time and the vertical axis the time penalty. The closer a solution to point (0, 0), the better. Assuming different weights for the two measures, different distance measures could also be considered. The average point of the search space of solutions is also depicted as a measure of how a "random" choice of deployment would possibly be. In small configurations this involves all the solutions of the search space; in large configurations a sample of 32,000 solutions.

Experiments for a Line: Line Configuration

The experiments performed for this configuration involve all four variants of the proposed algorithm. Specifically, we have experimented with (a) single dimension (left-to-right) *Line-Line* algorithm with the application of *Fix_Bad_Bridges* (denoted as *1d with Fix*), (b) single dimension (left-to-right) *Line-Line* algorithm without the application of *Fix_Bad_Bridges* (denoted as *1d no Fix*), (c) double dimension (left-to-right and right to left) *Line-Line* algorithm with the application of *Fix_Bad_Bridges* (denoted as *2d with Fix*), and (d) double dimension (left-to-right and right to left) *Line-Line* algorithm without the application of *Fix_Bad_Bridges* (denoted as *2d no Fix*).

Figure 14 presents four experiments for larger configurations (each reported in the figure). Due to the vastness of the search space,

Table 3. Experimental configuration

$MsgSize(O_i, O_{i+1})$	*0.006660 Mbits* with probability *25%* *0.057838 Mbits* with probability *50%* *0.163208 Mbits* with probability *25%*
Line_Speed(S_i, S_{i+1}) *(for Line topologies)*	*10 Mbps* with probability *25%* *100 Mbps* with probability *50%* *1000 Mbps* with probability *25%*
$C(O_i)$	*10 M* cycles with probability *25%* *20 M* cycles with probability *50%* *30* M cycles with probability *25%*
$P(S_i)$	*1 GHz* with probability *25%* *2 GHz* with probability *50%* 3 *GHz* with probability 25%

the reported average concerns 32,000 randomly sampled solutions. The different variants of the algorithm do not show significant differences, while presenting satisfactory solutions with respect to both the time penalty and the execution time. Observe how far from the optimal solution the average solution is placed (upper right part of each figure). Similar behaviour was obtained in experiments for smaller configurations of workflows with 8 operations over network topologies of 3 servers.

Experiments for a Line – Bus Configuration

We have conducted all classes of experiments with all the proposed algorithms participating for the configuration of linear workflows executed over a network bus. In Figure 15, we depict comparative results of the employed algorithms by computing the average solution of 50 experiments. We test workflows of 19 operations over network topologies of (a) 5, (b) 10, and (c) 15 servers connected through a bus whose line speed takes one out of the following values: 1, 10, 100 Mbps. Both *Tie Resolver* algorithms provide some improvements in both dimensions, whereas the *FL- Merge Message's Ends* improves the execution time to a certain extent by deteriorating the load balance. The *HeavyOps-LargeMsgs* algorithm produces quite acceptable execution times, esp. for small bus capacities and practically seems to be the more stable solution compared to all the others. It is interesting that the behavior of the *HeavyOps-LargeMsgs* algorithm remains quite stable even when the fraction of operations to servers (denoted as *K*) increases.

In terms of the quality of the solution, *HeavyOps-LargeMsgs* produces (2.9%, 12%) deviations for execution time/time penalty for 1Mbps bus, and (29%,0.3%) for 100 Mbps bus.

As an overall result, we can safely argue that *FL-Tie Resolver2* seems to provide quite fair

