Predicting Resource Allocation and Costs for Business Processes in the Cloud

Toni Mastelić†, Walid Fdhila*, Ivona Brandić† and Stefanie Rinderle-Ma*
*Faculty of Computer Science, University of Vienna, Austria
†Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria
Email: {toni, ivona}@ec.tuwien.ac.at, {walid.fdhila, stefanie.rinderle-ma}@univie.ac.at

***Abstract*—By moving business processes into the cloud, business partners can benefit from lower costs, more flexibility and greater scalability in terms of resources offered by the cloud providers. In order to execute a process or a part of it, a business process owner selects and leases feasible resources while considering different constraints; e.g., optimizing resource requirements and minimizing their costs. In this context, utilizing information about the process models or the dependencies between tasks can help the owner to better manage leased resources. In this paper, we propose a novel resource allocation technique based on the execution path of the process, used to assist the business process owner in efficiently leasing computing resources. The technique comprises three phases, namely process execution prediction, resource allocation and cost estimation. The first exploits the business process model metrics and attributes in order to predict the process execution and the required resources, while the second utilizes this prediction for efficient allocation of the cloud resources. The final phase estimates and optimizes costs of leased resources by combining different pricing models offered by the provider.**

I. Introduction

Business processes are means to implement and execute business logic of an enterprise [2]. Relevant tasks are orchestrated in the right order (control flow) while exchanging data (data flow). Moreover, the organizational aspect is reflected by assigning resources such as users, machines, or services to tasks during runtime. All relevant information about a business process can be described by a process model, which is used by a task scheduler for controlling the execution of the process. The scheduler decides on the path the process will take based on control/data flow, and utilizes the underlying resources for executing and synchronizing the tasks along the execution path.

Traditionally, a process owner holds and utilizes an in-house fixed set of resources used for executing business processes. These resources are managed and scheduled by the owner, which entails several issues related to flexibility and scalability, as well as the tremendous costs that engenders the maintaining of these resources. With the advent of cloud computing, business processes can benefit from the full potential of scalable and elastic cloud resources [3], as well as on-demand pricing models. However, if the same allocation strategies are applied as with in-house resources this can lead to high resource redundancy, which can still result in high costs. Therefore, efficient resource allocation and cost optimization based on the execution path of the process is still required by the owner.

While there is a body of work on resource allocation and management in the BPM context [4], [5], [6], only lately more attention has been paid to process scheduling in the cloud [7], [8], [9], [10]. The main focus is to propose techniques to optimize the actual scheduling and deployment of cloud resources for business processes, from the perspective of the cloud provider. However, such approaches only reduce costs for the provider, while a customer, i.e., a process owner can still end up with high costs due to low utilization of leased resources and poor selection of pricing models.

In our work, we adopt the perspective of a process owner, and formulate the problem as follows: *How can a business process owner estimate and minimize costs for allocating cloud resources by utilizing the prediction of the process execution path?* The prediction considers different metrics and the behavior of a process during the runtime. Consequently, resource allocation must satisfy all the constraints and the requirements of the tasks. Finally, cost estimation should provide different pricing options and highlight the best one.

We take the offline approach by predicting the execution path and estimating resource requirements prior the actual deployment. The approach is used for resource capacity and budget planning before moving business processes to cloud, as well as for offline resource scheduling, i.e., defining lease contracts and associating tasks with resources. The approach comprises three phases/contributions for achieving the above defined goal, namely:

- **Business process metrics and prediction** - we utilize process model metrics and attributes in order to derive resource requirements based on the predicted process execution path.
- **Resource allocation** - we minimize resource requirements by reusing the resources of sequentially executing tasks that are obtained from the prediction phase.
- **Cost estimation and optimization** - we estimate the lowest cost leasing option based on the resource allocation and pricing models offered by the cloud provider.

The paper is structured as follows. Section II describes a motivating example and summarizes the challenges, and Section III presents the fundamentals. Sections IV, V and VI detail the three phases of our approach, while Section VII evaluates the methods. Finally, Section VIII discusses the related work, and Section IX summarizes the paper.

978-1-4673-7275-6/15 $31.00 © 2015 IEEE
DOI 10.1109/SERVICES.2015.16

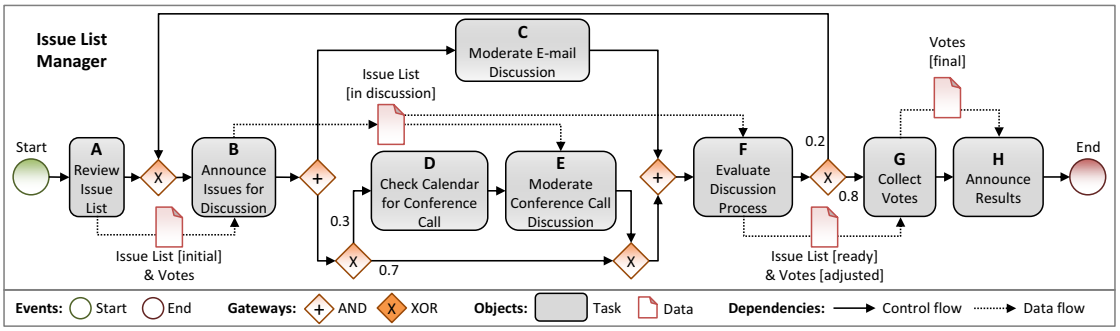

Figure 1: E-mail voting for resolving issues [1]

II. Motivating Example

The example depicted in Figure 1 describes an E-mail voting process presented in BPMN 2.0 specification. The example represents a simplified version of the process found in [1], and illustrates a process for resolving issues through e-mail votes. The grey boxes describe the tasks to be executed, while the diamonds define the relationship between them and the order of their execution; i.e., the control flow. The dashed links define the data flow and the items to be exchanged between the tasks.

