Multi-dimensional Resource Allocation for Data-intensive Large-scale Cloud Applications

Foued Jrad[1], Jie Tao[1], Ivona Brandic[2] and Achim Streit[1]
[1]*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany*
[2]*Information Systems Institute, Vienna University of Technology, 1040 Vienna, Austria*
{*foued.jrad, jie.tao, achim.streit*}*@kit.edu, ivona@infosys.tuwien.ac.at*

Keywords: Cloud Computing, Multi-Cloud, Resource Allocation, Scientific Workflow, Data Locality.

Abstract: Large scale applications are emerged as one of the important applications in distributed computing. Today, the economic and technical benefits offered by the Cloud computing technology encouraged many users to migrate their applications to Cloud. On the other hand, the variety of the existing Clouds requires them to make decisions about which providers to choose in order to achieve the expected performance and service quality while keeping the payment low. In this paper, we present a multi-dimensional resource allocation scheme to automate the deployment of data-intensive large scale applications in Multi-Cloud environments. The scheme applies a two level approach in which the target Clouds are matched with respect to the Service Level Agreement (SLA) requirements and user payment at first and then the application workloads are distributed to the selected Clouds using a data locality driven scheduling policy. Using an implemented Multi-Cloud simulation environment, we evaluated our approach with a real data-intensive workflow application in different scenarios. The experimental results demonstrate the effectiveness of the implemented matching and scheduling policies in improving the workflow execution performance and reducing the amount and costs of Intercloud data transfers.

1 INTRODUCTION

With the advantages of pay-per-use, easy-to-use and on-demand resource customization, the Cloud computing concept was quickly adopted by both industry and the academia. Over the last decade, a lot of Cloud infrastructures have been built which distinguish themselves in the type of offered service, access interface, billing and SLA. Hence, today Cloud users have to make manual decisions about which Cloud to choose in order to meet their functional and non-functional service requirements while keeping the payment low. This task is clearly a burden for the users because they have to go through the Web pages of Cloud providers to compare their services and billing policies. Furthermore, it is hard for them to collect an maintain the all needed information from current commercial Clouds to make accurate decisions.

The raising topic of Multi-Cloud addresses the interoperability as well the resource allocation across heterogeneous Clouds. In this paper we focus from a user perspective on the latter problem, which has been proved to be NP-hard. A challenging task for the resource allocation is how to optimize several user objectives like minimizing costs and makespan while fulfilling the user required functional and non-functional SLAs. Since in Multi-Cloud data transfers are performed through Internet between datacenters distributed in different geographical locations, another challenge is how to distribute the workloads on these Clouds in order to reduce the amount and cost of Cloud-to-Cloud (Intercloud) data transfers.

Today, related work on Cloud resource allocation is typically restricted to optimizing up to three objectives which are cost, makespan and data locality (Deelman et al., 2008). Attempts to support other objectives (e.g. reliability, energy efficiency) (Fard et al., 2012) cannot be applied directly to Multi-Cloud environments. However, the support for more SLA constraints like the Cloud-to-Cloud latency and Client-to-Cloud throughput, which both have high importance for data-intensive Mutli-Cloud applications, is still missing.

Motivated by the above considerations, we propose a multi-dimensional resource allocation scheme to automate the deployment of large scale Multi-Cloud applications. We facilitate the support of multi-

ple generic parameters by applying a two stage allocation approach, in which the target Clouds are selected using an SLA-based matching algorithm at first and then the application workloads are distributed to the selected Clouds using a data locality driven scheduling policy. For the SLA-based matching, we adopted from the economic theory a utility-based algorithm which scores each non-functional SLA attribute (e.g. availability, throughput and latency) and then calculates the user utility based on his payment willingness and the measured Cloud provider SLA metrics. Overall, our optimization is done with four objectives: makespan, cost, data locality and the satisfaction level against the user requested non-functional SLA.

In order to validate our resource allocation scheme, we investigate in this paper the deployment of data-intensive Multi-Cloud worlflow applications. Our idea of supporting Multi-Cloud workflows comes from the observation of following scenario: A customer works on several Clouds and stores data on them. There is a demand of jointly processing all of the data to form a final result. This scenario is similar to the collaborative work on the Grid. For example, the Worldwide LHC Computing Grid (WLCG) [1] involves more than 170 computing centers, where the community members often work on a combined project and store their data locally on own sites. A workflow application within such a project must cover several Grid resource centers.

We evaluated our allocation scheme using a Mutli-Cloud workflow framework, we developed based on the CloudSim (Calheiros et al., 2011) simulation toolkit. The reason for applying a simulator rather than real Clouds is that our evaluation requires different Cloud platforms with various properties in infrastructure Quality of Service (QoS). In addition, the simulator allows us to fully validate the proposed concept with various scenarios and hence to study the developed resource matching and data locality optimization schemes.

The experimental results show that our multi-dimensional allocation scheme offers benefits to users in term of performance and cost compared to other policies. Overall, this work makes the following contributions:

1. A multi-dimensional resource allocation scheme for large scale Multi-Cloud applications.
2. An efficient utility-based matching policy to select Cloud resources with respect to user SLA requirements.
3. A data locality driven scheduling policy to minimize the data movement and traffic cost.
4. An extensive simulation-based evaluation with a real workflow application.

The remainder of the paper is organized as follows: Section 2 presents the related work on data locality driven workflow scheduling. Section 3 describes the Multi-Cloud workflow framework used to validate our approach. Section 4 and Section 5 describe the functionality of the matching and scheduling algorithms implemented in this work. Section 6 presents the simulation environment and the evaluation results gathered from the simulation experiments. Finally, Section 7 concludes the paper and provides the future work.