Figure 14. Results for the Line-Line Algorithm in various configurations

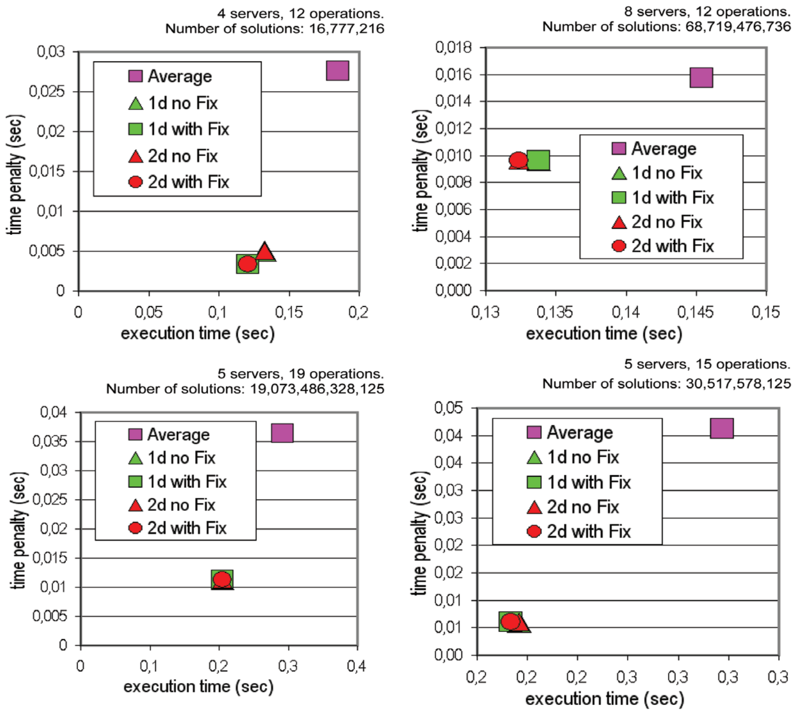
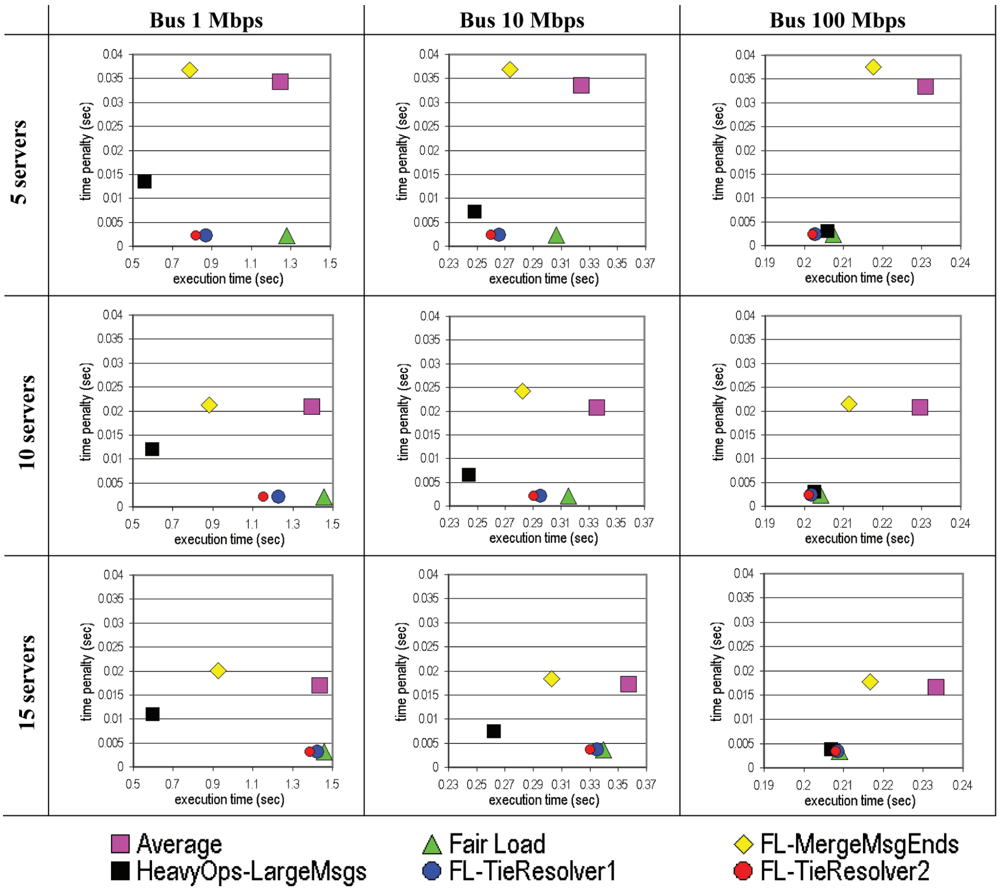


Figure 15. Line – Bus algorithms with 19 operations in the workflow



solutions, whereas the *HeavyOps-LargeMsgs* algorithm is slightly worse in this category, but provides consistently good execution times in all configurations.

