Efficient Deployment of Web Service Workflows

Konstantinos Stamkopoulos Evaggelia Pitoura Panos Vassiliadis
Dept. of Computer Science, University of Ioannina, Ioannina, Hellas
{kostamko, pitoura, pvassil}@cs.uoi.gr

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, we assume that a service provider has several servers over which web service operations can be deployed. Then, given a workflow of web services and the topology of the servers, the most efficient mopping of operations to servers must 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. We study different topologies for the workflow structure and the server connectivity and propose a suite of greedy algorithms for each combination.

1. Introduction

A web service is an interface that describes a collection of operations provided through the internet and accessed through standard XML messages [ACKM04]. 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* (specified in appropriate languages such as BPEL or WSFL) that combine intermediate service results towards achieving a more complex goal.

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. 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 simple 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: In Section 2, we start with a formal definition of the problem. In Section 3, we introduce algorithms for the deployment of web service operations at the appropriate servers. In Section 4, we present experimental results and in Section 5, we discuss related work. Finally, in Section 6, we summarize our findings and discuss issues of future research.

2. Problem formulation

In this section, we start with a motivating example to show the nature and importance of the appropriate deployment of web service operations and then move on to formally define the problem.

2.1. 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 Fig. 1. Once the meeting has been conducted, the system registers any prescribed medicines and communicates through operations at social security agencies to register the assignment of medicines to patients. For lack of space, we avoid the detailed description of operations; 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 activities* that regulate which operations are to be invoked depending on the state of the workflow.

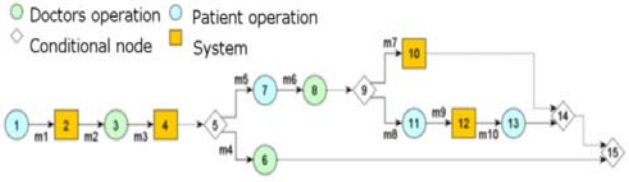

Figure 1. Exemplary workflow.

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.

2.2. Formal definition of the problem

In this subsection, 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.

Assume a finite set of web service operations $\boldsymbol{O} = \{O_1, O_2, ..., O_M\}$ and a finite set of servers $\boldsymbol{S} = \{S_1, S_2, ..., S_N\}$. The term "operation" refers to WSDL operations (i.e., modules that can 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., the output of o_p is used as input to o_n. A workflow is a directed digraph 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 $o_n\}$. Plainly speaking, a workflow is a graph, with operations being the nodes of the graph and XML messages being modelled as the edges of the graph. In the case of web service *operations* (as opposed to web services), we make the reasonable assumption that each pair of operations can be connected through only one message. A network of servers is a graph $\boldsymbol{N(S, L)}$, where $L=\{(s_i, s_j) \mid s_i, s_j \in S, \exists$ connection among server s_i with server $s_j\}$. 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. The latter are the ones performing specific tasks for the workflow, whereas the former control the flow of execution. We consider three types of decision operations/nodes, namely *AND, OR, and XOR*. 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*. Plainly speaking, 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 (like node 5 in Fig. 1), the semantics are as follows: (a) *AND* nodes involve the execution of all their outgoing paths with a rendezvous 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.

Assume a cost model $\boldsymbol{Cost(W)}$ 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 a set of servers* $\boldsymbol{S}$*, such that the operational cost is minimized (and the constraints* $\boldsymbol{C}$ *are met).* Formally, the mapping is modeled as a finite set $\boldsymbol{Mapping} = \{r_1, r_2, ..., r_M \mid \forall i=1,2,...,M: r_i$ a rule of the form $o \rightarrow s, o \in O$ and $s \in S\}$ and the goal is to find the mapping with the minimal $\boldsymbol{Cost(W)}$ that respects $\boldsymbol{C}$.

3. Proposed Algorithms

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

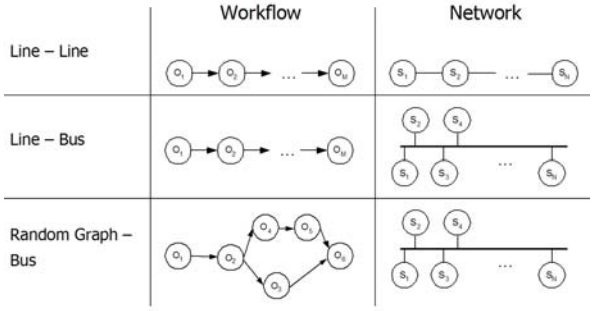

Figure 2. Examined configurations.

