3.1 Experience management for task placements in a cloud.

Eric Kübler [12] and Mirjam Minor[13]

Abstract: The execution of workflows in a cloud is more and more popular, and new business concept based on this combination emerge. However, the task to control a cloud in such a way, that the rented cloud resources match the requirements for the currently executed workflows is difficult. Simple solutions struggle with over-, and under-provisioning problems or lack the needed flexibility for the new business concepts. A smart concept for cloud management should use knowledge about the characteristic of the executed task to improve the resource utilization of the cloud. In this paper we present our approach for a CBR based concept for cloud management that reuses experience on proper cloud configurations. We introduce our similarity function for task placements in a cloud and illustrate the approach with some sample workflows form the music mastering domain.

Keywords: Cloud, CBR, MAC/FAC

1 Introduction

Cloud computing offers nearly infinite resources on-demand on a pay as you go pricing model [MG]. Therefore it is not surprising that more and more business models are based on the use of cloud computing. One field that can benefit from cloud computing is the execution of workflows. A workflow is defined by the Workflow Management Coalition [Co] as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". A tasks, also called activity is defined as follows: {*A process [...] consists of one or more activities, each comprising a logical, self-contained unit of work within the process. An activity represents work, which will be performed by a combination of resource (specified by participant assignment) and/or computer applications (specified by application assignment)*"[Co, p.14]. The execution of a workflow in a cloud means that the cloud provides the workflow with resources and software that are required to complete the tasks of the workflow. This could be for example the provision of a virtual machine with an installed office software for a task during that a human actor has to write a letter. Another example is the deployment of a web

[12] Goethe University, Department of Business Informatics, 60325 Frankfurt am Main, ekuebler@cs.uni-frankfurt.de

[13] Goethe University, Department of Business Informatics, 60325 Frankfurt am Main, minor@cs.uni-frankfurt.de

server with a certain web service that renders images, for a task that automatically processes images. We call the assignment of tasks to its cloud resources a task placement.

The creation of a task placement can be trivial, if every task just get its own cloud resources. However, this is very ineffective and will lead to over-provisioning of resources and consequently to unnecessary costs for the user. The problem of finding an optimal assignment of tasks to cloud resources, where the resource utilization is optimal, can be modeled as a bin packing problem, which is NP complete. The problem is even harder, because the bins (in this case virtual machines or containers like docker) can vary in their size (different amount of available memory, disk space ...), where the objects (the tasks) not just have one value for their needed requirement (size), but can have several different requirements (for CPU, memory, Linux Kernel version, for example), that all need to be fulfilled. Further, it is not desirable to execute all tasks with the same requirements on the same resources. This will lead to a state, where the execution of all tasks are slowed down. This so-called under-provisioning can lead to future problems (for example deadline problems or frustrated users) and should be avoided too. It is required to find a good balance between over- and under-provisioning of resources [Ar]. To find such a balance is generally a problem. In general, the management of resources is an important aspect for cloud computing [Ba]. There are plenty of approaches for cloud management in the literature.

The simplest methods to provide resources is the static way. This means, the system does not adjust itself to a changing situation. Obviously, this will lead to under- or over-provisioning [SD]. A more dynamic approach is required. The range for such approaches is great and spans from rather simple, rule-based approaches such as observations on the number of open connections [PM] to complex algorithms [Qu].

All of the above approaches have the problem, that they are not very flexible when it comes to a change of the used cloud or workflow management system. Many do not consider available knowledge about the tasks and cloud configuration. Thus, it takes quite long to compute a proper task placement. To handle this complex problem in a reasonable time and to avoid over- and under-provisioning, it is necessary to use the knowledge about the tasks to manage the cloud resources properly.