2 RELATED WORK

Over the past decade, the scheduling of data-intensive workflows is emerged as an important research topic in distributed computing. Although the support of data locality have been heavily investigated on Grid and HPC, only few approaches apply data locality for Cloud workflows. A survey of these approaches is provided in (Mian et al., 2011). In this section we focus on works dealing with data locality driven workflow scheduling in Multi-Cloud environments.

The authors in (Fard et al., 2013) adopted from the game theory a dynamic auction-based scheduling strategy called BOSS to schedule workflows in Multi-Cloud environments. Although their conducted experiments show the effectiveness of their approach in reducing the cost and makespan compared to traditional multi-objective evolutionary algorithms, the support for data locality is completely missing in their work. Szabo et al. (Szabo et al., 2013) implemented a multi-objective evolutionary workflow scheduling algorithm to optimize the task allocation and ordering using the data transfer size and execution time as fitness functions. Their experimental results prove the performance benefits of their approach but not yet the cost effectiveness. Although the authors claim the support for Multi-Cloud, in their evaluation they used only Amazon EC2-based [2] Clouds. Yuan et al. (Yuan et al., 2010) proposed a k-means clustering strategy for data placement of scientific Cloud workflows. Their strategy clusters datasets based on their dependencies and supports reallocation at runtime. Although their simulation results showed the benefits of the k-means algorithm in reducing the number of data movements, their work lacks the evaluation of the execution time and cost effectiveness. In (Pandey et al., 2010) the authors use particle swarm

[1] http://lcg.web.cern.ch

[2] http://aws.amazon.com/ec2

optimization (PSO) techniques to minimize the computation and traffic cost when executing workflows on the Cloud. Their approach is able to reduce execution cost while balancing the load among the datacenters. The authors in (Jin et al., 2011) introduced an efficient data locality driven task scheduling algorithm called BAR (Balance and reduce). The algorithm adjusts data locality dynamically according to the network state and load on datacenters. It is able to reduce the completion time. However, it has been tested only with MapReduce (Dean and Ghemawat, 2008)-based workflow applications.

The examination of the previous mentioned works shows that the support of data locality in the scheduling improves the performance and minimizes the cost of workflow execution on Cloud. A more clever solution allowing the support of more generic workflow applications is to implement task scheduling as part of a middleware on top the Cloud infrastructures. In such way, it would be possible to support more user-defined SLA requirements, like latency and availability in the task scheduling policies. Since, a proper SLA-based resource selection can have also significant impact on the performance and cost, we investigate in this work the effect of different resource matching policies on the scheduling performance of data-intensive workflows. The cost effectiveness of such matching policies in Multi-Cloud environments have been investigated in (Dastjerdi et al., 2011), but they have not been evaluated with scientific workflows.

In contrast to the previous works, the resource allocation scheme introduced in this paper allows the execution of workflows on top of heterogeneous Clouds by supporting an SLA-based resource matchmaking combined with a data locality driven task scheduling. In addition to a cost evaluation, we study the impact of the matching and scheduling on data movement and makespan using a real scientific application.

3 BACKGROUND

In order to validate and evaluate our multi-dimensional resource allocation scheme we use a broker-based workflow framework, we implemented in a previous work (Jrad et al., 2013b) to support the deployment of workflows in Multi-Cloud environments. In this section we describe briefly the main components of the framework.

Figure 1 depicts the architecture of the Multi-Cloud workflow framework developed on top of Cloudsim. The Cloud Service Broker, as shown in the middle of the architecture, assists users in finding the suitable Cloud services to deploy and execute their workflow applications. Its main component is a Match-Maker that performs a matching process to select the target Clouds for the deployment. A scheduler assigns the workflow tasks to the selected Cloud resources. The architecture includes also a Data Manager to manage the data transfers during the workflow execution. The entire communication with the underlying Cloud providers is realized through standard interfaces offered by provider hosted Intercloud Gateways. A Workflow Engine deployed on the client side, delivers the workflow tasks to the Cloud Service Broker with respect to their execution order and data flow dependencies.

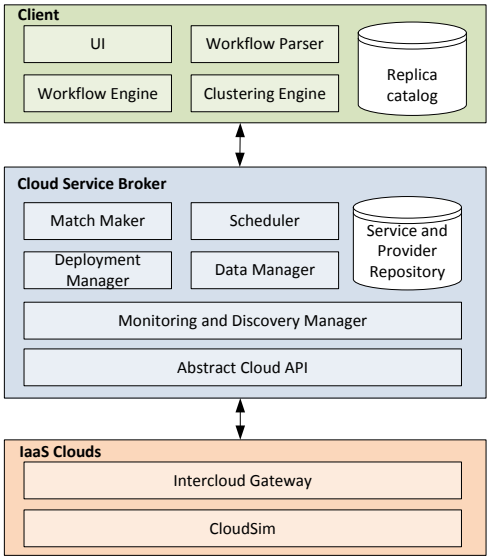

Figure 1: Multi-Cloud workflow framework architecture.

In order to execute workflows using the framework, the Workflow Engine receives in a first step a workflow description and the SLA requirements from the user. After parsing the description, the Workflow Engine applies different clustering techniques to reduce the number of workflow tasks. The reduced workflow and the user requirements are then forwarded to the Broker. In the following, the Match-Maker selects the Cloud resources that can fit the user given requirements by applying different matching policies. After that all the requested virtual machines (VMs) and Cloud storage are deployed on the selected Clouds, the Workflow Engine transfers the input data from the client to the Cloud storage and then starts to release the workflow tasks with respect to their execution order. During execution, the scheduler assigns each task to a target requested VM according

to different scheduling policies while the Data Manager manages the Cloud-to-Cloud data transfers. A Replica Catalog stores the list of data replicas by mapping workflow files to their current datacenter locations. Finally, the execution results are transferred to the Cloud storage and can be retrieved via the user interface. The requested VMs and Cloud Storage are provisioned to the user until the workflow execution is finished. However, if the same workflow should be executed many times (e.g. with different input data), a long-term lease period can be specified by the user.