We have experimented with different types of workflow and server topologies. Regarding the topology of the workflow, we have considered linear and random graph topologies. The network of servers forms either a linear topology (mainly for initial experimental reasons) or a bus topology. In Fig. 2, we depict the combinations that were eventually considered as valid cases. In all our deliberations, we assume *N* servers and *M* operations. For lack of space, we provide informal descriptions for the algorithms that are simple to formalize. The formal descriptions are found in [StPV06].

3.1 Exhaustive algorithm

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 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.

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. $T_{trans}(op_i, op_j) = \sum_a \frac{MsgSize(opi, opj)}{Line\_Speed(Sa, Sb)}$, $(s_a,s_b) \in Path(Server(op_i), Server(op_j))$
$T_{proc}(op)$	Processing time of a deployed operation *op*. $T_{proc}(op) = \frac{C(op)}{P(Server(op))}$
$MsgSize(op_i, op_j)$	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. $Load(s) = \sum_j T_{proc}(O_j)$
$T_{comm}(op_i, op_j)$	Assuming $(op_i, op_j) \in E$, the communication time between operations op_i and op_j, $T_{comm}(op_i, op_j) = \sum_a T_{prop}(Sa, Sb) + T_{trans}(op_i, op_j)$, $(s_a,s_b) \in Path(Server(op_i), Server(op_j))$
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 $Time\_Penalty = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\lvert Load(s_i) - Load(s_j) \rvert}{(1/2) \times N \times (N-1)}$
$T_{execute}$	Execution time of workflow ***W***. $T_{execute} = \sum_{j=1}^{M} T_{proc}(O_j) + T_{comm}^{(total)}$

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 a single 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.* To achieve this, it is desirable to allocate as many neighboring operations as possible to each server, provided that the server is not overloaded. By doing so, the fraction of messages sent over each communication line is expected to be reduced. Similarly to the above observation, *minimization of the number of messages exchanged between servers* is also desirable.

3.2 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.

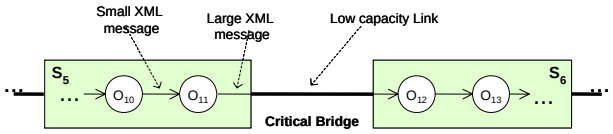

Figure 3. Critical Bridge.

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).

First, the algorithm computes the ideal load per server. Then, it starts assigning the operations of ***W*** to the servers of ***W*** starting from the first operation/server on the left. When a server comes as close as possible to its ideal load, the algorithm considers the next server. 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 a contiguous operation. Fig. 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).

Algorithm *Fair Load – Tie Resolver for Cycles*
Input: a workflow of web service operations $W(O, E)$, with $O = (O_1, O_2, \ldots, O_M)$ and a server configuration $N(S, L)$, with $S = \{S_1, S_2, \ldots, S_N\}$ and L all the combinations of server pairs with the same network costs (bus)
Output: a mapping M of O to S
Begin

$Sum\_Cycles = \sum_{i=1}^{M} C(O_i)$, $Sum\_Capacity = \sum_{i=1}^{N} P(S_i)$

$Ideal\_Cycles(S_i) = Sum\_Cycles \times \frac{P(S_i)}{Sum\_Capacity}$, $i = 1, \ldots, N$

$Servers\_List = (s_1, s_2, \ldots, s_N) = (S_1, S_2, \ldots, S_N)$

$Operations\_List = (o_1, o_2, \ldots, o_M) = (O_1, O_2, \ldots, O_M)$

Sort Servers_List so that $\forall\ i = 1, \ldots, N\text{-}1$ $Ideal\_Cycles(s_i) \geq Ideal\_Cycles(s_{i+1})$

Sort Operations_List so that $\forall i = 1, \ldots, M\text{-}1$ $C(o_i) \geq C(o_{i+1})$

Initialize M *to a random Mapping*

while *Operations_List is not empty do*
  $gain_1 = Gain\_Of\_Operation\_At\_Server(o_1, s_1, M)$
  $i=2$
  while $C(o_1) = C(o_i)$ *and* $i \leq M$ *do{*
    $gain_2 = Gain\_Of\_Operation\_At\_Server(o_i, s_1, M)$
    if $gain_2 > gain_1$ *{*
      $swap(o_1, o_i)$
      $gain_1 = gain_2$ *}*
    i++
  } //end inner while
  $M = M - \{o_1 \rightarrow Server(o_1)\}$
  $M = M \cup \{o_1 \rightarrow s_1\}$
  Delete o_1 *from Operations_List*
  $Ideal\_Cycles(s_1) -= C(o_1)$
  Move s_1 *in Servers_List so that* $\forall i = 1, \ldots, N\text{-}1$: $Ideal\_Cycles(i) \geq Ideal\_Cycles(i+1)$
  Continue with new $Servers\_List = (s_1, s_2, \ldots, s_N)$
} //end outer while
return M
End

Figure 4. Algorithm Fair Load – Tie Resolver for Cycles.

