Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2305

Energy-efficient Distribution of Workflow-based Mobile Applications

Jörg Belz

Course of Study:	Computer Science
Examiner:	Prof. Dr. K. Rothermel
Supervisor:	Dipl.-Inf. Stefan Föll
Commenced:	November 1, 2010
Completed:	May 3, 2011
CR-Classification:	C.2.4, H.4.1

Contents

List of Figures

1 Introduction

1.1 Workflows

Traditionally, most organizations have been organized by function, such as purchasing, sales, shipping and accounting. Generally, these functions are not stand-alone units, but are part of a larger process, for example selling an item to a customer. These processes can be automated by workflow management systems [Deh03] [SM01]. The field of understanding and streamlining workflow models models originated from business organization [BGW09], but the very same concepts can be applied to any distributed system. In this paper, we focus on distributed systems found in pervasive computing scenarios where human users interact with infrastructure services within a workflow environment. With mobile devices in place, power consumption is a crucial matter and we present methods to optimize the workflow execution with respect to the power consumed. For example, a workflow in pervasive computing may consist of the following steps:

1. The infrastructure determines a suitable worker for the task
2. The worker is notified by the infrastructure
3. The worker takes a picture
4. The pictures need to be diagnosed.
5. Based on the diagnose, the user gets different instructions and the next steps are determined

Figure 1.1 displays these steps in a graph. We call the steps activities. Each activity represents a task that is part of a high level process, in this case examining an object. The activities call services that carry out the task. The services are executed on a host, in our scenario this could be the device or the infrastructure. The order in which they are executed during the process is not arbitrary because activities may depend on each other, e.g. a picture cannot be diagnosed before it was taken. We call this order *control flow*, and the information they exchange *data flow*. Together, the control and data flow form a *workflow model* (or simply workflow).

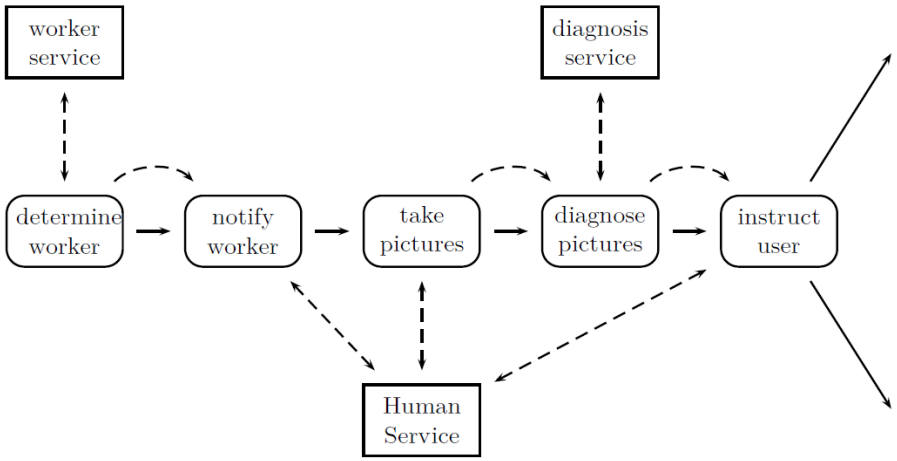

Figure 1.1: Workflow Example

1.2 Problem Description

1.2.1 Power Consumption

Power consumption is a crucial matter in scenarios involving mobile devices because applications that cause a substantial drain of the mobile device's power source induce costs in form of time necessary to recharge the battery or of carrying an additional battery. Accordingly, our goal is to make the workflow process as energy efficient as possible. For our model, we assume that the energy necessary for managing the control flow is neglectible to the optimization, and focus on the energy consumed by the workflow entities themselves. Execution of services on the infrastructure does not induce power drain on the mobile device. This leaves two possible actions that have an impact on consumption during the workflow execution:

- Transferring data over the wifi network between the device and the infrastructure
- Executing services on the mobile device

The energy required for transferring data depends on the type of network used. Transfering data using a GSM/3G-based network for example is considerably more energy draining than a transfer on an IEEE 802.11 network. Several energy models have been proposed [SSM][BBV09][RZ07]. However, the properties of an energy model highly depend on the device used [ZTQ+10]. This applies in particular to cellular networks [RZ07], which makes the models difficult to compare.

1.2.2 Activity Distribution

We assume there exist three type of services, which differ on which host $\Psi = \{IS, H\}$ they can be executed.

1. Infrastructure-only services that can be executed only on the infrastructure host IS (for example queries that require access to a database on a server in the infrastructure)
2. Device-only services that can be executed only on the device H (for example user input or geo location services). We also call these services human services.
3. Services that may be executed on either IS or H (such as route planning). We call them IS/H services.

The latter type of services generate either transfer or execution costs, depending on where they are eventually executed. The energy necessary to transfer the data and the energy required for executing the service on the device are generally different. Furthermore, activities only need to transfer data between each other if they are executed on different hosts. Therefore, the selection of the host where the activity is executed affects the total energy cost of executing the workflow that we would like to minimize. We define a distribution as a function $\mu : A \to \Psi$ that maps each activity to a host. The problem we face is finding a distribution μ_{opt} with the lowest possible energy cost for executing the workflow.

The next chapter provides a more formal description of the system model that we will use. In chapter 3 we introduce an algorithm for distributing activities in a static workflow and extend it so it can be applied to partly finished workflow executions. This makes it necessary to predict part of the instance, therefore we propose prediction methods for the control and data flows in chapter 4. Finally, we compare the performance of the distribution gained by prediction to the actual optimal energy cost in an evaluation (chapter 5).

2 System Model

2.1 Workflow Model

In the definition we use, a workflow model W_{mod} consists of the following elements:

- Activities
- Control flow links
- Data flow links
- Services
- Conditions

We will now describe these elements and how they work together in the workflow:

2.1.1 Activities

Activities are nodes representing tasks to be carried out in the workflow process. We denote the set of activities as A. The activities themselves do not actually execute the tasks they represent themselves, instead they call services. Therefore, each activity $a \in A$ has exactly one service $S(a)$ assigned that it can communicate with using the data flow.

2.1.2 Control Flow Links

The control flow defines the other in which activities are executed. It is modelled by directed control flow links $l = (a, b)\, a, b \in A$ between activities. The start activity designates the start of the workflow execution, and the end activity is the last activity to be executed. To be able to form a wider array of control flows, we allow for conditional split and parallel splits in the workflow. They are initiated by split activities which have two successors and end in join activities with two predecessors. In conditional splits, the successors exclude each other, thus only one of the two branches will be executed. Parallel splits on the other hand imply a parallel execution of the branches. Parallel execution means that both branches can be executed independently according to their individual control flow, but the activity in which they join again can only be executed after both branches finished execution. We use a function $type : a \in A \rightarrow \{A_{normal}, A_{csplit}, A_{cjoin}, A_{psplit}, A_{pjoin}, A_{start}, A_{end}\}$ to distinguish different kinds of activities. The control flow can be defined as a set L_C of control flow links

and a set A of activities, with a function $type$. We use an inductive definition starting with a basic control flow with one activity between the start and end activities:

$$\begin{aligned} L_C &= \{(a,b),(b,c),(c,d)\} \\ A &= \{a,b,c\} \\ type(a) = A_{start}, type(b) &= A_{normal}, type(c) = A_{end} \end{aligned}$$

is a valid control flow.
Then we allow replacing an activity by two activities, increasing the length of the control flow. Figure 2.1 illustrates this replacement of an activity A: Let L_C, A, $type$ be a valid control flow with $(x,y),(y,z) \in L_C$, $type(y) = A_{normal}$. Then

$$\begin{aligned} L'_C &= (L_C \setminus (y,z)) \cup \{(y,u),(u,z)\} \\ A' &= A \cup u \\ type &= type \cup u \mapsto A_{normal} \end{aligned}$$

is also a valid control flow. Finally, we can also replace an activity in L_C, A, $type$ with a conditional split or parallel split structure:

$$\begin{aligned} L'_C &= (L_C \setminus (x,y),(y,z)) \cup \{(x,t),(t,u),(u,w),(t,v),(v,w),(w,z)\} \\ A' &= (A \setminus y) \cup \{t,u,v,w\} \\ type &= type \cup \{t \mapsto A_{csplit}, u \mapsto A_{normal}, v \mapsto A_{normal}, w \mapsto A_{cjoin}\} \end{aligned}$$

or

$$\begin{aligned} L'_C &= (L_C \setminus (x,y),(y,z)) \cup \{(x,t),(t,u),(u,w),(t,v),(v,w),(w,z)\} \\ A' &= (A \setminus y) \cup \{t,u,v,w\} \\ type &= type \cup \{t \mapsto A_{psplit}, u \mapsto A_{normal}, v \mapsto A_{normal}, w \mapsto A_{pjoin}\} \end{aligned}$$

also form valid control flows.

2.1.3 Data Flow Links

The set data flow links L_D describes the data dependencies between activities. If the execution of activity a_i requires data generated during execution of activity a_k, we create a data link from a_k to a_i. Furthermore, we need to add data links from $S(a_k)$ to a_k and from a_i to $S(a_i)$ to model the communication between the activities and their services. Thus, data links always get added in triplets of data links that belong to the same data flow. To distinguish the different data flows an activity can be part of, we assign the same data flow identifier $id \in ID_D$ to each of the three data links belonging to the same data flow.

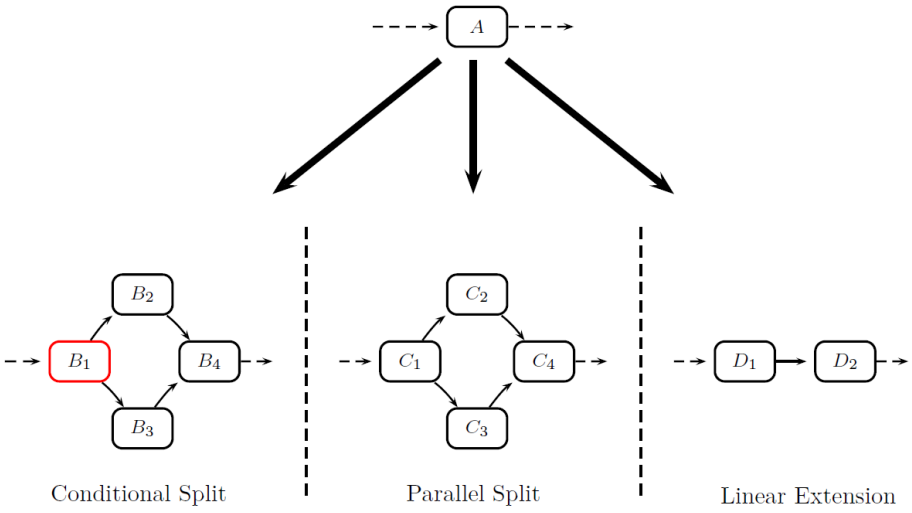

Figure 2.1: Inductive definition of control flow

2.1.4 Services

Services are called by activities to perform an action. The action may require data input and may produce data output, modelled by the data flow. The data output may or may not depend on the input. All services are executed on a host in $\Psi = \{IS, H\}$. In our model, we assume that there exists only one device host H. If there are several different mobile devices in use during the workflow, more than one device node would be required. We do not account for that possibility, however the same principles we present can still be applied. Note that we do not restrict the number of services on each host, thus there may be several different services available on the infrastructure and the device host.