4 SLA-BASED MULTI-CLOUD MATCHING POLICIES

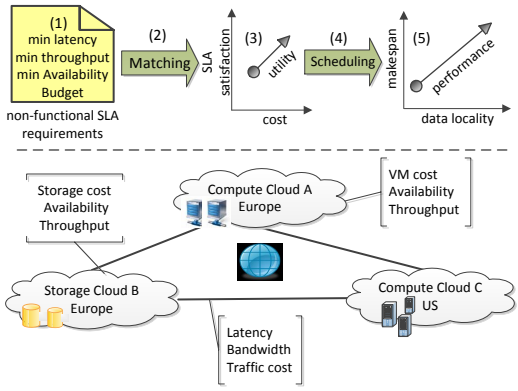

Figure 2: Multi-dimensional resource allocation scheme.

As illustrated in Figure 2, our multi-dimensional resource allocation is performed in five steps. After that the user gives his SLA requirements and budget (step 1), a matching process is started (step 2), where the functional and non-functional SLA requirements are compared to the measured Cloud SLA metrics as well as their service usage costs. The selection of the optimal Clouds (step 3) is then performed using a utility-based matching algorithm with the goal to maximize the user utility for the provided QoS. In the following step, The application workloads are distributed to the requested compute resources using a scheduling policy. For this purpose, a data-locality scheduling scheme is used to achieve a minimal data movement and improve the overall application performance (step 5). In the following subsections we describe in details the used utility-based matching algorithm. In addition, we describe another simple matching algorithm, we used for a comparative study with the utility-based algorithm. The used scheduling policies are described in the next Section.

4.1 Sieving Matching Algorithm

On the search for efficient matching algorithms, we implemented a simple matching policy called Sieving. Given a service list forming the requested Mutli-Cloud service composition and a list of candidate datacenters, the Sieving matching algorithm iterates through the service list and selects randomly for each service a candidate datacenter, which satisfies all functional and non-functional SLA requirements. Therefore, for each selected datacenter the measured SLA metrics and his usage price should be respectively within the ranges specified by the user in his SLA requirements and budget. In addition, the algorithm checks if the current datacenter capacity load allows the deployment of the requested service type. However, it may be possible that the result set is empty in case that none of the Cloud providers fulfill the requested criteria. The following pseudo code describes in detail the functionality of the Sieving algorithm:

Algorithm 1: Sieving matching.

```
Input: requiredServiceList, DatacenterList
For each service S in requiredServiceList Do
 For each datacenter D in DatacenterList Do
  If S in offers(D) and isDeployableIn(D) Then
    If price(D) <= budget(S) &
       availability(D)>=requiredAvilability(S) &
       throughput(D)>=requiredThroughput(S) Then
       add D to CandidatesList
    Endif
  Endif
 Endfor
 If sizeof(CandidatesList) >0 Then
   Choose random datacenter D from CandidatesList
   CloudCompositionMap.add(S,D)
 Else return null
 Endif
Endfor
Output : CloudCompositionMap
```

4.2 Utility-based Matching Algorithm

A major issue of the above described Sieving algorithm is the lack of flexibility in the matching of non-functional SLA attributes. Hence, it cannot handle use cases like availability is more important than throughput or select well qualified providers while keeping the total costs low. In addition, the network connectivity between the Clouds and traffic costs are ignored in the matching. Therefore, we adopted from the Attribute Auction Theory (Asker and Cantillon, 2008) a new economic utility-based matching algorithm, which takes the payment of customers as the focus. In a previous work (Jrad et al., 2013a),

we compared the utility algorithm with Sieving and validated its cost benefits in matching single Cloud services. This work, is our first attempt in using the utility-for matching Multi-Cloud service compositions.

The main strategy of the utility-algorithm is to maximize the user profit for the requested service quality by using a quasi-linear utility function (Lamparter et al., 2007). The user preferences for the non-functional SLA attributes are modeled by weighted scoring functions, whereas all functional requirements must be fulfilled similar to Sieving. Let Q be the set of m required non-functional SLA attributes q with $q \in Q = \{1, ..., m\}$. The utility of a customer i for a required service quality Q with a candidate Cloud composition j is computed as follows:

$$U_{ij}(Q) = \alpha_i F_i(Q) - P_j, \tag{1}$$

where α_i represents the *maximum willingness to pay* of consumer *i* for an "ideal" service quality, and $F_i(Q)$ the customer's *scoring function* translating the aggregated service quality attribute levels into a relative fulfillment level of consumer requirements. P_j denotes the total price that has to be paid for using all Cloud services in the Cloud composition j with:

$$P_j = T * C^j_{VM} + D_{st} * C^j_{st} + D_{tr} * C^j_{tr} \tag{2}$$

where C^j_{VM}, C^j_{st} and C^j_{tr} denote respectively the charged compute, storage and traffic cost with composition j, whereas D_{st} and D_{tr} denote respectively the amount of data to be stored and transferred. The lease period T is defined as the time period in which the Cloud services are provisioned to the user. The scoring function $F_i(Q)$ is defined as follows:

$$F_i(Q) = \sum_{q=1}^{m} \lambda_i(q) f_i(q) \rightarrow [0,1], \tag{3}$$

where $\lambda_i(q)$ and $f_i(q)$ denote respectively the relative assessed weight and the fitting function for consumer *i* regarding the SLA attribute *q*, where $\sum_{q=1}^{m} \lambda_i(q) = 1$. The fitting function maps properly to the user behavior each measured SLA attribute to a normalized real value in the interval [0,1] with 1 representing an ideal expected SLA value. An example for three non-linear fitting functions is provided in Section 6. The aggregated SLA values for the Cloud composition are calculated from the SLA values of the component services by applying common used aggregation functions as in (Zeng et al., 2003) presented in Table 1.

