DOI 10.12694/scpe.v20i3.1556
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 20, Number 3, pp. 541–562. http://www.scpe.org © 2019 SCPE

E-DPSIW-FCA: ENERGY AWARE FCA-BASED DATA PLACEMENT STRATEGY FOR INTENSIVE WORKFLOW

RIHAB DEROUICHE[*] ZAKI BRAHMI[†] MOHAMMED MOHSEN GAMMOUDI [‡] AND SEBASTIÁN GARCÍA GALÁN [§]

Abstract. Intensive Workflows are composed of large number of complex tasks and require a large amount of data located in different Storage Computing Servers (*SC*). The data movement between *SC* causes high communication and data movement cost. In this paper, a data placement strategy based on Formal Concept Analysis approach (E-DPSIW-FCA) is proposed aiming to reduce the data movement, the consumed energy, and the workflow execution cost. FCA allows to group the maximum of data and tasks in an hierarchical structure called lattice concepts. These concepts are mapped to the appropriate *SC*. The navigation through the hierarchy of concepts is considered as a solution of the case when the data group size exceeds the *SC* storage capacity. The simulations results show that E-DPSIW-FCA can achieve better results than the K-means [4] and genetic algorithm [14] based approaches.

Key words: Data Placement, Intensive Workflow, Cloud Computing, FCA, Energy, Communication, Computing, Granularity, Network, Level

AMS subject classifications. 68M14, 68P20

1. Introduction.

Workflows are composed of a sequence of operations and declared as a work of person or a group [1]. They have been used in a number of scientific applications, which generate massive amount of data every day, such as astronomy [2] and bioinformatics [3]. These workflows are potentially intensive and comprise hundreds or thousands of complex tasks and big datasets which need to be stored and, make its processing very complex [4]. For instance, the Cybershake workflow, which is used to calculate probabilistic seismic hazard curves for several geographic sites in the southern California area, has an average execution time that can be up to 8 hours and 51 minutes for running 2 083 325 tasks and generates a huge amount of datasets [5] [6]. So, in order to process and store these huge amounts of workflow data, Cloud providers are deploying large number of Computing and Storage devices to address and satisfy the ever increasing user's requirements for more computing capacity, storage and memory. As well as that, they offer different Cloud storage resources including Amazon Web Services (AWS) which offers various kinds of cloud storage systems [7]. For example, Elastic Block Store (EBS) provides persistent block storage volumes for use with Amazon Elastic Compute Cloud (Amazon EC2) instances whereas the data are delivered as data block [8]. Amazon Elastic Compute Cloud (Amazon EC2) instances and the International Business Machines (IBM) offered a block storage with up to 12 TB in capacities [9].

The execution of a workflow task requires a massive volume of datasets, which are physically distributed and stored in multiple *SC*. Thus, data movement between *SC* will be inevitable and it would generate a significant data movement cost due to the difference in the location between the data processing task and the necessary dataset. Consequently, transfer large amounts of data between *SC* increases notably the consumed energy by the networking devices, communication links, hence the increasing of the workflow execution cost. Further, processing such a large size of moved data for executing workflows in the Cloud stands as a challenge [1]. According to [10], a major portion of the consumed energy by the data center is utilized to maintain interconnection links and network equipment operations. This consumed energy account for up to 50% of the total energy consumption of a data center [11] and it is caused mainly by switches, Local Network Area infrastructure (or LAN), routers, etc [12]. Reducing the energy consumption of a Data Center Networks (DCNs) is considered as an essential step for advancing energy efficiency in Cloud Computing paradigm [11]. It is considered hence as a challenge [1].

To address data placement problem, many works had been proposed such as [3] [4] [13] [25], etc. However, to the best of our knowledge, most existing data placement studies have considered storing data in data centers

[*]Faculty of Sciences of Tunis, RIADI-GDL Laboratory, Tunisia, (derouicherihab@gmail.com).
[†]Taibah University, KSA, RIADI-GDL Laboratory, Tunisia, (zakibrahmi@gmail.com).
[‡]ISAMM, University of Mannouba, RIADI-GDL Laboratory, Tunisia, (gammoudimomo@gmail.com).
[§]Higher Polytechnic School of Linares, University of Jéan, Spain, (sgalan@ujaen.es).

without taken into account the granularity of data centers used resources, and the network aspect of a data center infrastructure. More specifically, they did not cope with the consumed energy incurred by the networking devices, the communication links and by the computing devices during the data placement and movement. Thus, we are interested in proposing a strategy for data placement that considers the aspects cited previously. To resolve this data placement problem, it is important to manage effectively these datasets in order to minimize both the energy consumption and the total data movement cost efficiently during the workflow execution. This can be achieved by placing and distributing intelligently these datasets in order to reduce the consumed energy by computing, storage and communication devices as well as reducing the cost of using these devices.

In this paper, a novel Data Placement Strategy based on Formal Concept Analysis (E-DPSDIW-FCA) is proposed. Indeed, contrarily to others studies, aiming to group the maximum of dependent datasets, our approach operates at the granularity of a data center while considering its different levels of communication (routers, switches, etc.), so that, to provide an efficient energy.

Overall, our main contributions are summarized as follows:

1. Applied the FCA approach in a new field which is massive data placement. Indeed, we used the FCA approach to identify the tasks-datasets associations (Formal concepts) which indicate the dependency among datasets and tasks,
2. Proposed a data placement algorithm to distribute and place original data and tasks based on their dependencies in SC. For this end, as to save lattice computing time, firstly, we propose a K-means based algorithm that allows dynamically clustering the input scientific workflows in order to identify the most convenient algorithm to apply for the lattice concept computing. Second, we reduce the size of the initial formal context. Additionally, an heuristic is defined by exploiting lattice concept level, and this in order to avoid examining all concepts at the process of mapping datasets to SC.
3. Proposed a new data transfer cost model closely to real Cloud environment by using data slices.
4. Proposed a novel energy model to evaluate the energy consumption by communication, computing and storage devices,
5. Formulated a mono-objective mathematical optimization model for workflow data placement taking into account both the communication energy and the computing energy consumption, the data movement and the quality metrics of server as explicit decision criteria, and
6. Implemented our algorithm and evaluate its performances with some scientific intensive workflows.

The remainder of this paper is organized as follows: in the section 2, we perform the system modeling and the problem formulation. In section 3, we detail our proposed solution and justify our choice for FCA. Section 4 demonstrates the simulation results and the evaluation of our approach. Section 5 highlights the major related works addressing the data placement problem and the existing energy models strategies. Finally, we conclude our work and we give some perspectives.

2. System Modeling and Problem Formulation.

2.1. Workflow Modeling. A workflow W is modeled as: $W = (T, D^{\text{in}}, TS, DS)$. $T = \{t_i | i = 1, ..., n\}$ represents the set of workflow tasks with n the number of tasks. $D^{\text{in}} = \{d_i^{\text{in}} | i = 1, ..., m\}$ represents the amount of original datasets to be processed by each workflow task t_i and m is the number of datasets that are consumed by such workflow task as input datasets. Each dataset d_i^{in} has a size denoted as $size(d_i^{\text{in}})$. This size is defined in some pre-determined unit such as mega-bytes, gigabytes or tera-bytes. Each task t_i requires a set of data denoted as $D^{\text{in}}(t_i)$ from the set D^{in} to satisfy its execution. TS: $D^{\text{in}} \rightarrow T$ is a function dataset-task that returns the set of workflow tasks that consume such datasets as their inputs. DS: $T \rightarrow D^{\text{in}}$ is a function task-dataset that returns the set of datasets that are consumed by such task as its inputs.

2.2. Cloud Computing Environment Modeling. A Cloud Computing environment is a 3-tuple: $CC = (DCN, SC, Scap)$ where

(i) DCN represents the datacenter network architecture, that consists of a set of Computing and Networking devices. In addition, the computing devices are the servers (storage computing) denoted as SC and the networking devices are the network switches and routers denoted as SWs.

(ii) $SC = \{sc_i | i = 1, ..., s\}$ represents a set $'s'$ of storage computing servers from the Cloud environment. As mentioned above, SC may be virtual machines, dedicated storage sites, Amazon S3, Elastic Block Store, etc. Each sc_i has a storage capacity denoted as $Cap(i)$.

The communication link among the SC is defined as the bandwidth demand or the traffic load between two SC sc_i and sc_j. Besides, this communication link is expressed as: $BW(sc_i, sc_j)$.

A computing server sc_i is defined by three quality metrics (M_i) used to evaluated its performance denoted as: $M_i = (C_{storage}, C_{processing}, C_{DataTransfer})$ [63].

Actually, our goal lying behind evaluating the performance of server sc_i consists in reducing the data transfer cost, the storage cost and the processing cost. Hence, the goal is to minimize the sum Q_i:

$$Q_i = C_{storage} + C_{processing} + C_{DataTransfer} \tag{2.1}$$

where:

$C_{storage}$ $= T_{remain} \times D^{in} \times CostperStorage$ represents the cost of data storage where T_{remain} is the time that the data D^{in} is remaining in sc_i, D^{in} represents the stored data in sc_i and $CostperStorage$ is the cost of hosting time per second. For instance, the simple storage service Amazon S3 offers a range of storage classes designed for different use cases. There are three highly durable storage classes including Amazon S3 Standard for general-purpose storage of frequently accessed data, Amazon S3 Standard - Infrequent Access for long-lived, but less frequently accessed data, and Amazon Glacier for long-term archive [34]. The storage pricing of Amazon S3 varies by region. For instance, in the standard storage in the region of the US West (Northern California) for 50 TB/month, the price is \$0.026 per GB. However, in South America (Sao Paulo), the price is \$0.0405 per GB [34].

$C_{processing}$ $= (T_{proc} + T_{wait}) \times CostperProcess$ indicates the cost of processing where T_{proc} is the time needed to process a dataset of a task, T_{wait} is the time waiting by a dataset to be processed and $CostperProcess$ is the cost of processing in million instructions per second ($MIPS$).