The algorithm *Line-Line* 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.

3.3 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.

Function *Gain_Of _Operation_At_Server*
Begin
gain = 0
if $O_i \in (O_2, O_3, \ldots, O_M)$ *and* $\{O_{i-1} \rightarrow S_j\} \in M$ *then*
    gain += MsgSize(O_{i-1}, O_i)
if $O_i \in (O_1, O_2, \ldots, O_{M-1})$ *and* $\{O_{i+1} \rightarrow S_j\} \in M$ *then*
    gain += MsgSize(O_i, O_{i+1})
return gain
End

Figure 5. Function Gain of Operation at Server

Fair Load. The simplest of all the involved variants is tuned to obtain the best possible load distribution. *Fair Load* 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.

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) operates as *Fair Load* with respect to its basic principle (Fig. 4). 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 (Fig. 5). 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) 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) extends $FLTR_2$ by adding an extra test during the deployment decision. If the assignment of an operation to a server 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.

Heavy Operations – Large Messages. Algorithm *Heavy Operations–Large Messages* 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. 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) 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 $O(M \times logM + N \times logN + MN)$ for *Fair Load*, and $O(M \times (M \times logM + N \times logN + MN))$ for the rest of the algorithms. In the algorithm *Heavy Operations–Large Messages*, *MN* becomes *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. Specifically, 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 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).

4. Experiments

In this section, we present experimental results for the assessment of the proposed algorithms. We mainly focus on the topologies where the network involves a bus; any insights from the experiments of a Line-Line configuration are discussed in the context of the two other cases.

4.1 Experimental methodology

We have varied several parameters of the configurations. We use the results of [HGSL+05] and [NgCG04] to determine appropriate values for our experiments. In [NgCG04], 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 (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 1.67 MHz CPU). Then, we set simple web service operations to 5M cycles, medium operations to 50M cycles and heavy operations to 500 M cycles.

We have grouped our experiments in three classes. In all experiments, we measure the execution time and the load distribution of the workflow. In class **A**, we vary the link capacity and the size of the messages exchanged. In class **B,** we vary the CPU power of the servers and the workload of the workflow. In class **C,** we change all the variables of the problem. Due to lack of space, we only report our findings for class C experiments. Table 6 lists the different values employed in this class of experiments.

To assess the quality of our solutions, we have performed sampling of solutions with configurations with varying number of servers (3-5) and operations (5-19). We report worst case numbers of 50 experiments over a configuration of 5 servers and 19 operations. Each sample involved *32.000* potential solutions over search spaces that spanned from *32.000* to 10^{19} solutions.

4.2 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.

Table 6. Experimental configuration for Class C experiments.

$MsgSize(O_i, O_{i+1})$	*0.006660 Mbits* with probability *25%* *0.057838 Mbits* with probability *50%* *0.163208 Mbits* with probability *25%*
$Line\_Speed(S_i, S_{i+1})$	*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%

In Fig. 6, we depict our results for the Class C experiments. The horizontal axis of each diagram depicts the execution time and the vertical axis the time penalty. The closer a solution is to point (0, 0), the better it is. Assuming different weights for the two measures, different distance measures could also be considered.

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

4.2 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 Fig. 7, we depict the overall performance of our algorithms and in Fig. 8 the detailed results organized per graph structure. 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.

5. Related Work

Related work has quite extensively dealt with similar problems, although we are not aware of any results on the problem of optimal service deployment so far.