A candidate Cloud service composition j is optimal if it is feasible and if it leads to the maximum utility value with:

$$U_{ioptimal}(Q) = \max_{j=1}^{n} U_{ij}(Q) \tag{4}$$

Table 1: Aggregated SLA attributes of N component services S.

SLA attribute	Aggregation function
Throughput	$\min_{i=1}^{N} Th(S_i)$
Latency	$\overline{Lat(S_{ij})} \begin{cases} i \in \{1,\ldots,N-1\} \\ j \in \{i+1,\ldots,N\} \end{cases}$
Availability	$\prod_{i=1}^{N} Av(S_i)$

where n is the number of possible Cloud service compositions. Hence, the match-making problem can be formulated with a search for the Cloud composition with the highest utility value for the user. As this kind of multi-attribute selection problems is NP-Hard, we apply a single objective genetic algorithm combined with crossover and mutation operations to search for the optimal candidates. To evaluate each candidate we use as objective function the utility function in equation 1. Additionally, we use a death penalty function to penalize candidates, which not satisfy the service constraints and to discard Cloud compositions with a negative utility. In order to include the Cloud-to-Cloud latency and the traffic cost in the utility calculation, we model each candidate service composition as a full connected undirected graph (see Figure 2), where the nodes represent the component services and the edges the network connectivity between them.

5 DATA LOCALITY DRIVEN WORKFLOW SCHEDULING

To support the data locality during the workflow execution, we implemented two dynamic greedy-based scheduling heuristics, which distribute the workflow tasks to all the user requested VMs allocated by the previous described matching policies (see Section 4). In the following subsections we describe in details the functionality of the implemented policies.

5.1 DAS Scheduler

We implemented a Data-Aware Size-based scheduler (DAS) capable of scheduling tasks to the provisioned VMs running in different Cloud datacenters with respect to the location of the required input data. In a first step, the algorithm iterates through the workflow tasks and calculates for each task the total size of the required input files found in each matched datacenter and store the result in a map data structure called task data size affinity $T_{sizeaff}$. If we assume that task T has m required input files $freq_i$, $i \in \{1, ..., m\}$ and there are k matched datacenters, the task affinity of

the datacenter dc_j, $j \in \{1,...,k\}$ is calculated using the following equation:

$$T_{sizeaff}(dc_j) = \sum_{freq_i \in dc_j} size(freq_i), \quad (5)$$

After sorting the task affinity map by the data size values in descending order, the policy assigns the task to the first free provisioned VM running on the datacenter dc_{cand} containing the maximum size of located input files, where:

$$T_{sizeaff}(dc_{cand}) = \max_{j=1}^{k} T_{sizeaff}(dc_j) \quad (6)$$

In case that all the provisioned VMs in the selected datacenter are busy, the algorithm tries the next candidate datacenters in the sorted map to find a free VM. So that, a load balancing between the datacenters is assured and an unnecessary waiting time for free VMs can be avoided. The implemented data-aware scheduling policy is described using the following pseudo-code:

Algorithm 2: DAS/DAT Scheduler.

```
Input: requestedVMList, TaskList, schedulingPolicy
For each task T in TaskList Do
  processAffinity(T)
  If schedulingPolicy=DAS Then
     Taff=sizeAffnityMap
     sort Taff by size in descending order
  Elseif schedulingPolicy=DAT Then
     Taff=timeAffnityMap
     sort Taff by time in ascending order
  Endelse
  For each entry in Taff Do
    site=entry.getkey()
    For each vm in requestedVMList Do
      If vm.getStatus()=idle
         & vm.getDatacenter()=site Then
         schedule T to vm
        scheduledTaskList.add(T)
        Break
      Endif
    Endfor
  Endfor
Endfor
Output: scheduledTaskList
```

Algorithm 3: Function processAffinity(Task T).

```
Input: matchedDatacenterList, Replica catalog R
For each datacenter D in matchedDatacenterList Do
  Time=0; Size=0;
  inputFileList = T.getInputFileList()
  For each file in inputFileList Do
      maxBwth=0
      siteList = R.get(file)
      If siteList.contains(D) Then
         size=size+file.getSize()
      Else
        For each site in siteList Do
         If region(site)=region(D) Then
            bwth=Cloud-to-Cloud Intra-continental
         Else
            bwth=Cloud-to-Cloud Inter-continental
         Endelse
         If bwth>maxBwth Then
            maxBwth=bwth
        Endfor
      Endelse
      time=time+file.getSize()/maxBwth
  Endfor
  sizeAffnityMap.add(D, size)
  timeAffnityMap.add(D, time)
Endfor
Output sizeAffnityMap, timeAffnityMap
```