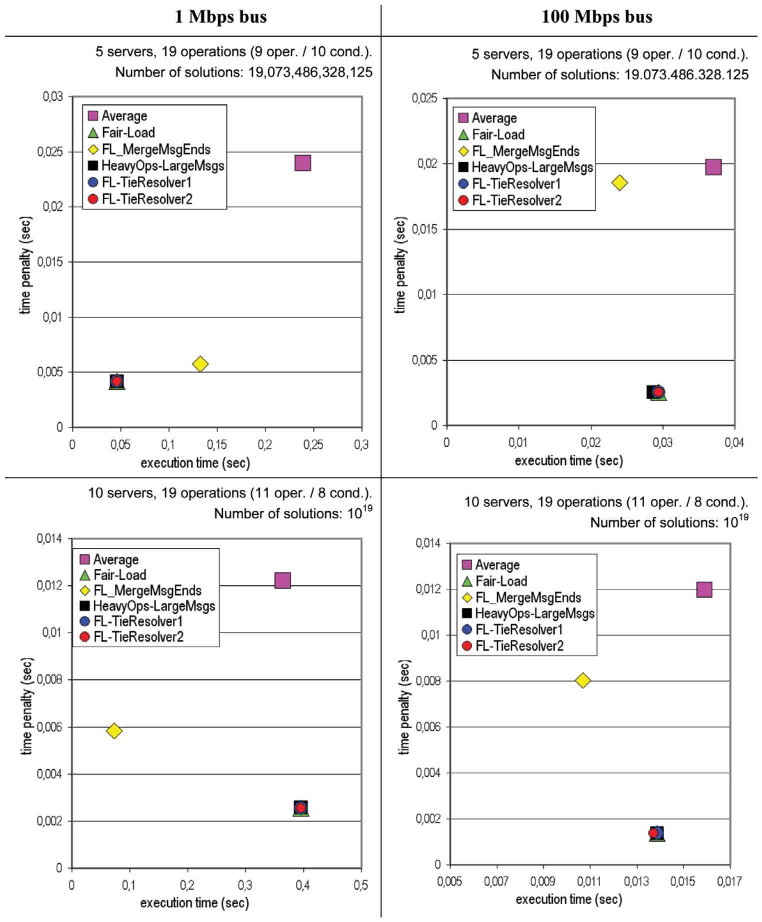
Experiments for a Random Graph – Bus Configuration

In the case of workflows with random graph structures, we have discerned three cases: (a) bushy, (b) lengthy and (c) hybrid graphs. Bushy graphs have a high percentage of decision nodes (and are therefore shorter in length, but with a higher fan-out). Lengthy graphs have a small percentage of decision nodes and involve lengthy paths. Hybrid graphs are somewhere in the middle. Specifically, bushy graphs involve a 50%-50% balance of decision/operational nodes, lengthy graphs involve a 16%-84% balance and hybrid graphs a 35%-65% one.

In Figure 16 we see four experiments for bushy workflows of 19 operations over topologies of (a) 5 or (b) 10 servers connected through a bus whose line speed takes one out of the following values: 1, 100 Mbps. The difference is in the number of servers and the speed of the bus. Algorithm *Fair Load–Merge Messages' Ends* seems to perform better than the rest, especially when the speed of the bus is slow.

In Figure 17 we see the respective experiment for lengthy workflows. It is interesting to see that in contrary with the previous experiment, *Fair Load–Merge Messages' Ends* can

Figure 16. Graph – bus algorithms for bushy workflows



give very bad solutions. The most reliable algorithm for this category seems to be *HeavyOps-LargeMsgs*. It is also interesting that *Fair Load* can be better than its tie-resolver improvements.

In Figure 18, we depict the average values of 50 experiments for workflows of 19 operations over 10 servers. As one can see, the results are not very different from the ones for the previous topology. For almost all configurations, the *HeavyOps-LargeMsgs* algorithm appears to be a clear winner: it is consistently the best choice in terms of execution time and it also appears to be the quite close to the best solutions in terms of fairness. *FL-Merge Message's Ends* appears to be quite close in terms of execution time (in fact, in individual experiments it has occasionally outperformed *HeavyOps-LargeMsgs*), still it is quite unstable with respect to its fairness.

In terms of the quality of the solution, *HeavyOps-LargeMsgs* produces (29%, 1.8%) deviations for execution time/time penalty for the 1Mbps bus, and (0%, 0%) for the 100 Mbps bus.

Summary of Experimental Findings

In summary, our experimental findings are as follows:

1. In the *Line-Line* configurations, all algorithms behave well and give a solution quite close to the optimal one. The variant of double direction Line-Line algorithm appears to perform slightly better, whereas the fixing of bad bridges appears to work only when the messages are really heavy for the network connections involved.
2. In the *Line-Bus* configurations, the algorithm *Fair Load–Tie Resolver*$_2$ is the most appropriate for fair solutions, whereas the algorithm *Fair Load–Merge Messages' Ends* and *Heavy Operations–Large Messages* are the most appropriate whenever execution time is more important.
3. In the case of *Graph-Bus* configurations, the same observations still hold. In the case of bushy workflows, the algorithm *Fair Load–Merge Messages' Ends* is better than *Heavy Operations–Large Messages*, especially with respect to the execution time.