In this paper we introduce a CBR approach for task placements in cloud computing, that uses knowledge about the tasks and the workflow structure. The idea of CBR is that similar problems have similar solutions [AP]. The idea of using CBR for cloud management is not new. The work of Maurer et al. [MBS] applies CBR to implement automatic cloud management. Aamodt and Plaza describe a case as follows: "*In CBR terminology, a case usually denotes a problem situation. A previously experienced situation, which has been captured and learned in a way that it can be reused in the solving of future problems*" [AP]. In cloud management, this is to reuse problem solving knowledge on the cloud resources. In this work, a case is a task placement with problems. These problems could be for example violated Service Level Agreements (SLAs), missing web services or unused resources. A solution is a new task placement, that solves the problems. A sample solution is a newly started VM with the missing web service, the shutdown of the unused resources or the increasing of the resources to avoid future SLA violations. To retrieve similar problems (cases), a similarity function determines the similarity between two cases. Due to the fact that CBR only requires the similarity function to receive other, similar problems and their similar solution, the time and computational effort is relatively low. We introduce our similarity function for task placements in cloud computing, discuss in short some alternatives within the function and give an example how the function will be applied. The CBR approach will be embedded into our Workflow Cloud Framework (WFCF). WFCF is a connector based integration framework to integration workflow management tools with cloud computing.

2 Similarity of task placements

In this section we describe the structure of our cases, the MAC/FAC method and the similarity function for our cases.

2.1 Case representation

As mentioned before, a task placement is the assignment of the currently active tasks to cloud resources. Fig. 1 gives an example. In this example, the tasks *Task2* and *Task3* are active, where *Task 1* is already done. Task3 is assigned to a container named *CON2* where *Task2* is assigned to a VM named *VM2* and *Task4* is assigned to *CON3*. Assigned means that *CON2*, *CON3* and *VM2* host the software that the tasks needs to execute, for example a web service or an Office Suite. The task calls the web service, or the user uses a remote desktop connection to work with the Office Suite.

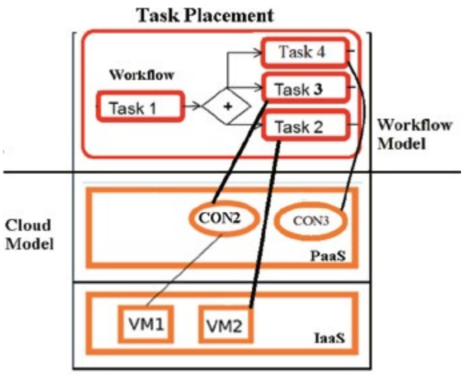

Fig. 1: A simple illustration of a task placement with a task assigned to a VM and another to a VM.

A case is a task placement with some problems. This could be for example violations of Service Level Agreements (SLA) or the violation of internal constraints for example that there should be no unused cloud resources. The solution is a new task placement that solves the problems.

2.2 Relevance of case parts for retrieval

Next, we discuss what should be important for the similarity of two task placements. A task placement has plenty of parameters that could be considered for the similarity. In this work, we consider a cloud node either as a virtual machine (VM), or a container. Both have a set of resources, this can include, but is not limited to, CPU, GPU, memory, disc storage, network capacity and different kinds of installed software. For comparison, even two cloud nodes with the same set of resources (same CPU, installed software etc.) could considered as different, if the resource utilization is different. For example, a cloud node with 4GB of memory and a utilization of 100% of the memory, should be more similar to a node with 4GB memory and 90% utilization than a node with 4GB memory and a utilization of 10%. Though, a VM and a container share some attributes, it makes a difference if a Node is a VM or a Container. For example, a container can be migrated relatively easy from one VM to another, even if the VMs are hosted on different cloud providers. This is not so easy and sometimes even impossible for an entire VM. So the solution for a VM can not be always the migration to an other host, this is possible for a container. In this case, it is necessary to propagate the new URL or IP address to the workflow.

Therefore, to compare two cloud nodes, it is important to distinguish whether it is a VM or container, to know the set of resources and the utilization of the hardware resources.