$C_{DataTransfer}$ is the cost of transferring data within region, across regions or over Internet. Consider sc_i a storage computing server which communicates with server sc_j, data transfer is charged for every gigabyte moved from sc_i to sc_j. Consider DT is the quantity of data to be transmitted between the two servers in gigabytes. The amount of data transferred is divided into slices, and each slice is defined by: (1) its size which is defined by an interval with max and min bounds, (2) its transfer cost depending on its size. For instance, Microsoft Azure uses several data size intervals according to data to be transferred, such as $[5TB - 10TB]$, $[50TB - 150TB]$, etc [58]. Hence, to be more closely to real environment, we propose a new way to compute data transfer cost. Indeed, we define the data transfer cost provided by a sc_i as a set S of n slices $S = \{s_i | i = 1, ..., n\}$. Indeed, each slice s_i is defined as a triplet $< s_i^{max}, s_i^{min}, p_i >$, with p_i is the price of the slice $\left[s_i^{max}, s_i^{min}\right]$. To note, for the latest slice, we have considered only s_n^{min}. For example, a data size over 500 TB, Microsoft Azure considered only the 500 TB [58]. Thus, the monthly data transfer cost is calculated as follows:

$$C_{DataTransfer}(sc_i) = \sum_{i=1}^{n-1} CostSlice(s_i) + (DT - s_n^{min})p_i \tag{2.2}$$

where: $CostSlice(s_i)$ represents the cost of data slice transfer from sc_i to any sc as indicated in the Eq. 2.3:

$$CostSlice(s_i(sc_i, sc_j)) = \begin{cases} (s_i^{max} - s_i^{min}) * p_i \text{ if } (DT_i^{'} > (s_i^{max} - s_i^{min})) \\ \text{otherwise }, DT_i^{'} * p_i \end{cases} \tag{2.3}$$

where $DT_i^{'}$ is the rest of data size for each used slice:

$$DT_i^{'} = DT_{i-1} - (s_{i-1}^{max} - s_{i-i}^{min}).$$

For instance, for the same data slice interval (5TB - 10TB), the prices of data transfer for the Central US is \$0.087 per GB, however, for the East Asia is evaluated as \$0.12 per GB for Microsoft Azure [58]. p_i depends also on the regions where the destination server is located. To note, a region is a geographic location in which public Cloud providers' data centers reside [59]. For example, Amazon Web Services (AWS) operates regions in the United States, South America, etc [59]. Indeed, there's a cost of moving data across servers within the same region, and a different (generally greater) cost for data transfer across servers outside that region. Hence, according to the region, the cost is computed. First, if sc_i and sc_j are within the same region. Then, the traffic will not be charged and p_i is free. Second, if sc_i and sc_j are not in the same region, then costs apply and all outbound traffic is charged. This cost is denoted as $PriceAcrossRegion$. Finally, if the two servers communicate over Internet, so all internet traffic from sc_i to sc_j is bound to costs denoted as $PriceOverInternet$, which are the most greater. For instance, data going out of Azure data centers beloging to the interval (5GB - 10TB) to France Central are charged to \$0.084 per GB. Contrarily, for the same destination region and for the interval (50GB- 150Tb), the cost is evaluated as \$0.07 per GB [58]. (iii)$Scap : SC \to R^+$ is an available storage resource capacity function. $SS(sc_i)$ with $sc_i \in SC$ determine the maximum available storage capacity of sc_i in the CC environment. It may be measured in mega-bytes, giga-bytes or tera-bytes.

2.3. Data Center Network Architecture. In this scope, we have considered the three-tier Cisco data center network architecture [35] to avoid the problem faced with the two tier one when scaling up the number of servers, the network links in the core tier become over-subscribed. We have defined the link distance between sc_i and sc_j as *Physical Link Distance* ($PLD(sc_i, sc_j)$). Further, in order to determinate the communication cost, we assigned a *weight* for every (PLD). This link weight may be any practical measure such as link latency, number of hops or number of switches, etc. In our experiments, the PLD is defined as the number of switches on the routing path from source sc_i to destination sc_j denoted as $Nber_{\text{Switch}}$ and we have considered the fan-out of the access switches ($p0$) as well as the fan-out of the aggregation ones ($p1$) [36]. It can be expressed as follows [36]:

$$Nber_{\text{Switch}}(sc_i, sc_j) = \begin{cases} 0, \text{ if } i = j, \text{and} \\ 1, \text{ if } \lfloor \frac{i}{p_0} \rfloor = \lfloor \frac{j}{p_0} \rfloor, \text{and} \\ 3, \text{ if } \lfloor \frac{i}{p_0} \rfloor \neq \lfloor \frac{j}{p_0} \rfloor \wedge \lfloor \frac{i}{p_0 p_1} \rfloor = \lfloor \frac{j}{p_0 p_1} \rfloor, \\ 5, \text{ if } \lfloor \frac{i}{p_0 p_1} \rfloor \neq \lfloor \frac{j}{p_0 p_1} \rfloor, \end{cases} \tag{2.4}$$

$\lfloor x \rfloor$ is called floor function that gives the largest integer less than or equal to x [37]. $\wedge$ is a AND binary operator. In some cases, the communication between SC induces to increase the energy consumption in the communication elements (routers, switches, etc). Indeed, in a data center, the number of switches involved in the execution of a task is proportional to the position of SC to run the task [38]. Consequently, while examining the two active states and turned off of switch, the consumed energy by a network switch at times t, denoted as $Energy_{\text{Switch}}(t)$, is inspired from [38] and it is evaluated using the following formula:

$$Energy_{\text{Switch}}(t) = \alpha \times [P^{base}_{\text{Switch}}(t) + n_{activeport}(t) \times P^{activeport}_{\text{Switch}}(t)] \tag{2.5}$$

where $\alpha \in [0, 1]$; if $\alpha = 0$, it means that the switch is running, otherwise the switch is turned off. $P^{base}_{\text{Switch}}(t)$ represents the power consumed by the fixed parts of the Switch chassis and linecard and it is expressed as follows:

$$P^{base}_{\text{Switch}}(t) = P^{chassis}_{\text{Switch}}(t) + P^{linecard}_{\text{Switch}}(t) \tag{2.6}$$

$n_{activeport}$ is the number of active ports at time t. $P^{activeport}_{\text{Switch}}(t)$ represents the power consumed by an active port.

2.4. Problem Formulation. The large number of tasks in Intensive Workflow (IW) needs sometimes to process more than one dataset that may be stored in different SC. Moving these datasets increases the workflow execution time while consuming a high energy. As a result, the data movement between SC becomes a bottleneck that leads to limit the overall system performance [7] and increases the total cost of the system [39]. Figure 2.1 shows a sample scientific workflow instance that specifically describes our research questions. This scientific workflow consists of four tasks $\{t_1, t_2, t_3, t_4\}$, five input datasets $\{d_1, d_2, d_3, d_4, d_5\}$ initially existed in

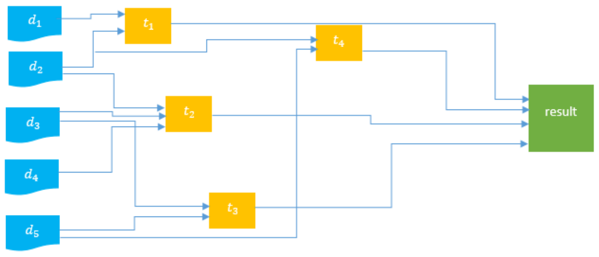


FIG. 2.1. *A Simple instance of a Workflow.*

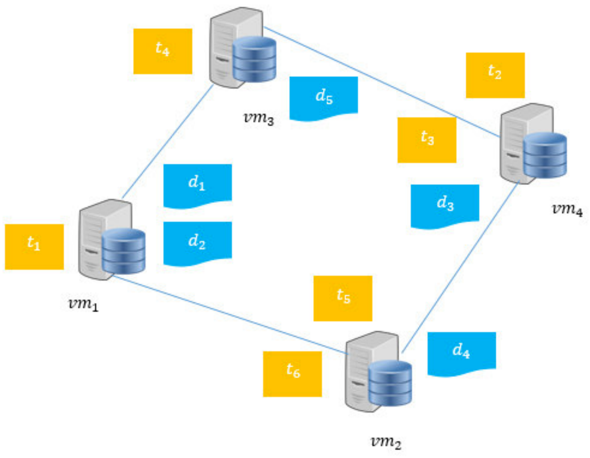


FIG. 2.2. *A virtual machine configuration in the Cloud with four virtual machines for workflow of Fig. 2.1.*

the system. In this figure, the data flows from dataset d_2 to t_1 and t_2 mean that d_2 will be used by both t_1 and t_2 and d_3 will be used by both t_2 and t_3. As shown in Fig.2.2, the task t_1 as well as the datasets d_1 and d_2 are assigned to the virtual machine vm_1. In same manner, the task t_4 and the dataset d_5 are assigned to vm_3. The tasks t_2 and t_3 and the dataset d_3 are placed in vm_4. Once the workflow tasks are executed, the task t_2 needs to transfer the datasets d_2 from vm_1 to vm_4 and d_4 from vm_1 to vm_4 to insure its execution. This data movement may increase the execution time and the energy consumption incurred during the communication between VMs.

2.5. Objective function. The data placement problem consists to providing a mapping of these datasets to SC that satisfies a set of objectives such as reducing the data transfer cost/time. Accordingly, given a workflow W, we formulate our data placement using mono-objective optimization problem formulation that aims to minimize the overall energy consumption and the cost of the workflow. The energy includes the communication and computing energy consumption of used devices. The cost of the workflow includes its data transfer cost, the data storage and the data processing cost. Hence, it is described by the Eq. 2.7:

$$\begin{aligned}
&min((\textstyle\sum_{i=1}^{m}\sum_{j=1}^{s} x_{ij} * \omega_1 * Q_j) + \omega_2 * TDM(W) \\
&+\omega_3 * WorkflowEnergy) \\
&st. \\
&\forall i = 1..m, \forall j = 1..s, x_{ij} \in [0,1]. \\
&\forall j = 1..s, \textstyle\sum_{i=1}^{m} size(i) \leqslant Cap(j) \\
&\forall i = 1..m, \textstyle\sum_{j=1}^{s} x_{ij} = 1 \\
&\textstyle\sum_i \omega_i = 1 \\
&\forall j = 1..s, \forall c = j..s-1, \forall i = 1..m, size(i) < BW(j,c)
\end{aligned} \tag{2.7}$$

s and m represent the number of SC and the number D^{in} respectively. x_{ij} is a boolean variable that indicates which dataset i is assigned to which server j. It is equal to 1 if the dataset i is assigned to the sc_j and to 0 otherwise. We assume that each dataset can be assigned to only one sc_j as indicated by Eq. 2.7. BW is the bandwidth demand or the traffic load between sc_j and sc_c. ω_1, ω_2 and ω_3 are the weight values that indicate the degree of importance of evaluated metrics of server which are described above, the total data movement and the total workflow energy consumption defined below and the costs parameters in the fitness function (Eq. 2.7) with their sum is equal to 1.