2.1.5 Conditions

Conditions are necessary for conditional splits to determine the successor activity. Conditions can depend only on the data transferred by incoming data edges of the activity.

2.2 Workflow Instance

The workflow model describes the structure of a workflow process. A workflow instance W_{inst} describes a concrete instantiation of this structure. It consists of

- A set of activities $A_{history} \subseteq A$ that represent the activities that have been executed.

- A function $content : l \in L_D \rightarrow \mathbb{N}$ that assigns a numeric value depicting the content of the data flow on edge l.
- A function $size : l \in L_D \rightarrow \mathbb{N}$ that allocates a size in kb to each data edge l.

We assume the service's output is non deterministic. This assumption is reasonable in many cases, for example the user can enter a different input or the database result can differ each time a workflow is executed. Accordingly, for a single workflow model there can exist many different such workflow instances. The following section describes how these instances are generated.

2.2.1 Execution of Workflows

A workflow is executed by executing a set $A_{hist} \in A$ and at the same time building the functions that determine the content and size of data flows. A_{hist} can easily be build by gradually adding the activities on the the control flow path. For each content/value pair of a data edge, we store an object that holds these values. In our execution model, we assume that data is only transferred if both activities participating in the data flow have been executed. Hence, if there is a data link from a servicefunction on H to a servicefunction on IS over activities $a, S(a) = S_H$ and $b, S(b) = S_{IS}$, the data between any nodes (H and a, a and b and b and IS) is transferred at the moment when b is executed, and only if a is in A_{hist}. Therefore, it is necessary to store the data generated by the first service until both activities have been executed. We store the content/value pair of each 'unfinished' data flow in the set $dataedges_{open}$, and the pairs for which both activities have been executed in the set $dataedges_{complete}$.

The activities in A_{hist} and the order they are executed in are determined by the control flow which is managed by a set A_{next} of activities to be executed next. Algorithm 1 outlines how the control flow is processed.

Initially, the set A_{next} only contains the start activity. After an activity has been executed it will be moved to the history and its successor(s) are added to the set A_{next}. Parallel and conditional splits have two successors. In case of parallel splits, both are added to A_{next}. For conditional splits, we determine the successor by evaluating the condition of the conditional split activity. All other activities have exactly one successor, except for the end activity. In line 6 we perform a check if the activity is ready to execute. This is important in the case of parallel joins, because they may only be executed after both predecessors have finished execution.

Algorithm 1: Workflow Execution

```
1  dataedges_complete = ∅
2  dataedges_open = ∅
3  A_next = {a_start}
4  while A_next ⊃ ∅ do
5      foreach a ∈ A_next do
6          if a is ready to execute then
7              foreach incoming data link l from other activity to a do
8                  if there is value/content pair p for l in data_open then
9                      move p to data_complete
10                 end
11             end
12             execute the service S(a) with the incoming data links as input
13             store the resulting content/value pair(s) in data_open
14             if a is conditional split then
15                 evaluate condition and at successor to A_next
16             end
17             else if a is parallel split then
18                 add both successors to A_next
19             else
20                 add successor of a (if any) to A_hist
21             remove a from A_next
22         end
23     end
24 end
```

3 Distribution Algorithms

In general, a distribution algorithm calculates an activity distribution μ for a workflow instance so that the energy cost minimized. In this chapter we describe the distribution algorithm introduced in [FFH] and propose an extension of it which adopts an instance based approach. For the sake of clarity, we call the original distribution algorithm *workflow based distribution algorithm* and our extended algorithm *instance based distribution algorithm*.

3.1 Workflow Based Distribution

As the distribution depends on the data flow edges, it must be given a workflow instance to base the calculation on. For this purpose, the authors of the original distribution algorithm assume a workflow instance W_{inst} based on average data flows of a set of past instances. Therefore, for each workflow model, the algorithm suggests exactly one activity distribution which is applied for all instances.
The distribution algorithm consists of two steps, creating a cost graph and finding a minimum cut in this graph. We will now outline these steps briefly, a more thorough description including an optimality proof can be found in [FFH].

3.1.1 Creation of the Cost Graph

The goal is to find a distribution with minimum costs. To this end, a cost graph is constructed. The cost graph resembles the workflow graph, but with weighted, undirected edges representing energy costs that replace the data edges. Furthermore, it does not have any control edges since these do not represent costs. Let $\Theta(x, y)$ be the total data amount to transfer between two nodes $x, y \; in A \cup \Psi$. $\Theta(x, y)$ can be easily computed by adding the data transfer volumes all data edges between x and y given by the *size* function. Now, to construct the cost graph all parallel data edges of the workflow model need to be transformed into a single, undirected cost edge. This single edge is then weighted with energy necessary to transfer the data between the nodes, given by $E_T(\Theta(x, y))$. Additionally, for all cost edges between an activity a and an infrastructure service that can also be executed on the device with execution cost $E_X(S(a))$, we compare $E_X(S(a))$ to $E_T(\Theta(x, y))$. If the execution cost is lower, the weight is set to the execution cost instead of the transfer cost.

3.1.2 Computation of the Minimum Cut

The actual activity distribution is acquired by computing a minimum cut for each device node. The cuts are calculated between each node $h \in H$ and an artifical node created by merging the the infrastructure node IS and the remaining device nodes $g \in H, g \neq h$. Each of these cut partitions the set of activities into two subsets, the set of activities that is part of the device partition is then the set of activities to be executed on the corresponding device.

3.2 Instance Based Distribution

For the workflow based distribution algorithm it is necessary to predict the instance. The prediction methods generally become more accurate the more information they have about the workflow instance. Therefore, being able to run the distribution algorithm several times during a workflow's execution is likely to improve the distribution with respect to energy costs.

The possible benefit of an instance based prediction is illustrated in figure 3.1. It shows part of a workflow where two conditional split branches meet in the join activity A_3. The numbers on the data edges denote energy costs. The infrastructure service of A_3 can be executed on the device with execution cost 30, the other edges represent energy costs for transferring data. Depending on which branch was chosen, the actual instance may look like either figure 3.2 or 3.3. If we had distributed the workflow prior to the execution, the distribution would assign either H or IS to A_3. Let us assume that A_3 was assigned to the human service. Then, in case A_1 is executed before A_3, a transfer cost of 20 would be necessary to transfer the data from the infrastructure to the device. However, assigning A_3 to the infrastructure in this case would result in 0 cost of the cut between H and IS. But a fixed assignment of $A_3 \rightarrow IS$ is also not always preferable, since if the A_2 branch is chosen, distributing both A_2 and A_3 to the device would be more optimal (cost of 20) than distributing A_3 on the infrastructure (cost of 80).

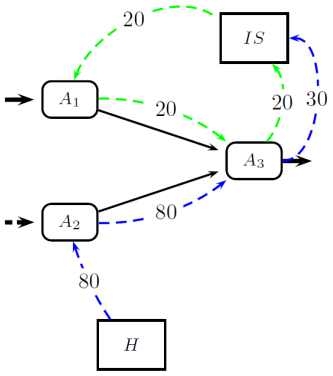

Figure 3.1: Conditional split example

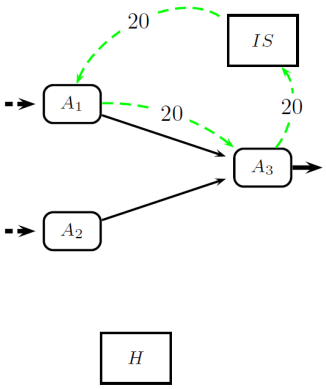

Figure 3.2: A_1 branch chosen in figure 3.1

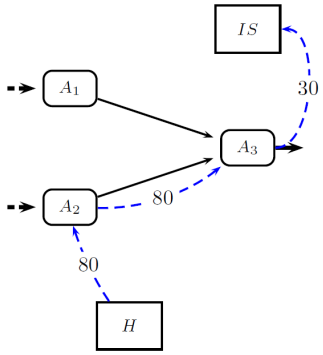

Figure 3.3: A_2 branch chosen in figure 3.1

A static distribution is not always optimal in this example. However, during the workflow execution, using a suitable prediction method it can be determined which branch is chosen before the host that activity A_3 should be executed on needs to be fixed. Therefore, we could always choose the optimal distribution. The distribution algorithm presented in chapter 3 however can only be applied to instances that have not begun execution. If we applied it to a partly finished (and distributed) workflow instance, it would assume that it can freely distribute all activities, whereas the distribution of the activities already executed (A_{hist}) cannot be changed. Hence, the algorithm has to work under the constraint that there already exists a partly defined distribution function $\mu : A \rightarrow \Psi$ and it cannot change these distributions. We account for these distributions by removing the distributed activities from the complete cost graph C (algorithm 2).

However, we cannot simply remove the activities A_{hist} and the attached cost edges. This is because there may be cost edges between activities in A_{hist} and activities that are still to be executed in the instance. Removing these edges would affect the distribution of these activities. Therefore, if an activity a has been distributed to $\mu(a)$, we can only remove those edges that are connected to activities which are also in A_{hist}. The remaining cost edges

Algorithm 2: Distribution of executed activities

```
1  Let C be a cost graph
2  Let A_hist be a set of executed activities and μ an activity distribution
3  foreach a ∈ A_hist do
4      foreach edge e connected to a in C do
5          let v be the node that e connects with a
6          if v is an activity b ∈ A_hist then
7              remove e
8          else if v = μ(a) then
9              remove e
10         else
11             rewire e to connect v and μ(a)
12         end
13     end
14     remove a from C
15 end
```

represent energy costs between the host to which a was distributed and a future activity. Consequently we 'rewire' this edges' endpoint from a to $\mu(a)$.

Example

We assume the workflow given in figure 3.4 where A_1 to A_3 have already been executed, and distributed according to $\mu : \mu(A_1) = H, \mu(A_2) = H, \mu(A_3) = IS$.