DISCUSSION

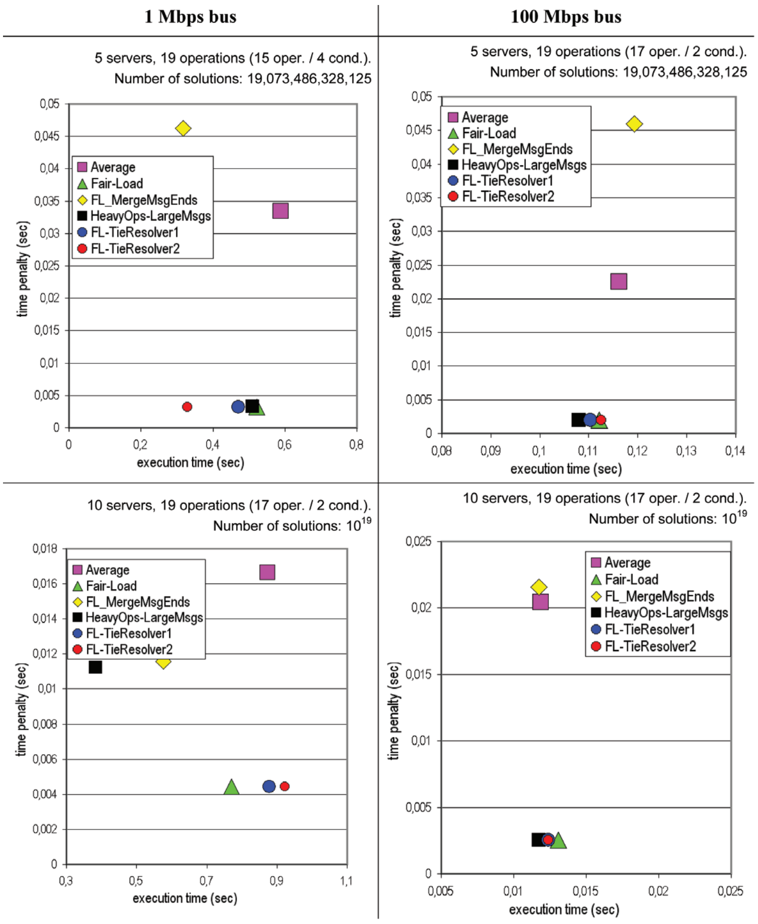
Summary of findings, importance and implications. Summarizing the proposed method, we can argue that there is indeed the possibility of finding efficient deployments of services to servers for composite web service workflows. We have examined different, reasonable topologies of servers and discovered that more than one algorithm can be applied to provide good solutions to the service deployment problem.

This result is important since it covers an open gap in the related literature. As already mentioned, although related research deals with the problems of guaranteeing quality-of-service characteristics once the allocation of services to servers is performed. This work provides the means to the administrator to explore alternative possibilities for this deployment and in fact, it produces solutions of good quality as already demonstrated at the experimental section. Both researchers and practitioners can employ the proposed algorithms as a first step before the subsequent fine tuning that the rest of the methods discussed in the related work section provide.

Fitness within the broader perspective. Taking a step back and looking at the wider view of both research and practice in the area of software systems engineering, the software challenge of the near future is dealing with the dramatically increasing complexity and scale of the systems that we build (Northrop et al., 2006). Software systems that rely on Internet technologies in particular must cope with various constantly growing dimensions of scale including load, heterogeneity, broad distribution, cardinality of services that are available and should be coordinated. Therefore, from this boarder perspective the most important factor that affects the way web services should evolve in the near future is the matter of scale. Apparently, the movement towards a service-oriented architecture of software at the web will drive more and more people and organizations to export functionality at the web. As a service provider exports more and more functionality via services over the web, the more popular, complicated and sophisticated the composite workflows that the users construct will be. Thus, it is highly important that efficient execution is achieved in the back-stage infrastructure of a service provider.

The contribution of this work can serve as the basis for pursuing performance improvements in the case of service compositions that span the borders of several systems. The advent of Web 2.0 brought us the notion of user-defined *mashups* (see Benslimane, Dustdar, & Sheth, 2008; Maximilien, Ranabahu, & Gomadam, 2008) that integrate services possibly from several providers via a user-friendly graphical user interface that allows a naïve user to try and construct an application in an on-line fashion. Thus, the