5.2 DAT Scheduler

The second scheduling policy called Data-Aware Time-based (DAT) scheduler has a strong similarity with the previous described DAS policy except in the method of calculating the task affinity. Instead of calculating the maximum size of existing input files per datacenter, the algorithm computes the time needed to transfer the missing input files to each datacenter and stores the transfer time values in a Map data structure called $T_{timeaff}$, which is calculated using the following equation:

$$T_{timeaff}(dc_j) = \sum_{freq_i \notin dc_j} transfertime(freq_i), \quad (7)$$

As target datacenter, the algorithm chooses the datacenter, which assures the minimum transfer time by sorting the affinity map in the ascending order.

$$T_{timeaff}(dc_{cand}) = \min_{j=1}^{k} T_{timeaff}(dc_j) \quad (8)$$

For each workflow task the algorithm (see function processAffinity() above) iterates through the matched datacenters and checks the existence of local input files. For each missing input file, it calculates the needed time to transfer the file from a remote location to that datacenter. For this, it fetches the replica catalog and selects a source location, which assures the maximum bandwidth and consequently the minimal transfer time to that datacenter.

6 EVALUATION

In order to evaluate our proposed multi-dimensional resource allocation scheme with a real large scale workflow, we conducted a series of simulation experiments. The simulation setup and results are presented in the following subsections.

6.1 Simulation Setup

We implemented the matching and scheduling policies presented in Section 4 and 5 as part of the Multi-Cloud simulation framework. As workflow engine, we use WorkflowSim (Weiwei and Ewa, 2012), a CloudSim-based version of the Pegasus WfMS [3]. In addition, we use the opt4j (Lukasiewycz et al., 2011) genetic framework to perform the utility-based matching. For all the conducted simulation experiments, we configured 20 compute and 12 storage Clouds located in four world regions (Europe, USA, Asia and Australia). Each compute Cloud is made up of 50 physical hosts, which are equally divided between two different host types with respectively 8 and 16 CPU cores. In order to make the simulation more realistic, we use real pay-as-you-go prices for computation, storage and network traffic for each modeled Cloud. Traffic inside the same datacenter is free. The real Cloud SLA metrics values (average for last 3 months) for availability and Client-to-Cloud throughput were acquired through CloudHarmony[4] network tests from the same client host. In order to consider the network traffic and latency in the matching and scheduling, we defined based on the location of the datacenters three constant bandwidth and latency values, which are presented in Table 2. The use of synthetic values is justified by the lack of free accessible Cloud-to-Cloud network metrics from current Cloud benchmarking tools.

Table 2: Defined Cloud-to-Cloud latency and bandwidth values.

Cloud-to Cloud	local transfer	intra-continental	inter-continental
Bandwidth	100 Mbit/s	30 Mbit/s	10 Mbit/s
Latency	10 ms	25 ms	150 ms

With the help of the framework, we modeled the following use case for a workflow deployment on Multi-Cloud. A user located in Europe requests 10 VMs of the type *small* and 10 VMs of the type *medium* and one storage Cloud to store the workflow input and output data. The configuration of each VM type is reported in Table 3. We assume that all the VMs located in the same datacenter are connected to a shared storage. The Workflow Engine transfers at execution start the input data from the Client to the Cloud storage. The output data is stored in the Cloud storage when the execution is finished. The Data Manager fetches the Replica catalog and transfers all the missing input files before each task execution from their source datacenters with the maximal possible bandwidth. We configured the Workflow Engine to release maximal 5 tasks to the broker in each scheduling interval (default value used in Pegasus).

Table 3: VMs Setup; 1 CPU Core: 1GHZ Xeon 2007 Processor of 1000 MIPS; OS: Linux 64 bits.

VM Type	Cores	RAM (GB)	Disk (GB)
small	1	1.7	75
medium	2	3.75	150

It is worth to mention that if the user executes the same workflow multiple times, our implemented data locality scheduling scheme reuses the existing replicated input data in order to save on data transfers and costs. For simplicity, we consider in our evaluation only the first run of the workflow. For collecting the simulation results we repeated each of the experiments ten times from the same host and then computed the average values.

6.2 Montage Workflow Application

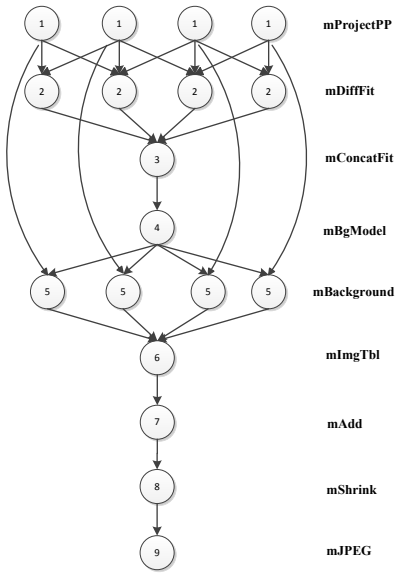

Figure 3: A sample 9-level Montage workflow.

Montage (Berriman et al., 2004) is a data-intensive (over 80% I/O operations) workflow application used to construct large image mosaics of the sky obtained from the 2MASS observatory at IPAC [5]. A sample directed acyclic graph (DAG) of a 9 level Montage workflow is illustrated in Figure 3. The tasks at the same horizontal level execute the same

[3] http://pegasus.isi.edu/

[4] http://www.cloudharmony.com

[5] http://www.ipac.caltech.edu/2mass/

Table 5: User preferences expressed using fitting functions f and relative weights λ; $\gamma = 0.0005$; $\beta = 1 - \gamma$.