To find the resulting graph, we proccess the activities in the order they were executed:

1. A_1 was executed on H. We merge A_1 with H (figure 3.5): The data edges between A_1 and H (colored red) can be removed because A_1 is executed on H and thus no energy cost is induced. The edges between A_1 and other activities (colored green) are changed so that the endpoint A_1 is replaced by H. The resulting graph is shown in figure 3.6.
2. After executing A_2 on H, the edge $IS \to A_1$ is changed to $IS \to H$. This represents the energy costs of the data A_2 which is executed on H needs from the infrastructure service. The edge from A_2 to A_3 is also rewired by replacing its origin A_2 with H, resulting in the graph in figure 3.7.
3. Finally, we execute A_3 on the infrastructure, which creates two additional edges between the infrastructure and the service node. The energy cost of all edges connecting the hosts H and IS directly equals the energy consumed so far in the workflow execution. They can be removed for the distribution algorithm since they don't affect the distribution itself, but only the total cut cost.
4. Now, the resulting graph (figure 3.8) can serve as input for the mincut calculation of the distribution algorithm that distributes the remaining activity A_4.

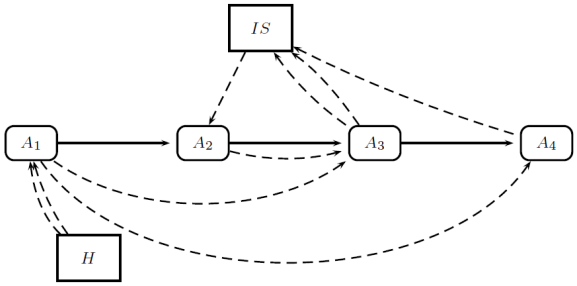

Figure 3.4: Instance based distribution - workflow model

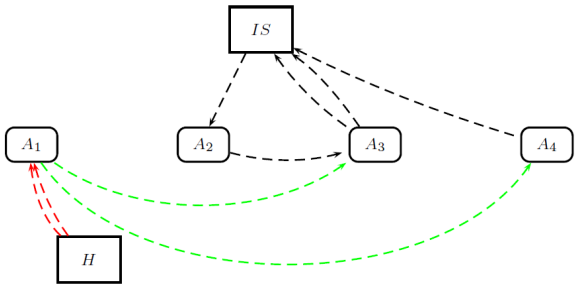

Figure 3.5: Example - Step 1

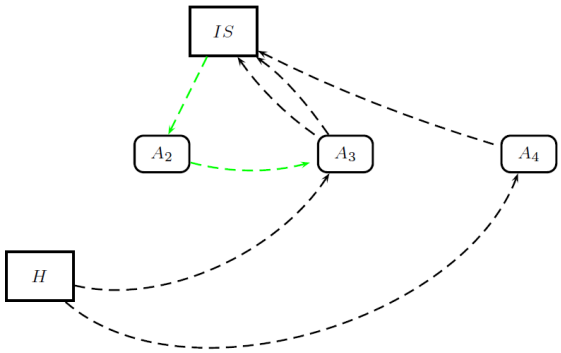

Figure 3.6: Example - Step 2

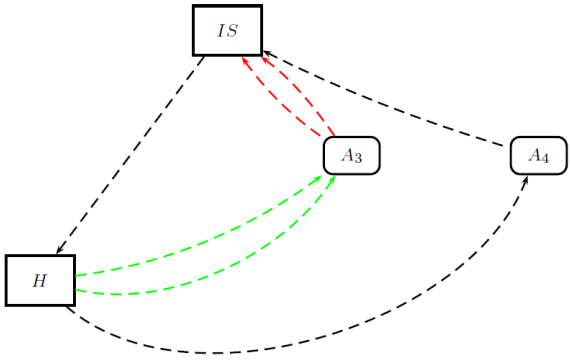

Figure 3.7: Example - Step 3

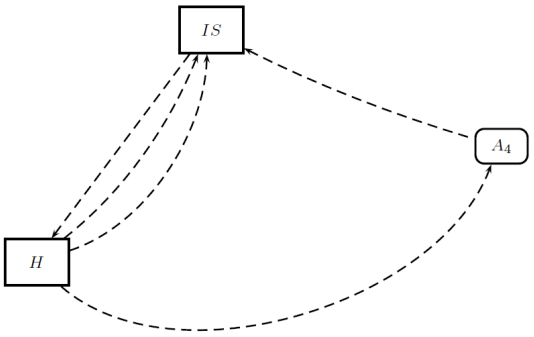

Figure 3.8: Example - Step 4

4 Prediction

If the workflow instance is not known, it needs to be estimated. A workflow instance was defined as a set of activities $A_{history}$ that have been executed in the control flow and as functions *content* and *size* that allocates content and a size to each data flow. Both the control and data flow need to be predicted. A predictor basically transforms a partly finished workflow instance into a finished workflow instance, usually based on the control and data flow of the input instance (figure 4.1). Additionally, the predictor has access to a set T of finished workflow instances. We call this set training set. The predictor can use the control and data flow information of these instances to predict the new instance.

4.1 Prediction Model

We propose a model where the control and data flow is first predicted separately, and then merged (figure 4.2). This makes it possible to combine various methods of prediction.

A control flow predictor is a function that, given a partly finished workflow instance, assigns a probability value to every activity:

$$pre_C : (a \in A, w_{inst} \in W_{inst}) \rightarrow [0, 1]$$

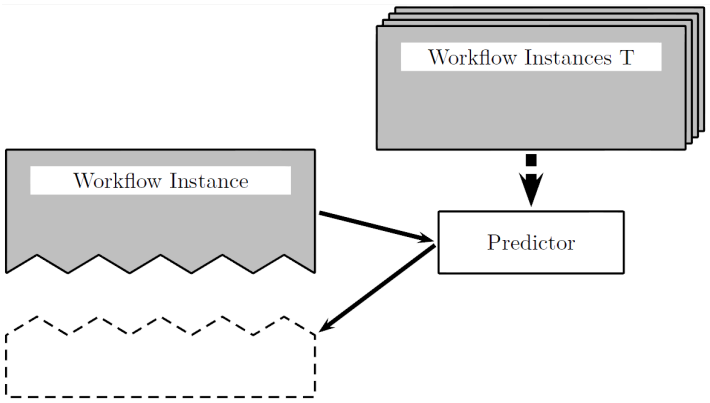

Figure 4.1: Prediction of an instance

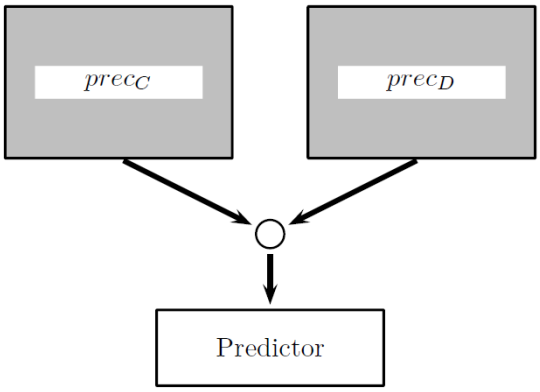

Figure 4.2: Prediction model

A data flow predictor predicts the size of data edges, under the condition that the activities using the edge are executed.

$$pre_D : (e \in L_D, w_{inst} \in W_{inst}) \rightarrow \mathbb{N}$$

Based on these, we generate a predictor

$$pre : (e \in L_D, w_{inst} \in W_{inst}) \rightarrow \mathbb{N}$$

that is based on the values given by pre_D and takes the probabilities pre_C into account. This is necessary because if one activity connected to a data edge is not executed, the data edge will not transfer any data. A straightforward way to achieve this is to weight the data edge by the probability that both activities are executed:

$$pre(e) = pre_D * P(a_i a_k) \quad e = (a_i, a_k) \in L_D,\ a_i, a_k \in A$$

We cannot simply multiply $pre_C(a_i) * pre_C(a_k)$ to get $P(a_i a_k)$ because the probabilities of executing the activities a_i and a_k are not independent. For example, if a_k is the only successor of a_i, in all workflow instances where a_k is executed, a_i will be executed as well, and the probability that a_i and a_k will both be executed is simply $pre_C(a_i)$. Instead, we need to calculate $P(a_k a_i)$ using conditional probabilities:

Let e be a data edge from a_i to a_k. Note that this implies that a_i is preceding a_k in the control flow, because a_k has a dependency on a_i. The probability $P(a_k a_i)$ that both are executed is then given by

$$P(a_k a_i) = P(a_k | a_i)\, pre_C(a_i)$$

This follows from conditional probability. We need to calculate $P(a_k|a_i)$, which is the probability that a_k is executed given a_i has already been executed. This equals the probability

$P_{path}(a_i, a_k)$ that there will be a path from a_i to a_k in the workflow instance assuming that a_i is part of the instance. A path from a_i to a_k is a sequence of activities $(a_i, a_{i+1}, ...a_k)$ on the control flow path . It can be found e.g. by breadth first search of the control flow graph.

Due to conditional and parallel splits in the workflow graph, there may be several possible paths $(a_i, ..., a_k)$ from a_i to a_k. However, we can show that it is sufficient to find only one such path: It is clear that paths may be different only if they contain different branches of parallel or conditional splits. Paths that differ only by a different branch of a parallel split actually represent the same control flow since in a workflow instance, both branches are executed. Therefore, we must not consider them being different paths. That leaves us with paths varying due to conditional splits. These actually represent different workflow executions and therefore, the total probability equals the sum of the individual path probilities. We can simplify the calculation of the sum. Let X and Y be two different paths from a_i to a_k with probabilities p_x and p_y. Let p_{xy} the probability of the sequence of activities that is in both X and Y, and p_{cx} and p_{cy} the probability of the branch exclusive the X respectively exklusive in the Y path. Then $p_x = p_{cx} * p_{xy}$ and $p_y = p_{cy} * p_{xy}$. The sum $p_x + p_y$ then equals $p_{cx} * p_{xy} + p_{cx} * p_{xy} = p_{xy} * (p_x + p_y)$. However, conditional on execution of a, it must hold that $p_x + p_y = 1$ because there are two branches of which one has to be chosen. Thus, $P_{path}(a_i, a_k) = p_{xy}$ with p_{xy} being the probability of the common part of all parts between a_i and a_k.

In order to calculate $P_{path}(a_i, a_k)$, we first purge the distinct elements of any path from a_i to a_k. This is done by removing all branches of conditional splits, for which there exist both a split and a join activity in the path. Afterwards, we calculate $P_{path}(a_i, a_k)$ as follows by dividing the probability of each activity through the probability of its predecessor in the purged path of length n:

$$P_{path}(a_i, a_k) = \prod_{2}^{n} \frac{pre_C(a_j)}{pre_C(a_{j-1})}$$

Example

Figure 4.3 shows part of a workflow, with the numbers denoting the absolute probability pre_C for each activity to be executed. We assume execution is at activity A_1 and we want to calculate $P(A_3A_9) = P(A_9|A_3) * pre_C(A_3)$. $pre_C(A_3)$ is already given (0.8). The probability $P(A_9|A_3)$ equals the probability of any path from A_3 to A_9. It is sufficient to find one path, e.g. $(A_3, A_5, A_6, A_8, A_9)$ and purge this path. The result is shown in figure 4.4.