The issue list manager sets up a list of issues (task A) that will be discussed through the voting process. When the list is ready, the issues are announced (task B) and the discussion process is launched through e-mail exchanges (task C) and a possible conference call (tasks D and E). The discussion is then evaluated (task F) and the votes are collected (task G). The votes are calculated and the results are finally announced and published (task H).

It should be noted, that each of these tasks requires a specific application and an amount of resources to be executed. For example, task E might require conference terminals for each participant of the conference call, which would result in several small machines being allocated. Commonly, the business process owner would allocate all the resources required for executing the entire process.

However, D and E might not be executed, so their resources would be wasted. This provides a trade-off between selecting the most probable path, or playing it safe and allocating redundant resources. Furthermore, if task D also requires small machines, they might be reused for task E, which would reduce the number of redundant machines. Finally, if resources for task B are leased from the provider until the end of the entire process, they can be reused for any task after B, as the resources are already paid for.

Based on the discussion, we define 3 research questions:

- How can business process owner utilize knowledge about the process in order to predict an execution path and the required resources for this path?
- How can the owner utilize this prediction of the resource requirements in order to efficiently allocate them on a cloud infrastructure?
- How can the owner use the allocation in order to derive the cheapest pricing scheme from the cloud provider?

III. Fundamentals

This section introduces the main concepts and definitions related to business processes and cloud infrastructure. They are used throughout the paper for formulating the context of the problem, solution and evaluation.

A. Business Process Model

A process model defines the relationship between the tasks of an organization needed to achieve a business goal. This relationship may characterize either the control or data flow structure; i.e. control or data dependencies between the process tasks. A process is represented by a directed acyclic graph where nodes are tasks, gateways or events and edges are data or control dependencies (Figure 1).

Definition 1 (Process Model): A process model is a tuple $(\mathcal{O}, \overrightarrow{c}, \overrightarrow{d}, \mathcal{D})$ where:

- $\mathcal{O}$ is a set of objects which can be partitioned into disjoint set of tasks $\mathcal{T}$, gateways $\mathcal{G}$ and events $\mathcal{E}$
- $\mathcal{G}$ is a set of gateways which can be partitioned into disjoint sets of exclusive XOR gateways and parallel AND gateways.
- $\mathcal{E}$ is a set of events which can be partitioned into disjoint sets of $start$ and end events.
- $\mathcal{D}$ is the set of data objects.
- $\overrightarrow{c}$: $\mathcal{O} \rightarrow \mathcal{O}$ is a control flow relation between the objects.
- $\overrightarrow{d}$: $\mathcal{T} \times \mathcal{T} \rightarrow \mathcal{D}$ is a data flow relation between the tasks.

In the following, we assume that the choice patterns (XOR gateways) of a process model are annotated with branching probabilities; i.e., the probability of choosing a branch. In practice, this can be determined using mining techniques of previous executions of the process model.

Definition 2 (Task): A task t is a unit of work performed to complete a job and is defined as a tuple ($name$, $input$, $output$, δ, $\mathcal{R}$), Where:

- $input \subset \mathcal{D}$ with $\forall d \in input, \exists t'$ with $\overrightarrow{d}(t', t) = d$.
- $output \subset \mathcal{D}$ with $\forall d \in output, \exists t'$ with $\overrightarrow{d}(t, t') = d$.
- δ is the average time for executing t.
- $\mathcal{R}$ is the set of resources required by the task t; e.g. storage, CPU.

B. Cloud Infrastructure Model

Cloud represents computing and storage resources, i.e., machines that can be leased by a process owner, and are used for satisfying resource requirements of the process tasks, as shown in Figure 2. In the figure, the first task requires a set of resources, which can be satisfied by the two machines as they have sufficient configuration. The machines represent a cloud infrastructure and are configured according to certain flavors.

Definition 3 (Cloud model): A cloud model is a tuple $(\mathcal{M}, \vec{\mathcal{F}})$ where:

- $\mathcal{M}$ is a set of machines m connected over the network. Each machine is defined as a tuple $m(h, f)$, where h indicates a type, e.g., storage or a computing machine, while f defines a flavor, namely a resource capacity. In the real-world, a machine can be any type of isolated computing resources such as a physical or virtual machine (VM), or a container, e.g., two VMs running on top of a physical machine as shown in Figure 2.
- $\mathcal{F}$ is a set of flavors f, a preconfigured package of resources such as RAM, CPU, storage and network bandwidth, defined in ordinal scale, e.g., small, medium and large VMs. Figure 2 shows two flavors, where only larger one can satisfy resource requirements of the task, namely 32G of storage, 4 cpus and 4G of RAM.

Finally, we define a mapping between the resources required by a task and the resources provided by a cloud infrastructure, i.e., machines, through the definition and configuration of flavors.

