Minimizing Human Interaction Time in Workflows

Christian Hiesinger, Daniel Fischer, Stefan Föll, Klaus Herrmann, Kurt Rothermel
University of Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätstr. 38, D-70569 Stuttgart
{chrisitan.hiesinger,daniel.fischer,stefan.foell,klaus.herrmann,kurt.rothermel}@ipvs.uni-stuttgart.de

***Abstract*—Many business scenarios require humans to interact with workflows. To support humans as unobtrusively as possible in the execution of their activities, it is important to keep the interaction time experienced by humans as low as possible. The time required for such interactions is influenced by two factors: First, by the runtime of the services that are used by a workflow during an interaction. Second, by the time required to transfer data between workflow servers and services that may be distributed in a global network. We propose an algorithm that computes a suitable distribution of a workflow in such a network. The goal of our algorithm is to minimize the time required for interactions between a human and a workflow. Current approaches in the domain of workflow optimization pay little attention towards optimizing a workflow to increase the usability for humans. We show the feasibility of our approach by comparing our algorithm with two non-distributed approaches and a distributed approach which is based on a greedy algorithm and show that our algorithm outperforms these approaches.**

***Index Terms*—Workflow distribution, human interaction, pervasive workflows.**

I. Introduction

By using workflows, organizations are able to automate and optimize their business processes [1]. In many business scenarios, activities have to be executed by humans. Therefore, it is important to integrate humans into workflows.

Such integration can adhere to different patterns. In the simplest of cases, a human is notified by the workflow system about currently available activities and provides some sort of feedback when he has completed them. However, more complex interaction patterns may require a human to query the workflow system for information throughout the execution of the activities. The time needed to provide the desired information is experienced by the human as waiting time. An important design principle from the area of Pervasive Computing is to support humans as unobtrusively as possible [2], [3]. Therefore, applying workflows in this area requires that such waiting times are minimized.

The time is dependent on two factors: First, on the runtime of services that need to be executed in order to provide a human with the desired information. Second, on the time required to transfer data between workflow servers and service hosts participating in the execution of a workflow. New technologies like Cloud Computing support organizations in focusing on their core business [4] and lead to the necessity of using remote service providers within workflows. However, communicating data to remote networks may create extensive waiting times due to limited bandwidth and significant propagation delay.

In this paper, we propose an algorithm for distributing a workflow over a set of workflow servers such that the interaction time experienced by humans is minimized. Existing approaches for workflow optimization do not take this factor into account [5], [6], [7]. Our algorithm is based on a two-phase list-scheduling approach. In phase 1, an initial distribution is computed that is based on estimated values for activity execution and data transfer times. In phase 2, the initial solution is refined based on a hill-climbing algorithm. We show that our algorithm improves the interaction time between humans and workflows by up to 80% compared to an approach in which the complete workflow is run on a single machine. Furthermore, we report an improvement of up to 10% compared to an existing greedy approach. We also show that our algorithm scales better with an increasing number of tasks compared to this approach.

The remainder of this paper is organized as follows. In Section II, we introduce our system model and define the problem of workflow placement in a formal way. Thereafter, in Section III, we discuss related work in the area of workflow placement. In Section IV, we describe our placement algorithm that we evaluate in Section V. Finally, we conclude the paper and give some outlook on future work in Section VI.

II. System Model and Problem Description

In this section, we describe our system model in a formal way. Our goal is to distribute a workflow among local networks (*domains*) administered by various service providers which are connected to each other via a global network. We split our system model into a network model and a workflow model. Thereafter, we formalize our goal of minimizing the time to execute so called *human interaction patterns* as an optimization problem.

A. Network Model

We assume a set of domains D, each representing a local network consisting of a set of hosts. A domain $d \in D$ provides a set of *service types* $\mathcal{S}_d$. Services of the same type may be available (replicated) in different domains and $\mathcal{S} = \bigcup_{\forall d \in D} \mathcal{S}_d$. Note that we model the functionality a human h provides as a special service $s_h \in \mathcal{S}$. We assume a workflow server and a *domain controller* in each domain. The

This research has been supported by FP7 EU-FET project ALLOW (contract number 213339).

domain controller serves as an information service. It provides the link properties of all relevant communication links and a service discovery mechanism. This can be achieved by means of an overlay network between all domain controllers in the network. Services, workflow server and domain controller may be replicated inside a domain to allow for provisioning of quality of service guarantees, but for simplicity we treat each of them as being unique within the respective domain.