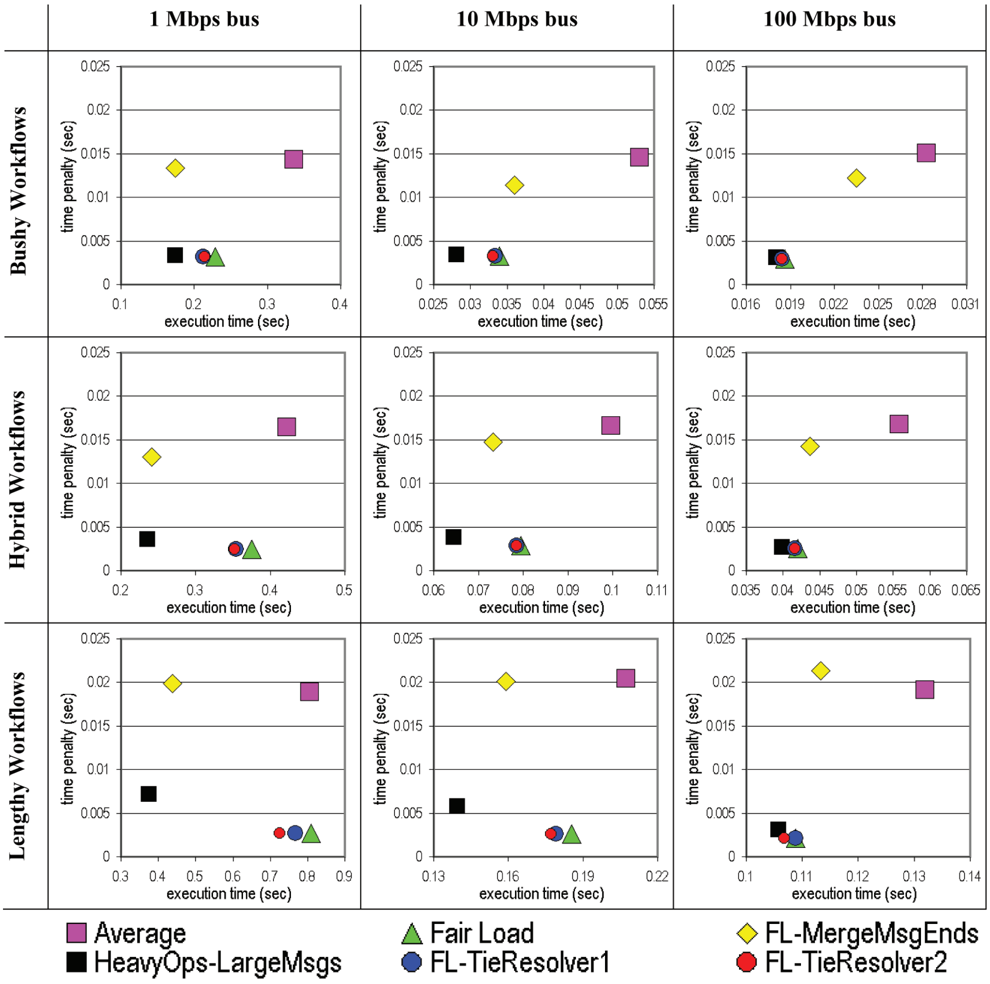
Figure 17. Graph – bus algorithms for lengthy workflows



development task moves from the experienced programmer to the less skilled users (see Firat, Wu, & Madnick, 2009; Kongdenfha, Benatallah, Vayssière, Saint-Paul, & Casati, 2009). There is a plethora of mashup applications like Yahoo Pipes (http://pipes.yahoo.com/), Google Maps (http://maps.google.com/), IBM Damia (http://services.alphaworks.ibm.com/damia/), and Microsoft Popfly (http://www.popfly.com/). The on-line nature of such applications makes the optimization of the execution of a workflow more and more important. Fast algorithms that deduce memory allocation, execution and choice of services in an on-line fashion, appropriate for end-users are highly important. This paper can provide a useful basis for subsequent research in this direction.

Limitations and possibilities for further research. A clear limitation of the existing method is the fact that related research has not matured with respect to the cost models for web service execution. Currently, we do not have the experimental findings (also due to the youth of this technology) to be able to predict with accuracy the behavior of a web service. Therefore the designer needs to perform micro-benchmarks to assess the cost of the operations. Moreover, phenomena of load bursts are very

Figure 18. Graph – bus algorithms organized per graph structure



common in the internet; this is where the fine-tuning fits. Still, some considerations for burst-handling have not been covered and could be explored as part of future work. Again, before that, we need some good experimental evidence of how bursts happen over the internet – to the best of our knowledge no such data sets exist.

Further Extensions Other extensions for this work involve the case of multiple workflows (instead of just a single one) and the detailed study of the proposed algorithms whenever user-defined constraints are given. For instance, apart from the overall execution time, the response time of individual operations can also be considered as part of the cost model.

CONCLUSION

In this paper, we have dealt with the problem of discovering the best possible deployment of the operations of a certain workflow given its structure and a topology of servers. To the best of our knowledge, this is the first attempt towards the issue. Thus, our approach fits nicely with existing efforts that are concerned with the fine-tuning of the workflow to obtain desired levels of quality of service as it provides the necessary first step of service deployment, before the fine-tuning can start. We have measured efficiency in terms of two cost functions that concern the execution time of the workflow and the fairness of the load on the servers. We have studied different topologies for the workflow structure and the server connectivity and proposed greedy algorithms for each combination. Our experiments indicate that algorithm *HeavyOps-LargeMsgs* is a good choice for all the considered configurations.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers of this paper, as well as the reviewers of the first version of this paper (accepted in SEIW 2007) for their constructive comments. This research was co-funded by the European Union in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the 3rd Community Support Framework of the Hellenic Ministry of Education, funded by 25% from national sources and by 75% from the European Social Fund (ESF).

REFERENCES

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services concepts, architecture and applications*. New York: Springer.