2.5.1. Total Data Movement for a given workflow W: $TDM(W)$. Consider the case where the dataset d_m^{in} is stored in sc_i. However, the the task t_x exists in sc_j. Actually, on assume that the d_m^{in} is required to be transferred from sc_i to sc_j in order to satisfy the execution of the task t_x, hence, we define the Data Movement denoted as DM as follows:

$$DM(d_m^{\text{in}}, sc_i, sc_j) = \begin{cases} 0, \text{ if } sc_i \ and \ sc_j are \ in \ the \ same \ region \\ Size(d_m^{\text{in}}), \text{ if } sc_i \ and \ sc_j \ are \ in \ different \ regions \end{cases} \tag{2.8}$$

Therefore, the total data movement TDM for a given workflow W is defined as the total data transferred while executing the whole workflow tasks. It is calculated using the following formula:

$$TDM(W) = \sum_{t_x=1}^{n} \sum_{d_m^{\text{in}} \in DS(t_x)} DM(d_m^{\text{in}}, sc_i, sc_j) \tag{2.9}$$

n represents the number of tasks. When the servers sc_i and sc_j are in different regions, requested data must be transferred from sc_i to sc_j, so data traffic is generated accordingly.

2.5.2. Workflow Energy Consumption: $WorkflowEnergy$. Energy consumed during workflow execution includes both the energy consumed by network devices (CE) and computing and storage ($CompE$) devices, otherwise:

$$WorkflowEnergy = CE + CompE \tag{2.10}$$

Modeling the Workflow Communication Energy: CE. As the total amounts of workflow datasets can be very large, it may severely increase the consumed energy by the network devices (switches) and by the communication links. Thereby, communication energy depends on the total data being transferred, hence, the total data movement was included in the cost modeling of workflow communication consumption Energy. Recently, several works focused on controlling energy consumption of communication links in data center have been proposed aiming to reduce energy such as [11] [55] [62], etc.

In our paper, we propose a communication energy model which has to:
(i) Focus on a three-tiered data center network hierarchy described in Sect. 2.3,
(ii) Take into account the networking devices that consume a significant portion of overall energy consumption in data center in order to save energy,
(iii) Consider the consumed energy by the communication links, and
(iv) Operate at the granularity of the networking devices that may be either switch, router, etc.
For these reasons, our model is inspired from both works in [11] and [54] since the communication energy models proposed in these works meet our requirements. In our model, we assume that each computing storage server communicates with other servers through switches and a dedicated (contention free) reliable link that operates at the transmission rate of $R(i)(bits/s), i = 1, ..., s$ with s is the number of SC. Hence, our communication energy model is expressed as follows:

$$CE = \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} cost(sc_i, sc_j) * DM(d_s^{\text{in}}, sc_i, sc_j) + \sum_{c=1}^{s} \varepsilon_{\text{net}}(c) \tag{2.11}$$

where $cost(sc_i, sc_j)$ is the function computing the communication cost between any two SC. It is defined as the product of the number of switches on the routing path from sc_i to sc_j as described in Eq. 2.4 and the consumed energy by each switch as indicated by Eq. 2.5. Formally, it is expressed as:

$$cost(sc_i, sc_j) = Nber_{Switch}(sc_i, sc_j) \times Energy_{\text{Switch}}(t) \tag{2.12}$$

$\varepsilon_{\text{net}}(j)$ is the consumed energy by the one-way transmission plus switching operation is described in the following formula:

$$\varepsilon_{\text{net}}(j) = P_{\text{net}}(j) \times \frac{size(d_j^{\text{in}})}{R_j} \tag{2.13}$$

$D(j) = \frac{size(d_j^{\text{in}})}{R_j}$ is the delay corresponding to the one-way transmission with $size(d_j^{\text{in}})$ is the dataset size to be moved to other SC and $R(j)$ is the transmission rate. P_{net} is the power consumed by the one-way transmission plus switching operation which depends on the corresponding transmission rate $R(j)$, the bandwidth $BW_j(Hz)$, the noise spectral power density $N_0(j)(BW/Hz)$, (non negative) gain g_j of the j^{th} link and the power consumed by the j^{th} end-to-end connection in the idle mode P_{idle} [29].

$$P_{\text{net}}(j) = \xi_j(2^{\frac{R(j)}{BW_j}} - 1) + P_{idle}(j) \tag{2.14}$$

with $\xi_j = \dfrac{N_0(j) \times BW_j}{g_j}, j = 1, ..., s$

Modeling the Workflow Computation Energy: *CompE*. Several approaches have been proposed to handle reducing computing energy consumption [11] [54] [56], etc. For the computation energy model, we have applied the high level approach and it is inspired from [54] since their model is general and it avoids relying on only processor element which increase the portability of the model and the simulation speed [57]. Hence, our power consumption of a computing server or CPU is linearly related to the server's utilization rate α and the associated power consumption.

$$CompE = (1 - \alpha) \cdot P_{CPU_{idle}} + \alpha \cdot P_{CPU_{Full}} \tag{2.15}$$

where $\alpha \in [0, 1]$ represents the utilization rate of CPU, being $\alpha = 0$, the machine is in an idle state and the associated power consumption is $P_{CPU_{idle}}$, whereas $\alpha = 1$ the machine is in a full state and the associated power consumption is $P_{CPU_{full}}$. For the idle state, the power consumption $P_{CPU_{idle}}$ is expressed as:

$$P_{CPU_{idle}} = A \cdot C \cdot f_{idle} \cdot v_{idle}^2 \tag{2.16}$$

where V_{idle} and f_{idle} denote respectively the voltage V and frequency f when the CPU is in idle state. The power consumed by CPU in the full state $P_{CPU_{full}}$ corresponds to:

$$P_{CPU_{full}} = A \cdot C \cdot f_{full} \cdot v_{full}^2 \tag{2.17}$$

where V_{full} and f_{full} are the voltage V and frequency f when the CPU is in full state.

3. Proposed solution: E-DPSIW-FCA.

Our data placement strategy targets at finding a minimal number of SC where the maximum of datasets and tasks are grouped based on their dependencies. It is based essentially on the FCA approach. Note that reader can review the paper [15] to fully understand the mathematical foundations of FCA approach. In the following, we will justify our choice for this approach.

3.1. Choice of the FCA approach.

The choice of FCA approach is motivated by the following reasons: (i)Apart from the mathematical foundation of the FCA approach, the notion of the concept represents faithfully the notion of tasks grouped based on their common attributes. These attributes represent the common datasets that tasks need for their execution, (ii)Because FCA results could be manipulated by some operators to navigate in Galois-lattice structure, we need it to consider relationships among concepts, (iii)Its effectiveness to deal with the problem faced during the placement of data when the group data size exceeds the storage capacity of SC, that is considered as a compromise for prior data placement strategies. However, using the hierarchy between the lattice concepts, this problem could be resolved by navigating among the different concepts. Furthermore, to understand our solution, we need first to mention the necessary used definitions.

3.2. Definitions.

Extension, Intention and Size of a Concept c_i: This definition was given in [15]. Consider C is a set of p Lattice concepts. Each concept c_i is defined by the couple: $c_i =< Ext, Int >$; with Ext is called the extension of the concept which is the tasks group. Int is called the intent of the same concept which represents the input data. $c_i.Int$ refers to the set of data existed in the intent of the concept c_i. The size of the concept c_i is defined as:

$$Size(c_i) = \sum_{d_i^{\text{in}} \in |c_i.Int|} Size(d_i^{\text{in}}), d_i^{\text{in}} \in D^{\text{in}} \tag{3.1}$$

Sparse Formal Context: The set of Tasks T and the set of original datasets D^{in} are used in order to generate the Formal Context. Indeed, a formal context that contains more number of Zero elements than non-Zero elements is known as *Sparse Formal Context*. More specifically, to check whether a formal context which has O as the number of objects (or entities), A as the number of attributes (or properties) is *sparse*, we need to verify the total number of zero. If this count is more than $(O * A)/2$, we consider this context as *sparse*, otherwise, it is considered as *dense* context.

Level of concept ($Level(c_i)$): In order to improve the solution execution time in [15], we propose to use the level of a concept in a Galois-lattice. It's is defined as the cardinality of its intention in our case. The level of a concept c_i is determined as:

$$Level(c_i) = |c_i.Int| \tag{3.2}$$

It's necessary to note that while navigating in the lattice from the top going bottom, the level of concepts increases. The high level in a Galois-lattice is defined as the maximum cardinality of the all concepts intention. In fact, a high level in a lattice of concepts is described as:

$$High_{level} = max_{j=1}^{p}(Level(c_j)) \tag{3.3}$$

p is the concepts number in the Lattice. Note that the concepts located in the lattice middle have the highest level.

Weight of a Concept ($P(c_i)$): We improved the weight definition in [15] by considering in this scope the granularity of the data center resources. Specifically, we have replaced the data center with the computing storage server in the data placement. Besides, the weight $P(c_i)$ of a concept is denoted as follows:

$$P(c_i) = \frac{|c_i.Ext|}{|T|} \times \frac{|c_i.Int|}{|D^{\text{in}}|} \times \frac{MaxCap(sc_i) - Size(c_i)}{|MaxCap(sci) - Size(c_i)|} \tag{3.4}$$

The measure of the concept weight allows us to have the concepts that contain a maximum number of tasks and datasets. The third factor is used to verify if the concept exceeds c_i the maximum storage capacity of sc_i.