We assume that each domain is able to communicate to any other domain via the Internet. We denote the bandwidth and propagation delay between two arbitrary domains $d_1, d_2 \in D$ as $\beta(d_1, d_2)$ and $\delta(d_1, d_2)$, respectively. The bandwidth and propagation-delay between two hosts in the same domain is assumed to be constant and denoted as $\beta(d, d)$ and $\delta(d, d)$, respectively. We assume that $\forall d \in D$, $\forall d_i, d_j \in D$, $d_i \neq d_j$: $\beta(d, d) \gg \beta(d_i, d_j) \wedge \delta(d, d) \ll \delta(d_i, d_j)$, i.e. inter-domain delay dominates intra-domain-delay which reflects typical communication properties found in interconnected LANs.

B. Workflow model

A workflow is a directed acyclic graph $W = (A, s, C, \rho, \theta_A, \theta_D)$. A denotes the set of activities in the workflow. The functionality of an activity is defined by means of the function $s : A \rightarrow \mathcal{S}$ which maps an activity $a \in A$ to a required service type $s \in \mathcal{S}$.

The control flow is specified by means of the set $C \subset A \times A$ and defines the logical order of activities. We refer to activities that model conditional or parallel behaviour as *structural activities*. A conditional and parallel split is modeled as an activity with more than one outgoing control flow link. The set of outgoing control flow links of an activity $a \in A$ is denoted as C_a. For a given control flow link $c = (a_i, a_j)$, ρ_c is the probability that a_j will be executed if a_i has been executed. This value can be derived from execution traces of the workflow. For a conditional split, the workflow is executed following only a single alternative, i.e. the conditions $|C_a| > 1$ and $\sum_{c \in C_a} \rho(c) = 1.0$ hold. For a parallel split, all outgoing branches are executed in parallel, i.e. the conditions $|C_a| > 1$ and $\forall c \in C_a : \rho_c = 1$ hold. The latter also holds for all other links originating from a non-structural activity.

The average amount of data that needs to be transferred from a workflow server executing an activity $a \in A$ to a service required by a is denoted as $\theta_S(a)$. We assume that the values of $\theta_S(a)$ cover the amount of data required for the input parameters as well as for the result of the respective service call. Similarly, $\theta_A(a_1, a_2)$ specifies the average amount of data that has to be exchanged between two activities $a_1, a_2 \in A$. We assume that the functions θ_A and θ_S are defined either by means of estimations by a workflow designer or by learning them from past executions of the workflow. In the following, we refer to communication relationships between activities as well as between an activity and a service as *data links* and subsume them in the sets $L_{\text{AA}} \subset A \times A$ and $L_{\text{AS}} \subset A \times S$, respectively.

A *Human Interaction Pattern* (HIP) is a connected subgraph of a workflow. It starts with a single entry activity which expects input data from a human and ends with a single exit activity which generates output data for the same human. This is a natural assumption as we focus on interaction patterns that resemble queries. An example for a HIP is given in Figure 1. Arrows show control flow links while circles and rectangles represent activities and services, respectively. The single entry activity is a_1 (a conditional split). The single exit activity is a_4. Note that there may be several HIPs in a single workflow.

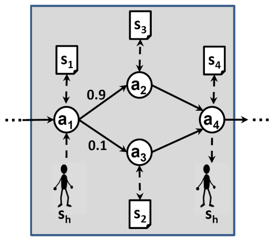


Fig. 1. Human Interaction Pattern

C. Problem description

Our goal is to find a mapping function $\mu : A \rightarrow D$ of activities to domains that minimizes the average required communication time for all HIPs in a workflow. We focus only on activities that are part of a HIP. All other activities may be distributed according to other optimization goals (e.g. network load). Each execution of the workflow takes only a single *route* through the workflow. A route is a connected subgraph of a workflow that contains all activities visited in one execution. For example, a_1, a_2, a_4 as well as a_1, a_3, a_4 are both valid routes in the HIP shown in Figure 1.