Andrews, T., Curbera, F., Dholakia, H., Golang, Y., Klein, J., Leymann, F., et al. (2003). *Business process execution language for Web services version 1.1*. Retrieved from http://www.ibm.com/developerworks/library/ws-bpel/

Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services Mashups: The New Generation of Web Applications. *IEEE Internet Computing*, *12*(5), 13–15. doi:10.1109/MIC.2008.110

Burstein, M., Bussler, C., Finin, T., Huhns, M., Paolucci, M., & Sheth, A. (2005). A semantic Web services architecture. *IEEE Internet Computing*, *9*, 52–61. doi:10.1109/MIC.2005.96

Cardoso, J., Sheth, A., Miller, J., Arnold, J., & Kochut, K. (2004). Quality of service for workflows and Web service processes. *Journal of Web Semantics*, *1*(3), 281–308. doi:10.1016/j.websem.2004.03.001

Chen, Y., Zhou, L., & Zhang, D. (2006). Ontology-supported Web service composition: An approach to service-oriented knowledge management in corporate services. *Journal of Database Management*, *17*(1), 67–84.

Cherkasova, L., & Gupta, M. (2004). Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change. *ACM/IEEE Transactions on Networking, 12*(5), 781-794.

Cherkasova, L., & Phaal, P. (2002). Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, *51*(6), 669–685. doi:10.1109/TC.2002.1009151

Constantinescu, I., Binder, W., & Faltings, B. (2005). Optimally distributing interactions between composed semantic Web services. In A. Gomez-Perez & J. Euzenat (Eds.), *Proceedings of the 2nd European Semantic Web Conference* (pp. 32-46).

Conti, M., Kumar, M., Das, S. K., & Shirazi, B. A. (2002). Quality of service issues in internet Web services. *IEEE Transactions on Computers*, *51*(6), 593–594. doi:10.1109/TC.2002.1009145

Ding, Y., Fensel, D., & Klein, A. B. O. (2002). The semantic Web: yet another hip? *Data & Knowledge Engineering*, *41*(3), 205–227. doi:10.1016/S0169-023X(02)00041-1

Erickson, J., & Siau, K. (2008). Web Services, service-oriented computing, and service-oriented architecture: separating hype from reality. *Journal of Database Management*, *19*(3), 42–54.

Firat, A., Wu, L., & Madnick, S. (2009). General Strategy for Querying Web Sources in a Data Federation Environment. *Journal of Database Management*, *20*(2), 1–18.

Geer, D. (2003). Taking steps to secure web services. *IEEE Computer*, *36*(10), 14–16.

Gillmann, M., Weikum, G., & Wonner, W. (2002). Workflow management with service quality guarantees. In D. DeWitt (Ed.), *ACM SIGMOD International Conference on Management of Data* (pp. 228-239).

Gravano, L., & Papakonstantinou, Y. (1998). Mediating and meta searching on the internet. *A Quarterly Bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering*, *21*(2), 28–36.

Head, M., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., van Engelen, R., et al. (2005). A benchmark suite for SOAP-based communication in Grid Web services. In W. Kramer (Ed.), *ACM/IEEE Conference on High Performance Networking and Computing* (pp. 19-20).

Kongdenfha, W., Benatallah, B., Vayssière, V., Saint-Paul, R., & Casati, F. (2009). Rapid development of spreadsheet-based web mashups. In *Proceedings of the 18th International Conference on the World Wide Web* (pp. 851-860).

Laoutaris, N., Telelis, O., Zissimopoulos, V., & Stavrakakis, I. (2006). Distributed selfish replication. *IEEE Transactions on Parallel and Distributed Systems*, *17*(12), 1401–1413. doi:10.1109/TPDS.2006.171

Leff, A., Wolf, J. L., & Yu, P. S. (1993). Replication algorithms in a remote caching architecture. *IEEE Transactions on Parallel and Distributed Systems*, *4*(11), 1185–1204. doi:10.1109/71.250099

Lewis, P. M., Bernstein, A. J., & Kifer, M. (2001). *Databases and transaction processing: An application-oriented approach*. Reading, MA: Addison-Wesley.

Leymann, F., & Roller, D. (2000). *Production workflow: concepts and techniques*. Upper Saddle River, NJ: Prentice Hall.

Maximilien, E. M., Ranabahu, A., & Gomadam, K. (2008). An Online Platform for Web APIs and Service Mashups. *IEEE Internet Computing*, *12*(5), 32–43. doi:10.1109/MIC.2008.92

Maximilien, E. M., & Singh, M. P. (2004). A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, *8*(5), 84–93. doi:10.1109/MIC.2004.27

Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., & Longstaff, T. (2006). *Ultra Large Scale Systems. The Software Challenge of the Future. Software Engineering Institute*. Pittsburgh, PA: Carnegie Mellon.