Minimum Data Coverage (MDC): Let $C = \{c_1, c_2..., c_p\}$ the set of p concepts. C is MDC if it covers all datasets. Formally, we define MDC of C by:

$$MDC(C) = \begin{cases} 1, \text{ if } \bigcup_{1\leq i\leq p} c_i.Int = D^{\text{in}} \text{ and } \bigcap_{1\leq i\leq p} c_i.Int = \oslash \\ 0, \text{otherwise} \end{cases} \tag{3.5}$$

Candidacy of a Concept c_i ($Candidate(c_i)$): A concept c_i is called candidate concept, if it has a maximum weight and it covers the whole datasets. We define the candidacy of c_i as follows:

$$Candidacy(c_i) = \begin{cases} 1, \text{ if } P(c_i) \text{ is maximum and } MDC(c_i) = 1 \\ 0, \text{ otherwise} \end{cases} \tag{3.6}$$

Feasibility of a Concept c_i (Feasibility (c_i)): A concept is feasible if the total size of its datasets is small enough to be assigned to one storage computing server to be able to assign them. It is defined as follows:

$$Feasibility(c_i) = \begin{cases} 1, \text{ if } \exists sc_i \text{ where } \sum_{d_i^{\text{in}} \in cc_i.Int} \\ Size(d_i^{\text{in}}) \leqslant Cap(sc_i) \\ 0, \text{ otherwise} \end{cases} \tag{3.7}$$

Score of Dataset d_i^{in} (Score (d_i^{in})): For each dataset $d_i^{\text{in}} \in D^{\text{in}}$, let $SC_{d_i^{\text{in}}}$ denotes the set of SC that contained d_i^{in}.Thus, the score of dataset d_i^{in} is defined as:

$$Score(d_i^{\text{in}}) = |\{t_i \in T \text{ such as } R_{t_i} \cap SC_{d_i^{\text{in}}} = \{d_i^{\text{in}}\}\}| \tag{3.8}$$

R_{t_i} is a partial order relation which gives all the datasets used by a task t_i. Different from the solution in [15], the datacenter in the feasibility formula (3.7) and the score definition (3.8) was replaced by the storage computing server.

Tasks dependency $dependency^T(t_i, t_j)$: $dependency^T(t_i, t_j)$ represents the dependency between the tasks t_i and t_j. To note, two tasks are dependent if they use the same datasets to satisfy their execution. Consequently, the number of dependency between the tasks t_i and t_j is the number of data that are used by both t_i and t_j which is defined as follows:

$$dependency^T(t_i, t_j) = Count(D^{\text{in}}(t_i) \cap D^{\text{in}}(t_j)) \tag{3.9}$$

where: $D^{\text{in}}(t_i) \subseteq D^{\text{in}}$, $D^{\text{in}}(t_j) \subseteq D^{\text{in}}$ are the datasets that the tasks t_i and t_j respectively require to execution. $Count$ returns the number of data used by both the tasks t_i and t_j.

Quantity of transferred data: (QtDT(d^{in}_{com}, c_i, c_j)) Consider two concepts c_i and c_j, and d^{in}_{com} a common original dataset used by the two concepts: $d^{\text{in}}_{com} = \{c_i.Int\} \cap \{c_j.Int\}$. We have to verify the condition of the minimum data coverage $\bigcap_{1 \leq i \leq p} c_i.Int = \oslash$ which represents the set of elements which are in c_i or c_j, or in both c_i and c_j. Then, we have to decide where to place the common dataset between the two concepts c_i and c_j. Indeed, our choice is based essentially on finding the best placement which requires a minimum of: (i)energy consumed during communication as described in Eq. 2.11, (ii)total data movement (Eq. 2.9) hopping through network devices that will eventually reduces the network overhead. The quantity of transmitted data between two concepts is described in the Algorithm 1 where sc_i, sc_j are the concepts' computing storage servers c_i and c_j. $Count(t_i)$ and $Count(t_j)$ are the number of tasks presents in the concepts c_i and c_j respectively that require the common dataset d^{in}_{com}. $DM(d^{\text{in}}_{com}, sc_i, sc_j)$ and CE are described above in (Eq. 2.8) and (Eq. 2.11).

Algorithm 1 Quantity of transferred Data function

```
1: function QtDT($d^{\text{in}}_{com}, c_i, c_j, sc_i, sc_j$)          ▷ $d^{\text{in}}_{\text{com}}$ is placed in $c_i$
2:     if $d^{\text{in}}_{com} \in \{c_i.Int\}$ then
3:         $QtDT(d^{\text{in}}_{com}, c_i, c_j, sc_i, sc_j) = \sum_{d^{\text{in}}_j \in c_j.Int} Size(d^{\text{in}}_j) - Size(d^{\text{in}}_{com})) \times (Count(t_j))t_j \in c_j.Ext(t_j) + DM(d^{\text{in}}_{com}, sc_i, sc_j) + CE$
4:     end if
5: end function
```

Overall, we aim to place the independent data in a minimum number of SC, subsequently, in the lowest level of communication in order to reduce the data movement between SC and minimizing the workflow communication energy. Indeed, finding the appropriate storage computing server to place d^{in}_{com} among sc_i and sc_j described by the Algorithm 2.

Algorithm 2 Appropriate Placement's Algorithm

```
1: function PLACEMENT($QtDT(d^{\text{in}}_{com}, c_i, c_j, sc_i, sc_j)$, $QtDT(d^{\text{in}}_{com}, c_j, c_i, sc_i, sc_j)$)
2:     Output: Var $AppropriatePlacement$
3:     if QtDT($d^{\text{in}}_{com}, c_i, c_j, sc_i, sc_j$) < QtDT($d^{\text{in}}_{com}, c_j, c_i, sc_i, sc_j$) then
4:         $AppropriatePlacement = sc_i$
5:     else
6:         $AppropriatePlacement = sc_j$
7:     end if
8: end function
```

3.3. Proposed Data Placement Solution. In our previous work, a data placement strategy to dataset-datacenter mappings was proposed. It was just a start towards using FCA for the data placement problem. Based on this work, we have further investigated the characteristics of FCA approach to improve our previous work steps by: (i)classify input workflows and reduce formal context in order to save time when building the

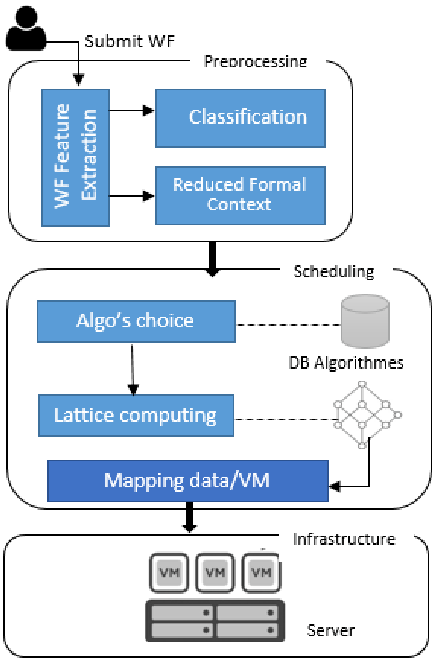


Fig. 3.1. *Overview of our proposed solution E-DPSIW-FCA.*

lattice, and (ii)exploit the lattice concepts level notion to define an heuristic in order to prevent analyzing all the concepts during the candidate concepts selection. Fig. 3.1 presents the architecture of E-DPSIW-FCA:

3.3.1. Workflow Features Extraction. We have resorted to use an XML parser to parse the workflow features, which is stored in XML format. Indeed, the XML parser extracted all workflows tasks, a set of input datasets required for their execution and their sizes and also the dependencies between these tasks.

3.3.2. Pre-processing. We improved our previous work by adding this new step, which consists of two phases:

Workflows Classification: In this stage, we classify the input workflows in order to identify the most convenient techniques or algorithms to be used for the generation of Galois-lattice. Thus, we have to consider the characteristics of the input workflows for the choice of these algorithms. Indeed, these characteristics include the following metrics: the number of datasets, and the dependency between the tasks of the workflow [63]. Based on data dependencies between the tasks, we defined a set of metrics to initially classify the input workflows as workflows with high dependency and workflows with low dependency. Noteworthy that this classification allow us identifying the dense formal context and this is in the case of high dependency workflows and the sparse formal context in the case of ow dependency workflows. Further, based on the types of these formal context, the lattice construction algorithms are recommended. For example, the Godin algorithm [43] has a good performance with a sparse formal context. Note that we have used a clustering algorithm to be able to automatically classify the input workflows.

High Dependency Workflows: This category of workflows includes multiple dependencies between their tasks, since these tasks require for their execution the same datasets. These dependencies are reflected by intensive data transfer between tasks. This type of workflows generates a significant execution time and high-energy consumption.

Low Dependency Workflows: These workflows contain essentially a few number of dependent tasks (See Def. 3.9). Thus, the amount of data communication between these tasks is quite small. This type of workflows maximizes the parallelism of tasks execution which allows saving both the total execution time and energy consumption. For instance, we consider the Eq. 3.10 to identify whether the input workflows with high dependency or with low dependency.

$$\frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} dependency^T(t_i,t_j)}{n} \geq Limit_{dependency} \tag{3.10}$$

where n is the overall number of tasks. $dependency^T(t_i, t_j)$ is defined above (def. 3.9). $Limit_{dependency}$ is the dependency threshold between tasks. In fact, beyond this threshold, one can consider the workflow with a high dependence between tasks and below, the workflow is considered with a low dependence. Noteworthy that is the formal context represents the data dependency relationships of a input workflow.

The performance of Lattice generation algorithms depends essentially on the formal context's density. Thus, a K-means based algorithm [62] that allows clustering the input workflows based on their feature (datasets, tasks, tasks dependencies) is used. As described in [63], this algorithm allows us to identify the most convenient algorithms to apply for the lattice concepts generation. K-means aims to partition an input set of N points into K clusters by finding a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. The squared error metric and more details about the K-means algorithm could be found in [62]. The main steps of K-means algorithm are as follows:

1. Select an initial partition with K clusters; repeat steps 2 and 3 until cluster membership stabilizes,
2. Generate a new partition by assigning each pattern to its closest cluster center, and
3. Compute new cluster centers.

This algorithm requires three user-specified parameters: number of clusters K, cluster initialization, and distance metric. The number of clusters in this work is limited to two defining the type of the workflow dependency (high, low). The definition of initial clusters is accomplished using the Eq. 3.10 determining a first classification of the workflows. Then, we use the graph distance measure defined in [62] to evaluate the similarity degree between the DAG graphs of the input workflows.