More important than the cloud nodes are the tasks that are currently executed with the cloud nodes. One of the goals of our WFCF framework is the careful use of cloud resources for the execution of workflow tasks. Without any task, there is generally no need for any cloud nodes. That means, that the driving force of the task placement are the tasks. To determine the needed cloud resources for a task, we introduced in one of our previous works the concept of task characteristics in cloud computing [KM]. In short, the idea is to label tasks with its needs and give a hint of the foreseeable resource usage. In our current work, we extend the idea of characteristics so that for example the characteristic "compute intensive" now has a value of 0 to 4 to indicate how intensive the task uses the CPU and not just a binary value of 0 or 1 to determine if the task is compute intensive or not. Other characteristics that determine if a task is long or short running, were replaced by a values that contain the minimal, maximal and average execution time.

Another important aspect should be the problems that a placement has. As mentioned before, this could be for example violations of Service Level Agreements (SLA) or the violation of internal constraints for example that there should be no cloud node active that is not in use.

2.3 MAC/FAC approach

It is clear that this can lead to many parameters to compare for similarity. And at this point we even haven't discussed the similarity of sets of tasks and sets of cloud nodes. For performance reason, we use the MAC/FAC Principle as introduced in [FGL], to distribute the effort. The idea is to collect fast a set of few promising case candidates (MAC step) and investigate the similarity of the candidates in more detail in a second step (FAC step).

As mentioned before, the most important aspect for similarity are the tasks. Therefore in the MAC step we compare the currently active tasks of the problem case with the currently active tasks in cases stored in the case base. The currently active tasks is represented by a set of vectors. At this point it is tempting to define a similarity for two tasks. Since tasks are Vectors, the hamming distance or the euclidean distance can be used to determine similarity. But in this case, we have to consider the similarity between two sets of vectors. There are some metrics for defining the similarity between two sets, for example the Hausdorff metric [HKR93] or the sum of minimum distances [EM97]. The problem with these heuristics is, that it is very easy to create a case where the similarity is very high but the sets are by intuition very dissimilar. For example in one set are only equal tasks. This could be for example a set with only a single type of image rendering tasks and the other set contains different tasks but one rendering task and a mapping did map all rendering task of the first tasks to the single rendering tasks of the second set, then the similarity would be 1 for these two sets. of course, this is not desired. Another option is not to use a metric for building a single mapping, but to build all possible mappings for the vectors and chose the mapping with the minimum weight like the Kuhn-Munkers algorithm [Ku55, Mu57]. Though, this method is very compute intensive as mentioned in [AFS93].

Our solution for a fast approach that is not that much vulnerable for special cases, uses an intersection of the task sets to determine the similarity. Let T_1 and T_2 be a Set of Tasks, then is the function $sim_T(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$. The benefit of this function is, that it covers the difference between the size of the sets, as well as the actually equal tasks. This will help in the FAC step to find more relevant task placements in a reasonable amount of time. Beside the tasks, it is important to consider the problems that a task placement has. As mentioned before, this could be SLA or constraint violations. The SLAs and constraints are stored in a a vector, are ordered by name and contains the number of violations for each SLA or constraint. We call this an SLA vector. To compare two SLA vectors, we first make sure that both vectors have the same parameters, which means the same SLAs and constraints. Let $slav_1$ an SLA vector with the sla $slav1 = \{SLA1_1, SlA2_1\}$ and $slav_2 = \{SlA2_2, SLA3_2\}$ with the values $SLA1_1 = 1, SLA2_1 = 3, SLA2_2 = 2, SLA3_2 = 1$. In a first step we add in both SLA vectors the missing parameters but set their value to 0. The new vectors are $slav'_1 = \{SLA1_1, SlA2_1, SLA3'_1\}$ and $slav'_2 = \{SLA1'_2, SlA2_2, SLA3_2\}$ with $SLA1'_1 = 1, SLA1'_2 = 0, SLA2_1 = 3, SLA1_2 = 2, SLA3_2 = 1, SLA3'_1 = 0$. Now we can compute the euclidean distance between two SLA vectors. Let $q_i \in SLA1$ and $p_i \in SLA2$, than is the euclidean distance for the SLA vectors

$$d_{sla}(p,q) = \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} (q_i - p_i)^2} .$$