	Availability		Latency		Throughput	
Scenario	λ_{av}	$f(av)$	λ_{lat}	$f(lat)$	λ_{th}	$f(th)$
EU	$\frac{3}{10}$	$\frac{\gamma}{\gamma+\beta e^{(-0.9(av-84))}}$	$\frac{6}{10}$	$\frac{\gamma}{\gamma+\beta e^{(0.2(lat-100))}}$	$\frac{1}{10}$	$1-\beta e^{-0.2th}$
EU-US	$\frac{2}{5}$	$\frac{\gamma}{\gamma+\beta e^{(-0.9(av-84))}}$	$\frac{2}{5}$	$\frac{\gamma}{\gamma+\beta e^{(0.2(lat-150))}}$	$\frac{1}{5}$	$1-\beta e^{-0.6th}$

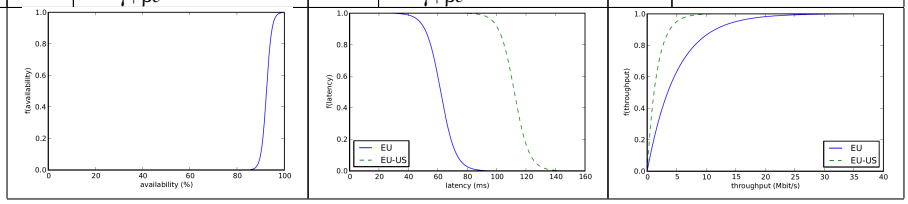

Table 4: User non-functional SLA Requirements; Availability (av); Latency (lat); Throughput (th).

deployment scenario	min av (%)	max lat (ms)	min th (Mbit/s)
EU	95	50	12
EU-US	95	100	2

executable code (see right side of the figure) with different input data. For all our conducted experiments we imported with the help of the WorkflowSim Parser a real XML formatted Montage trace executed on the FutureGrid testbed using the Pegasus WfMS. The tasks runtime and files' size information are imported from separate text files. The imported trace contains 7463 tasks within 11 horizontal levels, has 3 GB of input data and generates respectively about 31 GB and 84 GB of intermediate output and traffic data.

6.3 Simulation Scenarios

For the purpose of evaluation, we modeled two simulation scenarios. In the first scenario, named "EU-deployment", all the requested VMs and storage Cloud are deployed in the same user region (Europe). In the second scenario, named "EU-US deployment", all the 10 VMs of type *small* are deployed on Clouds located in the US region, while the 10 VMs of type *medium* and the storage Cloud are located in Europe. The non-functional SLA requirements for both scenarios, given in Table 4, express the user desired ranges for availability, latency and throughput in order to deploy the workflow with an acceptable quality. These values are consumed by the Sieving matching algorithm described in Section 4 to select the target Clouds for the workflow deployment.

The fitting functions and the relative weight for each SLA parameter, both required for the utility-based matching, are given in Table 5. As we can see from the table, the user prefers for the first scenario Clouds with low Cloud-to-Cloud latency values to advantage the data transfer time, while for the second scenario availability is equally weighted with the latency and the Client-to-Cloud throughput has more significance.

The maximum user payment willingness for each VM type as well as for storage and network traffic are given in Table 6.

Table 6: Maximal payment for VMs, storage and traffic.

VM small ($/hour)	**VM medium** ($/hour)	**storage** ($/GB)	**traffic** ($/GB)
0.09	0.18	0.1	0.12

6.4 Impact of Clustering

As mentioned before we use clustering to reduce the scheduling overhead of large scale workflows on Multi-Cloud. A well suited clustering technique for the Montage workflow structure is horizontal clustering. Herewith tasks at the same horizontal level are merged together into a predefined number k of clustered jobs. Table 7 shows the resulted total number of clustered jobs for each used k.

Table 7: Total Number of clustered jobs with different cluster numbers k.

k	20	40	60	80	100
Jobs number	92	152	212	272	332

In order to evaluate the impact of clustering with respect to increasing cluster number k, we measured in a first experiment the total time needed to execute a single run of the sample Montage workflow and the total consumed time for data transfer in minutes for the "EU-deployment" scenario. For this experiment, we configured the Cloud Service Broker to use utility-based matching together with DAS as scheduling policy. For an accurate calculation of the execution time, we extracted from the workflow trace the real delay

overhead resulted from clustering, post-scripting and queuing. Figure 4 illustrates the results achieved.

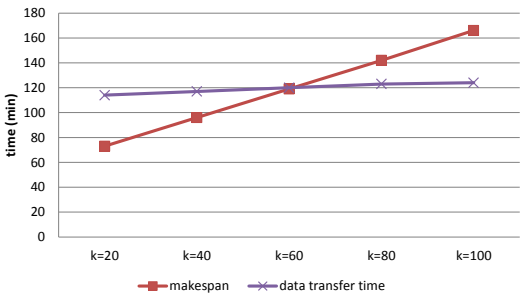

Figure 4: Workflow makespan and total data transfer time with utlity+DAS EU for different cluster numbers k.

As depicted in the figure, the continually increase of k results in a steady increase of the workflow execution time. This demonstrates that our allocation scheme scales well with increasing number of the clustered workflow jobs. Although the transfer time is nearly constant, as file transfers are performed with a high throughput and low latency values because the matched Clouds are close to the user, we observed a slow decrease of the transfer time for small numbers of k. In the latter case more files are merged in a clustered job, so that the amount of Intercloud transfers is heavily reduced. For all the next conducted experiments, we fixed the cluster number to k=20, as it gives us the best results in term of makespan and consequently execution cost.

6.5 Makespan Evaluation

We repeated the previous experiment with the "EU" and "EU-US" scenarios with the different matching and scheduling policies presented in Section 4 and 5. For a comparative study we executed the workflow using a simple Round Robin scheduler, which schedules tasks to the first free available VMs regardless of their datacenter location and the from the literature well established "Min-Min" scheduler (Freund et al., 1998), which prioritizes tasks with minimum runtime and schedule them on the *medium* VM types. The results with k=20 for all the possible combinations are shown in Figure 5.