Reducing Formal Context. From the set of Tasks T and the set of original datasets D^{in} extracted in step 3.3.1, we generate the Formal Context $FC = (T, D^{\text{in}}, I)$. I is the binary relationship between T and D^{in}, it is set to 1 if task t_i needs the dataset d_i^{in} for its execution, 0 otherwise. Further, the Formal Context indicates the dependency between tasks. Tow tasks t_i and t_j are dependent if only if they share one or more datasets. This step is new compared to our previous solution. Since the concept of Formal context is the central of FCA approach and its size has a significant influence on the structure of concept lattice as well as time and space complexity [60] when building the lattice, hence, it is important to reduce this formal context. In general, many techniques were proposed for this purpose. Thus, we have applied a simple way of reduction, which was suggested by [42]. Its principle consists of the junction of lines and/or columns. Indeed, if for two objects g and h are equal ($g = h$) then g and h can be replaced by one single object; dually, if for two attributes m and n are equal ($m = n$), then m and n can be replaced by one single attribute [42].

3.3.3. Choice of the lattice Construction Algorithm. According to [61], the lattice construction algorithms are recommended in terms of density/sparseness of underlying formal contexts as follows: (i)When the formal context is **small** and **sparse** as denoted in 3.2 **Godin** [43] algorithm is a good choice in this case, (ii)However, when contexts become **denser**, the algorithms such as **Norris** [44], **NextClosure** [45] and **Close by One** [46] should be applied, and (iii)Though, in case of *average density* context, **Bordat** [47] algorithm performs well.

3.3.4. Building of Galois-lattice. Our claim is to group the maximum of datasets and tasks together based on their dependencies. This group is called Formal Concept referred to FCA. Moreover, in order to represent all the formal concepts and their relationships, we build the Galois-lattice from the reduced context $FC = (T, D^{\text{in}}, I)$ and depending on the generation algorithm chosen in the above step. Then, we proceed to the reduction of the initial generated Galois-lattice in order to find the minimum number of concepts that have grouped a maximum datasets and tasks and eliminate the unnecessary and redundant concepts without loss of the information. In our solution, we eliminate all concepts having a single intention (or null) since the tasks belonging to these concepts require only one dataset to be processed. The recovery of this dataset is possible from other lattice concepts.

3.3.5. Choice of Candidate Concepts based on Concept Level. It refers to extract from the Galois-lattice, all the candidate concepts that, together, have a maximum weight and can cover the entire set of attributes (D^{in}). Nevertheless, we have proposed in this paper a new heuristic, which exploit the notion of level defined in 3.2 in order to avoid examining all of them. Actually, we will firstly organize all the concepts by level in the Lattice and then choose the concepts existed in the middle since they have the highest weight and cover all the data. After selecting concepts and before affecting them into appropriates SC, we have to verify if two candidate concepts, which are kept, have a common dataset. If it is the case, we will choose the placement of the common dataset in one storage computing server of the candidate concepts by comparing the quantity of transferred datasets in the two cases according to the algorithms1 and 2.

3.3.6. Mapping Datasets to Storage Computing Servers. In this paper we will take into account the granularity of the utilized resources in the data center and its different communication levels during the placement of IW data. Its basic is to assign the datasets of each concept candidate cc_i (See Eq. 3.6) and feasible (See Eq. 3.7) to the appropriate sc_i to prove that the difference between its storage capacity and the size of the concept cc_i is the minimum among every $sc_i \in SC$. If any concept among the candidate concepts isn't more feasible with what is left as free SC, we will proceed to assigning its sub-concepts denoted as $SubC(cc_i) = \{scp_1, scp_2, ..., scp_s\}$. We treat this sub-concepts in the following manner: we select the small size dataset as the first sub-concept to be placed later in the appropriate sc_i in order to avoid the transfer of larger datasets. Then, we put the rest of datasets in a second sub-concept. In this data assigning step, we proceed to the elimination of already assigned data from others sub-concepts in order to prevent their redundancy. The stop condition is that all the datasets of cc_i are placed ($cc_i.Int = \oslash$) and $\forall cc_i \in SubC(cc_i)$, we have $scp_i.Int = \oslash$. Finally, the result of this step is the list m of SC where the datasets are allocated.

3.3.7. Data Replication. The data replication technique is considered in our approach as a solution to ensure the minimization of the average cost of all tasks. This technique was detailed in our previous work [15]. Briefly, it is based on two steps which are: i) Identification of data to replicate based on the score of each dataset $Score(d_i^{\text{in}})$ (Eq. 3.8), and ii) Replication of important datasets.

3.4. E-DPSIW-FCA algorithm. The E-DPSIW-FCA algorithm is outlined in Algorithm 4. The algorithm 4 describes the proposed strategy. In the first step, we start by classify the input workflows by applying the K-means algorithm (KmeansCluster). Then, we generate the formal context $FContext$ from the workflow tasks and datasets using *generateFormalCxt* function. We apply the algorithm suggested in [42] for the reduction of the generated Formal $FContext$ using *reduceFCxtbyGanter(FContext)* to have our $ReducedFCxt$. We generate the Lattice Galois from $ReducedFCxt$ by applying the appropriate classification algorithm as described above by the use of *generateLattice* function. This generated lattice, in turn, will be simplified using *simplifyLattice* function. We retain then the candidate concepts from this simplified lattice by the use of *selectCpCand* function. In the next step, for each candidate concept, we verify its feasibility using *feasible* function. If the concept is feasible, we first apply the function *findAppropriateServer* to find the most appropriate storage computing server where we store this datasets concept using the function *affecte*. Then, we add this server to the best servers list. If the concept isn't feasible, we find its sub-concepts using *findSubCpts* function and then we update the list of candidate concepts with the function *update(CandidateConcepts, SubConcepts* to perfom the same treatment again. In the last step, for each storage computing server $sc_i \in Placement$, we verify if its datasets need to be replicated using the function *score*. If it is the case, the replication is done by the use of *replicate* function. $WF.D^{\text{in}}$ stands for the data D^{in} of the input workflows WF.

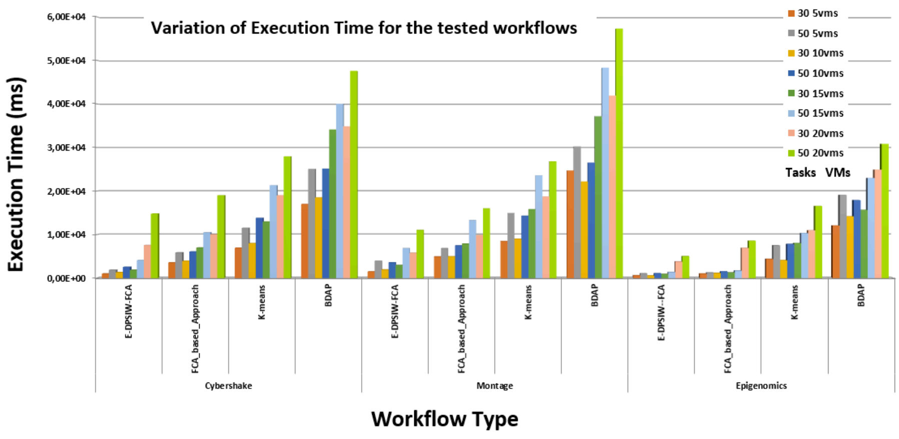
4. Experiment Simulation. In our experiments, we run a set of scientific workflows as the sample tests from [48] notably earthquake science (*CyberShake*) [49], astronomy (*Montage*) [50] and biological genetics (*Epigenomics*) [51]. They were chosen because they represent a wide range of application domains and they have distinct structures and differ greatly in the number and size of datasets. For example, Montage is I/O intensive, CyberShake is memory intensive, and Epigenomics is CPU intensive. The Montage project is an astronomy application that delivers science-grade mosaics of the sky [50]. The CyberShake is used to calculate Probabilistic Seismic Hazard curves for several geographic sites in the Southern California area [52]. Epigenomics maps short DNA segments collected with high-throughput gene sequencing machines to a previously structured reference genome [51]. We have used the execution traces of the tested workflows in [48], which are stored in

Algorithm 4 E-DPSIW-FCA Algorithm

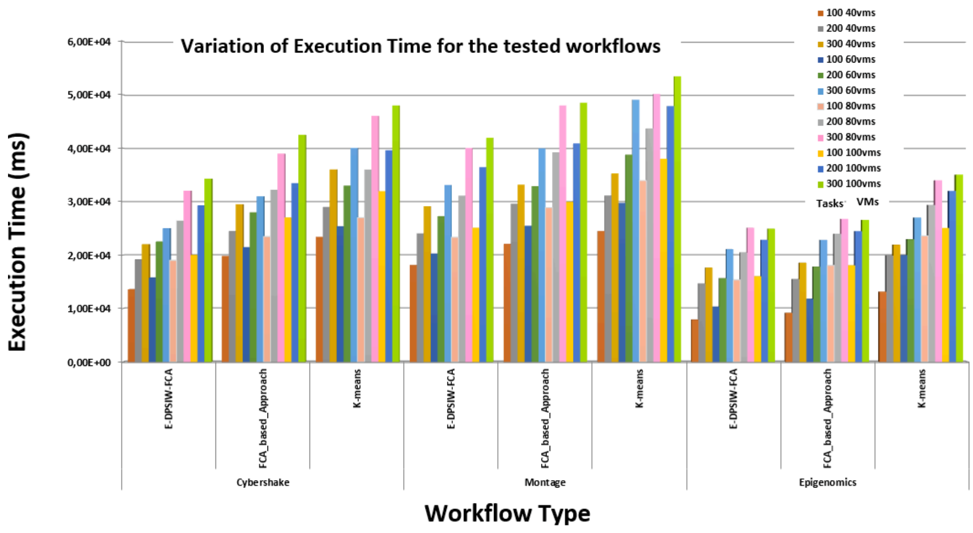
```
Input: $WF$                                                  ▷ The input Workflow
    $setofServers = sc_1, sc_2, ..., sc_n$                   ▷ List of $SC$
    $CandidateConcepts = \oslash$                            ▷ List of candidate concepts
    $SubConcepts = \oslash$                                  ▷ List of sub-concepts
    $FContext = \oslash$                                     ▷ Formal context to generate
    $ReducedFCxt = \oslash$                                  ▷ Reduced Formal context
    $lattice = \oslash$                                      ▷ Galois-lattice to build
    $simplifiedLattice = \oslash$                            ▷ Galois-lattice simplified
Output: Var $Placement$                                      ▷ List of couple $< sc_i, d_j >$
1: int $capacity = 0$;
2: $intworkflow - type = KmeansCluster(WF)$;
3: $FContext = generateFormalCxt(intworkflow - type.T, intworkflow - type.D^{\text{in}})$;
4: $ReducedFCxt = reduceFCxtbyGanter(FContext)$;
5: $lattice = generateLattice(ReducedFCxt)$;
6: $simplifiedLattice = simplifyLattice(Lattice)$;
7: $CandidateConcepts = selectCpCand(simplifiedLattice)$;
8: for $candidate \in CandidateConcepts$ do
9:     if $feasible(candidate)$ then
10:        $sc_j = findAppropriateServer(sc_i, setofServers)$;
11:        $capacity = Cap(sc_j)$
12:        $capacity = (min(Cap(sc_i), size(candidate))$          ▷ $\forall i = 1..n.$
13:        $affecte(candidate, sc_j)$
14:        $add(sc_j, Placement)$
15:        $sc_j.Cap = sc_j.Cap - Size(candidate)$
16:    else
17:        $SubConcepts = findSubCpts(candidate)$
18:        $update(CandidateConcepts, SubConcepts)$;
19:        for $sc_i \in Placement$ do
20:            for $d_i \in sc_i.WF.D^{\text{in}}$ do
21:                if $score(d_i^{\text{in}})$ then
22:                    $replicate(d_i^{\text{in}})$
23:                end if
24:            end for
25:        end for
26:    end if
27: end for
```