We can now calculate the probability of this path, which equals the probability that there is any path from A_3 to A_9:

$$P((A_3, A_5, A_8, A_9)) = \frac{0.3}{0.8}\frac{0.8}{0.8}\frac{0.8}{0.8} = 0.375$$

The probability that both A_3 and A_9 are executed conditioned on the knowledge available at A_1 is then $0.8 * 0.375 = 0.3$.

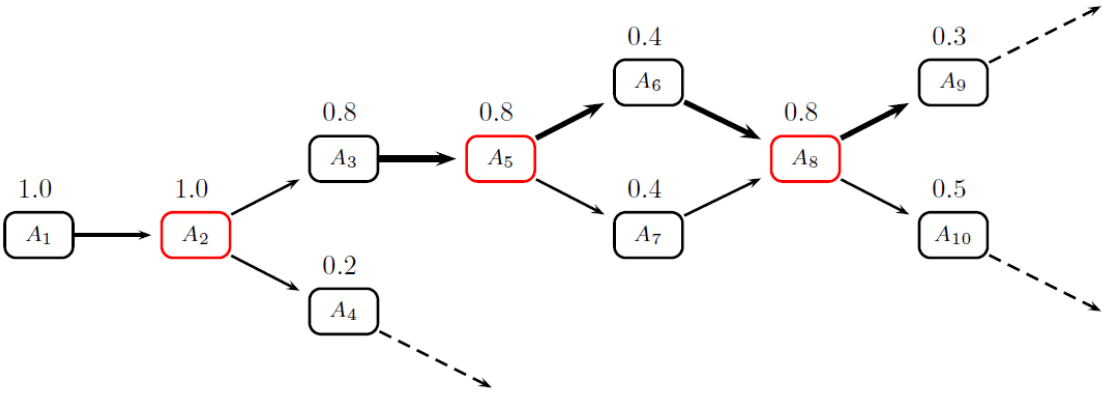

Figure 4.3: Path before purging

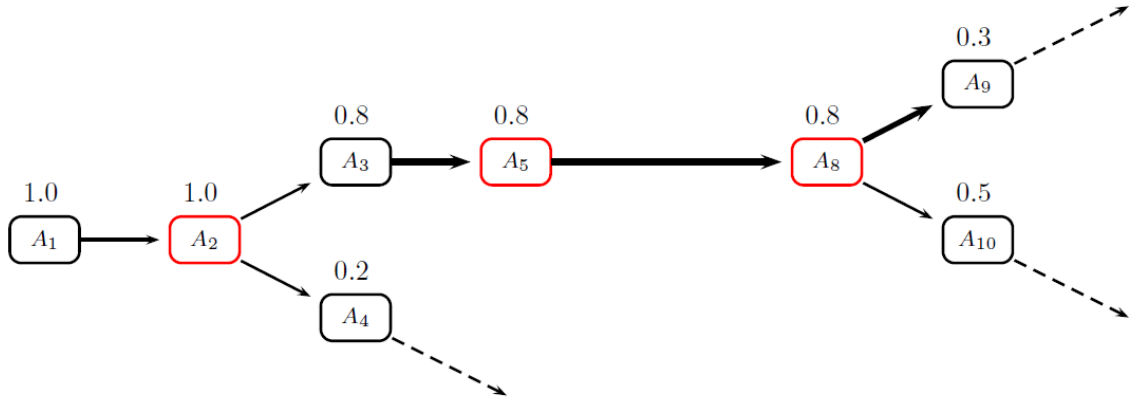

Figure 4.4: Path after purging

4.2 Control Flow Predictors

4.2.1 Conditional Probability Predictor (CPP)

The conditional probability predictor Pre_{ap} uses a training set $T = (t_1, t_2, ..., t_n) t_i \in W_{inst}$ of workflow instances to estimate each activity's execution probability. The unconditioned probability that an activity a is executed is given by

$$p(a) = \frac{\text{number of } t_i \text{ with } a \text{ in history}}{n}$$

If there is nothing known about the instance w_{inst} that is executed, $Pre_{ap}(a, w_{inst}) = p(a)$. However, if the history A_{hist} of the instance is non-empty, the execution probability of these activities equals 1. Furthermore, this affects the probabilities of other activities as well since some control flow paths can now be eliminated. The probabilities of an activity a given by the probability that there is a path from the execution state of w_{inst} to a. This probability can be calculated by finding any such path, purging it and calculating the path probability like in the previous section.

$$Pre_{cpp}(a, w_{inst}) = \begin{cases} 1 & \text{if } a \text{ has been executed in } w_{inst} \\ p(a) & \text{path probability from } w_{inst} \text{ to } a \end{cases}$$

4.2.2 Condition Evaluation Predictor (CEP)

The condition evaluation predictor CEP is based on the conditional probability predictor CPP, but tries to refine its estimations by prematurely evaluating conditional splits, thus determining the successor activity. This is possible when the split's condition depends only on data edges that are already known.

In the following, we develop the condition evaluation predictor which tries to determine the successors of conditionals splits by prematurely evaluating conditions.

Without loss of generality, let b be the actual successor of a conditional split activity a, and c be the successor not chosen by the condition. Based on this knowledge, $p(b)$ and $p(c)$ can be refined: $p(b) = p(a)$, $p(c) = 0$. Additionally, all activities following b and c also have to be updated with the new probabilities: Let B be the set of all activities between (but not including) a and the corresponding join d in the 'b-branch', and C those activities in the 'c-branch'. The new probabilities p of these activities are calculated as follows:

$$p'(x) = \begin{cases} \frac{p(x)}{p(a)} & x \in B \\ 0 & x \in C \end{cases}$$

We do not need to update the join activity or any of its successors. The decision which branch to take does not influence the execution probability of these activities because the execution probability of the join activity always equals the execution probability of the split activity.

4.2.3 Binary-CPP, Binary-CEP and ALL

Since we do not know if weighting the edges' costs actually improves prediction, we introduce variations which weight the costs only by 0 or 1:

$$Pre_{bcpp}(a, w_{inst}) = \begin{cases} 1 & \text{if } Pre_{cpp}(a, w_{inst}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$Pre_{bcep}(a, w_{inst}) = \begin{cases} 1 & \text{if } Pre_{cep}(a, w_{inst}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, to verify that predicting probabilities provides an improvement at all, we define the predictor ALL that assigns the probability 1 to all activities:

$$Pre_{all}(a, w_{inst}) = 1$$

4.3 Data Flow Predictors

4.3.1 Average Volume Predictor

The average volume predictor P_{avp} predicts an unknown data flow's size simply by taking the average value of this data flow from the history, unless the value is already known to the running workflow instance. Let $n(l \in L_D)$ be the number of workflow instances in T in that the data edge l was generated. Then the predictor is defined as

$$P_{avp}(l \in L_D, w_{inst} \in W_{inst}) = \begin{cases} \frac{1}{n(l)} \sum_{t \in T} size_t(l) & \text{if } l \text{ is unknown in } w_{inst} \\ size_{w_{inst}}(l) & \text{if } l \text{ is known in } w_{inst} \end{cases}$$

4.3.2 Linear Regression Predictor

The average volume predictor treats a service function as a black box and merely examines its output. The predictor introduced in this section takes another approach by actually trying to infere the service function's output volume from given input. Hence, this model requires that the output volume somehow depends service function's incoming data volume:

Assumption 1 *Let V_k be the size of the k-th input data flow of a service function s. The output size of s depends linearly on the input volumes:*

$$out_s(V_1, ..., V_m) = \beta_0 + \sum_{k=1}^{m} \beta_k V_k$$

The unknown β_k describe the effect of a one unit increase in V_i on out_s. The estimators $\hat{\beta}_k$ for β_k can be found using e.g. the ordinary least square method or a maximum likelihood estimation [Wooo8]. If not all V_k are known, the model cannot be applied and falls back to the average data volume predictor:

$$P_{reg}(l \in L_D, w_{inst} \in W_{inst}) = \begin{cases} size_{w_{inst}}(l) & \text{if } l \text{ is known in } w_{inst} \\ \hat{\beta}_0 + \sum_{k=1}^{m} \hat{\beta}_k V_k & \text{if all } V_k \text{ are known in } w_{inst} \\ P_{avp}(l, w_{inst}) & \text{otherwise} \end{cases}$$

The restriction to linear relationship between the input and output makes it comparably easy to solve calculate the coefficients $\hat{\beta}_k$[VHMK97]. On the other hand it does not allow to accurately predict service functions that depend on the input values in a strong non-linear manner. However, we merely introduce this predictor as an example of estimating the service function. In real-world applications, knowledge about the specific service function's behaviour can be used. For instance, the service function's output might also depend on the content of the incoming data like the conditions in our system model do.

5 Evaluation

5.1 Workflow Generation

Since we do not have authentic workflow models available for the evaluation, it is necessary to generate them. The workflow generation can be separated into three steps:

1. Generation of the control flow
2. Generation and assignment of services
3. Generation of the data flow

5.1.1 Control Flow

The control flow generation is using a mechanism based on the inductive definition of the control flow (section 2.1.2). It first creates a basic control flow chain with $L_{initial}$ activities and then randomly chooses an activity to replace it with a linear extension, conditional split or parallel split structure. The number of insertions of each structure is given by R_{linear}, $R_{parallel}$ and $R_{conditional}$. We first insert conditional and parallel splits, and afterwards extend the control flow by randomly inserting new activities (figure 5.1). For the conditional split structure, we introduce an additional parameter L_{csplit} which defines the initial length of each branch. This is necessary to ensure the conditional splits have a significant effect on the workflow execution.

5.1.2 Services

After adding activities to the workflow, we need to assign them services they communicate with. We assume there exist three kinds of services: a human service that can only be executed on the device, infrastructure services that can only be executed on the infrastructure and an IS/H service that can be executed on both, with additional execution costs for execution on the device. Each activity is assigned one of these three services with probabilities $P_{IS}, P_H, P_{IS/H}$.