Definition 4 (Resources): Resource requirement r is a tuple $r(f, b, e)$, where:

- f is a flavor, as shown in Figure 2 where resource requirements of the task are mapped to the flavor of the machines.
- s defines *start* time when the resources are required.
- e defines *end* time when the resources can be freed.

Both s and e are derived from the δ attribute of a task. Finally, a task can require a set of resources $\mathcal{R}$, where each requirement $r \in \mathcal{R}$ can have different flavor f.

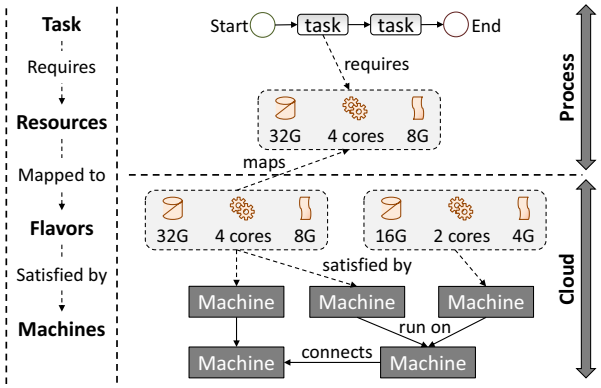

Figure 2: Cloud model

IV. Business Process Metrics and Prediction

In this section, we present the metrics that are used to predict the execution path of a business process, as well as the resource requirements of each task.

Communication overhead*:* represents the amount of data to be transferred between a pair of tasks. We assume that communication overhead can be measured in bytes. We also assume that the average size in bytes of data d is known, and we write $size(d)$ to denote this size. We define the functions $\alpha(t)$ and $\beta(t_i, t_j)$ as follows:

Function 1 (Execution Number $\alpha(t)$): The function $\alpha(t)$ represents the number of times a task t is executed in a single instance of the business process.

Function 2 (Follow Probability $\beta(t_i, t_j)$): Function $\beta(t_i, t_j)$ represents the probability that a task t_i follows a task t_j for a single instance of the business process.

Details about how to calculate the execution number of a task and the follow probability between two tasks in a process model can be found in [11]. The communication overhead is given by the following equation:

$$co(t_1, t_2) = \Sigma_{d \in \vec{d}\,(t_1,t_2)} \alpha(t_1) \times \beta(t_1, t_2) \times size(d) \quad (1)$$

In the equation, the communication overhead between two tasks t_1 and t_2 is equal to the sum of the data sent from t_1 to t_2 multiplied by (i) the number of the execution of t_1; i.e., $\alpha(t_1)$ (e.g. in a loop), (ii) the probability that t_2 follows t_1, and (iii) the size of data d. The communication overhead is used to estimate the storage requirements by the owner.

In general, an execution of a process follows the order and the dependencies defined in its model. During runtime, instances can take different paths based on the choices they make (e.g., XOR pattern). A **path** represents a set of tasks that can be executed by a single instance, from $start$ to end following the control flow of the process . The resources required by a given instance depend exclusively on its execution path. Therefore, predicting the path that will be taken by a given instance is used for estimating the amount of the required resources. In this context, we use two different approaches as follows:

Optimistic Path*:* represents the path with the highest occurrence probability. To determine the optimistic path, we parse the process model from $start$ to end and for each choice gateway, we choose the branch with the highest probability. Tasks with a low execution probability are not considered in this path. The optimistic path is given by Equation 2, where $|\vec{c}(o_i, o_j)|$ represents the value (probability) of the control dependency between the objects $o_i$and o_j (an object can be a task or a gateway).

$$OP(M) = argmax_{P \in M}[\prod_{o_i, o_j \in P} |\vec{c}(o_i, o_j)|] \quad (2)$$

It should be noted that the calculation of the optimistic path does not take into consideration the resource requirements (e.g., communication overhead).

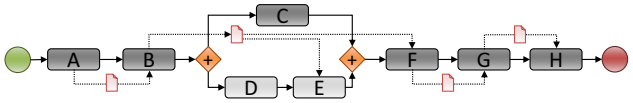

Figure 3: Critical path (all tasks) and Optimistic path (dark colored tasks)

Critical Path*:* represents the path that requires the maximum of resources. To determine the critical path, we parse the process model from *start* to *end* and for each choice gateway (XOR), we choose the branch with tasks that require the maximum of resources. The amount of resources required by a task is weighted by its execution time and the average execution number. For example, in a choice between two tasks t_1 and t_2 that require resources R_1 and R_2, and have the execution times δ_1 and δ_2 respectively, the task that belongs to the critical path is given by $max_{\{t_1,t_2\}}(|R_1| \times \delta_1 \times \alpha(t_1), |R_2| \times \delta_2 \times \alpha(t_2))$. The calculation of the critical path is given by Equation 3.

$$CP(M) = argmax_{P \in M}[\Sigma_{t \in P}|R_t| \times \delta_t \times \alpha(t) + \Sigma_{(ti,tj) \in P}\, co(t_i, t_j)] \quad (3)$$

Since we deal with two types of resources; i.e., computing resources and storage, then it is possible to have two different critical paths of the same process model; i.e., one for each resource type. For example, consider a choice between two tasks t_1 and t_2, which require R_1 and R_2 as computing resources, and R'_1 and R'_2 as storage resources, respectively. If we assume that $R_1 < R_2$ but $R'_1 > R'_2$, then the critical path in term of storage is the one containing t_1, and the critical path in terms of computing resources is the one containing t_2. In this case, we use the both critical paths, estimate their costs (cf. next Sections), and chose the one the most costly. The critical and optimistic paths of our example scenario are shown in Figure 3.