Because we do not know this route in advance, we cannot optimize our mappings for it. Therefore, we solve the most general case and aim for minimizing the expected execution time of a HIP. Let R be the set of all routes of a HIP. The probability for the execution of $r \in R$ can be calculated as $\varrho_r = \prod_{\forall c \in r} \rho_c$. Furthermore, let φ_r^μ be a function that defines the execution time for r under the mapping μ. Then, our goal is to find a mapping μ such that the following sum is minimized:

$$\sum_{\forall r \in R} \varrho_r \cdot \varphi_r^\mu \tag{1}$$

In the following, we describe how φ_r^μ is calculated. In a parallel split, only the branch which results in the largest execution time is relevant for the overall execution time of the flow. We refer to the subgraph of r that contains only this longest branch for every parallel split as the *critical path* of a route. The time to execute a route of a HIP is influenced by three factors: The time κ_A required to transfer data between the activities, the time κ_S required to transfer data between activities and their mapped services, and by the time κ_X required to execute the corresponding services. Thus, we have $\varphi_r^\mu = \kappa_A + \kappa_S + \kappa_X$.

For the computation of κ_A, we have to distinguish two cases. First, two activities that exchange data may be mapped to the same domain and, thus, to the same workflow server according to our system model. In this case, κ_A is negligible because no data has to be sent over the network. Second, two activities may be mapped to different domains. In this case, the time required equals the sum of the propagation delay of the communication link between the respective domains and the time required for transmitting the data on the respective data flow link. Let $L_{crit} \subseteq L_{AA} \cup L_{AS}$ be the set of data links on the critical path of a route r, then

$$\kappa_A = \sum_{\forall (a_i,a_j) \in L_{crit}} \delta(\mu(a_i), \mu(a_j)) + \frac{\theta_A(a_i, a_j)}{\beta(\mu(a_i), \mu(a_j))}. \quad (2)$$

For the computation of κ_S, we proceed analogously. Given an activity a placed in domain d, let $\xi(a)$ be a function that returns the domain, among all domains that provide an instance of service type $s(a)$, which can be accessed with lowest interaction time (ideally, $\mu(a) = \xi(a)$). Then,

$$\kappa_S = \sum_{\forall (a,s) \in L_{\text{crit}}} \delta(\mu(a), \xi(a)) + \frac{\theta_S(a)}{\beta(\mu(a), \xi(a))}. \quad (3)$$

We assume that service providers guarantee a certain execution time as part of a SLA. For the computation of κ_X, we accumulate the expected runtime required for the services mapped to the activities on the critical path.

Our problem is a generalization of the problem of task allocation in heterogeneous distributed systems (TAHDS) which is known to be NP-hard [8].[1] Therefore, we propose to use a heuristic algorithm to solve the problem because an exhaustive search of an optimal placement for example by means of backtracking is not feasible.

III. Related Work

Bauer and Dadam [6] propose an algorithm for assigning workflow activities to servers in order to reduce network load. They introduce a cost model based on estimated execution data and employ a greedy algorithm. First, each activity is greedily placed on a workflow server that minimizes the cost for its execution. Then, a hill-climbing algorithm is used to eliminate data transfers between neighbouring activities which have been placed on different servers. In this approach activities are initially placed without taking their data links into account. Thus, it is unlikely that suitable sets of service providers are found in our scenario. We will show that our heuristic performs better than a version of this algorithm adapted to our problem. We refer to this adapted version as *Greedy* approach.

Son et al. [5] propose an algorithm for minimizing communication cost based on multi-level graph partitioning. A workflow is divided into several fragments. However, their solution assumes homogeneous communication links which is not valid in the Internet.

In parallel computing, tasks have to be assigned to CPUs in order to optimize their execution. Many solutions assume that activities are not depending on each other, which leads to a Bin-Packing problem. Obviously, this assumption does not hold for our problem. More sophisticated approaches apply list scheduling algorithms [8], [9]. We adopted the basic idea of these algorithms while dropping their basic assumption of homogeneous communication links.

In the area of grid computing, list scheduling algorithms are employed under the assumption of heterogeneous network links among loosely coupled computing systems [7], [10]. However, these algorithms assume variable task execution times to have a major influence on the overall execution of the task graph. In our scenario, we assume that tasks are services and that quality of service guarantees specify the time required for their execution. Hence, in our case the overall execution time mainly depends on the communication links between workflow servers, rendering respective algorithms like e.g. HEFT inappropriate.

