Integrated Data Placement and Task Assignment for Scientific Workflows in Clowds

Ümit V. Çatalyürek
Biomedical Informatics
Ohio State University, USA
umit@bmi.osu.edu

Kamer Kaya
Parallel Algorithms Group
CERFACS, France
kamer@cerfacs.fr

Bora Uçar
CNRS and ENS
Lyon, France
bora.ucar@ens-lyon.fr

ABSTRACT

We consider the problem of optimizing the execution of data-intensive scientific workflows in the Cloud. We address the problem under the following scenario. The tasks of the workflows communicate through files; the output of a task is used by another task as an input file and if these tasks are assigned on different execution sites, a file transfer is necessary. The output files are to be stored at a site. Each execution site is to be assigned a certain percentage of the files and tasks. These percentages, called target weights, are pre-determined and reflect either user preferences or the storage capacity and computing power of the sites. The aim is to place the data files into and assign the tasks to the execution sites so as to reduce the cost associated with the file transfers, while complying with the target weights. To do this, we model the workflow as a hypergraph and with a hypergraph-partitioning-based formulation, we propose a heuristic which generates data placement and task assignment schemes simultaneously. We report simulation results on a number of real-life and synthetically generated scientific workflows. Our results show that the proposed heuristic is fast, and can find mappings and assignments which reduce file transfers, while respecting the target weights.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Cloud computing, Grid computing*]; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Hypergraphs*

General Terms

Algorithms

Keywords

Scientific workflows; cloud computing; data placement; task assignment; hypergraph partitioning

DIDC'11, June 8, 2011, San Jose, California, USA.

1. INTRODUCTION

As a result of the increase in the complexity of problems encountered by scientific researchers, the need for computational power required by scientific applications is increasing exponentially. Although grids are widely being used for scientific applications [11, 12, 21], the emerging computational cloud [29] is a promising tool for executing applications that require immense amounts of storage and processing power. Recently, researchers are studying using cloud computing for scientific workflows and developing practical solutions [15, 17, 18, 27, 30].

A scientific application is usually modeled as a scientific workflow which shows the execution of an application step by step with a directed network. The direction of links shows the flow of the information, i.e., dependence of these steps. These steps are responsible for various tasks such as data mining and analyzing [21]. In this work, we will call a workflow *compute-intensive* when the amount of data used by the tasks is small and the time spent executing tasks dominates the computation, and *data-intensive* otherwise. Here, we are interested non-trivial, data-intensive workflows where computation and data transfer times are comparable on the available computational sites.

To execute a scientific workflow, its tasks and files need to be assigned and distributed to the execution sites. These files need to be placed carefully, since when a task is assigned to an execution site, some of its required files may not be available on the site. Hence, to start the execution of the task, some file transfers are required. Considering total file access in scientific workflows increased from the megabyte- to the petabyte-level [23], such file transfers may increase the makespan of the workflow and the cost of the execution drastically, especially for the data-intensive workflows. To avoid this, it is better to store a set of files in the same execution site if they are accessed by the same tasks. Then, after the data placement, if the tasks are assigned to this site, they can access the files locally without requiring a file transfer. One cannot map all of the files and tasks to a single execution site since the storage capacity of this site is limited. Besides, although no file transfer is required in this case, this will limit the task parallelism, thereby increasing the makespan.

The execution sites in the cloud usually have different characteristics such as the storage capacity/cost, computational power/cost and current load. In addition to these, the cloud user may have some preferences which give precedence to some execution sites. Hence, each execution site has a desirability for the files and tasks of the workflow. The data

placement and task assignment problems can be described as mapping the files and tasks to the execution sites such that the total data transfer is minimized and the storage and computational loads of the workflow are distributed, by taking the above-mentioned desirability criteria into account. Similar variants of this problem for scientific workflows have been investigated by other researchers [1, 30].

The data access patterns and characteristics of scientific workflows were investigated carefully by Shibata et al. [25, 26]. It is stated that there exist data-intensive workflows in practice such as Montage, an astronomical image mosaic engine (`http://montage.ipac.caltech.edu/`) for which the data operations take 99% of the whole execution time. They also state that after a data placement scheme, *locality aware scheduling* techniques which assigns a task to the execution site storing most of its required input files can increase the locality of the accessed files to 96%. However, for other scientific applications, in practice the efficiency of locality-aware scheduling reduces to 0.2%, and is largely dependent on the structure of workflow [25]. Hence, the data placement and task assignment problems are not independent from each other. If one needs to reduce the amount of file transfers by increasing the locality of the input data, it is advisable to consider the structure of the workflow in the task assignment phase.

There are several studies addressing the data placement and task assignment problems in cloud computing environments. Agarwal et al. [1] considered the problem of data placement in geographically distributed execution sites and proposed an automated scheme which takes several factors into account such as the data interdependencies and storage limits. Cope et al. [10] proposed various heuristics to obtain robust schedules for workflows which can tolerate excessive usage of storage resources. Pandey and Buyya [22] and Ramakrishnan et al. [24] considered the problem of scheduling data-intensive workflows in clouds. In both of these works, the files are assumed to be replicated in multiple execution sites. The former work proposed a Steiner tree based approach to minimize the makespan of the workflow, whereas the latter work considered the case with storage-constrained resources and tried to optimize the maximum disc usage. The closest study to this paper is by Yuan et al. [30], in which the authors tried to solve the data placement problem in collaborative cloud environments. We will describe their approach in Section 4.