Papazoglou, M. (2003). Web services and business transactions. *World Wide Web (Bussum)*, *6*(1), 49–91. doi:10.1023/A:1022308532661

Papazoglou, M., & Kratz, B. (2007). Web services technology in support of business transactions. *Service Oriented Computing and Applications*, *1*(1), 51–63. doi:10.1007/s11761-007-0002-3

Papazoglou, M., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research directions. *IEEE Computer*, *40*(11), 64–71.

Papazoglou, M., & van den Heuvel, W. J. (2007). Service oriented architectures: Approaches, technologies and research issues. *Very Large Database Journal*, *16*(3), 389–415. doi:10.1007/s00778-007-0044-3

Rezgui, A., Bouguettaya, A., & Eltoweissy, M. Y. (2003). Privacy on the Web: facts, challenges, and solutions. *IEEE Security and Privacy*, *1*(6), 40–49. doi:10.1109/MSECP.2003.1253567

Sakellariou, R., & Zhao, H. (2004). A low-cost rescheduling policy for efficient mapping of workflows on Grid systems. *Science Progress*, *12*(4), 253–262.

SOAP. (2003). *Simple Object Access Protocol version 1.2*. Retrieved from http://www.w3.org/TR/soap12-part0

Srivastava, U., Widom, J., Munagala, K., & Motwani, R. (2005). *Query optimization over Web services*. Retrieved from http://dbpubs.stanford. edu:8090/pub/2005-30

Vinoski, S. (2002). Web services interaction models, part 1: current practice. *IEEE Internet Computing*, *6*(3), 89–91. doi:10.1109/MIC.2002.1003137

WSDL. (2003). *Web services description language version 2.0*. Retrieved from http://www.w3.org/TR/wsdl20

Yu, Q., Liu, X., Bouguettaya, A., & Medjahed, B. (2008). Deploying and managing Web services: issues, solutions, and directions. *Very Large Database Journal*, *17*, 537–572. doi:10.1007/s00778-006-0020-3

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., & Chang, H. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, *30*(5), 311–327. doi:10.1109/TSE.2004.11

Konstantinos Stamkopoulos received his Bachelor's degree from the Department of Computer Science of the University of Ioannina, Greece in 2004 and his MSc in Computer Science from the same department in 2006. His research interests include web services, especially in their interaction with databases.

Evaggelia Pitoura received her B.Sc. from the University of Patras, Greece in 1990 and her M.Sc. and Ph.D. in computer science from Purdue University in 1993 and 1995, respectively. Since June 2005, she is an associate professor at the Department of Computer Science of the University of Ioannina, Greece where she leads the distributed data management group. Her publications include more than 70 articles in international journals and conferences and a book on mobile computing. She has also co-authored two tutorials on mobile computing for IEEE ICDE 2000 and 2003. She is recipient of the best paper award of IEEE ICDE 1999 and two "Recognition of Service Awards" from ACM.

Panos Vassiliadis received his PhD from the National Technical University of Athens in 2000. He joined the Department of Computer Science of the University of Ioannina in 2002. Currently, Dr. Vassiliadis is also a member of the Distributed Management of Data (DMOD) Laboratory (http://www.dmod.cs.uoi.gr/). His research interests include data warehousing, web services and database design and modeling. Dr. Vassiliadis has published more than 25 papers in refereed journals and international conferences in the above areas.

Apostolos Zarras received his B.Sc. in Computer Science in 1994 from the Computer Science Department, University of Crete. From the same department he received his M.Sc. in Distributed Systems and Computer Architecture. In 1999 he received his Ph.D in Distributed Systems and Software Architecture from the University of Rennes I. From 2004 until now he holds a position at the Department of Computer Science of the University of Ioannina. Apostolos Zarras has published over 20 papers in international conferences, journals and magazines. He is currently a member of the IEEE computer society. His research interests include middleware, model-driven architecture development, quality analysis of software systems and pervasive computing.

APPENDIX A: BACKGROUND TERMINOLOGY AND STANDARDS

SOAP. The Simple Object Access Protocol (SOAP) is the means that facilitates web service communication and the basis for the development of practically all other protocols in the area. SOAP specifies a message format for the communication with Web services along with the bindings to HTTP and SMTP protocols for the delivery of messages. The communication in SOAP is one-way and asynchronous. The messages are XML documents, consisting of *envelopes* comprising a *header*, with meta-information for the processing of the message and a *body* with the actual contents of the message.

WSDL. The Web Services Description Language (WSDL) is a specification language for the generation of XML documents that (a) *describe* and (b) *automate the deployment* of web services. In practice, WSDL documents play the role of IDL's in conventional middleware: they provide a description of the available interfaces of a web service and at the same time, they can serve as an input to the appropriate compiler to generate client stubs and server skeletons.