V. Resource Allocation

In the previous section we define business process metrics used for predicting execution path of the process, and derive *start* times s and *end* times e of the path tasks, as well as the amount of required resources. Software resources are not considered in the allocation as each task requires fresh software, and therefore each allocated machine has to be reimaged. However, hardware resources, i.e., machines can be reused for satisfying time constrained tasks by mapping the tasks to available machines. The mapping problem can be formulated as Fixed Job Scheduling (FJS) problem, also known as interval scheduling problem or k-track assignment problem [12]. Finally, it should be noted that the same approach can be applied for computing and storage machines utilized by the tasks.

Resource requirements*:* FJS defines a set $\mathcal{R} = \{r_1, ..., r_n\}$ of n jobs and each job r_i requires processing without interruption from a given start time s_i to a given end time e_i. Each machine, while available all the time, can process at most one job at a time. The objective is to determine the minimum number of machines needed to process all the jobs. In our case, jobs represent resources requirements of tasks. Furthermore, our problem includes different machine configurations, i.e., flavors, which make it strongly NP-hard [13]. Therefore, we apply heuristics in order to optimize allocation, more specifically we apply modified best-fit algorithm and combine it with FJS (Algorithm 1).

The algorithm receives a set of n requested resources $\mathcal{R}$, where each request r_i is defined with the *start* time s_i, *end* time e_i and flavor f_i. Output of the algorithm is a list of machines $\mathcal{M}$ where each machine m_j is scheduled to satisfy a subset of requested resources $\mathcal{R}_j \subseteq \mathcal{R}$. Each machine is defined with flavor f_j and a flag a_j, which is used to mark machine's availability during resource allocation within the algorithm. In line 1 of the algorithm, start time s_i and end time e_i of the requested resources are placed in a single vector, thus creating $2n$ endpoints, namely $\{p_1, p_2 ... p_{2n}\}$. Afterwards, from line 2 to 4 the endpoints are sorted by three attributes. Firstly, the endpoints are sorted in ascending order by time indicating *start* or the *end*. Secondly, if times are equal, the start times are set before the end time, in order to first release the expired resources and then allocate new ones. Finally, the endpoints are sorted by the execution time of the related resources in a descending order, so that longer requirements are served first.

Algorithm 1: Resource requirements prediction

Input: $\mathcal{R} = \{r_1, r_2, ...r_n\}$, where $r_i(s_i, e_i, f_i)$
Output: $\mathcal{M} = \{m_1, m_2, ...m_m\}$, where $m_j(\mathcal{R}_j \subseteq \mathcal{R}, f_j, a_j)$ so that $\mathcal{R} = \mathcal{R}_1 \bigcup ... \bigcup \mathcal{R}_c$ and $\mathcal{R}_1 \bigcap ... \bigcap \mathcal{R}_c = \emptyset$

1 Perform three level sort on $2n$ endpoints $(p_1 \leftarrow s_1, p_2 \leftarrow e_1, ...p_{2n-1} \leftarrow s_n, p_{2n} \leftarrow e_n)$
2 • sort in ascending order so that $p_1 \leq p_2 \leq ... \leq p_{2n}$
3 • if $p_k = p_{k+1} = ... = p_{k+w}$ then s is set before e
4 • if $p_k = p_{k+1} = ... = p_{k+w}$ then apply descending order by execution time so that $e_i - s_i \geq e_{i+1} - s_{i+1}$
5 $F \leftarrow \mathcal{F} \bigcup \{f_i\}$, where $i = [1, n]$ and f_i is unique
6 sort(F) in descending order
7 **foreach** $f \in \mathcal{F}$ **do**
8 **for** $i = 1$ *to* $2n$ **do**
9 **if** $(f_i = f)$ **then**
10 **if** $(p_k = s_i)$ **then**
11 $m_j \leftarrow getAvailable()$; `// Alg 2`
12 $\mathcal{R}_j \leftarrow \mathcal{R}_j \bigcup \{r_j\}$
13 $a_j \leftarrow false$
14 **else**
15 $a_j \leftarrow true$, where $m_j \in M$ and $r_i \in \mathcal{R}_j$
16 **return** $\mathcal{M}$

The set of distinct flavors $\mathcal{F}$ are extracted from the requested resources $\mathcal{R}$ (line 5) and sorted in a descending order (line 6). For each distinct flavor f_z in line 7 the algorithm performs allocation by going through the set of $2n$ endpoints (line 8). It first checks in line 9 if the current endpoint p_k belongs to the requested resources with a specified flavor f_z (line 9) and if the endpoint is the start

(line 10). If conditions are true function $getAvailable$ (line 11) gets first available machine m_j and takes the resource r_i to which the start time $p_k = s_i$ belongs to, and adds it to the schedule list $\mathcal{R}_j$ of machine m_j (line 12). In other words, it satisfies resource requirement r_i with the machine m_j. Finally, the flag a_j of the machine m_j is set to false, thus indicating that it is no longer available (line 13).