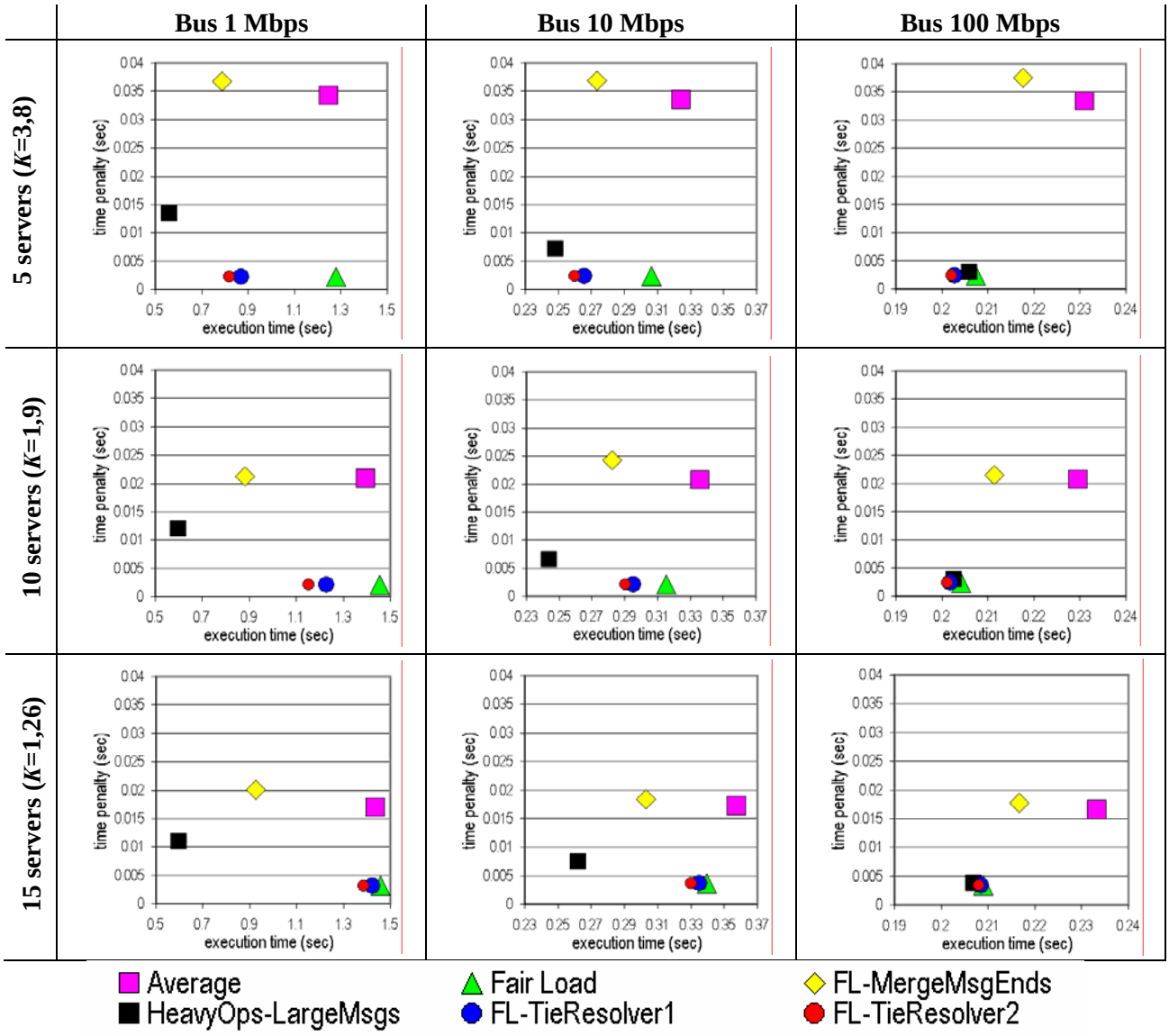

Figure 6. Line – Bus algorithms with 19 operations in the workflow.

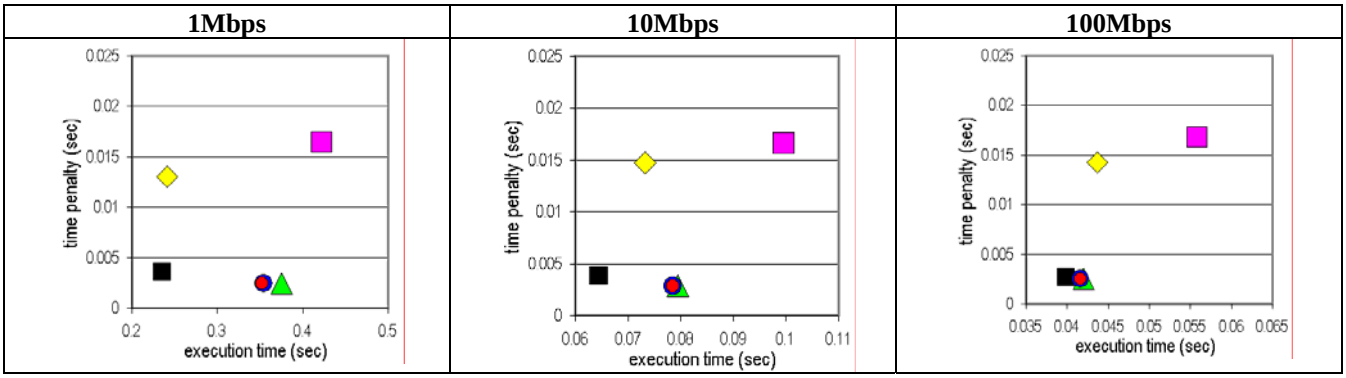

Figure 7. Random Graph – Bus algorithms.