Our contribution is two-fold. First, we combine data placement and task assignment problems and model the workflow as a hypergraph. With a hypergraph partitioning based formulation, we simultaneously find the data placement and task assignment schemes which distributes the storage and computational loads among the execution sites with respect to some pre-determined ratios. The schemes conform with each other and reduce the amount of file transfers. Second, we implement the hypergraph partitioner for our model by modifying PaToH [8], an existing hypergraph partitioning tool. This can be of a wider interest, because none of the existing hypergraph partitioning tools and models can encapsulate the requirements of the problem at hand.

The paper is organized as follows. In the next section, we formally describe the target framework by specifying the models for the workflow, the execution sites, and the objective function. In this section, we also give the notation and definitions for the hypergraph partitioning problem. The extended hypergraph partitioning problem and its use for the data placement and task assignment problem is described in Section 3. In Section 4, we summarize an existing method from the literature and its use in our problem, as it is close to the proposed method. Section 5 gives the experimental results, in which we report the performance of the modified existing method and ours on real-life workflows from Pegasus [5] and on synthetically generated workflows. Experimental results show that the proposed approach works well for real-life and synthetically generated workflows. Section 6 concludes the paper with some comments on possible future work.

2. FRAMEWORK AND BACKGROUND

Here, we briefly summarize the target data placement and task assignment framework that consists of a workflow model, a cloud model, and a cost model. We also give necessary background material on hypergraphs and hypergraph partitioning.

2.1 Workf ow model

The target workflow is represented as a two tuple $\mathcal{W} = (\mathcal{T}, \mathcal{F})$. Here, $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$ denotes the set of N dependent tasks, and $\mathcal{F} = \{f_1, f_2, \ldots, f_M\}$ denotes the set of M files. The computational load of a task t_i is given as $exec(t_i)$. Each task generates and requires subsets of $\mathcal{F}$ as inputs and outputs. The sets of files generated and required by a task t_i are denoted as $gen(t_i) \subseteq \mathcal{F}$ and $req(t_i) \subseteq \mathcal{F}$, respectively. Note that these two sets are disjoint for each task, and a task t_i depends on another task t_j if $gen(t_j) \cap req(t_i) \neq \emptyset$. The set of files accessed by t_i is denoted by $files(t_i) = gen(t_i) \cup req(t_i)$, and the tasks that access to a file f_k is denoted by $tasks(f_k)$. Files can have different sizes; the size of a file f_k is denoted by $size(f_k)$. The total size of the files used by a task t_i is denoted by

$$size(files(t_i)) = \sum_{f_k \in files(t_i)} size(f_k).$$

Finally, $|\mathcal{W}|$ denotes the total number of file access in the workflow $\mathcal{W}$, i.e., $|\mathcal{W}| = \sum_{t_i \in \mathcal{T}} |files(t_i)|$.

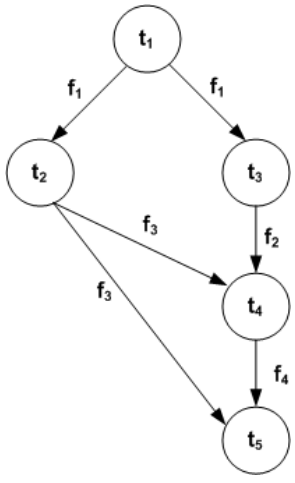

Figure 1: A sample workflow. Files are represented with edges; the labels on the edges are used to model that files are needed by two tasks.

A simple workflow $\mathcal{W}$ with $N = 5$, $M = 4$ and $|\mathcal{W}| = 6$ is given in Figure 1. In the figure, $files(t_2) = \{f_1, f_3\}$ and since $f_1 \in req(t_2)$ is generated by t_1, i.e., $f_1 \in gen(t_2)$, we say that t_2 depends on t_1.

Such workflow models arise in real-life applications. These include astronomical applications, seismic studies, epigenomics, and bioinformatics [5]; filtering streaming applications (see [2, 4] and the references therein). They are also identified as building blocks of complex workflows, see patterns 12 and 13 in [28].

2.2 Cloud model

The target computing platform is a cloud containing K execution sites $\{s_1, s_2, \ldots, s_K\}$. While executing a workflow $\mathcal{W} = (\mathcal{T}, \mathcal{F})$, each execution site is responsible for storing and managing of data and execution of the tasks assigned to it. We assume that replications may occur within execution sites but there is no file replication among the sites, and only a single site is used to store each file f_k. Communicating a file to another one may be seen as a replication (a temporary one), but we do not take advantage of this replication—a site receiving an input file will not forward that file to another one. We note that each generated file f_k is also stored only at a single site s_i, and this site can be different than the one to which the generator task of f_i is assigned. In that case, after the generation of f_k, it is transferred to s_i and this is counted as a file transfer. We use $files(s_i) \subseteq \mathcal{F}$ to denote the set of files that will be stored in s_i during the execution of $\mathcal{W}$. Similarly $tasks(s_i) \subseteq \mathcal{T}$ denotes the set of tasks that will be executed in s_i. Note that each pair of s_i and s_j are connected via Internet and if necessary, a file stored in s_i is transferred to s_j during the execution. We assume that the transferred file is erased from s_j after the execution of all tasks assigned to s_j and requiring that file finish.

The execution sites are heterogeneous in the sense that they have different characteristics such as storage capacity, computational power, cost, energy efficiency and availability. In addition to these, the user, who submits the workflow to the cloud, may have some preferences which give precedence to some execution sites. Hence, the scheduler needs to take these criterion into account before data placement and task assignment. Such information can be gathered by using the resource monitoring tools and techniques such as [6, 16]. We assume that for a given workflow, the storage and computational desirabilities of each execution site s_i are known; that is a user can specify these. Let $des_f(s_i)$ and $des_t(s_i)$ be these values, respectively. In our model, these parameters are referred to as target values and given as

$$\frac{size(files(s_i))}{size(\mathcal{F})} \quad \text{and} \quad \frac{\sum_{t_j \in tasks(s_i)} exec(t_j)}{\sum_{t_j \in \mathcal{T}} exec(t_j)} ,$$

for each site s_i. Note that since there is no file replication and each task will be assigned to a single execution site,

$$\sum_{i=1}^{K} des_f(s_i) = \sum_{i=1}^{K} des_t(s_i) = 1. \qquad (1)$$

That is, the target values in the model specify how much percent of the total file size and how much percent of the total computational load should be assigned to each execution site.