The size of the data flow generated by a service depends on the service type and incoming data flow(s):

$$size_{out} = \gamma_{service} * size_{in} + (1 - \gamma_{service})X \qquad X \sim N(\mu_{service}, \sigma^2_{service})$$

```
Input: Integer iterations
Output: Workflow
Create empty workflow wf
add linear control flow structure of length L_initial to wf
iterations=Max(R_linear, R_parallel and R_conditional)
for i = 1..iterations do
    if i < R_parallel then
        Replace random activity of wf with a parallel split structure
    end
    if i < R_conditional then
        Replace random activity of wf with a conditional split structure
    end
end
for i = 1..R_linear do
    Replace random activity of wf with a linear extension structure
end
return wf;
```

Figure 5.1: Function to generate control flow

The parameter $0 \leq \gamma_{service} \leq 1$ determines to what extend the size of an outgoing data flow depends on the incoming data flow(s). For $\gamma_{service} = 0$, the input and output of a service function is completely independent. The parameter may vary for infrastructure, human and is/h services. The mean μ and variance σ^2 of the normally distributed X can differ as well depending on the service. The content of the data flow that the service creates is a number generated randomly between 0 and 100 for all services.

5.1.3 Data Flow

The data flow models the data dependencies between activities. A data dependency is created by connecting two activities a and b with a data flow, that is, an edge from $S(a)$ to a, an edge between a and b and an edge from b to its service node $S(b)$. Data dependencies can only be created for activities a and b if a is a (possibly non-direct) predecessor of b. We generate them by iterating through all activities and generating one data flow for each. By choosing either a predecessor or a successor with probability 0.5, we avoid a bias of data links to the end or the start of the workflow:

The activity b is not chosen among all activities A, but within a range defined by $dlink_{min}$ and $dlink_{max}$: The path from a to b (or b to a) should contain at least $dlink_{min}$ and must contain at most $dlink_{max}$ activities. The minimum path length requirement can not always be met, e.g. when the start or the end of the workflow has been reached. If that is the case, the start / end activity is chosen as b. Conditional split activities are treated slightly differently: Because their condition requires an input, it has to be made sure that the activity b is a predecessor of

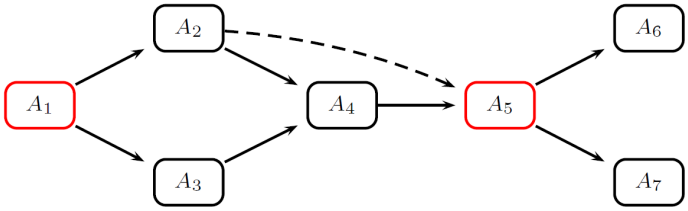

Figure 5.2: Evaluation of condition in A_5 depends on A_2

the conditional split activity a, and furthermore, that b is always executed when a is executed. Otherwise, a would not have an input to evaluate the condition with. This might be the case if b is part of a conditional split branch. Figure 5.2 shows such a case, where the conditional split A_5 depends on A_2. If however A_2 was not executed before A_5, the condition can not be evaluated due to missing data. So, instead of choosing any predecessor b of a within the range given, the path from b to a must not contain any conditional joins.

5.1.4 Conditions

The conditions of conditional splits depend on exactly one incoming data flow's value and compare this value to a value y that was chosen randomly between 0 and 100 during the generation of the workflow model. Let the incoming value be x, and b and c the possible successor activities of a. Then the successor of a is determined as follows:

$$succ(a) = \begin{cases} b & \text{if } x < y \\ c & \text{otherwise} \end{cases}$$

5.2 Evaluation Parameters

The following table summarizes the parameters used for evaluation:

$l_{initial}$	length of the control flow chain before replacements
$R_{linear}, R_{conditional}, R_{parallel}$	number of replacements for each structure
L_{csplit}	initial length of each conditional split branch
P_H, P_{IS}	probabilities for assigning a human service and an infrastructure service. $P_{IS/H} = 1 - P_H - P_{IS} \geq 0$
$\gamma_{IS}, \mu_{IS}, \sigma^2_{IS}, max_{IS}$	parameters for the size of data generated by infrastructure services
$\gamma_H, \mu_H, \sigma^2_H, max_H$	parameters for the size of data generated by human services
$\gamma_{IS/H}, \mu_{IS/H}, \sigma^2_{IS/H}, max_{IS/H}$	parameters for the size of data generated by IS/H services
$cost_{ex}$	execution cost for IS/H services
$dlink_{min}, dlink_{max}$	minimum / maximum path length of data links

For the evaluation, we created 300 workflow models with parameters randomly chosen in a parameter space defined as follows:

parameter	minimum values	maximum values
$l_{initial}$	15	20
$R_{linear}, R_{conditional}, R_{parallel}$	5 / 1 / 0	10 / 2 / 1
L_{csplit}	8	12
P_H, P_{IS}	0.3 / 0.3	0.5 / 0.5
$\gamma_{IS}, \mu_{IS}, \sigma^2_{IS}, max_{IS}$	$0/200kb/200kb/5000kb$	$1/3000kb/2000kb/5000kb$
$\gamma_H, \mu_H, \sigma^2_H, max_H$	$0/200kb/200kb/5000kb$	$1/3000kb/2000kb/5000kb$
$\gamma_{IS/H}, \mu_{IS/H}, \sigma^2_{IS/H}, max_{IS/H}$	$0/200kb/200kb/5000kb$	$1/3000kb/2000kb/5000kb$
$cost_{ex}$	equiv. to transfer of 2000 kb	equiv. 20000 kb
$dlink_{min}, dlink_{max}$	2 / 10	7 / 20

For each of these models, we executed 20 instances. Before running each model's instances, the predictors were trained with a training set of 100 simulated workflow runs of the corresponding model. 20 instances proved sufficiently large, considering that there are at most 2 conditional splits in the model. Unless stated otherwise, the predictors are executed every 4 steps during the workflow execution.
In our evaluation we will use a simplified variant of the energy model for EDGE introduced in [RZ07]. The simplification is necessary because in our simulations, we do not simulate time between transfers which would be necessary to model the different power states of the protocol.

5.3 Evaluation Results

5.3.1 Absolute Optimal Energy Costs

We will focus our evaluation on energy costs relative to the optimal cost. To get a rough idea about the quantity of the costs, we also computed the average value of the optimal energy costs for all instances. The average optimal cost of all instances that were run was around 3000 J. For comparison, a rechargeable lithium-ion battery of a current smartphone such as the HTC Desire has a nominal energy supply of around 5 Wh which equals 18.000 J.

5.3.2 Evaluation of Data Flow Predictors

We introduced two models for the data flow prediction, the average model which predicts the data flow by calculating the mean value of past data flows and the linear regression predictor, which tries to predict future flows on basis of current data flows. To compare the two approaches, we modified the model parameters so that $\gamma_H = \gamma_{IS} = \gamma_{IS/H} = 1$. Thus, the service output size exactly equals the summed input sizes, unless there is no input in which case the output is calculated randomly normally distributed according to the parameters. This setup provides optimal conditions for the linear regression. Figure 5.3 (page 38) compares the CPP and CEP with both data flow predictors. Despite the setup favors the linear regression predictor, the improvement over the average predictor is marginal. A possible reason is that the regression predictor only can provide an advantage over the average predictor for the prediction of an edge e if all edges that the service generating e depends on are known. Otherwise, it falls back to the average predictor resulting in equal predictions of both. An improvement might be achieved if the regression would base its prediction not only on known edges, but also on edges it predicted itself and that are not known yet. However, that way, errors in predictions would be amplified throughout the workflow unless the workflow's services perfectly followed the assumed linear model. Since the data flow predictors score almost equally well, we conduct the following control flow predictor evaluation using only the average data flow predictor.

5.3.3 Evaluation of Control Flow Predictors

Effect of Prediction Interval on CPP and CEP

The prediction interval defines how often the optimal distribution is recalculated during the execution of an instance. We expect that the less accurate the prediction is, the more its accuracy improves when decreasing the interval. This is because between the prediction steps, the predictor is 'blind' and cannot react on deviations in the control or data flow compared to its prediction. The accuracy of the predictor affects the performance of the distribution algorithm. We evaluated the conditional probability predictor and the condition evaluation predictor combined with the average data flow predictor. The same set of instances was

used for both predictors and all intervals. The results are displayed in figures 5.4 and 5.5 (pages 39 and 40) respectively. The ordinate indicates the accumulated percentage of workflow instances and the x-axis represents the cost increase compared to the optimal cost in percent. As expected, both the distribution calculated using CPP and the distribution using CEP significantly improve when using smaller prediction intervals. However, even if the distribution is calculated just once for all instances ($i = \infty$), in 30% of the instances the energy cost is equal to the optimal cost. In this case, CEP=CPP since lacking any data, CEP cannot evaluate any conditions and falls back to CPP. It is interesting to note that there is barely a difference between the CEP's performance for intervals 1 and 4. In both cases, it performs respectably, with less than 10% of the instances having an energy cost exceeding the optimal cost by more than 10%. On the other hand, if $i = 1$, the difference between CPP and CEP is neglectibly small. Thus, 'looking ahead' by evaluating the conditions only provides a considerable benefit if the prediction and distribution is not executed in every step.

Performance of Binary Variants

Figure 5.6 (page 41) displays a comparison between the CPP and CEP, the corresponding binary variants and the ALL predictor which assigns a probability of 1 to all activities. The CPP performs poor compared to the other predictors, and even the CEP's result is slightly worse than the simple ALL predictor. This provides strong evidence that weighting the predicted energy cost according to their probability is not advisable in workflow models generated in accordance with our system model. Only the binary CPP and binary CEP provide a (slight) advantage over the ALL predictor. In figure 5.7 (page 42), we examined how the performance of BCEP and ALL changes as the intervall is increased. Note the adjusted scale on the x-axis. The ALL predictor basically performs like the BCEP predictor with $interval = \infty$. This is reasonable since when predicting only once, $BCEP = BCPP$, and $BCPP = ALL$ if all activities were executed at some point in the training instances. The BCEP predictors perform better than the ALL predictors at intervalls 1 and 4 because it can rule out some data edges before the distribution. We also compared the binary predictors regarding the prediction intervals (figure 5.8, page 43). The difference here is much smaller than for the predictors weighting the costs with continous values between 0 and 1. The binary CEP performance is better than, but very close to the binary CPP.

Performance Compared to Centralized Distribution

If the execution is centralized, all non-human activities are distributed to the infrastructure. Figure 5.9 (page 44) exemplary shows much energy can be saved with the BCEP predicted distribution compared to the centralized approach. In around 50% of the instances, we can save around 30% or more with the distribution algorithm. Only for very few instances the centralized result is comparable to the optimized distribution.

Of course the result highly depends on the parameters chosen. If the probability for human services P_H is low and/or human services transfer much less data than the infrastructure services, the optimal distribution shows a tendency to the infrastructure, and accordingly, the centralized approach performs better. Figure 5.10 (page 45) shows the result of a second evaluation that has been conducted with decreased parameters for P_H and μ_H. Here, in around 20% of the instances, the centralized distribution's cost does not exceed the optimal energy cost.