The similarity function is now:

$$sim_{sla}(slav_1, slav_2) = \frac{1}{1 + d_{sla}(slav_{1,} slav_2)} .$$

The MAC step is a combination of the similarity of the current active tasks, and the problems, therefore we chose an aggregated similarity function

$$sim_{mac}(TP_{1,} TP_2) = \frac{sim_T(T_{1,} T_2) + sim_{sla}(slav_{1,} slav_2)}{2} ,$$

where TP_1 and TP_2 are task placements with $T_1, slav_1 \in TP_1$ and $T_2, slav_2 \in TP_2$. Depending on the test results, we may add some weights to the components of the similarity function, but for now we consider the tasks and the problems as equally important.

After a fast determination of promising candidates, the FAC step compares the candidates with the current problem situation in more detail. Here is the placement of the tasks, the resources of the cloud nodes and their utilization important, as well as the question, which tasks are to be executed next.

2.4 Similarity of tasks in placements

As shown before in Fig. 1, a task placement can be seen as a tree, if all VMs and containers that are not related to a VM, are assigned to an abstract "hardware" node, as shown in Fig. 2. This tree is unordered and labelled, where the labels are the resources that a cloud node contains. Graph isomorphism is NP complete as well known, but to compute the edit distance between two unordered labelled trees is also NP complete, as shown in [Zh96].

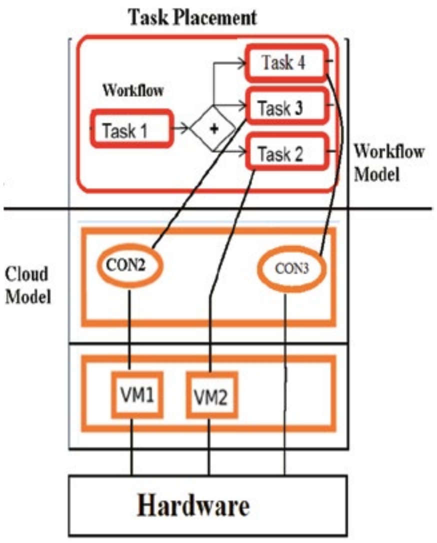

Fig. 2: Task placement as tree with hardware as the root and the tasks as leafs

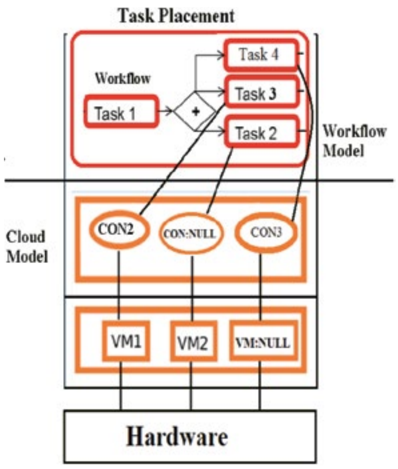

Fig. 3: Task placement as tree with the hardware as root and abstract nodes for a more generalized model

The problem is not easier, if we alter the tree and add new nodes with null values for the labels, to get a more generalized structure as shown in Fig. 3.

Instead we compare the vector of tasks and their related cloud nodes. We call this a *task_cloud_vector*. Such a vector contains a task, a container and a VM. As mentioned earlier, it is important not to compare containers with VMs.