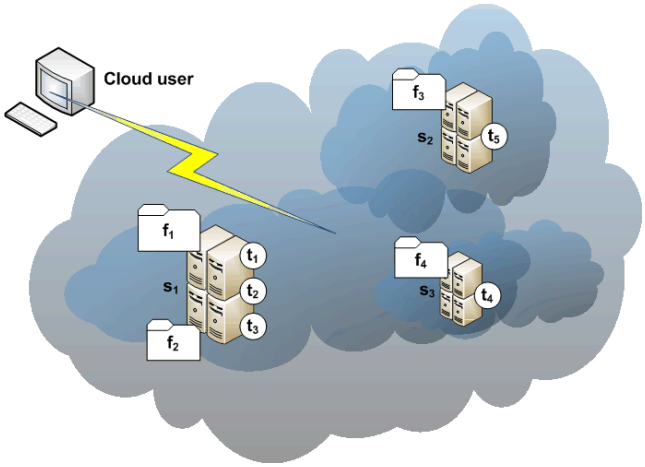

Figure 2: A simple cloud and sample data placement and task assignments for the workflow given in Fig. 1.

Figure 2 shows the topology of a simple cloud with three execution sites and a sample mapping of files and tasks of the workflow in Figure 1 to the sites. Note that each file is stored only at one site and each task is assigned to only one site. With this distribution, during the execution of the workflow, file transfers are be required. For example, the file f_3 needs to be transferred from the site s_1 to the sites s_2 and s_3. This is because, f_3 will be generated by t_2 in s_1 and will be used by t_4 and t_5 assigned to sites s_3 and s_2, respectively. Also, t_5 requires f_4 which will be generated at s_3. The file f_4 will therefore be transferred from s_3 to s_2.

2.3 Cost model and objective

The main objective of this paper is to reduce the total amount of file transfers between sites during the execution of a workflow. Hence, the objective of the target scheduling problem is to find a data placement and task assignment scheme in such a way that the total size of the transferred files is minimized and the target values $des_f(s_i)$ and $des_t(s_i)$ are respected for each execution site s_i. This formulation can be used to solve multiple problems. First, by minimizing the total amount of file transfers between sites, one reduces total costs of data transfers. Second, by selecting appropriate values for $des_f(s_i)$ and $des_t(s_i)$, one can also target computational load balance, hence minimizes the total execution time since the end-to-end execution time will be determined by the computation time of maximum loaded site plus the data transfer time.

Note that if a file f_k which is placed at a site s_i is required by some tasks assigned to another site s_j, f_k will be transferred from s_i to s_j only once. Hence, the total effect of this file transfer in the cost function will be $size(f_k)$. The problem is NP-complete as it includes the NP-complete graph partitioning problem [14, problem ND14] as a special case.

As an example, the total cost of the data placement and task assignment schemes in Fig. 2 for the workflow in Fig. 1 is

$$size(f_2) + 2 \times size(f_3) + size(f_4) . \qquad (2)$$

This is because f_2 will be transferred between (s_1, s_3); f_3 will be transferred between (s_1, s_2) where it is going to be

stored in s_2, and between (s_2, s_3); and f_4 will be transferred between (s_3, s_2). Note that f_1 will not be transferred since it is generated by t_1 and used by $\{t_2, t_3\}$ which all are assigned to s_1.

2.4 Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{E}$. Every net $n_j \in \mathcal{E}$ is a subset of vertices, i.e., $n_j \subseteq \mathcal{V}$. Weights can be associated with the vertices and costs can be associated with the nets. We use $w(v_i)$ to denote the weight of the vertex v_i and $c(n_j)$ to denote the cost of a net.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is nonempty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. That is, a K-way partition satisfies the following:

1. $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$,
2. $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for $1 \leq k < \ell \leq K$,
3. $\bigcup_k \mathcal{V}_k = \mathcal{V}$.

A K-way vertex partition of $\mathcal{H}$ is said to be balanced if

$$W(\mathcal{V}_k) \leq W_{avg} \times (1 + \varepsilon), \quad \text{for } k = 1, 2, \ldots, K. \qquad (3)$$

In (3), the weight $W(\mathcal{V}_k)$ of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in that part, i.e., $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$, W_{avg} is the average part weight, i.e., $W_{avg} = \sum_{v_i \in \mathcal{V}} w(v_i)/K$, and ε represents the allowed imbalance ratio.

In a partition Π of $\mathcal{H}$, a net that has at least one vertex in a part is said to *connect* that part. *Connectivity set* Λ_j of a net n_j is defined as the set of parts connected by n_j. *Connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* if it connects more than one part (i.e., $\lambda_j > 1$), and *uncut* otherwise (i.e., $\lambda_j = 1$). The set of cut nets of a partition Π is denoted as $\mathcal{E}_C$. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. For our purposes in this paper we use the *connectivity - 1* metric with the net costs:

$$cutsize(\Pi) \quad = \quad \sum_{n_j \in \mathcal{E}_C} c(n_j)(\lambda_j - 1) \; . \qquad (4)$$

The hypergraph partitioning problem can be defined as the task of dividing the vertices of a hypergraph into K parts such that the cutsize is minimized, while a given balance criterion (3) is met. The hypergraph partitioning problem is known to be NP-hard [20].