In case the endpoint $p_k = e_i$ is the *end* instead of the *start* (line 14), machine m_j corresponding to r_i is placed is released by setting a_j to true, hence flagging it as available (line 15). This means that machine m_j can now be used for other resource requirements. Once all the endpoints of the specific flavor are allocated, the algorithm returns to line 7 and starts allocating smaller flavors, since $\mathcal{F}$ is sorted in a descending order in line 6. Once all the flavors are allocated, the algorithm returns the set of machines $\mathcal{M}$ containing their schedules.

Reusing resources*:* Function $getAvailable$ in line 11 of Algorithm 1 defines how resources are reused. We define three strategies formulated as part of joined Algorithm 2 and shown in Figure 4 for reusing machine m_j with flavor f_j for satisfying resource requirement r_i with flavor f_i. Comments on the right of the algorithm define to which strategy the code line is applied to, namely:

(1) no reuse: always allocates a new machine for each resource requirement, as shown in Algorithm 2 with Lines 7, 8 and 9 where a new machine is allocated, added to the set of existing machines, and finally returned, respectively. This strategy serves as the baseline approach and commonly requires most resources as shown in Figure 4.

(2) $f_i = f_j$: reuses only the same size machines, as shown in Figure 4 where tasks B, D and F are sequentially executed on machine m_1. Available machines M are searched through (Line 1) and checked for the same flavor required by r_i (Line 2). If the machine is available (Line 4) it is returned by the function (Line 6). In case no machine satisfies the criteria, a new machine is allocated in line 6-8, the same as for strategy (1).

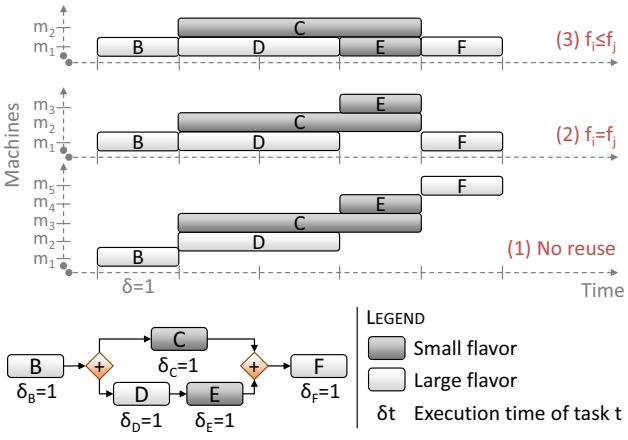

Figure 4: Resource Requirements through Time

(3) $f_i \leq f_j$: reuses larger machines for running smaller resource requirements, hence increasing the utilization of the same machines. For example, task E in Figure 4 requires small flavor, but was executed on a large machine m_1 as the latter was available during the requested time period. Similar to strategy (2), available machines are searched through (Line 1). However instead of accepting only the same size flavor, the larger flavors satisfy the criteria as well (Line 3). Note that this does not break the constraints as the larger machines can satisfy smaller resource requirements of tasks. Along with checking the availability of the machine (Line 4), the end time e_i is used to check if the current resource requirement can fit the time gap between the current moment and the s_k of the following requirement that is satisfied with the machine. Last four lines are equivalent to strategy (2).

Algorithm 2: Algorithms for machine reusage strategies

Input: getAvailable($\mathcal{M}, f_i, e_i$)
Output: m_j

```
1 foreach m_j ∈ M do                                   (2) (3)
2     if f_j = f_i then                                (2)
3         if f_j ≥ f_i then                            (3)
4             if a_j = true then                       (2) (3)
5                 if e_i ≤ next(s_k ∈ F_j)) then       (3)
6                     return m_j ;                     (2) (3)
7 m_j ← new m(f_i) ;                                   (1) (2) (3)
8 M ← M ⋃ {m_j} ;                                      (1) (2) (3)
9 return m_j ;                                         (1) (2) (3)
```

Resource deployment*:* Resource deployment includes deploying the required resources on top of the cloud infrastructure, e.g., deploying VMs on top of physical machines. Deployment is done by the cloud provider, who can benefit from business process metrics by achieving lower costs due to efficient resource planning. Resource (re)deployment algorithm is executed when existing resources are no longer needed or new resources are required, namely when a task starts or ends. Reusing the existing machines is done by simply deallocating the old ones and allocating new ones closer to data source, rather than re-imaging the machines and migrating them.

Here, we also discuss that the business process owner can also benefit from providing process metrics to the provider, as the provider can lower the costs and therefore offer a discount. On one hand, the trade-off for the owner is privacy/confidentiality of the business process, where the owner has to make some of the process information publicly available to the provider. On the other hand, the provider has to implement more complex deployment algorithms in order to process additional information. Algorithms such as [14] can be adapted for resource deployment and consolidation, while [15] can be used for data locality awareness. However, due to page restrictions, detailed algorithm is out of the scope of this paper.

VI. Cost estimation and optimization

Once the capacity and type of required resources are predicted, the costs for cloud deployment can be estimated. We consider two lease policies commonly practised:

- *Static* - a subscription type policy where all the resources are leased from the beginning, to ensure the whole execution of the business process.
- *Dynamic* - an on-demand policy where resources are leased when needed and released afterwards.