A task within the task_cloud_vector is represented by a sub-vector of parameters with their values, the characteristics and a set of tasks that could be executed next. The idea of this set is to consider the next tasks for a foresight of the workload that coming next. Therefore we use once again the euclidean distance to determine the similarity between the two characteristic vectors and the similarity for intersection for the next tasks. Let *TASK1, TASK2* are set of tasks and *t1, t2* are vectors of characteristics with *t1 ϵ TASK1, t2 ϵ TASK2*. The euclidean distance is then defined as

$$d_{task}(t1,t2)=\sqrt{\frac{1}{n}\times\sum_{i=1}^{n}(t2_i-t1_i)^2}\ .$$

Let *tn1 ϵ TASK1, tn2 ϵ TASK1* the set with the next task. The similarity function is now:

$$sim_{ntask}(TASK1,TASK2)=\frac{1}{1+d_{task}(T_{1,}T_2)}+\frac{|tn_1\cap tn_2|}{|tn_1\cup tn_2|}\ .$$

As shown below, we need a distance function for a special algorithm. Therefore we build the distance function with:

$$d_{ntask}(TASK1,TASK2)=(|TASK1\cup TASK2|)-(|TASK1\cap TASK2|)\ .$$

A cloud node contains its hardware resources, the utilization of the hardware resources and additional software or information, which are stored as a set of tags. Let *cn = (r, u,tag)* describe a cloud node, *r* is a vector of hardware resources (for example 4 cpu cores, 16GB Memory ect), *u* is a vector of resource utilization in percentage and *tag* is a set of tags (for example *tag = {windows8, tomcat7, jre7}*). For the distance between the resources, we use once again the euclidean distance $d_r(r1,r2)$, as well as for the utilization $d_u(u1, u2)$. The similarity functions is then again

$$sim_r(r1,r2)=\sqrt{\frac{1}{n}\times\sum_{i=1}^{n}(r2_i-r1_i)^2}\quad \text{and}\quad sim_u(u1,u2)=\sqrt{\frac{1}{n}\times\sum_{i=1}^{n}(u2_i-u1_i)^2}$$

To determine the similarity of the set of tags we build the intersection and compare it with the merged sets:

$$sim_{tags}(tag1,tag2)=\frac{|tag_1\cap tag_2|}{|tag_1\cup tag_2|}\ .$$

Similar to the next task we need a distance function for the tags:

$$d_{tags}(tag1\,,\,tag2) = (|tag1 \cup tag2|) - (|tag1 \cap tag2|)$$

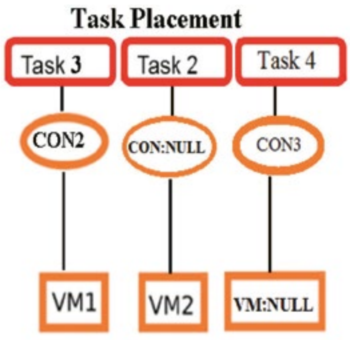

Fig. 4: Task placement as paths

The overall distance function for two task_cloud_vectors is:

$$d_{tcv}(tcv1\,,tcv2) = d_{tags}(tags1\,,tags2) + d_u(u1\,,u2) + d_r(r1\,,r2) + d_{ntask}(t1\,,t2)$$

Similar to the MAC step we might add weights in the future.

2.5 Similarity of entire task placements

After defining the similarity for two task_cloud_vectors, the next step is do define the similarity between two sets of vectors. For this we have chosen the Kuhn-Munkers algorithm (also called Hungarian algorithm) as described in [Ku55, Ku57]. This algorithm builds a minimum weight mapping for bipartite graphs, our in this case between two sets of vectors, where the edge weight is the distance between two task_cloud_vectors. In a first step a distance matrix must be built, that contains the distance from each task_cloud_vector in the first set to each task_cloud_vector in the second set. The Kuhn-Munkers algorithm requires a square matrix. If the two sets have a different number of vectors, we add to the smaller set dummy vectors and set their edge weight to infinite. Because of the strict selection in the MAC step, based on the intersection of the tasks, there should be not many dummy vectors in our matrix.