A recent variant of the above problem is the multi-constraint hypergraph partitioning [3, 9]. In this variant, multiple weights $w(v, 1), w(v, 2), \ldots, w(v, T)$ are associated with each vertex v, where T is the number of constraints. Let

$$W(\mathcal{V}_k, t) = \sum_{v \in \mathcal{V}_k} w(v, t)$$

denote the weight of part $\mathcal{V}_k$ for constraint t. Then a partition Π is said to be balanced if

$$\frac{W_{max}(t) - W_{avg}(t)}{W_{avg}(t)} \leq \varepsilon(t), \qquad (5)$$

$\forall t \in \{1, 2, \ldots, T\}$, where $W_{max}(t) = \max_k W(\mathcal{V}_k, t)$, $W_{avg}(t) = \sum_{v_i \in \mathcal{V}} w(v_i, t)/K$, and $\varepsilon(t)$ is a predetermined imbalance ratio for constraint t. We note that the current distribution of PaToH [8], a commonly used hypergraph partitioning tool, uses the same load imbalance parameter for all constraints.

3. THE PROPOSED APPROACH

We propose using a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ to represent the interaction among the tasks in a scientific workflow $\mathcal{W} = (\mathcal{T}, \mathcal{F})$. In this model, we have two types of vertices in $\mathcal{V}$: the task vertices, which represent the tasks in $\mathcal{T}$, and the file vertices which represent the files in $\mathcal{F}$. Hence, $|\mathcal{V}| = |\mathcal{T}| + |\mathcal{F}|$. For each vertex $v_i \in \mathcal{V}$, we have two weights $w(v_i, 1)$ and $w(v_i, 2)$. For a task vertex $v_i \in \mathcal{V}$ corresponding to $t_j \in \mathcal{T}$,

$$w(v_i, 1) = exec(t_j) \;\; \text{and} \;\; w(v_i, 2) = 0.$$

Similarly, for a file vertex $v_i \in \mathcal{V}$ corresponding to $f_k \in \mathcal{F}$,

$$w(v_i, 1) = 0 \;\; \text{and} \;\; w(v_i, 2) = size(f_k).$$

The nets in $\mathcal{E}$, which are subsets of $\mathcal{V}$, represent the files. If n_k represent the file f_k, it contains the file vertex corresponding to f_k and the task vertices corresponding to the set of tasks in $tasks(f_k)$. Finally, we set the cost of n_k to the size of the file, i.e., $c(n_k) = size(f_k)$.

Given a workflow and K execution sites, assume that we construct the hypergraph as described above and obtain a K-way partition Π. Since vertices represents tasks and files, we will decode partition Π as their assignments to the execution sites. Consider a cut net n_k, corresponding to f_k, with connectivity λ_k in Π. It is clear that λ_k denotes the number of sites that require the file f_k under the partition Π. Hence, $c(n_k)(\lambda_k - 1) = size(f_k)(\lambda_k - 1)$ represents the total transfer cost for f_k, as one of the sites in Λ_k holds f_k. That is the total cutsize of Π according to the (weighted) *connectivity - 1* metric in (4) is equal to the total amount of file transfers during the execution of the workflow if the tasks and the files are distributed with respect to Π.

Figure 3 shows the hypergraph corresponding to the sample workflow of Fig. 1 and a 3-way partition. The partitioning corresponds to the distribution of the tasks and the files in Fig. 2. In Figure 3, there are four nets and nine vertices where the white vertices represent the tasks and the gray vertices represent the files in the workflow. As described above, if the tasks and files are mapped according to the partitioning in the figure, the total amount of file transfers will be

$$\sum_{i=1}^{4} c(n_i)(\lambda_i - 1) = c(n_2) + 2 \times c(n_3) + c(n_4)$$

which is equal to the one given before (2). Notice that the cutsize is a function of the file sizes and the number of times they are communicated.

While minimizing cutsize, we have two constraints: we need to partition the task and file vertices in a way that the target values $des_f(s_i)$ and $des_t(s_i)$ are not exceeded for each execution site s_i. Hence, our model requires multi-constraint hypergraph partitioning with target part weights.

3.1 Multi-constraint hypergraph partitioning with target part weights

To the best of our knowledge the only publicly available tool that provides multi-constraint hypergraph partitioning

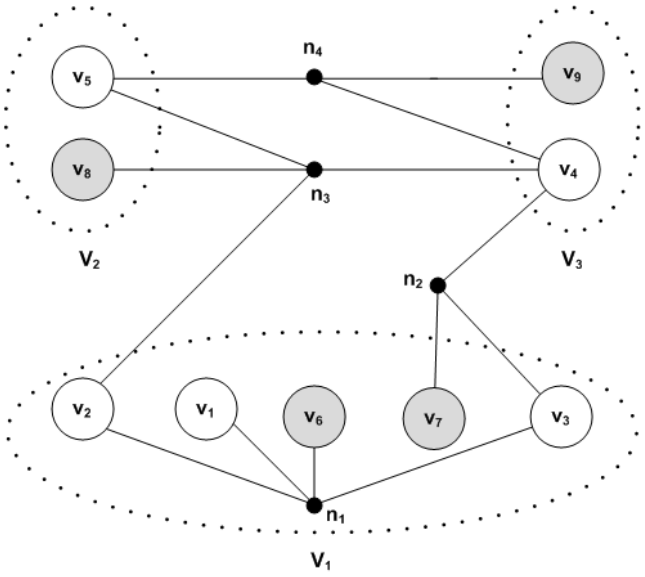

Figure 3: Corresponding hypergraph model for the workflow given in Fig. 1 and a simple 3-way partitioning respecting the distribution of tasks and files given in Fig. 2. The white and gray vertices represent, respectively, the tasks and the files in the corresponding workflow.

is PaToH [8]. However, our model in this paper also needed partitioning with non-unit net costs as well as the target part weights which was not available PaToH v3.1. In this, work we improved PaToH by implementing these features and also made them publicly available in v3.2.