Cloud providers usually offer several pricing models that are more convenient for the both leasing policies. Pricing models can be defined as a tuple $z(v, c)$, where v is the duration of the lease, and c is the price of the resources for that time period. In order to calculate a total price for all machines, the total lease time of each machine is divided into a set of smaller slices $\mathcal{L} = \{l_1, l_2, ...\}$, where each slice $l_k(s_k, e_k, c_k)$ has a duration $e_k - sk = v$ and price c_k based on the applied price model. Then, all slice prices are summed for each machine using the following equation:

$$C_{total} = \sum_{j=1}^{|M|} \sum_{k=1}^{|\mathcal{L}_j|} c_k \tag{4}$$

While a number of slices for *static* pricing policy can be calculated in a straightforward manner by dividing total execution time of a process with v and ceiling the result, the dynamic one requires algorithmic solution as a machine can be leased and released at any time. More specifically, each resource requirement that is satisfied by a machine must fall within a lease period, where lease period is defined by the pricing model. Therefore, we use the following algorithm applied on each machine for generating slices.

Algorithm 3: Estimating total lease time

Input: v, c, $\mathcal{R}_j$, where $\mathcal{R}_j \subseteq \mathcal{R}$, $\mathcal{R} = r_1, ..., r_n$, and $r_i(s_i, e_i, f_i)$
Output: $\mathcal{L}_j$

1 $p = s_1$
2 **foreach** $r_i \in \mathcal{R}_j$ **do**
3 **if** $(s_i - p) \geq v$ **then**
4 $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup l_k(s_k \leftarrow p, e_k \leftarrow p + v, c)$; $p \leftarrow s_i$
5 **while** $(e_i - p) > v$ **do**
6 $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup l_k(s_k \leftarrow p, e_k \leftarrow p + v, c)$; $p \leftarrow t + v$
7 $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup l_k(s_k = p, e_k = p + v, c)$
8 **return** $\mathcal{L}_j$

The *start* time s_1 of the first resource requirement $r_1 \in \mathcal{R}_j$ satisfied by machine m_j is taken as the start time of the lease; i.e., the start for the first slice, and is taken as current time p (Line 1). All resource requirements are looped (Line 2) and checked for their start times s_i and end times e_i. If start time s_i of r_i is after or at the expiration of the current slice (Line 3), the slice is added to the set $\mathcal{L}_j$, while the start time of the next slice p is moved to s_i (Line 4). If end time e_i exceeds the duration of the current slice (Line 5), the slice is added to $\mathcal{L}_j$ (Line 6). Furthermore, start time of the next slice p is set to the end of the last slice. This is repeated until e_i is within the duration of the slice. The final slice is added in Line 7, and $\mathcal{L}_j$ is returned in Line 8.

Here, we introduce a hybrid approach (Algorithm 4) where a client, rather than using a single pricing model, combines all offered models in order to derive a better deal, i.e., a lower price, while still satisfying all constraints. Lease time is divided into a set of slices $\mathcal{L}_j$ for each machine m_j using Algorithm 3 for the shortest pricing model $z_1(v_1, c_1)$ and fed into Algorithm 4. Rest of the pricing models $\mathcal{Z}$ are sorted in ascending order by their duration v_i in Line 1. Pricing models are looped in Line 2 in order to consolidate shorter slices into longer ones if the price of the former is higher than of the latter.

Algorithm 4: Hybrid pricing with variable slices

Input: $\mathcal{Z} = \{z_2, z_3...\}$, where $z_i(v_i, c_i), \mathcal{L}_j$
Output: $\mathcal{L}_j$

1 $Sort(\mathcal{Z})$ by v_i in ascending order
2 **foreach** $z_i \in \mathcal{Z}$ **do**
3 **foreach** $l_k \in \mathcal{L}_j$ **do**
4 $empty(\mathcal{B})$
5 **while** $duration(\mathcal{B}) \leq v_i$ **do**
6 $\mathcal{B} \leftarrow \mathcal{B} \cup l_{k+w++}$
7 **if** $(price(\mathcal{B}) > c_i$ **then**
8 $\mathcal{L}_j \leftarrow \mathcal{L}_j - \mathcal{B}$
9 $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup l_k(s_k \leftarrow s_k, e_k \leftarrow s_k + v_i, c_i)$
10 **return** $\mathcal{S}_j$

For each slice l_k (line 3) buffer $\mathcal{B}$ is emptied. Furthermore, the buffer is filled with slices starting from the slice l_k while its total duration is shorter or equal to the duration v_i of the current pricing model (lines 5 and 6). If the total price of slices in the buffer is greater than the price c_i of the current pricing model (line 7), the slices in $\mathcal{B}$ are removed from $\mathcal{L}_j$ (line 8) and replaced with a new longer slice with duration v_i and price c_i of the current pricing model (line 9). Once all slices are looped, a next longer pricing model is taken and applied on slices in order to achieve further consolidation. The final result is a set of variable slices $\mathcal{L}_j$ returned for each machine. The total price for all machines is again calculated with Equation 4.