After building the matrix, the Kuhn-Munkers algorithm successively improves the mapping between both sets. We will show a running example in the following section. To determine the similarity between two sets of vectors, after Kuhn-Munkers has finished, we build the sum of the edges between the sets is build. Since this is a distance function, the similarity function for Kuhn-Munkers is

$$sim_{km}(tcv1, tcv2) = 1 - \frac{1}{1 + d_{km}(tcv1, tcv2)}.$$

In [AFS93] is mentioned, that the run time of Kuhn-Munkers is *O(n^4)* where *n* is the number of task_cloud_vectors, but that the run time can be improved to *O(n^3)*. This is very compute intensive, in particular this has to be computed for each candidate, selected during the MAC step, therefor the selection of the MAC step should be very strict.

3 Illustrating Example

In this section we give a running example of our MAC/FAC similarity function. We use music mastering workflow as our application domain. An example workflow is given in fig. 5. This workflow contains several different tasks. The tasks sample rate, limiter, normalize, channels, fading and sample size require all a special web service for their own. For example, the task limiter needs the limiter web service (limiter_ws), where the task channels needs the channels web service (channel_ws) and so on.

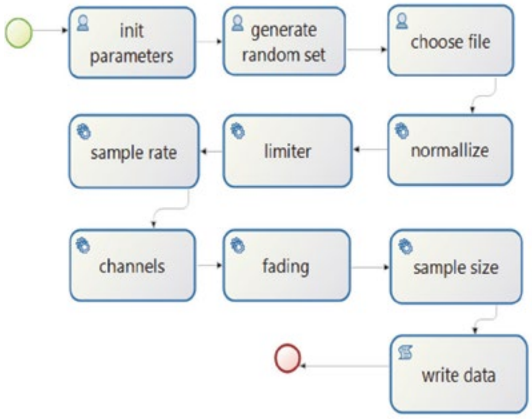

Fig. 5: Workflow form the music mastering domain

For our current problem case let us assume, that there are two workflow instances currently executed and therefore two tasks currently active. Fig 6 shows the placement for the problem situation. The labels *vector1-3* are important in the FAC step, when we compare the vectors. The problems in this case are, that the task *limiter* has no assigned cloud resources, where *vm3* has no assigned task. This is also noted in the SLA_vector. The notation for the MAC step is for the problem case:

problem = (task = (channels,limiter), sla_{vector} = (missing web service, unused resources)).

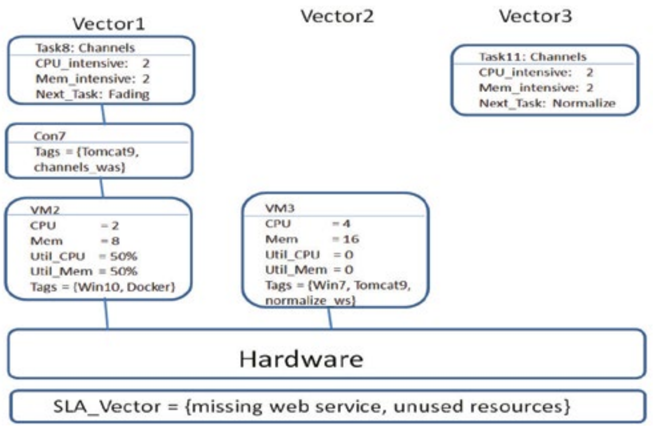

Fig. 6: Task placement of the problem case

In this example the Case Base has stored four old cases, called *case1, case2, case3, case4*. For the MAC step we first look at the currently active tasks and the SLA_vector of this cases.

- *case1 = (task = (limiter,normalize), sla_{vector} = (underprovisioning))*
- *case2 = (task = (limiter,channels), sla_{vector} = (missing webservice))*
- *case3 = (task = (normalize, fading), sla_{vector} = (missing web service, unused resources))*
- *case4 = (task = (sample_rat), sla_{vector} = (underprovisiong))*

Table 1 shows the computed similarity of the four cases to the problem case.