IV. Heuristic Placement Algorithm

We propose a 2-phase algorithm based on a list-scheduling approach in order to find a mapping μ that minimizes the runtime of HIPs. Since HIPs are independent of each other, we map each HIP separately.

As soon as activity a is mapped to domain d, activities with a communication dependency to a have to communicate with d. Thus, they should not be assigned to the best domains in a greedy fashion. Instead, we have to take this dependency into account and map each activity depending on its influence on the runtime of a HIP: The more influence it has on the runtime, the earlier it is mapped.

According to our system model, the bandwidth and propagation delay between domains vary and the time required for communication depends on the network link used. Additionally, there may be services which are available in many domains while other services are only available in few domains. Therefore, it is not known which data link is the most expensive (in terms of communication time) until all activities have been mapped. Therefore, we use a heuristic to estimate the cost of each data link before the actual mapping. Since HIPs are independent of each other, we run the algorithm for each HIP in a workflow as soon as it is known from which domain the human accesses the workflow.

In a first phase, we assign weights to the data links to reflect their estimated costs. Then, we sort the links in descending order of their weights to ensure expensive activities are mapped to domains first. To derive an initial mapping, we iterate over the sorted list and map the activities to domains such that their execution time is minimized. The final mapping is created by optimizing the initial mapping through hill-climbing. The overall algorithm (called *Link Weight Activity Assignment* (LWAA)) is depicted in Listing 1.

A. Weighting and Ordering

The weight of a data link l for the initial mapping is calculated by virtually placing l on each of the possible network links between any pair of domains and by calculating the average time consumed over all these virtual mappings.

Let $l_{AA} = (a_i, a_j)$ be a data link between two activities and let $l_{AS} = (a, s(a))$ be a data link between an activity and its required service. We distinguish between the average time

[1] see Appendix for proof of problem complexity

Listing 1 LWAA Algorithm

```
1: // Let $\hat{\mu}$ be the associative array that represents $\mu$
2: // At the beginning $\forall a : \hat{\mu}(a) = \perp$ holds
3: $L_{\text{DF}} = weightDataLinks(L_{\text{AA}} \cup L_{\text{AS}})$
4: /* Initial mapping */
5: while $L_{\text{DF}} \neq \{\}$ do
6:     $l$ = Link with highest weight in $L_{\text{DF}}$
7:     if $l \in L_{\text{AA}}$ then
8:         $handleActivityToActivityDataLink(l, L_{\text{DF}}, \hat{\mu})$
9:     else if $l \in L_{\text{AS}}$ then
10:        $handleActivityToServiceDataLink(l, L_{\text{DF}}, \hat{\mu})$
11:    end if
12:    $L_{\text{DF}} = L_{\text{DF}} \setminus \{l\}$
13: end while
14: /* Optimize mapping */
15: $hillClimbing()$
```

$weight_W(l_{AA})$ required to communicate between workflow servers and the average time $weight_S(l_{AS})$ required to access a service from a workflow server that controls the corresponding activity. To compute $weight_W(l_{AA})$, we consider every possible mapping of two activities a_i and a_j:

$$weight_W(l_{AA}) = \frac{1}{|D|^2} \sum_{\forall d_k, d_l \in D : k \neq l} \frac{\theta_A(a_i, a_j)}{\beta(d_k, d_l)} + \delta(d_k, d_l) \tag{4}$$

Note that if both activities are mapped to the same domain, no data needs to be transferred.

Similarly, we compute an estimate of the delay created by service data links. In this case, we consider all possible mappings and calculate the average transmission time:

$$weight_S(l_{AS}) = \frac{1}{|D|} \sum_{\forall d \in D} \frac{\theta_S(a)}{\beta(d, \xi(a))} + \delta(d, \xi(a)) \tag{5}$$

The resulting list of data links is sorted in descending order.

B. Initial mapping

For the initial mapping of each activity to a domain, the algorithm proceeds through the list of data links, in descending order of their weights and maps each activity that has not already been processed. Links connecting two activities (L_{AA}) and links connecting an activity to a service (L_{AS}) are handled differently (cf. Listing 1 lines 8 and 10).

Listing 2 shows the handler procedure for links $(a, s) \in L_{AS}$. This handler finds the domain d that exhibits the least cost for calling service s residing in d when placing activity a in d. Listing 3 shows how to handle a data link $(a, a) \in L_{AA}$. We aim at placing both activities in the same domain such that no data has to be transferred over the network. However, we also do not want to reduce the degree of freedom for the placement more than required. We have to distinguish three different cases: 1) None of the activities is mapped 2) Only one of the activities is mapped 3) Both activities are mapped.