XML-formatted files while providing information about these workflows, such as datasets size and dependencies, tasks flow, etc.

4.1. Simulation Setting. Our simulation environment was developed using CloudSimPlus toolkit an extension of CloudSim [53], which provides a platform of Cloud Computing infrastructures. The same workflows and CC environments are simulated on other contrast experiments described in the Sect. 5 which are the BDAP in [14] and the K-means strategy [4]. According to [14], the parameters of BDAP approach based GA are set as follows: the population size is 20, the maximum number of iterations is 20, the crossover probability is 0.8, the mutation probability is 0.5. In our experiments, we assume that the weight values that indicate the degree of importance of the quality metrics are equals. Our solution E-DPSIW-FCA, K-means and BDAP were implemented and evaluated for the three test workflows according to the following criteria: 1)Execution time of workflow tasks, 2)Workflow Communication energy cost that is defined in Sect. 2.11, 3)Total data movement defined in Sect.2.9, and 4)Workflow Computing energy which is already defined in Sect. 2.15. We did our



(a) 30 and 50 tasks.

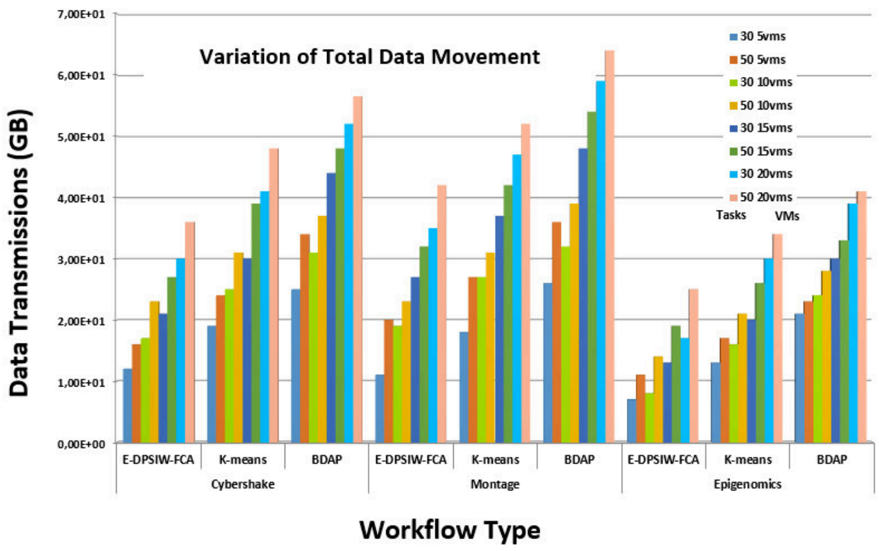


(b) 100, 200 and 300 tasks.

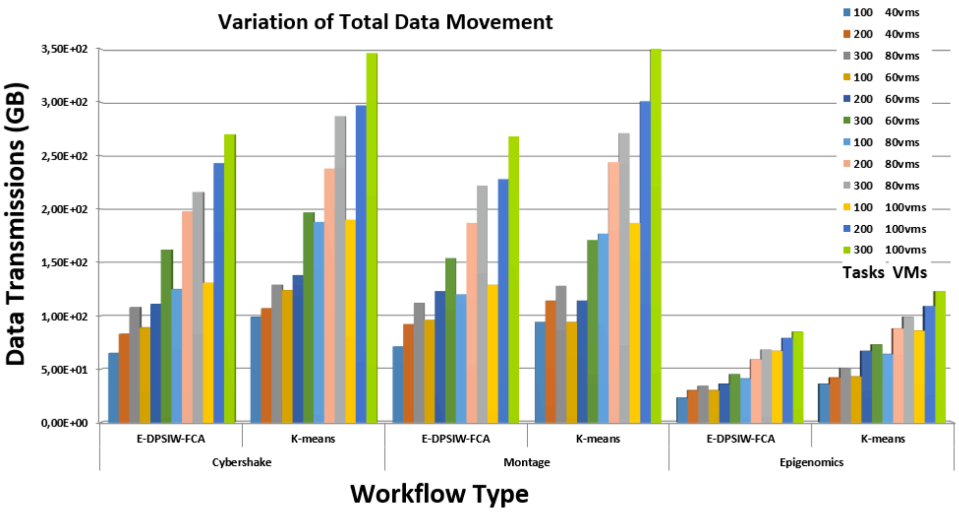
Fig. 4.1. *Total execution time by varying the number of Tasks.*

experiments for two different scenarios: firstly with varying the number of tasks and secondly with varying the number of VMs aiming to observe the effect of these variation on our performance indicators.

4.1.1. Scenario 1: Varying the number of Tasks. In this scenario, we have considered firstly four size number of VMs: 5, 10, 15 and 20 for the E-DPSIW-FCA, BDAP and K-means approaches. In this test, we select for each tested workflow a number of tasks in the range [30, 50]. Then, we test for E-DPSDIW-FCA and K-means, since the processing time of the genetic algorithm (BDAP) approach is higher and it trends to take longer to converge upon a solution. The number of VMs is in the range [40, 60, 80, 100, 120, 140, 160, 180, 200] with number of tasks set at 100, 200, 300 tasks. Note that each task can be executed on any virtual machine. With reference to Fig. 4.2, it can be seen that our solution has managed to reduce the total data movement between VMs compared to BDAP and K-means algorithms. This reduction is more valuable for the memory intensive workflows (Cybershake) and less remarkable in the case of CPU intensive workflows (Epigenomics) since its amount of data communication is quite small. Figure 4.3 confirms the ability of our strategy to reduce heavily the energy consumed during the communication between SC. Comparing the experiment results of K-means, BDAP and our strategy, in Fig. 4.4, in terms of Computing Energy Consumption, it would be found that the performance of our approach is the better.



(a) 30 and 50 tasks.

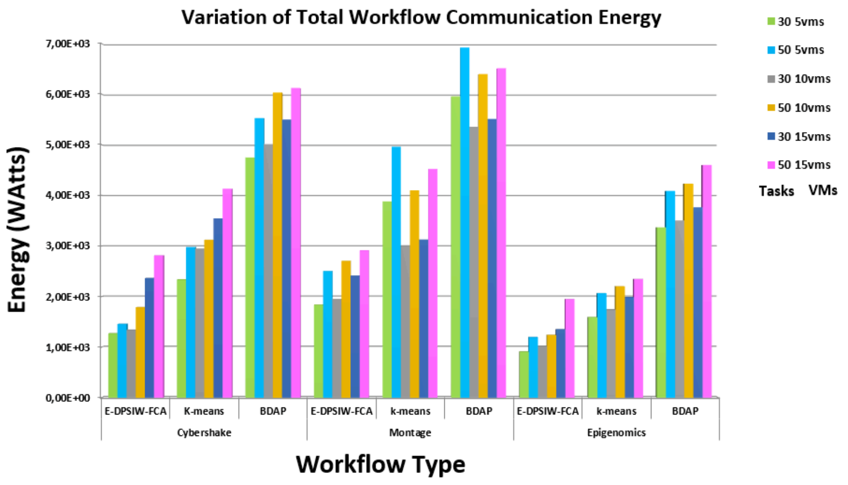


(b) 100, 200 and 300 tasks.

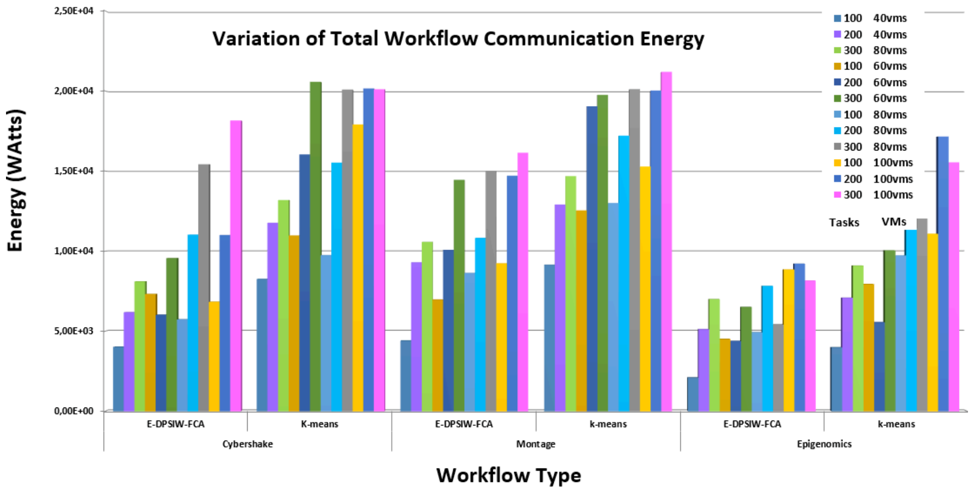
FIG. 4.2. *Total Data movement by varying the number of Tasks.*

4.1.2. Scenario 2: Varying the number of VMs. In this scenario, we did our experiments for the E-DPSIW-FCA and K-means with fixing the number of tasks to 300 and the number of VMs was set to [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]. It is worth noting that BDAP trends to take longer time to converge upon a solution and it blocks at 100 tasks, so it was excluded. Figure. 4.5 reveals well that as the number of VMs increases, FCA-based-Approach, K-means, and BDAP strategies' respective execution time prove to record a noticeable increase as compared to our approach. Further, it is clearly seen that our strategy reduces heavily the execution time for Epigenomics better than Cybershake, since Epigenomics maximizes the parallelism of tasks while saving time. Additionally, our approach performs well the execution time compared to our previous work FCA-based-Approach, especially in the case of Cybershake rather than Epigenomics.