It can be seen from the figure that for both scenarios the use of utility-based matching combined with the DAS or DAT scheduler gives the lowest execution time compared to Sieving. This result approves how the efficiency of the utility matching affects the scheduling performance. For the EU scenario, the utility algorithm have a tendency to deploy all the requested VMs on the cheapest datacenter to save cost

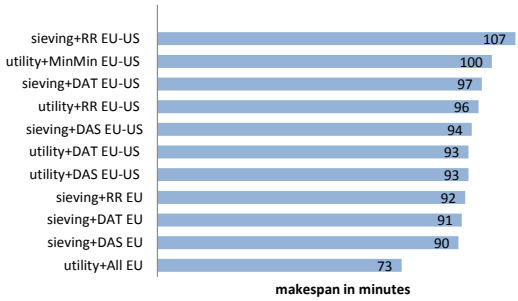

Figure 5: Workflow makespan with different scheduling and matching policies for k=20.

and minimize latency. So that, the user is charged only for the costs to transfer input/output data from/to storage Cloud. This explains the same makespan and Intercloud transfer obtained with different scheduling policies. We observed also that our implemented DAS and DAT scheduler outperform Round Robin and MinMin, especially for the EU-US scenario.

6.6 Intercloud Data Transfer

In order to assess the impact of our multi-dimensional scheme on reducing the Intercloud data transfers, we measured for the previous simulation experiment the size of Intercloud transfers and the ratio of transfer time over the total consumed processing time. The results for different combinations of matching and scheduling policies with k=20 are depicted respectively in Figure 6 and Figure 7. Note that for the Intercloud transfer time calculation, we use the previous defined "Inter-Continental" and "Intra-Continental" bandwidth values from Table 2.

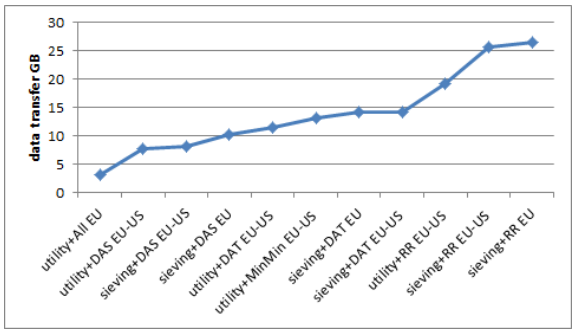

Figure 6: Total amount of Intercloud transfers with different scheduling and matching policies for k=20.

It can be seen from Figure 6 that the DAS scheduler keeps the Intercloud transfer size under 10 GB for both scenarios with utility and Sieving policies. Next to DAS on saving Cloud-to-Cloud data movements is DAT, whereas MinMin and Round robin occupy the last places. The evaluation results for the

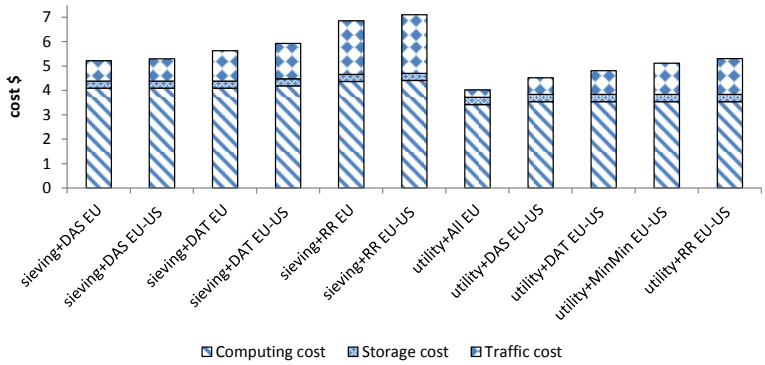

Figure 8: Total execution cost with Sieving (left) and utility (right) matching for k=20.

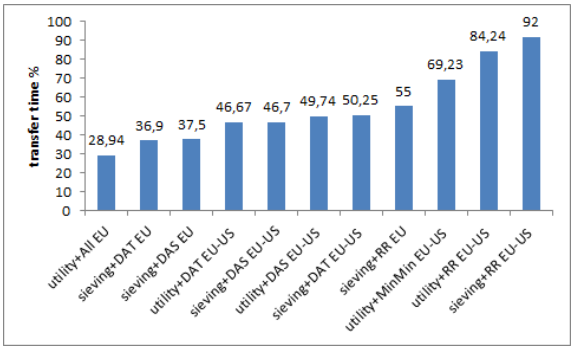

Figure 7: Percentage of data transfers with different scheduling and matching policies for k=20.

transfer time ratio approve also that DAS and DAT are able to reduce the transfer time ratio up to 37% and 50% respectively for the EU and EU-US scenario. On the contrary, the use of Round Robin and Min-Min scheduling regardless of the matching policy results in very high transfer ratios in particular for the EU-US scenario, which disadvantages consequently the workflow makespan. Therefore, data locality has more impact when Clouds are not close to user, as in this case latency and throughput affect more the transfer time.