Listing 2 Handle activity to service data link

```
procedure    $handleActivityToServiceDataLink(l, L_{DF}, \hat{\mu})$

1: $(a, s) := l$ //get corresponding service and activity
2: $\hat{\mu}(a) := MinArg_{d \in D : s \in \mathcal{S}_d} \delta(d, d) + \frac{\theta_S(a)}{\beta(d, d))}$
```

Listing 3 Handle activity to activity data link

```
procedure    $handleActivityToActivityDataLink(l, L_{DF}, \hat{\mu})$

1: $(a_i, a_j) := l$ //get activities the data link connects
2: if $(\hat{\mu}(a_i) = \perp) \wedge (\hat{\mu}(a_j) = \perp)$ // Both activities unmapped then
3:     if $\exists d \in D : s(a_i) \in \mathcal{S}_d \wedge s(a_j) \in \mathcal{S}_d$ then
4:         $a' := Merge(a_i, a_j)$
5:         // Sorted insert of new service data link
6:         $L_{\text{DF}} := L_{\text{DF}} \cup \{(a', s(a'))\}$
7:         // remove service links of $a_i$ and $a_j$
8:         $L_{\text{DF}} := L_{\text{DF}} \setminus \{(a_i, s(a_i)), (a_j, s(a_j))\}$
9:     end if
10: else if $(\hat{\mu}(a_i) \neq \perp) \wedge (\hat{\mu}(a_j) = \perp)$ //First activity mapped then
11:     if $s(a_j) \in \mathcal{S}_{\hat{\mu}(a_i)}$ then
12:         $\hat{\mu}(a_j) := \hat{\mu}(a_i)$
13:     end if
14: else if $\hat{\mu}(a_i) = \perp) \wedge (\hat{\mu}(a_j) \neq \perp)$ //Second act. mapped then
15:     if $s(a_i) \in \mathcal{S}_{\hat{\mu}(a_j)}$ then
16:         $\hat{\mu}(a_i) := \hat{\mu}(a_j)$
17:     end if
18: end if
19: // If both activities are mapped nothing has to be done.
```

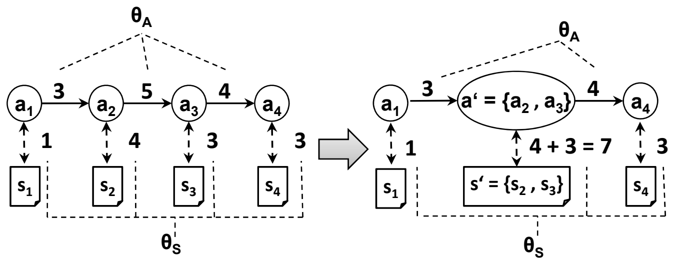


Fig. 2. Merging of unassigned activities

The first case is handled in lines $2-9$ of Listing 3: we check if there exists a domain hosting both service types required by the activities. If such a domain exists, we merge both activities.

The procedure of merging is depicted in Figure 2. The result of merging two activities $a_i, a_j \in A$ is a new activity a' with a corresponding data link to a virtual service $s(a') = s'$ which serves as a container for both $s(a_i)$ and $s(a_j)$. Each operation performed on s' has to be performed for all services in s'. This is illustrated in Figure 2. a_2 and a_3 are merged into a new activity a' with a data link to service type $s' = \{s_2, s_3\}$. The weight of the newly created data link is the sum of the weights of the original links. All other links remain unchanged. Note, that we do not map the merged activities to a domain right away as it may be merged with further activities.