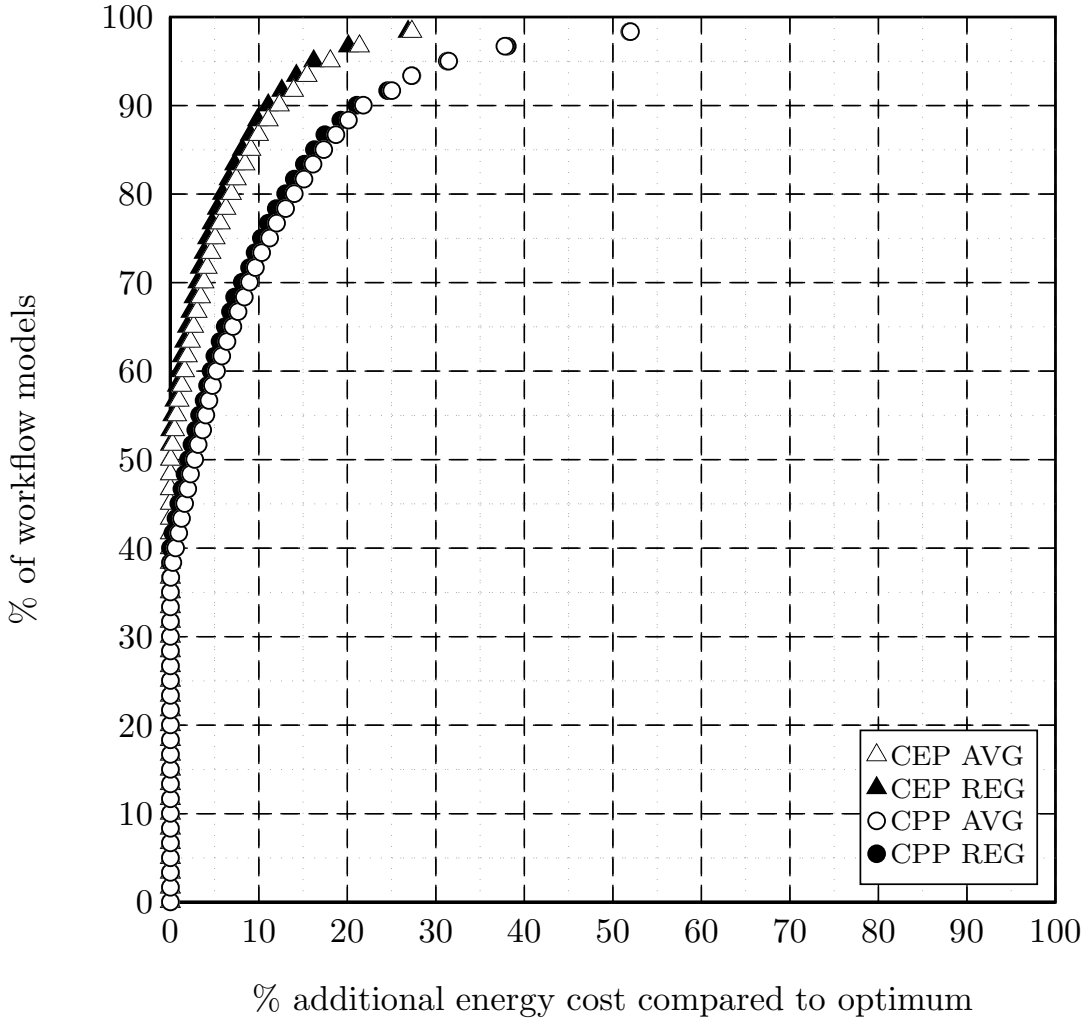

Figure 5.3: CPP/CEP performance for average and linear data prediction

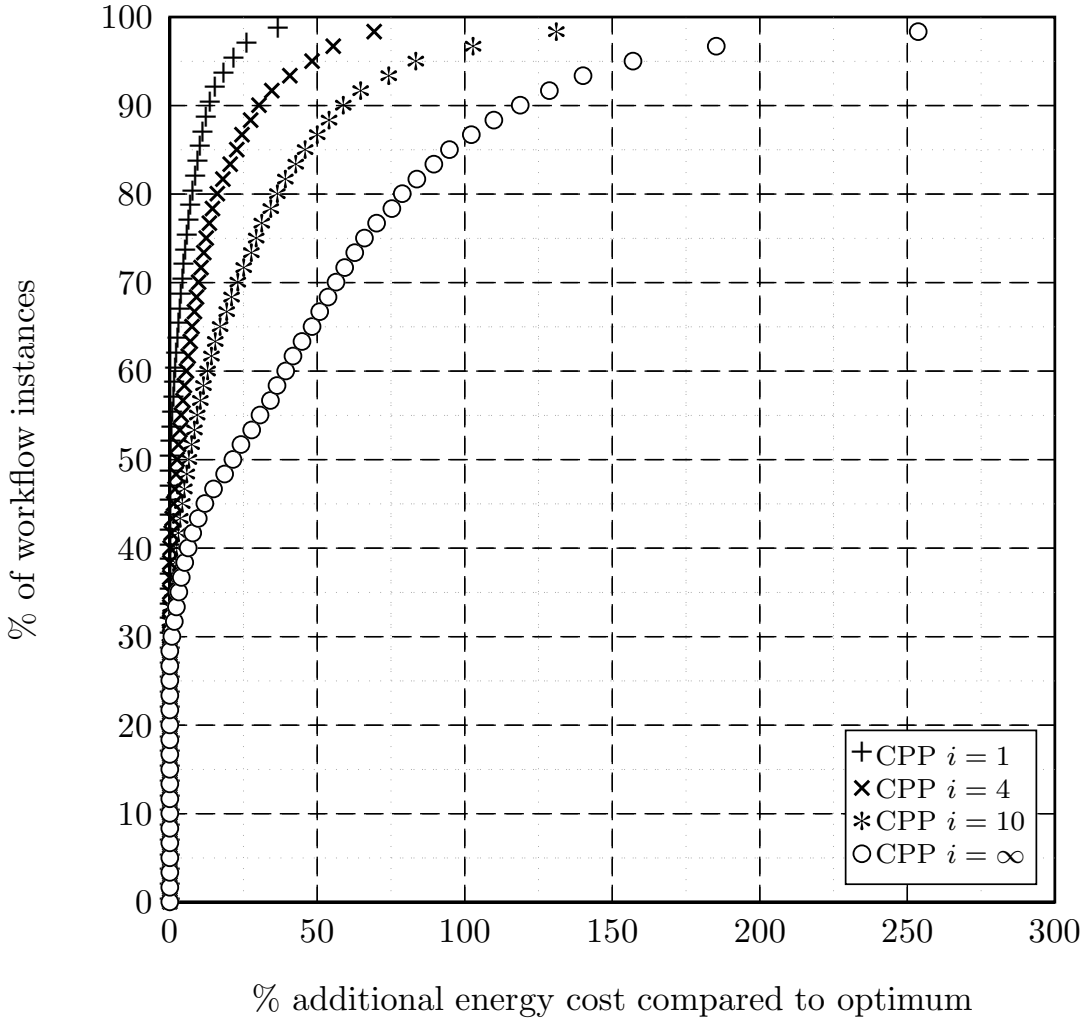

Figure 5.4: CPP performance for different prediction intervals

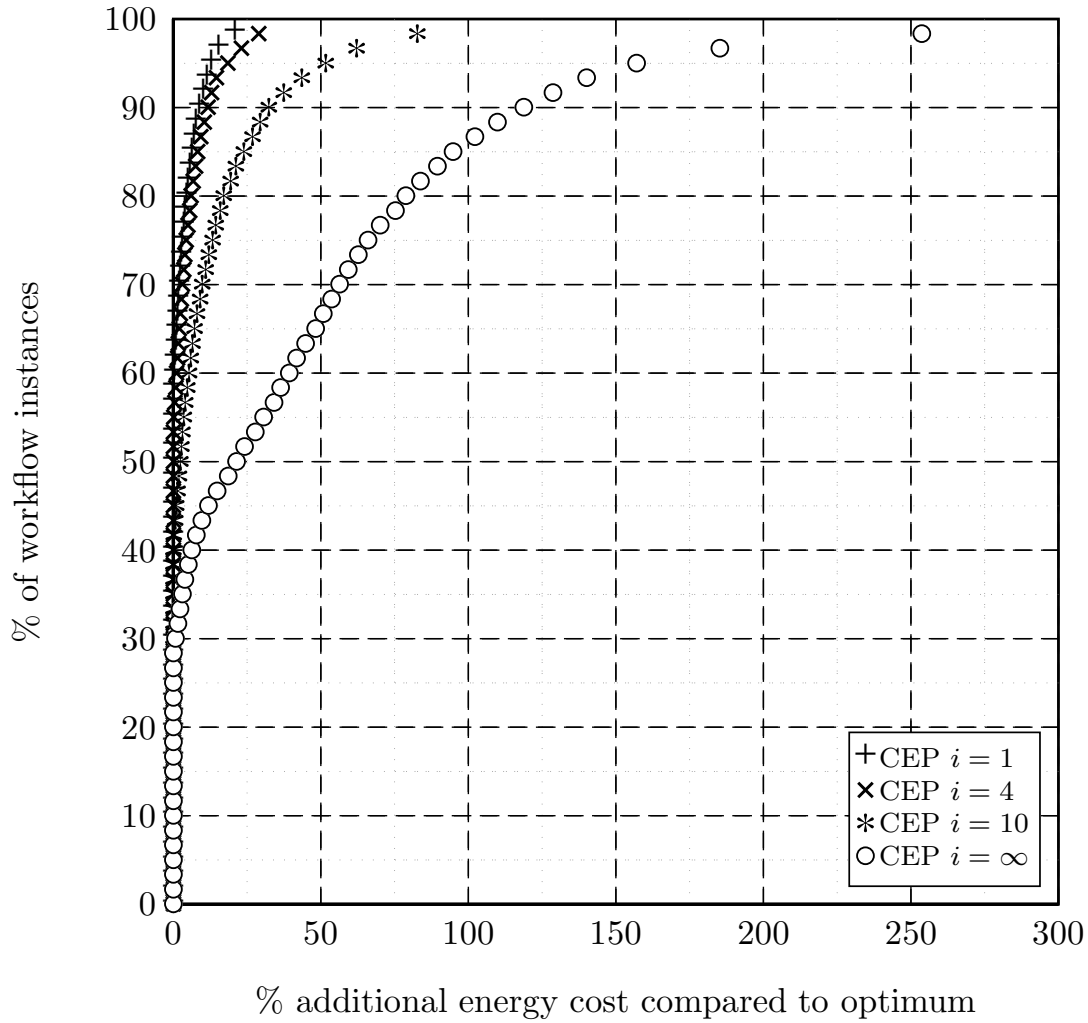

Figure 5.5: CEP performance for different prediction intervals

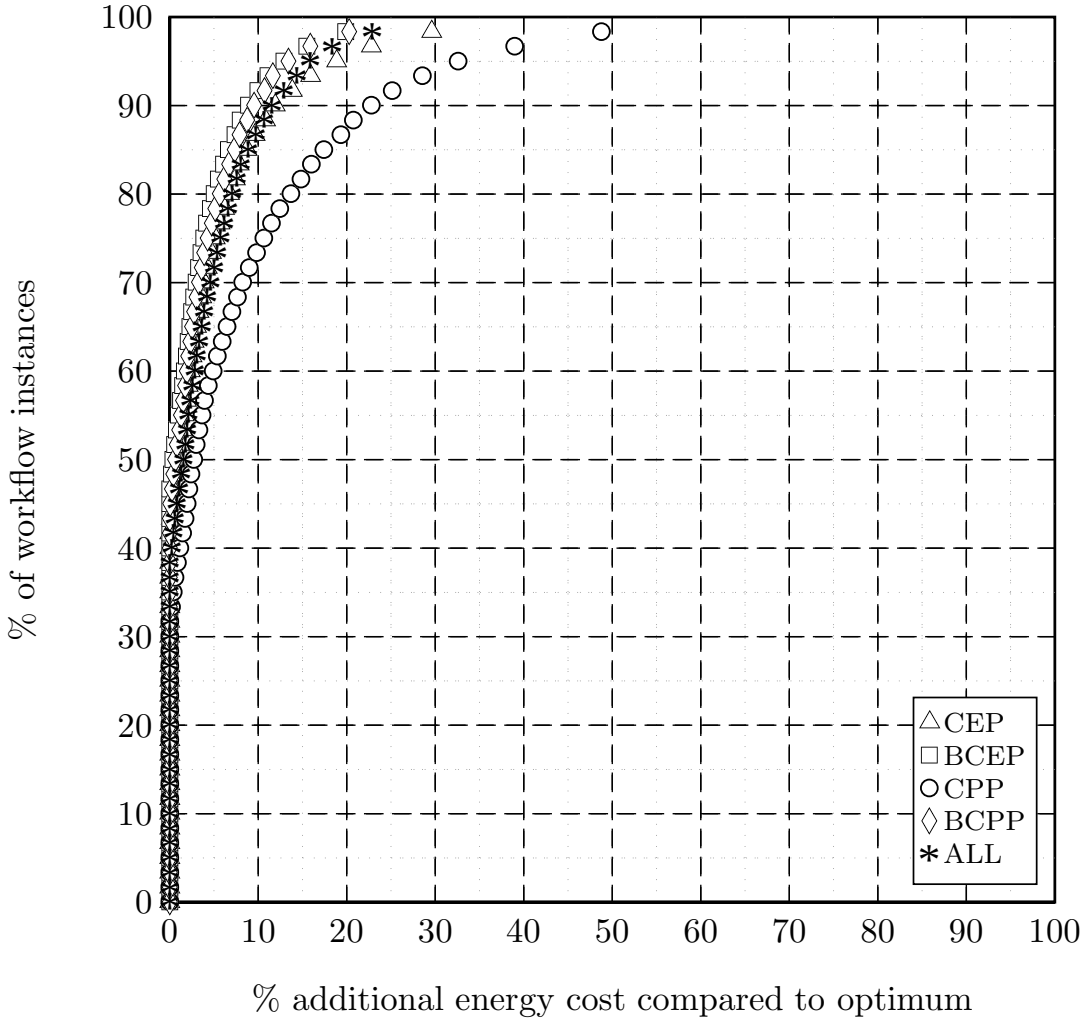

Figure 5.6: Performance of binary variants and the ALL predictor

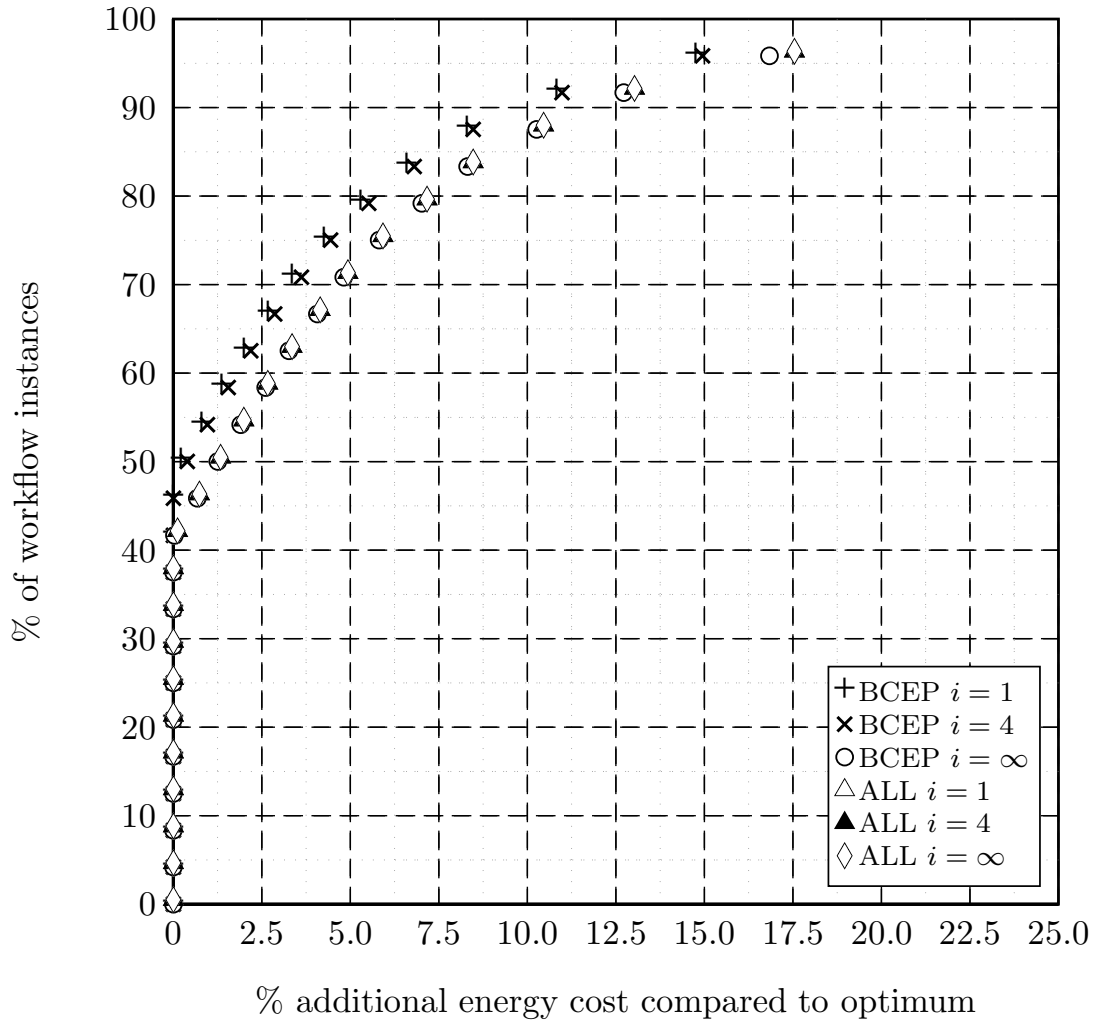

Figure 5.7: Performance of BCEP and ALL at different intervals

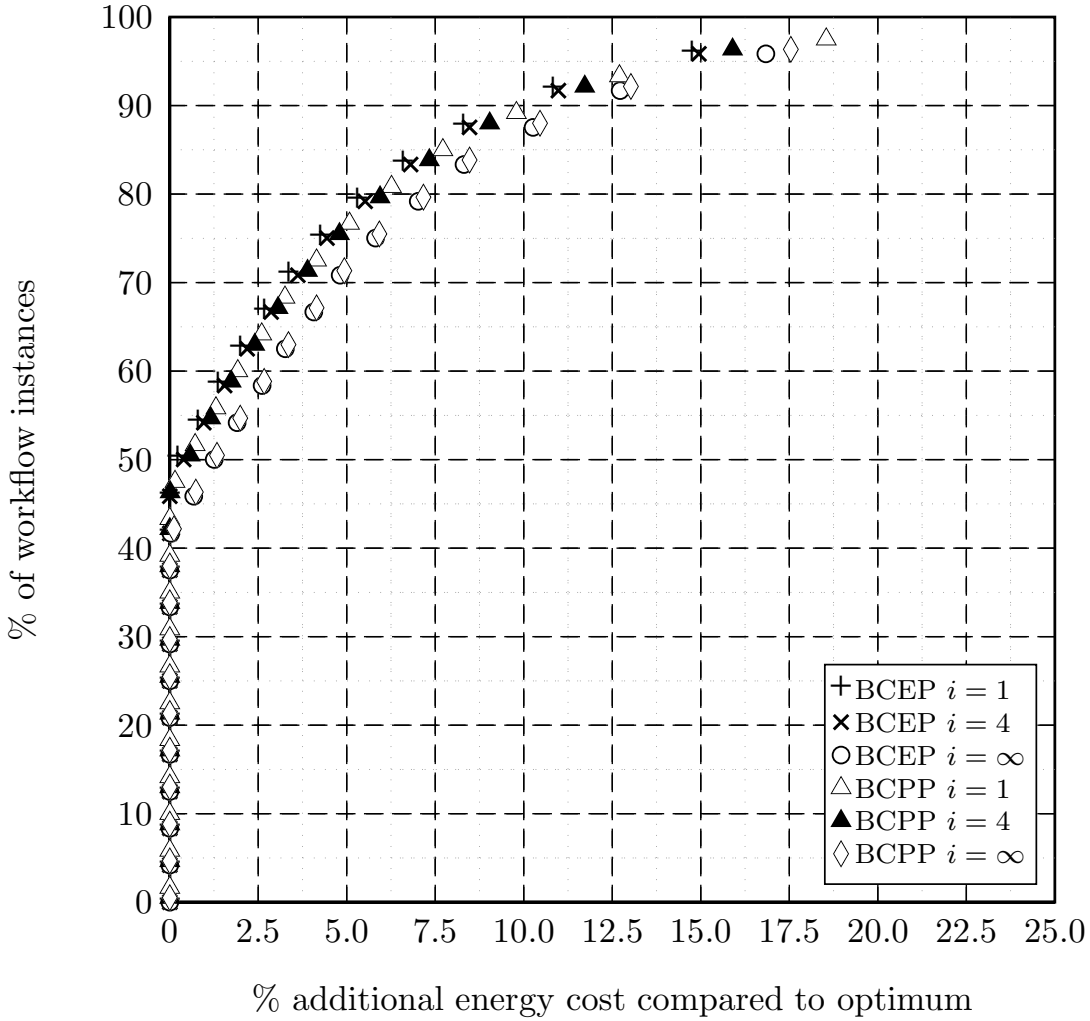

Figure 5.8: Performance of BCEP and BCPP at different intervals

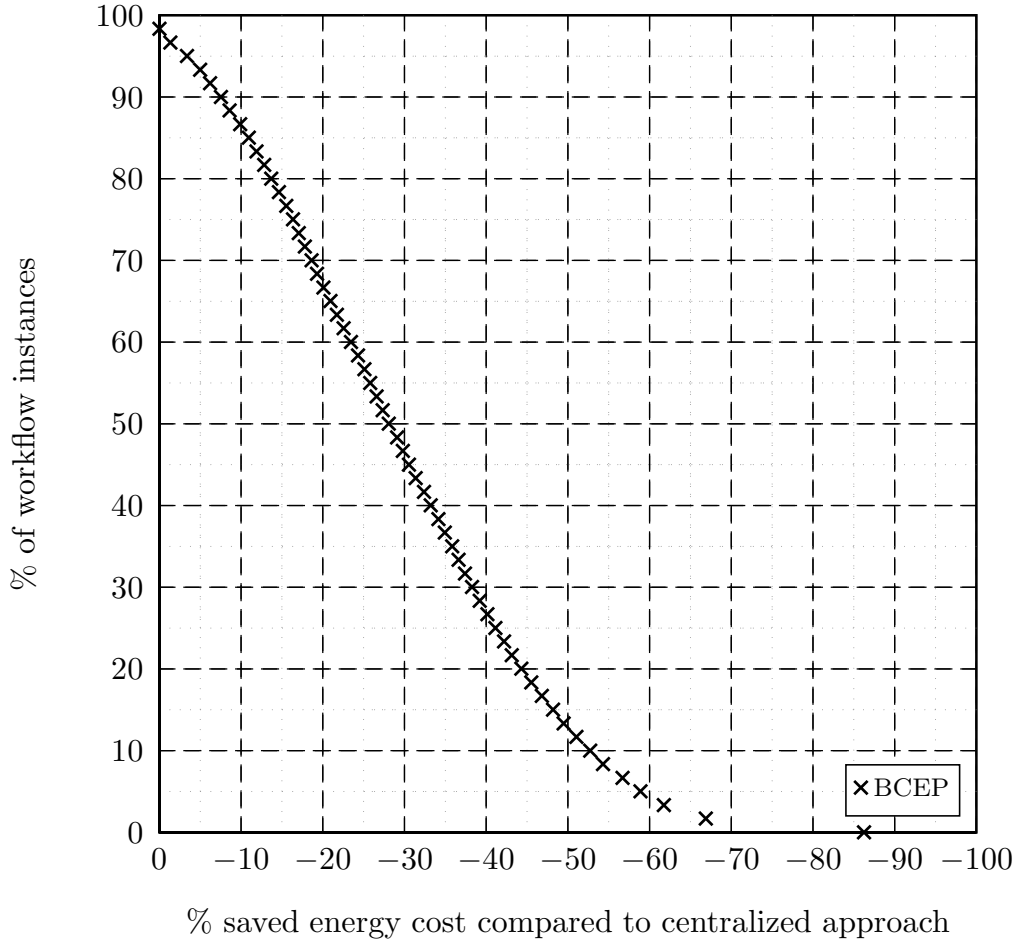

Figure 5.9: Performance of BCEP compared to centralized distribution

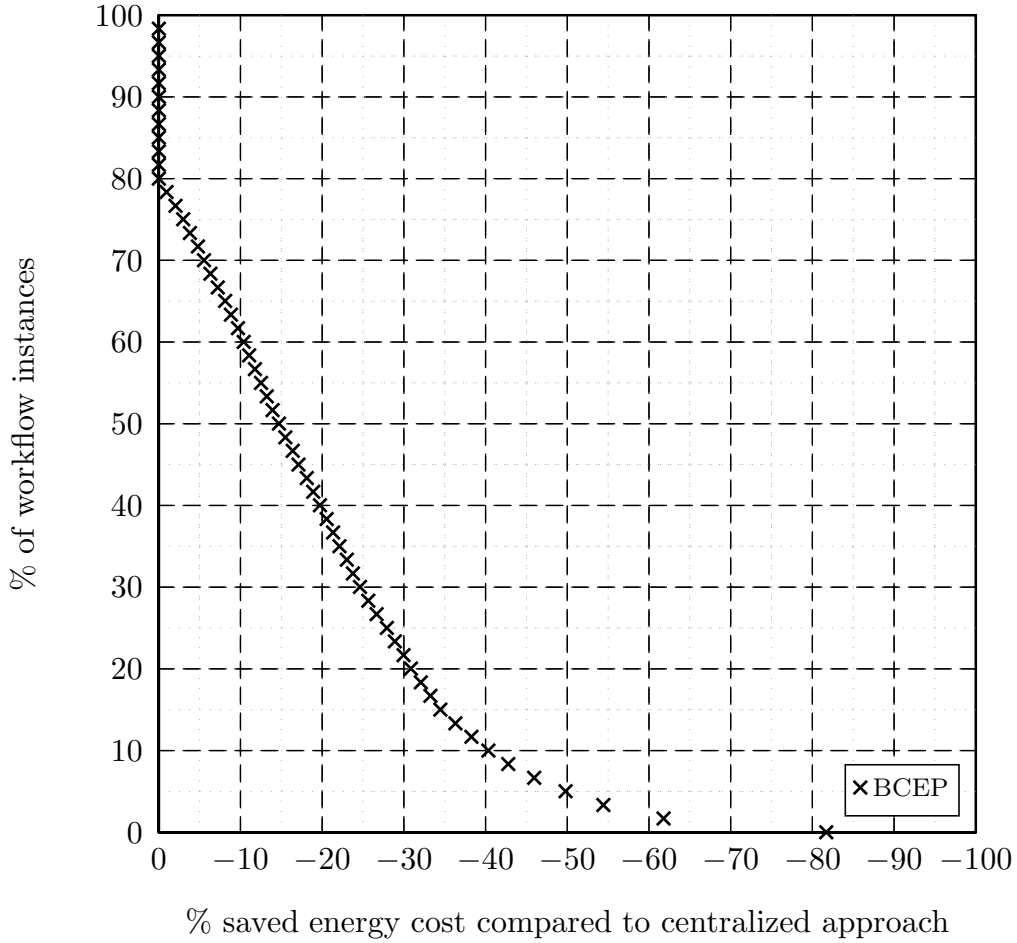

Figure 5.10: Performance of BCEP compared to centralized distribution with modified parameters

6 Summary and Conclusion

We introduced a method to apply the distribution algorithm repeatedly on a workflow instance. This allowed us to propose prediction methods which can take advantage of new information gathered during the workflow execution. From our evaluation results we can conclude the following:

- The workflow based distribution algorithm generally produces distributions with significantly lower costs than the centralized approach.
- Running the distribution algorithm repeatedly during the workflow execution improves the distribution with respect to energy costs.
- An average based data flow prediction provides a good basis for the predictor. Predicting the service's output according to its input (if known) results only in a minimal improvement even under optimal conditions.