Incorporating the net costs to PaToH's multi-constraint setting necessitated revising the codes in the three phases of the multilevel framework. In the coarsening phase, net costs are used to account for internal nets. In the initial partitioning phase, the algorithm GHGP [8] is modified to work with the net costs (and also the target weights). In the refinement phase, FM [13] heuristic is modified to ensure correct calculation of the cutsize and the vertex move gains.

Adding the target part weights feature necessitated adjusting the recursive bisection algorithm for appropriate propagation of target weights during the recursion. At each bisection, the total vertex weight is split among the first and second half of the parts according to the cumulative sum of the target weights within each half.

Adding the target weight feature also necessitated revisiting the vertex selection mechanisms during refinement with the FM heuristic [13]. The original FM heuristic is a multi-pass, local search algorithm. Starting from an initial solution, in each pass, the algorithm "searches" *neighbor* solutions by moving a vertex with the highest gain (reduction in the cut cost) to the "other" part, as long as this move does not violate the balance constraint given in (5). A moved vertex is locked for the rest of the pass to avoid revisiting the same solution. FM allows non-improving moves in order to overcome the local optima.

In our improved multi-constraint bisection code, we create multiple priority queues, as many as the number of constraints in the problem, and each vertex is inserted into only one of the queues. To make the weights of a vertex for each constraint comparable, we normalize them so that the total weight for each constraint is 1. That is, we set

$$w(v,t) = \frac{w(v,t)}{\sum_{v \in \mathcal{V}} w(v,t)},$$

so that $\sum_{v \in \mathcal{V}} w(v,t) = 1$ for each constraint t. Hence, the individual constraint weights of a vertex become comparable. A vertex whose t-th constraint is heaviest is inserted to t-th priority queue with its move gain. When selecting a vertex to move, first, current imbalance ratios for each constraint are computed by using the target part weights ($W_{tar}(k,t)$) and the user requested imbalance ratios, $\varepsilon(t)$ where $t \in \{1, 2, \ldots, T\}$. In order to make the imbalance ratios for all constraints comparable, we normalize the imbalance ratios with respected to the tightest imbalance (ε^*). That is, we first compute $\varepsilon^* = \min_t \varepsilon(t)$. Then, for each constraint, we compute the current (comparable) imbalance ratios ($\varepsilon_{cur}(t)$) as

$$\varepsilon_{cur}(t) = \frac{W_{max}(t) - W_{tar}(t)}{W_{tar}(t)} \frac{\varepsilon(t)}{\varepsilon^*} \tag{6}$$

Before each move, we sort the constraints in non-decreasing order of these imbalance values. Then visiting each constraint in this order, we check if the respected priority queue has a vertex that can be moved from the heavily loaded part to the least loaded part. We select the first vertex found in this search. Note that this heuristic move approach does not guarantee that each move leads to a partition that respects the user requested imbalance ratios (given in (5)), but it is a best effort algorithm to improve the balance. Therefore, feasibility of the move is checked after the vertex is selected. If the current partition satisfies the balance constraint in (5), but the move leads to a partitioning which does not satisfy the constraint, that move is not allowed in that pass. However, there might be cases where the current partition does not satisfy the balance constraint. For such cases, a move is allowed if it is improving the balance. Since our modified FM does multiple passes, all the vertices will be reconsidered for move in the next passes.

3.2 Integrated data placement and task assignment

By using the enhanced PaToH, we can integrate the data placement and task assignment problems and solve them as a multi-constraint hypergraph partitioning problem with target part weights. To do this, we first construct the hypergraph as described above and partition it by setting the target weight of each part to the corresponding execution site's desirability value. That is, we set

$$W_{tar}(k,1) = des_c(s_k)$$
$$W_{tar}(k,2) = des_f(s_k)$$

for each part $\mathcal{V}_k$ and the corresponding site s_k. Since PaToH partitions the vertices while respecting $W_{tar}(k,1)$ and $W_{tar}(k,2)$ for each part $\mathcal{V}_k$, the partition Π can be used to obtain the integrated data placement and task assignment scheme as follows. For each file vertex in part $\mathcal{V}_k$, the corresponding file is placed into the execution site s_k. Similarly, for each task vertex in part $\mathcal{V}_k$, the corresponding task is assigned to s_k. With this integrated scheme, the total file transfer is expected to be reduced since it is equal to the cutsize and minimizing this cutsize is the objective of the hypergraph partitioning problem.

4. A RELATED METHOD

Yuan et al. [30] present a data placement method for scientific workflows. To the best of our knowledge, this is the closest work to ours. Here we summarize their algorithm with some simplifications. We assume there is no fixed location data set.

Yuan et al. propose a strategy which includes heuristics for initial data placement, the assignment of generated data and tasks during workflow execution, and, if need be, adjusting the data placement during the execution.

In the initial placement phase, the authors cluster the files in such a way that highly related files are placed in the same site. For this purpose, a similarity matrix D is defined and built for the files. The entry d_{ij} shows the number of tasks that require both of the files f_i and f_j. This matrix is then symmetrically permuted and a diagonal entry p is found so that the bisection $\{f_1, \ldots, f_p\}$ and $\{f_{p+1}, \ldots, f_M\}$ optimizes a certain function (deemed to measure the quality of the clusters found). Then each subset of files is recursively partitioned into two until each subset can be fitted into an execution site's storage space.