We only merge if there exists a domain hosting the services required by *both* activities because the service access of a_2 and a_3 needs to be restricted to their own domain in order to save communication time. Using the workflow in Figure 2 (left), we explain the rationale behind this idea. Assume that $D = \{d_1, d_2\}$ with $S_{d_1} = \{s_1, s_2\}$ and $S_{d_2} = \{s_3, s_4\}$ and none of the activities is currently assigned to any domain. According to the ranking of data links, the link between (a_2, a_3) has to be processed first. If created a merger $a' = \{a_2, a_3\}$, we would have to map a' either to d_1 or d_2. Thus, either (a_2, s_2) or (a_3, s_3) would be mapped to an inter-domain link

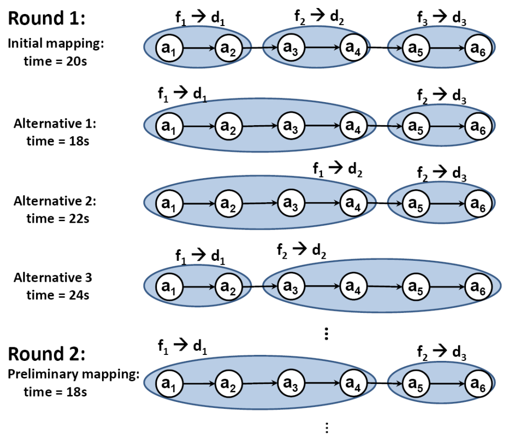

Round 1:
Initial mapping: time = 20s
$f_1 \rightarrow d_1$
$f_2 \rightarrow d_2$
$f_3 \rightarrow d_3$
Alternative 1: time = 18s
Alternative 2: time = 22s
Alternative 3 time = 24s
Round 2:
Preliminary mapping: time = 18s
a_1 a_2 a_3 a_4 a_5 a_6

Fig. 3. Hill-climbing

because there exists no domain hosting s_2 *and* s_3. The data transferred via the link (a_2, a_3) is either the output data of s_2 or the input data for s_3. Consequently, we would omit the time required to transfer the data between both activities. However, we would have to transmit the same amount of data via a communication link in the global network which would require the same amount of time that has been saved. Furthermore, through merging in this case we would limit the degree of freedom as, afterwards, we could not map a_2 and a_3 separately.

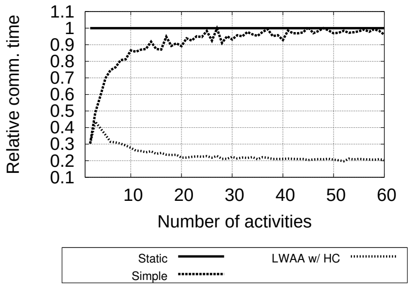
It may happen that only one of both activities is already mapped to a domain. This is covered in lines 10 to 18 of Listing 3. Analogously to the previous case, we map the unmapped activity to the domain of the already mapped activity only if this domain hosts the required service. Finally, it is also possible that both activities are already mapped to a domain. In this case, we do nothing since the currently handled data link must have a lower priority than the data links that led to a mapping of the respective activities to domains.

C. Optimized mapping

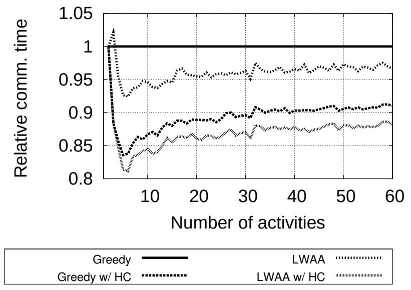
After the initial mapping is completed, we adjust it to the actual bandwidth/propagation delay in the network using a hill climbing algorithm in oder to further reduce the time consumed by transferring data via the global network. The principle of the algorithm is depicted in Figure 3.

First, we extract the clusters of the initial mapping. A cluster $A_F \subset A$ is the largest set of activities that form a connected graph with $\forall a_i, a_j \in A_F : \mu(a_i) = \mu(a_j)$. In Figure 3, there exist three clusters in the initial mapping, namely f_1, f_2 and f_3. We calculate the time required for the HIP if all activities of a cluster are mapped first to the domain of its preceding and then to the domain of its succeeding cluster. The possible alternatives and the time required for each alternative are depicted in rows 2 to 4 of Figure 3. The best alternative is selected as new preliminary mapping. This procedure is repeated until no mapping which requires less time can be found.

We only remap complete clusters because it is unlikely that remapping single activities results in a performance gain. If



(a) Non-partitioned placement



(b) Partitioned placement

Fig. 4. Comparison with other placement approaches

this would be the case the initial mapping would have come to a different conclusion.