- Weighting the predicted costs by probabilities does not improve the result. On the contrary, weighting all predicted costs unconditionally by 1 provides a better prediction as input for the distribution algorithm than weighting them by the activities' probabilities.
- However, the optimality of the distribution can slightly be improved by using prediction methods that predict the control flow, e.g. by checking for reachability of activities or evaluating conditional splits prior to their execution.

In around 40% of all instances we achieved a distribution already equivalent to the optimal distribution simply by distributing the activities once using unweighted average data flows as input to the predictor. Using more advanced control flow prediction methods and instance based distribution, this result can be improved. However, this improvement is rather small and we did not take into account the energy cost for predicting and calculating the distribution in our calculations. If these calculations are done on the infrastructure exlusively, an instance based approach with an appropriate prediction method reduces the energy costs compared to the workflow based distribution. We see potential for further improvement particulary in the prediction of the data flow where we used very basic methods. Improved control flow prediction seems to have only a marginal effect.

Bibliography

[BBV09] N. Balasubramanian, A. Balasubramanian, A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In A. Feldmann, L. Mathy, editors, *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement 2009, Chicago, Illinois, USA, November 4-6, 2009*, pp. 280–293. ACM, 2009. (Cited on page 8)

[BGW09] M. Bozkaya, J. Gabriels, J. M. E. M. van der Werf. Process Diagnostics: A Method Based on Process Mining. In *EKNOW'09*, pp. 22–27. 2009. (Cited on page 7)