VII. Evaluation

In order to demonstrate the feasibility of our approach by showing how it can be used for finding the most efficient allocation strategy and the lowest cost pricing scheme for the owner, we implement process prediction, resource allocation and cost estimation phases in Java and apply them on the use case scenario presented in Section II. We generate 30 different configurations for the scenario with random execution times of the tasks, and random resource requirements in order to get more diverse results. We calculate both optimistic and critical paths for each of the configurations, and use the process metrics for allocating resources by applying all three reuse strategies.

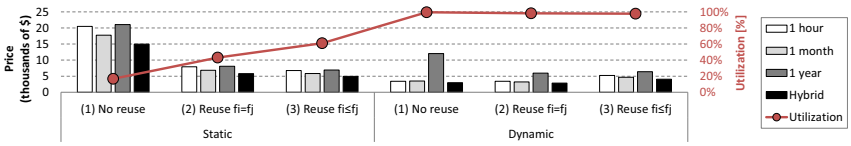

Figure 6: Resource allocation costs for the optimistic path and resource utilization for hybrid pricing scheme

Figure 5 shows average number and type of machines for each strategy and different paths. Generally, reuse strategy (3) requires least resources at one moment in time, as it can reuse already leased machines for satisfying new resource requirements, while (1) requires most as it always runs new machines. Additionally, a critical path represents a safer option as is can support any path. Consequently, it exhibits lower utilization due to redundant resources compared to the optimistic path. Due to space constraints, we present only results for the optimistic path further in the the evaluation.

Flavor	1 hour	1 month	1 year
micro	0.0150 $	0.0125 $	0.0100 $
small	0.0300 $	0.0251 $	0.0201 $
medium	0.0600 $	0.0501 $	0.0402 $
large	0.1660 $	0.1386 $	0.1137 $

Table I: Pricing models and prices for different flavors.

Furthermore, we utilize Amazon EC2 pricing models[1] (Table I) in order to estimate lease cost, namely we use on-demand model with duration of 1 hour, and reserved model with duration of 1 year. Moreover, we derive additional pricing model with duration of 1 month in order to show the diversity supported by our approach, and we apply all of them on resource allocation strategies. Figure 6 shows that costs for static leasing policy follow the amount of allocated machines (Figure 5), i.e., the more machines the higher the cost, as they are leased from the start until the end of a business process execution. Consequently, the resource utilization is thus low, namely 20-60% as shown in both Figures 5 and 6, while dynamic policy is always above 90%.

However, dynamic leasing policy provides lowest costs for reuse strategy (2), as only the same machines size are reused for satisfying new resource requirements. On one hand, reuse strategy (1) fails at sequential tasks, where machines used for the previous task have not yet expired, but are not reused for the next task, rather new ones are leased. On the other hand, reuse strategy (3) fails at reusing these machines as it reuses large ones (which are commonly more expensive) for satisfying small resource requirements, which results in higher total cost. Finally, in all tested settings the hybrid pricing scheme presented in this paper provides the best option for the customer, as it is able to combine best of all pricing models offered by the provider.

VIII. Related Work

In [16], an approach for estimating the minimum number of computing resources required for the execution of workflows within a given deadline is proposed. First, the approach was based on a balanced time scheduling heuristic algorithm (BTS), then it was extended using a partitioned balanced time scheduling algorithm (PBTS) [17]. The latter determines the best number of computing resources per time charge unit, while minimizing the total cost during the entire application lifetime. In comparison to our work, their approach considers workflows as simple directed graphs (DAG), and does not take into account the common control flow patterns; e.g.; choice or loops, and the data flow between the tasks. Finally, the approach does not consider the average execution time of tasks, but the deadline instead.

In [18], an empirical prediction model for adaptive resource provisioning in the cloud is proposed. The approach presents a prediction framework, that utilizes statistical models in order to predict resource requirements. The objective is to enable a proactive scaling in the cloud. The prediction is based on historical data through neural network based models. The approach deals with simple application and not business processes. In [19], a prediction-based dynamic resource allocation scheduling is proposed. The approach presents an online resource prediction and takes the VM workload variability into account. They consider that the VM resource demand is time-varying, and therefore use the ARIMA models [20] based on time series analysis to predict the CPU and memory usage. The prediction is based on statistical data collected from the resources and does not deal with workflow or business processes.

In [21], a forecasting solution for business processes is presented. The approach is based on two types of prediction: (i) prediction of metric values that active process instance will have at the end of their execution and (ii) prediction of aggregated metric values of future instances. This work is complementary to our work, and the predicted metrics can be used to enhance the resources allocation. In [22], only the challenges of how process metrics can be used for optimization, work queue management or resource allocation are presented. But no particular approaches about how to solve the challenges are given.

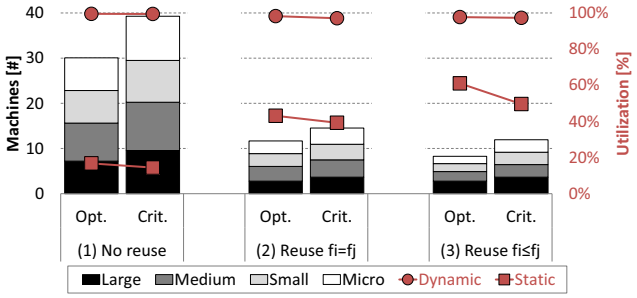

Figure 5: Number of machines allocated and their utilization for hybrid pricing scheme

[1]Amazon Web Services: http://aws.amazon.com/

In [7], a bi-criteria optimization for scheduling business processes in the cloud, while considering conflicting objectives was proposed. The approach considers the case where multiple instances run concurrently, and choses the optimized matching with the cloud resources. A similar approach has been proposed in [9], for optimizing resource provisioning costs, based on automatic leasing and releasing of cloud resources. In comparison with our approach, these works consider the actual deployment of resources by the cloud provider, while we adopt the perspective of the cloud customer and try to predict the cost of migrating a process to the cloud before the actual deployment.