6.7 Cost Evaluation

In this subsection, we evaluate the impact of used matching and scheduling policies on the traffic costs for both simulation scenarios. The previous experimental results show that the use of utility-based matching and data-aware scheduling heavily reduces the amount of Intercloud transfers and consequently the data traffic costs. This result is approved in Figure 8, in which the amount of compute, storage and traffic costs for different use cases is illustrated. Please note that for the costs calculation the VMs are charged on hourly basis. Therefore, the makespan of each scenario is rounded up to the nearest next hour. Also, we do not consider the additional license and VM images costs which can be charged by some Cloud providers. We found that the utility-based matching combined with the DAS scheduler benefits more the compute and traffic costs compared to the other use cases. For example, comparing utility and Sieving with DAS, respectively up to 25% and 15% cost-saving can be achieved with the EU and EU-US deployment.

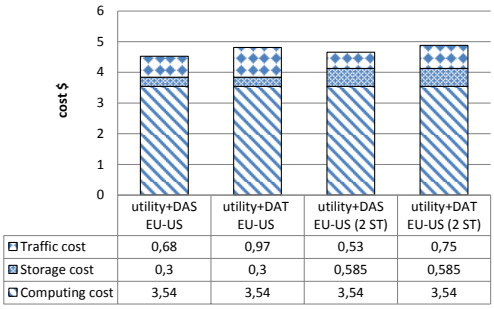

Figure 9: Total execution cost with one and two storage Clouds (ST) for k=20.

We conducted another experiment by adding another storage Cloud located in the US region for the EU-US scenario. The gathered cost results are depicted in Figure 9. For the purpose of comparison, we not consider the time needed to transfer the input files from the Client to the US located storage Cloud. It can be seen from the figure that the total costs for the two storage Clouds use-case are very close to the one storage use-case, even with the doubled storage cost. This is due to the tendency of the utility algorithm to deploy the US located VMs and the added storage Cloud on the same provider to save on traffic costs.

7 CONCLUSIONS

This work presented a two-stage multi-dimensional resource allocation approach for running data-intensive workflow applications on a Multi-Cloud environment. In a first phase, a utility-based matching policy selects the suitable Clouds for users with respect to their SLA requirements and payment willingness. In a second phase, a data locality driven scheduler brings the computation to its data to reduce the Intercloud data transfer.

We evaluated our approach using an implemented simulation environment with a real data-intensive workflow application in different usage scenarios. The experimental results show the benefits from utility-based matching and data locality driven scheduling in reducing the amount of Intercloud transfers and the total execution costs as well improving the workflow makespan.

In the next step of this research work, we will evaluate our multi-dimensional approach with other Mutli-Cloud real large scale applications like MapReduce. In addition, we will automate the collection of the newest SLA metrics from real public Clouds by extending the simulation framework to fetch them from third-party Cloud monitoring services. Also, we will include more accurate network models to make the simulation more realistic. Furthermore, we will investigate the use of adaptive matching and scheduling policies like in (Oliveira et al., 2012) in order to deal with resource and network failures on Clouds.

REFERENCES

Asker, J. and Cantillon, E. (2008). Properties of scoring auctions. *The RAND Journal of Economics*, 39(1):69–85.

Berriman, G. B., Deelman, E., Good, J. C., Jacob, J. C., Katz, D. S., Kesselman, C., Laity, A. C., Prince, T. A., Singh, G., and Su, M.-H. (2004). Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In Quinn, P. J. and Bridger, A., editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5493 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 221–232.

Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Dastjerdi, A. V., Garg, S. K., and Buyya, R. (2011). QoS-aware Deployment of Network of Virtual Appliances Across Multiple Clouds. *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 415–423.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. (2008). The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA. IEEE Press.

Fard, H. M., Prodan, R., Barrionuevo, J. J. D., and Fahringer, T. (2012). A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309.

Fard, H. M., Prodan, R., and Fahringer, T. (2013). A Truthful Dynamic Workflow Scheduling Mechanism for Commercial Multicloud Environments. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1203–1212.

Freund, R., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J., Mirabile, F., Moore, L., Rust, B., and Siegel, H. (1998). Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, pages 184–199.

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

Jrad, F., Tao, J., Knapper, R., Flath, C. M., and Streit, A. (2013a). A utility-based approach for customised cloud service selection. *Int. J. Computational Science and Engineering, in press*.

Jrad, F., Tao, J., and Streit, A. (2013b). A broker-based framework for multi-cloud workflows. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, MultiCloud '13, pages 61–68, New York, NY, USA. ACM.

Lamparter, S., Ankolekar, S., Grimm, S., and R.Studer (2007). Preference-based Selection of Highly Configurable Web Services. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 1013–1022, Banff, Canada.

Lukasiewycz, M., Glaß, M., Reimann, F., and Teich, J. (2011). Opt4J - A Modular Framework for Metaheuristic Optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland.

Mian, R., Martin, P., Brown, A., and Zhang, M. (2011). Managing data-intensive workloads in a cloud. In Fiore, S. and Aloisio, G., editors, *Grid and Cloud Database Management*, pages 235–258. Springer Berlin Heidelberg.

Oliveira, D., Ocaña, K. a. C. S., Baião, F., and Mattoso, M. (2012). A Provenance-based Adaptive Scheduling

Heuristic for Parallel Scientific Workflows in Clouds. *Journal of Grid Computing*, 10(3):521–552.

Pandey, S., Wu, L., Guru, S., and Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407.

Szabo, C., Sheng, Q. Z., Kroeger, T., Zhang, Y., and Yu, J. (2013). Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows. *Journal of Grid Computing*.

Weiwei, C. and Ewa, D. (2012). Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *The 8th IEEE International Conference on eScience*, Chicago. IEEE, IEEE.

Yuan, D., Yang, Y., Liu, X., and Chen, J. (2010). A data placement strategy in scientific cloud workflows. *Future Gener. Comput. Syst.*, 26(8):1200–1214.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 411–421, New York, NY, USA. ACM.