V. Evaluation

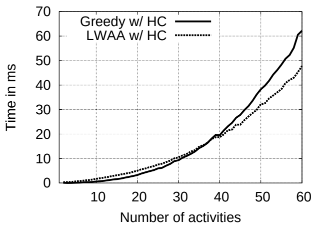
In this section, we describe our evaluation setup and results. We generate networks as well as workflows according to our models discussed in Section II. As the algorithm is executed for each HIP separately, we restrict the generation process to a workflow consisting of a single HIP. The number of activities between the human activities varies between 0 and 60.

We simulate 50 different domains. The bandwidth for communication within each domain is set to 1 GBit/s assuming a Gigabit Ethernet. For the communication between humans and workflow servers, we assume a 54 MBit/s WLAN connection. The bandwidth of communication links between domains is set to be between 2 MBit/s (E1) and 34 MBit/s (E3) to reflect the SLAs between service providers.

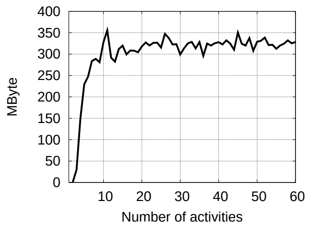
We assumed a uniform delay to simplify our simulation. Since the delay between domains is only influencing the ordering of the weighted data links, this does not change the qualitative results.

We have 200 service replicas drawn from 20 services according to a Zipf distribution. The services are randomly assigned to the 50 domains. We use a Zipf distribution because there may be few very popular services available in many domains, while there are many more specialized services which are only provided in a few domains.

For the generation of workflows we use a grammar that is able to generate sequences, conditional and parallel structures. The rules of the grammar are chosen randomly until the desired number of activities is reached. The values for Θ_S are generated randomly according to a uniform distribution with a maximum of 100 MByte to allow for a wide variety of data flow links. The values for Θ_A are implicitly defined by Θ_S to



(a) Time consumption



(b) Tolerable data amount per migr.

Fig. 5. Performance analysis

guarantee a consistent data flow meaning that all data received by an activity is sent via its outgoing data flow links to other activities. The assignment of activities to (human) services is also chosen uniform randomly.

We compare our algorithm with four other approaches:

- *Static* maps all all activities of a HIP to a random domain.
- *Simple* maps all activities of a HIP to the human's current domain.
- *Greedy* proceeds through all activities, searches for the domain reachable with the highest bandwidth hosting the next required service and maps the activity to this domain.
- *Greedy w/ HC* enhance *Greedy* with a subsequent hill-climbing (cf. Figure 3). This is an adapted version of the algorithm proposed by Bauer et al. [6].

In Figure 4(a), comparison of our distribution algorithm (*LWAA*) with the *non-partitioned approaches* (*Static* and *Simple*) is shown. We compared the relative gain of using our algorithm. The reference (at 1.0) is the *Static*. Figure 4(a) shows that, for an increasing number of activities per HIP, our algorithm quickly converges to around 20% of the time required for *Static*. This is because in the non-partitioned approaches a lot of expensive service calls have to use low quality network links. Thus, they consume a lot of time for communication as only the services that are located in the same domain can be accessed in a performant way.

Figure 4(b) depicts the effectiveness of our algorithm compared to the greedy approaches. The initial placement computed by our algorithm is between 12% and 16% better than the placement computed by the *Greedy* approach. This is due to the fact that our algorithm takes the data flow between activities into account and, thus, computes suitable clusters which is not done in the *Greedy* approach. The *Greedy w/ HC* approach performs better than the *LWAA* algorithm without subsequent hill-climbing. This is due to the fact that clusters are assigned to domains without taking the communication links of the individual domain into account. The results show that our algorithm is better compared to the *Greedy w/ HC* approach by 8% to 10% due to the better initial placement.

We compare our algorithm to the greedy approach in terms of the required computation time in Figure 5(a). Both algorithms show very similar execution times at first, until the effort for the subsequent hill-climbing starts to dominate at around 35 activities. This is because of the fact that *LWAA w/ HC* builds activity clusters, reducing the number of clusters left for the hill-climbing compared to *Greedy w/ HC*.

If the source activity of a data link is mapped to another domain than its target activity, data has to be transferred between workflow servers during the execution of a workflow. This process is called *migration*. The amount of data that has to be transferred in a migration may differ, for example, depending on whether the workflow management system has to transfer additional logging data for compensations. This is not accounted for in our simulation and would impact the performance of our algorithm negatively. Therefore, we measured the amount of data that can be sent additionally for each migration before the *Static* or *Simple* approach outperform our distribution approach. Figure 5(b) shows that this amount can be about three times the maximum amount of data that occurs in the data flow, indicating that considerable migration overhead can be tolerated by our algorithm.