Figure 4.6 exhibits the total data movement' tendencies recorded. In fact, our approach reduces the amount of data movement compared to other strategies. This result has its justification, since our approach avoids the movement of larger data and it provides multiple data placement choices, thereby, decreasing the amount of



(a) 30 and 50 tasks.



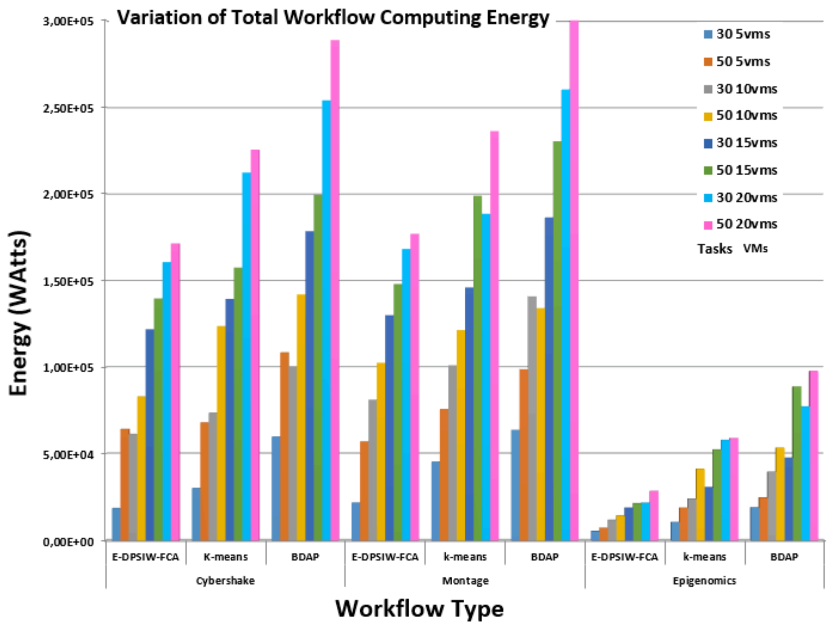
(b) 100, 200 and 300 tasks.

FIG. 4.3. *Total Workflow Communication energy by varying the number of Tasks.*

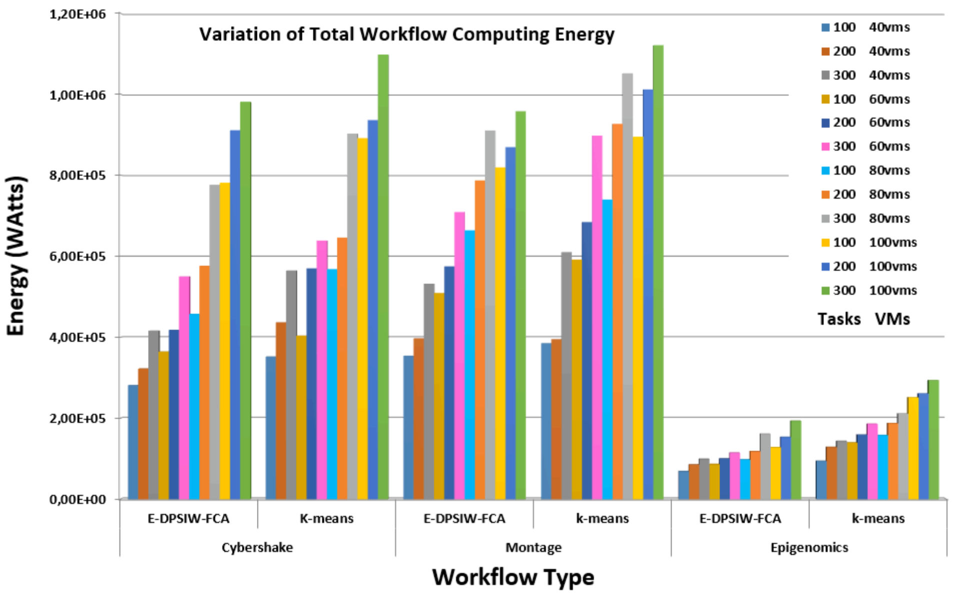
data transfer consequently. However, K-means considers data dependencies too much, which leads to larger data movement. Figure 4.7 indicates the additional energy consumption values incurred from the communication between VMs. As data movement between VMs leads to a communication energy consumption, based on this, it can be seen that our approach reduces the communication energy noticeably to BDAP and K-means. Worth citing that data movement reduction and the communication energy consumption decrease is more valuable for the memory intensive workflows (Cybershake and Montage) and less remarkable in the case of CPU intensive workflows(Epigenomics), since its amount of data communication is quite small. Similarly, as shown in Fig. 4.8, our approach reduces the computing consumption compared to other strategies.

5. Related Works. Being a critical challenging issue, data placement of IW has always attracted the attention of researchers. Several data placement strategies have been proposed in this issue and tend to be categorized under data dependency and graph-based methods.

Data dependency approaches have attempted to solve the data placement problem via application of the heuristic algorithms such as the Genetic Algorithm (GA), the Practical Swarm Optimization technique (PSO), the Ant Colony Optimization (ACO) and the hierarchical partition algorithms (based on Bond Energy Algorithm (BEA)) to group data based on their dependencies in a bid to reduce the total data movements



(a) 30 and 50 tasks.



(b) 100, 200 and 300 tasks.

Fig. 4.4. *Total Workflow Computing energy by varying the number of Tasks.*

throughout the workflow execution processes, such as [14] [17] [20], etc.

Graph-based approaches used, the data placement problem by modeling as a hypergraph or graph such as in [21], [22], [23] and [31]. Furthermore, we have identified a set of criteria in order to perform a comparison study (Table 5.1) of the data placement strategies above-mentioned which are:

(i) *objective* - which optimization criteria are exploited,
(ii) *technique used*- which concept is used to indicate the type of algorithm or method employed,
(iii) *modelization*- how authors model the data placement problem (graph, matrix),