IX. Conclusion and Future Work

In this paper, we presented an approach that helps a company estimating the migration cost of its business process into the cloud. In particular, we showed how process metrics and execution prediction can be utilized to chose the appropriate pricing model of a cloud provider. Additionally, we demonstrated how reusing leased resources can reduce the resource requirements and consequently the migration costs. The results proved that combining the pricing models offered by the same cloud provider leads to a lower costs for the process owner.

As future work, we plan to consider the dynamic aspects of business processes. In particular, we want to analyze the impacts of multiple concurrent instances of the same process on the resource requirements and accompanying costs. We also intend to investigate the provider's perspective and analyze how process metrics and prediction can be used for optimizing resource deployment. Finally, including other cloud attributes such as QoS can widen the feasibility of our approach.

Acknowledgment

The work described in this paper has been jointly funded through the Haley project (Holistic Energy Efficient Hybrid Clouds) as part of the TU Vienna Distinguished Young Scientist Award 2011, and the C^{3}Pro project I743 funded by the Austrian Science Fund (FWF).

References

[1] "Omg: Bpmn 2.0 by example, version 1.0." Online: http://www.omg.org/spec/BPMN/2.0/examples/PDF/, 2010.

[2] W. van der Aalst, A. ter Hofstede, and M. Weske, "Business process management: A survey," in *Business Process Management*, vol. 2678, pp. 1–12, 2003.

[3] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic business process management: State of the art and open challenges for BPM in the cloud," *Computers & Operations Research*, 2014.

[4] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data Knowl. Eng.*, vol. 70, no. 1, pp. 127–145, 2011.

[5] K. van Hee, A. Serebrenik, N. Sidorova, M. Voorhoeve, and J. van der Wal, "Scheduling-free resource management," *Data and Knowledge Engineering*, vol. 61, no. 1, pp. 59–75, 2007.

[6] A. Kumar, W. M. P. Van Der Aalst, and E. M. W. Verbeek, "Dynamic work distribution in workflow management systems: How to balance quality and performance," *J. Manage. Inf. Syst.*, vol. 18, no. 3, pp. 157–193, 2002.

[7] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Scheduling strategies for business process applications in cloud environments," *IJGHPC*, vol. 5, no. 4, pp. 65–78, 2013.

[8] S. Schulte, P. Hoenisch, C. Hochreiner, S. Dustdar, M. Klusch, and D. Schuller, "Towards process support for cloud manufacturing," in *Enterprise Distributed Object Computing Conference (EDOC)*, pp. 142–149, 2014.

[9] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal, "Self-adaptive resource allocation for elastic process execution," in *IEEE International Conference on Cloud Computing*, pp. 220–227, 2013.

[10] D. Schuller, M. Siebenhaar, R. Hans, O. Wenge, R. Steinmetz, and S. Schulte, "Towards heuristic optimization of complex service-based workflows for stochastic qos attributes," in *IEEE International Conference on Web Services (ICWS)*, pp. 361–368, 2014.

[11] W. Fdhila, M. Dumas, C. Godart, and L. Garcia-B., "Heuristics for composite web service decentralization," *Software and Systems Modeling*, vol. 13, no. 2, pp. 599–619, 2014.

[12] S. Zhou, X. Zhang, B. Chen, and S. L. van de Velde, "Tactical fixed job scheduling with spread-time constraints," *Computers & Operations Research*, vol. 47, 2014.

[13] A. W. Kolen and L. G. Kroon, "License class design: complexity and algorithms," *European Journal of Operational Research*, vol. 63, no. 3, pp. 432 – 444, 1992.

[14] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[15] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," in *Cluster, Cloud and Grid Computing (CCGrid), 11th IEEE/ACM International Symposium on*, pp. 295–304, May 2011.

[16] E.-K. Byun, Y.-S. Kee, J.-S. Kim, E. Deelman, and S. Maeng, "Bts: Resource capacity estimate for time-targeted science workflows," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 848–862, 2011.

[17] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1011–1026, 2011.

[18] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Gener. Comput. Syst.*, vol. 28, pp. 155–162, Jan. 2012.

[19] Q. Huang, K. Shuang, P. Xu, J. Li, X. Liu, and S. Su, "Prediction-based dynamic resource scheduling for virtualized cloud systems," *Journal of Networks*, vol. 9, no. 2, 2014.

[20] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[21] M. Castellanos, N. Salazar, F. Casati, U. Dayal, and M.-C. Shan, "Predictive business operations management," in *Databases in Networked Information Systems*, vol. 3433, pp. 1–14, 2005.

[22] M. Castellanos, F. Casati, M. Sayal, and U. Dayal, "Challenges in business process analysis and optimization," in *Technologies for E-Services*, vol. 3811, pp. 1–10, 2006.