When a task arrives during workflow execution, Yuan et al.'s algorithm greedily assigns it to the execution site which stores most of its required files. When a data file is generated, it is placed to the site which contains files having large cumulative similarity score with the subject file. If the placement becomes infeasible, then a new partition is computed by following the procedures of the initial data placement stage.

In our use case, the algorithm described above reads as follows. First it finds a placement of the data files such that highly related files are placed at a common site while respecting the target weight values. Then, each task t_i is assigned greedily to the site which contains the highest amount of files needed by the task t_i, of course, while respecting the target weight values of the sites for the task assignment.

Consider the standard undirected graph $G(D) = (\mathcal{V}, \mathcal{E})$ of the matrix D where each vertex in $\mathcal{V}$ corresponds to a file in $\mathcal{F}$ and each edge in $\mathcal{E}$ corresponds to the similarity of the respective files. The weight of each vertex $v_i \in \mathcal{V}$ which corresponds to the file f_i is set to $size(f_i)$. Consider a K-way partition of the vertices in $\mathcal{V}$ which reduces the total weight of the cut edges and respects the target part weights. Clearly, this is an effective heuristic for placing the files, as it uses the well-known edge cut metric and benefits from the state of the art graph partitioning tools such as MeTiS [19] (MeTiS has a subroutine for partitioning graphs with target part weights). The tasks can then be assigned to execution sites following an approach similar to the first-fit decreasing algorithm for the bin packing problem. The tasks are visited in decreasing order of their execution times, and a task t_i is assigned to a site s_i that can accommodate t_i and $size(files(s_i) \cap files(t_i))$ is the largest. This algorithm is referred as DP from now on. The method DP should be an improvement over the original algorithm of Yuan et al. as MeTiS is quite a successful tool.

5. EXPERIMENTAL RESULTS

The proposed data placement and task assignment tool is referred to as DPTA. The method was implemented in C programming language and it is available from the authors upon request. We tested the performance of DPTA and DP by running them on a set of benchmark workflows and on some other synthetically generated ones.

5.1 Data set

We have used six workflows from Pegasus [5]. The name and the characteristics of those workflows are given in Table 1 (the first six rows). These were the instances in each class of workflow with the highest number of tasks. These workflows from Pegasus web page `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator` have the names

- `CYBERSHAKE.n.1000.0`, referred to as C-shake in table;
- `GENOME.d.11232795712.12`, referred to as Gen-d,
- `GENOME.n.6000.0`, referred to as Gen-n,
- `LIGO.n.1000.0`, referred to as Ligo;
- `MONTAGE.n.1000.0`, referred to as Montage;
- `SIPHT.n.6000.0`, referred to as Sipht.

For a little larger scale experiments, we have generated three synthetic workflows, each having equal number of tasks and files. The characteristics of these workflows are given in Table 1 after the Pegasus data set. We assume that 20% of the files are the input files already exist before the start of the execution. The rest of the files are generated by a task in the workflow. Each file is requested by a random number of tasks selected from the set $\{0, 1, \ldots, 16\}$. This number is obtained by first selecting a number from a normal distribution with $\mu = 9$ and then rounding it using the *ceil*() function. If the random integer is smaller than 0 or larger than 16, we repeat the selection. The file sizes and the execution times are randomly selected from a normal distribution with $\mu = 20$ and $\sigma = 6$ in a similar fashion, and if the selected integer is smaller than 1 or larger than 40, we repeat the selection. After selecting the random values, to fix the *computation-to-communication* ratio of the workflow to one, we first compute the current ratio as

$$r = \frac{\text{total execution time}}{\text{total file access}} = \frac{\sum_{t_i \in \mathcal{T}} exec(t_i)}{\sum_{t_i \in \mathcal{T}} size(files(t_i))}$$

and then multiply the execution times with $1/r$ and use the *ceil*() function to round them up. Note that we are rounding the parameters since PaToH accepts integer weights by design.

The same normal distribution parameters, $\mu = 20$ and $\sigma = 6$, are also used to select the target values for execution sites. To compute $des_f(s_i)$ and $des_t(s_i)$, for each target value, we choose a random positive real value from the above distribution. Let

$$FDes = \sum_{i=1}^{K} des_f(i) \quad \text{and} \quad TDes = \sum_{i=1}^{K} des_t(i)$$

be the sums of the randomly selected target values. To satisfy (1), i.e., to make the sums equal to one, for each execution site s_i, we normalize its target values as

$$des_f(s_i) = \frac{des_f(s_i)}{FDes} \quad \text{and} \quad des_t(s_i) = \frac{des_t(s_i)}{TDes}.$$

Name	N	M	# files per task			# tasks per file		
			avg	min	max	avg	min	max
C-shake	1000	1513	3	1	5	2	1	92
Gen-d	3011	4487	3	2	35	2	1	736
Gen-n	5997	8887	3	2	114	2	1	1443
Ligo	1000	1513	6	2	181	4	1	739
Montage	1000	843	7	2	334	8	1	829
Sipht	6000	7968	65	2	954	49	1	4254
wf6k	6000	6000	9	1	18	9	1	17
wf8k	8000	8000	9	1	18	9	1	17
wf10k	10000	10000	9	1	19	9	1	17

Table 1: The data set contains six benchmark workflows (first six in the table) from Pegasus workflow gallery, and three synthetic ones.

5.2 Evaluation

To compare the performance of scheduling heuristics, the traditional way is simulating the schedules and find the makespan of the workflow. Instead, to compare DP and DPTA, we use the total amount of file transfers and the maximum deviation from the target values. Note that to obtain the makespan, we need the processor speeds, bandwidth, data management and scheduling strategies as well as other parameters. Since the main objective of the data placement algorithms is reducing the total amount of file transfers [30], we directly compare this value. Such a comparison is logical since, in a cloud computing environment, more reduction in the total amount of required file transfer usually leads to a shorter makespan and less cost for the execution.