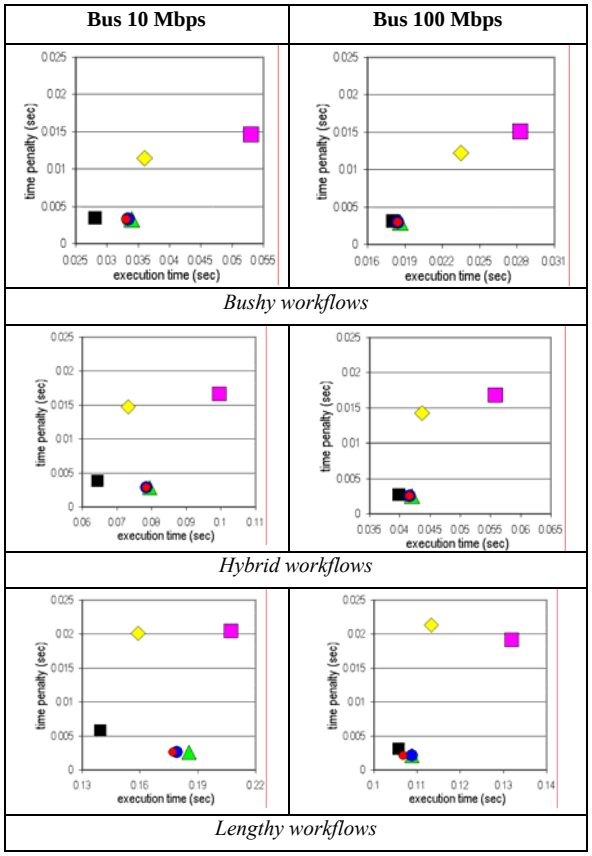

Figure 8. Graph – Bus algorithms organized per graph structure.

[LeWY93] and [LTZS05] deal with the problem of object replication and provide interesting insights on the dimensions of the problem and the gain functions. [CoBF05] and [SWMM05] 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. [CSMA+04], [GiWW02], [ZBD+03] and [SaZh04] consider the problem of achieving QoS properties, still they do not deal with the deployment of service instances (although [GiWW02] makes service replicas, and [ZBD+03] assumes communities of similar operation).

6. Conclusions

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. 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.

Future extensions of this work involve the case of multiple workflows (instead of just a single one). Other extensions involve a 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.

Acknowledgments. 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

[ACKM04] G. Alonso, F. Casati, H. Kuno, V. Machiraju. Web Services Concepts, Architecture and Applications. Sprinter, 2004.

[CoBF05] I. Constantinescu, W. Binder, B. Faltings. Optimally Distributing Interactions Between Composed Semantic Web Services. ESWC 2005: 32-46.

[HGSL+05] M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, M. Lewis. A Benchmark Suite for SOAP-based Communication in Grid Web Services. SC|05, Seatle WA. November 2005.

[LeWY93] A. Leff, J.L. Wolf, P.S. Yu. Replication algorithms in a remote caching architecture. IEEE Transactions on Parallel and Distributed Systems, 4(11), 1185-1204, Nov. 1993.

[LTZS05] N. Laoutaris, O. Telelis, V. Zissimopoulos, I. Stavrakakis. Distributed Selfish Replication. IEEE Transactions on Parallel and Distributed Systems, 2005.

[NgCG04] A. Ng, S. Chen, P. Greenfield. An Evaluation of Contemporary Commercial SOAP Implementations. AWSA2004. Melbourne, Australia, 2004.

[StPV06] K.Stamkopoulos, E. Pitoura, P. Vassiliadis. Efficient Deployment of Web Service Workflows (long version), 2006. Available at http://www.cs.uoi.gr/~pvassil/publications/2007_SEIW/SEIW07_long.pdf

[SWMM05] U. Srivastava, J. Widom, K. Munagala, R. Motwani. Query Optimization over Web Services. October 2005. Available at: http://dbpubs.stanford.edu:8090/pub/2005-30.