Case name	similarity
case1	5
case2	0,75
case3	0,75
case4	0

Table 1: Similarity of the cases in the case base to the problem case

In this example, *case2* and *case3* are chosen for a detail analysis in the FAC step.

The core of the FAC step is the analysis of the task_cloud_vectors and finding a fitting mapping. Fig. 7 show the task_cloud_vectors for *case2* and *case3*. In a first step we add to the problem case and the cases stored in the case base, null_tasks, null_container and null_VM to complete all Task_cloud_Vectors. Then we build the similarity matrix for the two cases with the problem case.

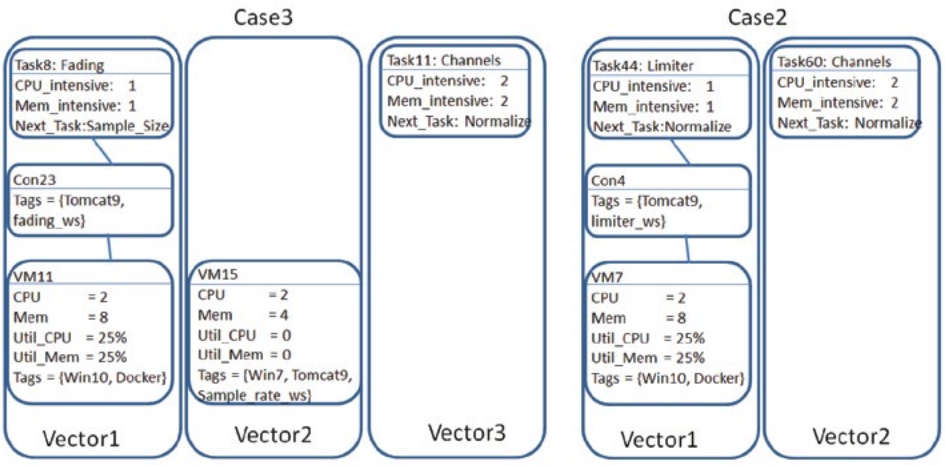

Fig. 7: The task_cloud_vectors stored in the case base

As mentioned before, The Kuhn-Munkers algorithm needs a square matrix, therefore we added to *case2* a third (dummy) vector, and set the distance to 999, because the distance to the dummy vectors should be infinite and therefore not preferable for the algorithm. The Kuhn-Munkers algorithm next search minimum distance for each column. Next, the algorithm reduces the value of each element in each column by the column minimum. Next, the row minimum is formed, similar to the column minimum and each element for each row is again reduced by the row minimum. In the next step the algorithm searches a combination of 0, so that each row and each column only contains one 0. Can such a combination found, that this is the optimal mapping and the algorithm is done. Else, if no valid mapping can be found at this point, the algorithm next mark the critical rows and columns and determine a minimum. Table 2 and 3 shows the distance matrix for *case2* and *case3* as well as the result of the Kuhn-Munkers algorithm for our example cases. For example: in *case2*, vector1 was mapped to vector1 of the problem case, therefore the distance was bordered. The sum of chosen mapping is for *case2 17,5 + 0 + 999 = 1016,5* where the distance for *case3* to the problem case is *17,5 + 12 + 0 = 29,5*. The most similar case to the problem case is also case3, which make sense. Both cases have much in common and only small difference in details.