WSDL descriptions are decomposed into two parts, or *perspectives*. In the *logical perspective*, α WSDL document specifies the interface of a web service. In the case of web services, the traditional middleware interface is replaced by a set of *messages* that the web service generates or receives. In practice, there are four modes that a service can use to interact with other programs: *one-way* (simply receives messages), *request-response* (sends a request message and waits for an answer), *solicit-response* (receives a message and responds appropriately) and *notification* (sends a message). A fault message can be sent, in some of these cases, whenever an error is encountered. This kind of modus operandi in WSDL is regulated by *operations*. In fact, operations are the counterpart of "interface" in traditional middleware. Operations are grouped by *port types*. Intuitively, a port type is a collection of operations, offered by the same organization, towards semantically similar functionalities. Port types are the coarser granule of logical organization in the WSDL specification. At the other end of the spectrum, the finest granule in the logical perspective involves message *parts*. Each message is an XML document organized in parts. Each part is assumed to be a typed granule of information; still, complex parts can be defined, too.

The logical perspective of WSDL documents is accompanied by the *physical* perspective, responsible for the description of the communication protocols and the ports where functionality is exposed. A *binding* specifies how a certain operation will communicate with the rest of the world; this is typically done via the HTTP or the SMTP (e-mail, that is) protocols. Technicalities of the message that is transported are also hidden behind the *style* of the binding. At the same time, a *port* (or *end-point*) offers a URI for a certain binding. Naturally, more than one bindings can be offered for the same operation and consequently, more than one ports. A *WSDL service* is a collection of ports, i.e., a set of URI's where functionality of the organization is exported. It is interesting that, in the WSDL specification, a service is an object in the physical perspective: the logical definitions are part of the specification but embodied in the overall service.

WSDL acts both as a service *description* language and a service *definition* language. As a description language, the WSDL documentation is a form of contract that describes the set of messages that the services guarantees to receive (input messages) and deliver (output messages). As a definition language, WSDL is employed as the means to generate the actual code that is exported in order to invoke the service. This follows a traditional IDL approach, where the development and invocation of an application are guided through an IDL definition both at the server and at the client (through the well-known stub/skeleton mechanism).

BPEL. The Business Process Execution Language (BPEL) is employed in order to specify abstract compositions of web services as well as their concrete coordination. The basic entity in the component model of BPEL is the *activity*. Activities can be classified as *basic* (simple, that is) and *structured* (composite). The orchestration of web service scenarios involves the possibility of combining services through *sequence*, *switch* (conditional routing), *pick* (non-deterministic choice), *while* and *flow* (parallel) activities. Data management is handled through a blackboard approach, where a set of "global" variable names handles the flow of data among activities. The selection of services involves the introduction of partner link types, i.e., roles that exchange messages and partner links, which are specific services materializing these roles. Exceptions are managed in a try-catch approach.

APPENDIX B: DETAILED INFORMATION ON DEVIATIONS FOR THE DERIVED SOLUTIONS

Table 4, 5 and 6 depict detailed results for the deviation from the optimal values for a large-size configuration.

Table 4. Line-Bus configuration. All solutions for 100 experiments with 8 operations, 3 servers

	1 Mbps Bus		**100 Mbps Bus**	
	$T_{execute}$	Time Penalty	$T_{execute}$	Time Penalty
Fair Load	70%	0.2%	44%	1%
FL-Tie Resolver$_1$	52%	0.2%	40%	1%
FL-Tie Resolver$_2$	48%	0.2%	37%	1%
FL-MergeMsgEnds	37%	34%	39%	32%
HeavyOps-LargeMsgs	17%	19%	42%	2%

Table 5. Line-Bus configuration. Sampling of 32,000 solutions for 50 experiments with 19 operations, 5 servers

	1 Mbps Bus		**100 Mbps Bus**	
	$T_{execute}$	Time Penalty	$T_{execute}$	Time Penalty
Fair Load	75%	0.08%	31%	0%
FL-Tie Resolver$_1$	32%	0.08%	26%	0%
FL-Tie Resolver$_2$	24%	0.08%	25%	0%
FL-MergeMsgEnds	26%	34%	32%	36%
HeavyOps-LargeMsgs	2.9%	12%	29%	0.3%

Table 6. Graph-Bus configuration. Sampling of 32,000 solutions for 50 experiments with 19 operations (65% operational, 35% conditional), 5 servers

	1 Mbps bus		**100 Mbps bus**	
	$T_{execute}$	Time Penalty	$T_{execute}$	Time Penalty
Fair Load	63%	0%	1.7%	0%
FL-Tie Resolver1	56%	0%	1.7%	0%
FL-Tie Resolver2	53%	0%	0.9%	0%
FL-MergeMsgEnds	34%	32%	4.5%	30%
HeavyOps-LargeMsgs	29%	1.8%	0%	0%