VI. Conclusion

We proposed an algorithm that minimizes human interaction time in workflow systems based on a list-scheduling approach for mapping activities to network domains. We compared our algorithm *LWAA w/ HC* to non-partitioned and greedy approaches and showed that it improves interaction time by up to 80%. Hence, *LWAA w/ HC* reduces the interaction time of humans with workflows significantly and, thus, increases the processing throughput considerably. This can result in competitive advantages in a business environment. Additionally, it helps opening areas like pervasive computing for workflow technologies since it renders workflow technology less obtrusive.

In our future work, we will investigate how workflow distribution can help minimizing the energy consumption of mobile devices used for interacting with the workflow. In this case, executing a partial workflow on such a device avoids the energy-intensive transfer of data to the infrastructure.

References

[1] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Prentice Hall International, 1999.

[2] M. Weiser, "The computer for the 21st century," in *Scientific American 265(3): 94-104*, 1991.

[3] S. Schuhmann, K. Herrmann, and K. Rothermel, "A Framework for Adapting the Distribution of Automatic Application Configuration," in *Proc. of the 2008 ACM Int. Conference on Pervasive Services (ICPS 2008), Sorrento, Italy, July 6-10, 2008*. ACM, Juli 2008, Konferenz-Beitrag, pp. 163–172.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California at Berkeley, Tech. Rep., February 2009.

[5] J. H. Son, S. K. Oh, K. H. Choi, Y. J. Lee, and M. H. Kim, "GM-WTA: an efficient workflow task allocation method in a distributed execution environment," *J. Syst. Softw.*, vol. 67, no. 3, pp. 165–179, 2003.

[6] T. Bauer and P. Dadam, "Efficient distributed workflow management based on variable server assignments," *Lecture Notes in Computer Science*, vol. 1789/2000, pp. 94–109, 2000.

[7] S. H. H. Topcuoglu and W. M. You, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel Distributed Systems*, vol. 13, pp. 260–274, 2002.

[8] Y. Kopidakis, "On the task assignment problem: two new efficient heuristic algorithms," *Journal of Parallel and Distributed Computing*, vol. 42, no. 9, pp. 21–29, 1997.

[9] A. Billionnet, M. C. Costa, and A. Sutter, “An efficient algorithm for a task allocation problem,” *J. ACM*, vol. 39, no. 3, pp. 502–518, 1992.
[10] A. Radulescu and A. J. C. Van Gemund, “Fast and effective task scheduling in heterogeneous systems,” in *Proc. of the 9th Heterogeneous Computing Workshop*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 229–238.

APPENDIX
PROBLEM COMPLEXITY

Our problem is a generalization of the problem of task allocation in heterogeneous distributed systems (TAHDS) which is known to be NP-hard [8]. In TAHDS, we have a set of processors P and a set of tasks T. Two arbitrary tasks $i, j \in T$ have communication costs c_{ij}, and e_{ip} represents the cost of executing task i on processor p. The problem is to find a mapping of tasks to processors such that the sum of communication and execution costs is minimized. Note that communication costs between two tasks only occur if they are placed on different processors.

In the following, we reduce TAHDS to our problem in order to show that our problem is NP-hard as well. We map T to A and P to D, i.e. each task corresponds to an activity and each processor to a domain. We define a unique service for each activity and replicate it on every domain. We set the propagation delay between and within domains to zero. Furthermore, we set the bandwidth within each domain to ∞, i.e. hosts within a domain can communicate instantly. The bandwidth between domains is set to a constant β_{const}. The time to execute service replica $s = s(a)$ running on domain d is set to e_{ip} where a is the activity corresponding to task i and d the domain corresponding to processor p. Θ_A is chosen such that $\Theta_A(a_i, a_j)/\beta_{const} = c_{ij}$ where tasks i and j correspond to activities a_i and a_j, respectively. Obviously, an algorithm that is capable to solve our problem is also able to solve TAHDS and hence, our problem is NP-hard. Therefore, we propose to use a heuristic algorithm to solve the problem because an extensive search of an optimal placement for example by means of backtracking is not feasible.