Vectors	Probcase vector1	Probcase vector2	Probcase vector3
Vector1	**17,5**	25	22
Vector2	12,5	13	**0**
Vector3	999	**999**	999

Table 2: Distance from the vectors of case2 to the problem case

Vectors	Probcase vector1	Probcase vector2	Probcase vector3
Vector1	**17,5**	25	22
Vector2	41	**12**	14
Vector3	29,5	28	**0**

Table 3: Distance from the vectors of case3 to the problem case

4 Conclusion

In this paper we presented our MAC/FAC approach for task placements in cloud computing and illustrated it with an example. The basic idea is to reuse problem solving knowledge from past task placements in order to mend SLA violations in recent task placements. Our MAC/FAC approach provides an efficient means to retrieve matching task placements. The concept of using this knowledge management approach for cloud management is promising. An illustrating example from the music mastering domain achieved good retrieval results. Even for a relatively complex query the results have been plausible; the best matching case indeed was very useful to solve the sample problem case. The insights from this realistic scenario serve as a preliminary proof-of-concept. However, experiments with a larger case base than in the illustrating example are required to provide further evidence for the feasibility of the approach. The improvement of the similarity functions by weights is a further issue. Our next steps are to implement the similarity functions and to conduct more experiments with WFCF.

1. Bibliography

[AFS93] Agrawal, Rakesh; Faloutsos, Christos; Swami, Arun: Efficient similarity search in sequence databases. In: International conference on foundations of data organization and algorithms. Springer, pp. 69–84, 1993.

[AP] Aamodt, A.; Plaza, E.: Case-based reasoning: Foundational issues, methodological variations and system approaches. 7(1):39–59.

[Ar] Armbrust, Michael; Fox, Armando; Griffith, Rean; Joseph, Anthony D.; Katz, Randy; Konwinski, Andy; Lee, Gunho; Patterson, David; Rabkin, Ariel; Stoica, Ion; Zaharia, Matei: A view of cloud computing. 53(4):50–58.

[Ba] Baun, C.; Kunze, M.; Nimis, J.; Tai, S.: Cloud Computing - Web-Based Dynamic IT Services. Springer.

[Co] Workflow management coalition glossary & terminology, last access 05-23-2007.

[EM97] Eiter, Thomas; Mannila, Heikki: Distance measures for point sets and their computation. Acta Informatica, 34(2):109–133, 1997.

[FGL] Forbus, Kenneth D.; Gentner, Dedre; Law, Keith: MAC/FAC: A Model of Similarity-Based Retrieval. 19(2):141–205.

[HKR93] Huttenlocher, Daniel P.; Klanderman, Gregory A.; Rucklidge, William J: Comparing images using the Hausdorff distance. IEEE Transactions on pattern analysis and machine intelligence, 15(9):850–863, 1993.

[KM] Kübler, Eric; Minor, Mirjam: Towards a Case-based Reasoning Approach for Cloud Provisioning. In: CLOSER 2016 - Proceedings of the 6th International Conference on Cloud Computing and Services Science, Rome, Italy 23-25 April, 2016. volume 2. SciTePress, pp. 290–295.

[Ku55] Kuhn, Harold W: The Hungarian method for the assignment problem. Naval research logistics quarterly, 2(1-2):83–97, 1955.

[MBS] Maurer, Michael; Brandic, Ivona; Sakellariou, Rizos: Adaptive resource configuration for Cloud infrastructure management. 29(2):472–487.

[MG] Mell, Peter; Grance, Timothy: The NIST Definition of Cloud Computing. p. 7-13.

[Mu57] Munkres, James: Algorithms for the assignment and transportation problems. Journal of the society for industrial and applied mathematics, 5(1):32–38, 1957.

[PM] Pousty, Steve; Miller, Katie: Getting Started with OpenShift. Ö'Reilly Media, Inc.".

[Qu] Quiroz, Andres; Kim, Hyunjoo; Parashar, Manish; Gnanasambandam, Nathan; Sharma, Naveen: Towards autonomic workload provisioning for enterprise grids and clouds. In: Grid Computing, 2009 10th IEEE/ACM International Conference on. IEEE, pp. 50–57.

[SD] Shoaib, Yasir; Das, Olivia: Performance-oriented Cloud Provisioning: Taxonomy and Survey. Abs/1411.5077.

[Zh96] Zhang, Kaizhong: A constrained edit distance between unordered labeled trees. 15(3):205–222, 1996.