We report the comparison of the two algorithms in Table 2. In this table, DP refer to the improved existing heuristic discussed in Section 4 and DPTA refers to the proposed simultaneous data placement and task assignment heuristic. For a given K-way task assignment, we compute

$$\max_k \left\{ \frac{\sum_{v_i \in \mathcal{V}_k} w(v_i, 1)}{W_{tar}(k,1) \times \sum_{v_i \in \mathcal{V}} w(v_i, 1)} \right\}$$

and report that number in the column `Tasks`. This number gives the maximum ratio of an execution site's computational load to its target computational load. Similarly, we compute

$$\max_k \left\{ \frac{\sum_{v_i \in \mathcal{V}_k} w(v_i, 2)}{W_{tar}(k,2) \times \sum_{v_i \in \mathcal{V}} w(v_i, 2)} \right\}$$

and report that number in the column `Files`. Again, this number gives the maximum ratio of an execution site's storage load to the target storage load. We also compute the total size of the communication and divide that to the total size of the files to measure the communication requirements of a given data placement and task assignment. The columns `Comm` contain those numbers. Therefore, the minimum number in the columns `Tasks` and `Data` can be 1.00, where smaller the better. The maximum number in the column `Comm` can be as high as $(K-1) \times \sum_{f_i \in \mathcal{F}} size(f_i)$ which happens when all files are needed in all execution sites. Clearly, one prefers small numbers in this column. Since partitioning tools we used for DP and DPTA (MeTiS and PaToH) contain randomized algorithms, therefore, for a given data set and a number of execution sites, DP and DPTA were run ten times and the average of those ten runs were listed in the table. The averages for the real world data set and the synthetic data set are reported by normalizing each entry to the corresponding entry in DP.

The load balance achieved by DP is remarkable, mainly thanks to tackling the two balance issues separately. On the other hand, DPTA optimizes the total file transfer amount much better than DP, resulting in about 38% improvement in the real world data set and about 50% improvement in the synthetic data set. The proposed algorithm DPTA also achieves significant balance in terms of tasks and files assigned to each execution site, where, to our surprise, its balance results for the synthetic data set are slightly better than those of DP.

We run the two heuristics on an Intel based MacBook Pro doted with a 2.53 GHz dual core processor, with 6 MByte of L2 cache to measure their execution times. The proposed DPTA method runs quite fast. In all workflows of the data set, with $K = 64$, the integrated approach finds a data placement and task assignment scheme in less than 3 seconds. The DP heuristic is quite fast too; excluding the construction of data similarity score, it run in less than 7 seconds for the largest workflow (wf10k). Hence, one can safely ignore the placement overhead since the execution times of these heuristics are negligibly small compared to the actual execution time of the real workflows in cloud.

6. CONCLUSION

We have proposed a heuristic called DPTA for optimizing the execution of scientific workflows in the Cloud. The proposed heuristics reduce the communication cost while distributing the input and output data and the tasks to execution sites according to a set of user-specified ratios. The heuristic was based on a hypergraph model and its partitioning for which there were no publicly available tool support. We have enhanced PaToH [8] to encapsulate the arising partitioning problem. We have also implemented an existing method, with ameliorations, and compared the performance of DPTA with respect to that method on some benchmark workflows from Pegasus [5] workflow gallery and on some synthetic workflows. We have seen around 38% improvement in the communication cost with DPTA for the Pegasus workflows and close to 50% on the synthetic ones. Both heuristics are quite fast in practice.

In some cases, some input files and the final destination of generated files (output) of a workflow may be specified, due to users' preferences or system requirements. A common technique in addressing such requirements is to include fixed vertices into the hypergraph model. We have not yet added this feature to PaToH, but we think that this could be a useful feature for data placement problems in the Cloud.

Some other work remains to be done in order to adapt the proposed heuristic to dynamic workflows whose execution pattern changes in time. If the changes are gradual, then the proposed heuristic can be run at certain time intervals to reassign the tasks and files for the upcoming computations. More adequately, the heuristic can be used within the repartitioning methods (see for example [7]) to reassign the tasks and files while reducing the associated migration costs.

Our current approach yields a two-phase scheduling of the workflows, where in the first phase data placement and task assignment are solved in an integrated manner, and

Data	K	DP			DPTA		
		Tasks	Files	Comm	Tasks	Files	Comm
C-shake	4	1.000	1.388	0.123	1.199	1.619	0.119
	8	1.002	1.388	0.294	1.192	1.465	0.489
	16	1.005	1.554	0.613	1.553	1.733	0.809
	32	1.031	2.865	0.780	1.932	2.670	0.882
Gen.d	4	1.001	1.001	1.025	1.034	1.021	0.300
	8	1.024	1.208	0.983	1.018	1.075	0.664
	16	1.102	1.636	1.686	1.488	1.614	1.509
	32	1.238	3.168	3.286	1.573	2.709	2.992
Gen.n	4	1.000	1.000	0.871	1.004	1.000	0.079
	8	1.000	1.002	0.980	1.013	1.004	0.152
	16	1.019	1.030	0.706	1.021	1.112	0.278
	32	1.134	1.097	0.891	1.055	1.096	0.538
Ligo	4	1.000	1.004	0.677	1.380	1.171	0.067
	8	1.001	1.017	0.975	1.193	1.269	0.144
	16	1.004	1.032	1.322	1.110	1.397	0.149
	32	1.019	1.034	1.611	1.109	1.315	0.227
Montage	4	1.003	1.007	0.932	1.002	1.001	0.564
	8	1.063	1.006	1.564	1.007	1.006	0.863
	16	1.181	1.254	1.931	1.023	1.121	1.153
	32	1.248	2.108	2.312	1.137	2.374	1.568
Sipht	4	1.000	1.001	1.223	1.000	1.000	0.604
	8	1.000	1.002	1.850	1.003	1.004	1.300
	16	1.000	1.030	3.781	1.016	1.014	2.923
	32	1.001	1.031	7.224	1.059	1.037	5.515
Average		1.000	1.000	1.000	1.124	1.048	0.615
wf6k	16	1.008	1.030	4.546	1.005	1.002	2.044
	32	1.036	1.030	5.407	1.009	1.003	2.765
	64	1.348	1.030	6.032	1.130	1.052	3.184
wf8k	16	1.007	1.030	4.603	1.004	1.002	2.208
	32	1.026	1.030	5.462	1.009	1.003	2.975
	64	1.218	1.030	6.066	1.099	1.032	3.118
wf10k	16	1.003	1.030	4.614	1.003	1.001	2.076
	32	1.016	1.030	5.472	1.007	1.003	2.757
	64	1.141	1.030	6.095	1.176	1.074	3.228
Average		1.000	1.000	1.000	0.968	0.989	0.501