[Deh03] J. Dehnert. *A Methodology for Workflow Modeling*. Master's thesis, TU Berlin, Fakultät IV Elektrotechnik und Informatik, 2003. (Cited on page 7)

[FFH] D. Fischer, S. Foell, K. Herrmann. Energy-efficient Workflow Distribution. (Cited on page 17)

[RZ07] A. Rahmati, L. Zhong. Context for Wireless: Context-sensitive energy-efficient wireless data transfer. In *Proceedings of the Fifth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, MobiSys '07, pp. 165–178. USENIX Association, San Juan, Puerto Rico, 2007. doi:10.1145/1247660.1247681. (Cited on pages 8 and 34)

[SM01] A. Sharp, P. Mcdermott. *Workflow Modeling: Tools for Process Improvement and Application Development*. Artech House Publishers, 2001. (Cited on page 7)

[SSM] A. Shye, B. Scholbrock, G. Memik. Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures. (Cited on page 8)

[VHMK97] P. Van Hentenryck, D. McAllester, D. Kapur. Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM J. Numer. Anal.*, 34:797–827, 1997. doi:http://dx.doi.org/10.1137/S0036142995281504. (Cited on page 29)

[Woo08] J. Wooldridge. *Introductory Econometrics: A Modern Approach (with Economic Applications, Data Sets, Student Solutions Manual Printed Access Card)*. South-Western College Pub, 4 edition, 2008. (Cited on page 29)

[ZTQ+10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, L. Yang. Accurate online power estimation and automatic battery behavior based power

model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pp. 105–114. ACM, New York, NY, USA, 2010. doi: http://doi.acm.org/10.1145/1878961.1878982. (Cited on page 8)

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Jörg Belz)