Table 2: Comparison of improved heuristic DP for the data placement and task assignment problems, and the proposed heuristic DPTA. The averages for the real world data set and the synthetic data set are reported by normalizing each entry to the corresponding entry in DP.

in the second phase tasks are executed obeying the dependencies among them. We hypothesize that such a scheduling framework is extremely effective for data-intensive workflows, both in terms of computational complexity and performance. However, naturally, another future work direction is to develop a single-phase scheduling approach that would take all of the constraints of the problem and directly minimizes the makespan of the workflow.

7. REFERENCES

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: automated data placement for geo-distributed cloud services. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[2] K. Agrawal, A. Benoit, F. Dufossé, and Y. Robert. Mapping filtering streaming applications with communication costs. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 19–28, New York, NY, USA, 2009. ACM.

[3] C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, May 2008.

[4] A. Benoit, U. Catalyurek, Y. Robert, and E. Saule. A Survey of Pipelined Workflow Scheduling: Models and Algorithms. Technical Report RR-LIP-2010-28, LIP, ENS Lyon, France, Sep 2010.

[5] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10, 2008.

[6] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. Resource monitoring and management with OVIS to enable HPC in cloud computing environments. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2009.

[7] U. V. Catalyurek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen. A repartitioning hypergraph model for dynamic load balancing. *J. Parallel Distrib. Comput.*, 69:711–724, August 2009.

[8] Ü. V. Çatalyürek and C. Aykanat. PaToH: A multilevel hypergraph partitioning tool, version 3.0. Technical Report BU-CE-9915, Computer Engineering Department, Bilkent University, 1999.

[9] Ü. V. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *ACM/IEEE SC2001*, Denver, CO, November 2001.

[10] J. M. Cope, N. Trebon, H. M. Tufo, and P. Beckman. Robust data placement in urgent computing environments. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–13, Washington, DC, USA, 2009. IEEE Computer Society.

[11] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In M. D. Dikaiakos, editor, *Grid Computing*, volume 3165 of *Lecture Notes in Computer Science*, pages 131–140. Springer Berlin / Heidelberg, 2004.

[12] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. Askalon: A development and grid computing environment for scientific workflows. In I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 450–471. Springer London, 2007.

[13] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[15] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 640–645, Washington, DC, USA, 2008. IEEE Computer Society.

[16] H. Huang and L. Wang. P&P: A combined push-pull model for resource monitoring in cloud computing environment. *Cloud Computing, IEEE International Conference on*, 0:260–267, 2010.

[17] G. Juve and E. Deelman. Scientific workflows and clouds. *Crossroads*, 16:14–18, March 2010.

[18] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Data sharing options for scientific workflows on Amazon EC2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–9, Washington, DC, USA, 2010. IEEE Computer Society.

[19] G. Karypis and V. Kumar. *MeTiS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*. University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[20] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley–Teubner, Chichester, U.K., 1990.

[21] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system: Research articles. *Concurrency and Computation: Practice and Experience*, 18:1039–1065, August 2006.

[22] S. Pandey and R. Buyya. Scheduling data intensive workflow applications based on multi-source parallel data retrieval in distributed computing networks. `http://www.cloudbus.org/reports/MultiDataSourceWorkflowCloud2010.pdf`. last accessed, February 2011.

[23] S. Pandey and R. Buyya. Scheduling and management

techniques for data-intensive application workflows. In *Data Intensive Distributed Computing: Challenges and Solutions for Large-scale Information Management*, USA, April 2010. IGI Global.

[24] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi. Scheduling data-intensiveworkflows onto storage-constrained distributed resources. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, pages 401–409, Washington, DC, USA, 2007. IEEE Computer Society.

[25] T. Shibata, S. Choi, and K. Taura. File-access characteristics of data-intensice workflow applications. In *Proc. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 522–525. IEEE, 2010.

[26] T. Shibata, S. Choi, and K. Taura. File-access patterns of data-intensive workflow applications and their implications to distributed filesystems. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 746–755, New York, NY, USA, 2010. ACM.

[27] Y. Simmhan, R. Barga, C. van Ingen, E. Lazowska, and A. Szalay. On building scientific workflow systems for data management in the cloud. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 434–435, Washington, DC, USA, 2008. IEEE Computer Society.

[28] W. M. P. van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14:5–51, July 2003.

[29] A. Weiss. Computing in the clouds. *netWorker*, 11:16–25, December 2007.

[30] D. Yuan, Y. Yang, X. Liu, and J. Chen. A data placement strategy in scientific cloud workflows. *Future Generation Computing Systems*, 26:1200–1214, October 2010.