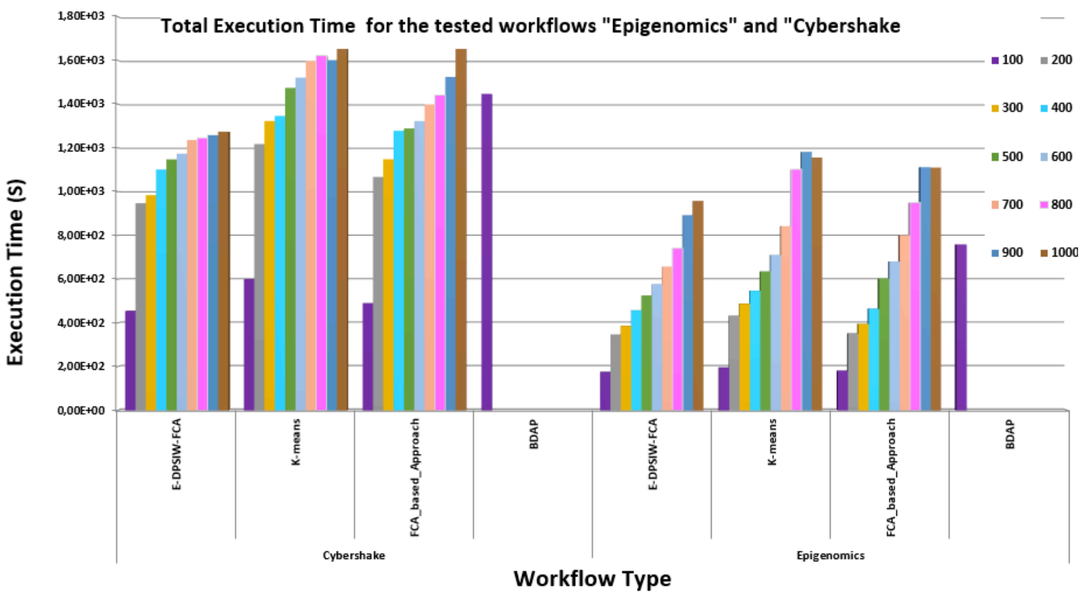


Fig. 4.5. *Total Execution Time by varying the number of VMs.*

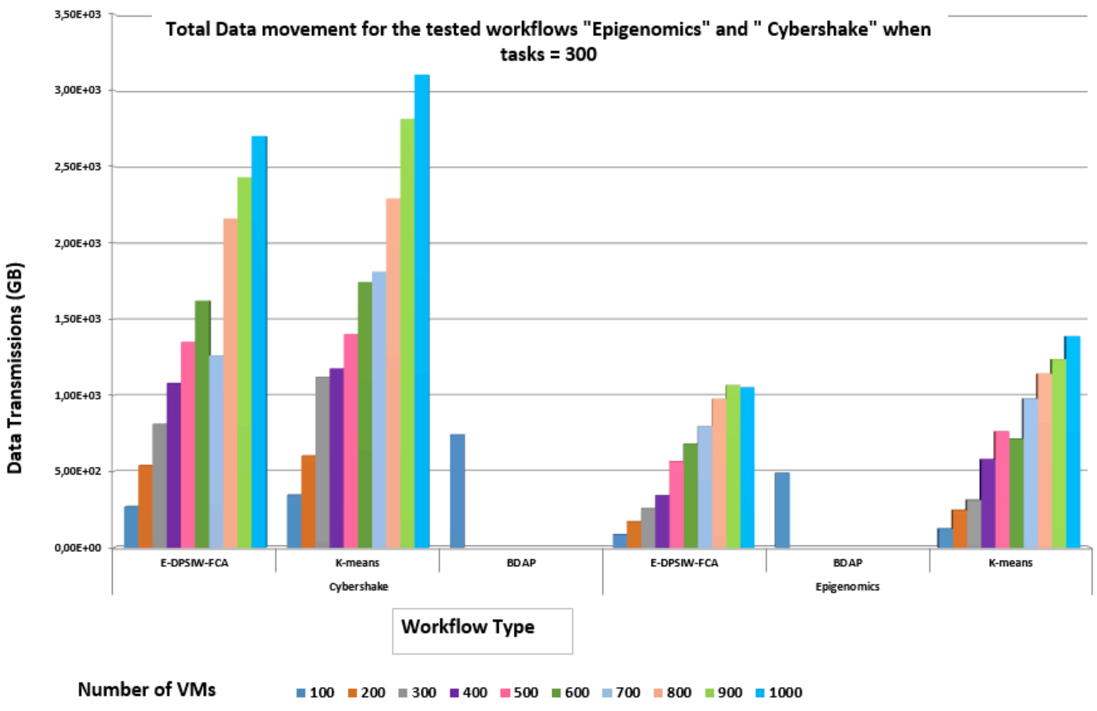


Fig. 4.6. *Total Data movement by varying the number of VMs.*

(iv) *type of datasets*- datatsets can be original (Inp), intermediate data Int), produced by running workflow, or fixed data Fix, stored in specific data centers according to their size or their management needs, flexible data $Flex$, that the system can flexibly decide where to store them,

(v) *data center infrastructure*- what is the granularity of the resources considered (virtual machines, servers, etc.),

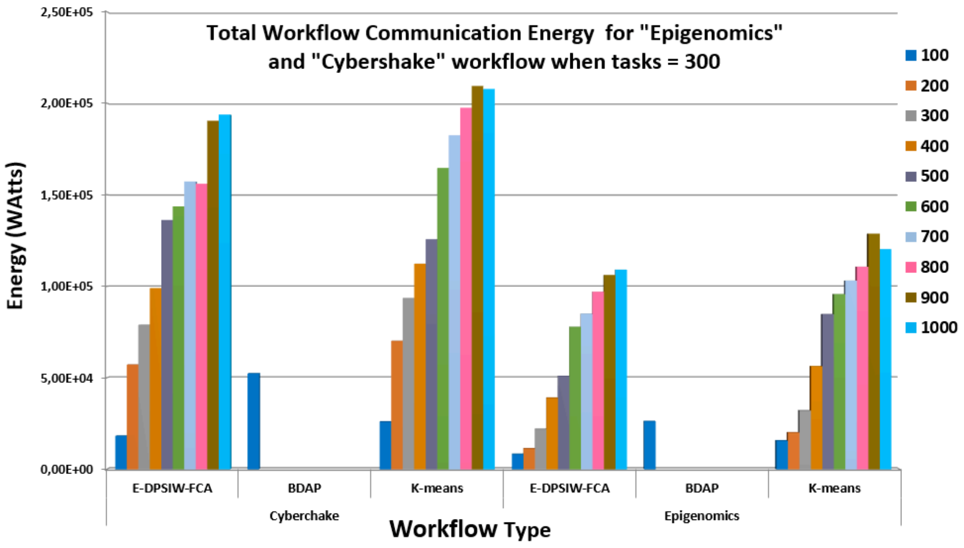


FIG. 4.7. *Total Workflow Communication energy for "Cybershake" and "Epigenomics" workflows by varying the number of VMs.*

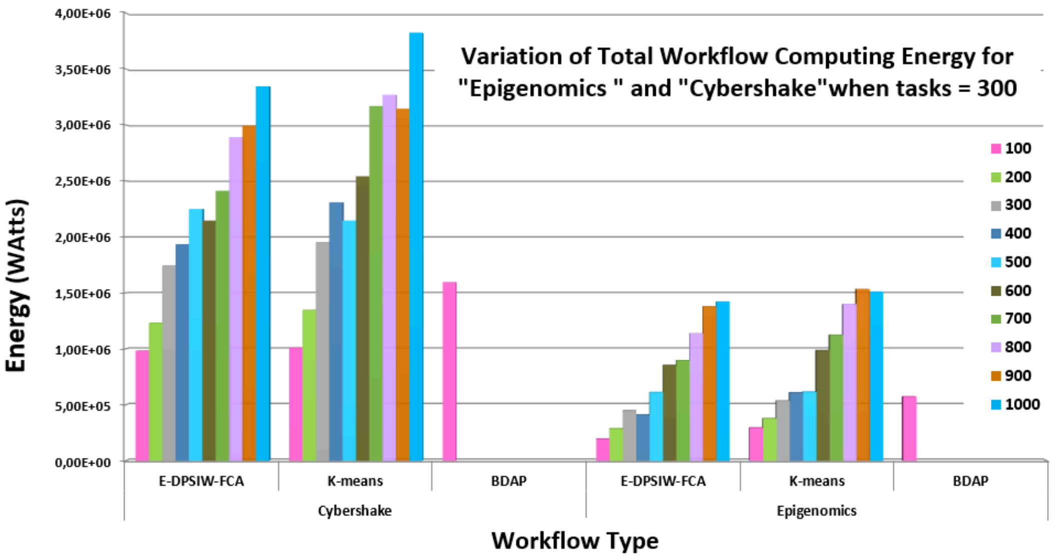


FIG. 4.8. *Workflow Computing energy for "Cybershake" and "Epigenomics" workflows by varying the number of VMs.*

(vi) *network infrastructure*- which network devices are supported (switches, routers, etc.).

From Table 5.1, we can make the following observations:

(i) Most approaches (72.72%) focus mainly in reducing data movement. Nevertheless, non-one of the above-mentioned approaches do consider the energy consumption of network devices and communication links during the workflow execution.

(ii) The vast majority of the presented works did not provide information about the network infrastructure (68.18%). However, only (4.54%) operates at the lowest level of network data center and consider its communication levels during data placement.

(iii) 22.72% of works did not provide information about data center infrastructure. Contrarily, few approaches (13.63%) ([14], [26] and [27]) considered the granularity of the data center resources, and solve the

Table 5.1
Comparative Table of Existing Data Placement Strategies

Approach	Objective	Technique Applied	Modelization	Datasets Type	Data Center Infrastructure	Network Infrastructure
[14]	data movement communication cost	GA	Matrix	Inp, Int Fixd	VMs	-
[16]	number of data transfers	GA	Matrix	Inp, Int	Data center	-
[17]	execution time data transfer time	GA	Matrix	Inp	Data center	Bandwidth
[18]	data transmissions data transfer time	GA	Graph	Inp, Int	Data center	Bandwidth
[19]	data transfer number	Hierarchical partionning Clustering Algorithm + PSO	Matrix	Inp, Fix, Flex	Data center	-
[20]	amount of transferred data	PSO + BEA	Matrix	Inp, Fix, Int	Data center	-
[21]	data movements number	PSO based on GA	Graph	Inp, Int	Data center	-
[22]	data transfer cost	Discrete PSO (DPSO)	Graph	Inp, Int Fix, Flex	Data center	Bandwidth
[23]	transmission time	PSO + GA	Graph	Inp, Int	Data center	Bandwidth
[24]	data security data transfer time	ACO	-	Inp, Int, Fix, Flex	Data center	-
[4]	data movement	K-means + Recursive partitioning	Matrix	Inp, Int Fix, Flex	Data center	-
[25]	total data scheduling	-	Matrix	Inp, Int	Data center	-
[26]	data movement	BEA	Matrix	Inp	VMs	-
[27]	data movement time consumption	heuristic tree-to-tree	-	Inp, Fix	Servers	Bandwidth
[27]	overall data access cost	subgradient optimization heuristics algorithm	-	Inp	Data center	Bandwidth
[29]	execution time, data movement	BEA	Matrix	Inp	Data center	-
[30]	Data transfer time	heuristic GA	Matrix	Inp	Data center	-
[31]	average query span	HPA	Hypergraph	-	Data center	-
[32]	Total amount of file transfers	HPA	Hypergraph	Inp	Data center	-
[33]	data movement	Data Correlation + BEA	Matrix	Inp	Data center	-
[3]	average query spans Execution time	FCA	Formal Context	Inp	Data center	-

data placement problem at the VM level.

Overall, to remedy these shortcomings cited above, we propose in this scope an extension of our previous work [12] that makes allowances for:

(i) The placement of datasets into SC while considering the energy consumption in computing, storage and communication devices.
(ii) Taking into account the architecture and the characteristics of the physical network interconnecting SC.
(iii) The communication between SC, while examining all the communication levels of data center architecture.

6. Conclusion. In this paper, a data placement strategy based on the FCA approach (E-DPSIW-FCA) is proposed to operate within the context of data intensive workflows. E-DPSIW-FCA minimized the total workflow communication and computing energy consumption and the total data movement between SC during the execution of the workflow tasks. Our experiments based on three types of scientific workflows showed that our strategy outperformed other strategies on reducing greatly the data movement and both computing and communication energy consumption as well as the data transfer cost. In our solution, we have considered the data placement for executing a single workflow. However, in the real world, multiple workflows can be executed concurrently. Thus, as future work, we plan to extend our strategy to consider the data placement of multiple workflows simultaneously, which is recognized today as an important challenging issue.

REFERENCES

[1] E. Deelman and A. Chervenak, Data Management Challenges of Data-Intensive Scientific Workflows. *Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 687-692, 2008.

[2] Sh. Zhang, Ch. Zhu, J. K. O. Sin and P. K. T. Mok, A Novel Ultrathin Elevated Channel Low-temperature PolySi TFT. *IEEE Electron Device Letters*, **20**, vol. 20, no. 11, pp. 569-571, 1999.

[3] K. Bousselmi, Z. Brahmi and M.M. Gammoudi, QoS-aware scheduling of Workflows in Cloud Computing environments. *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 737-745, 2016.

[4] D. Yuan, Y. Yang, X. Liu and J. Chen, A Data Placement Strategy in Scientific Cloud Workflows, *Future Generation Computer Systems*, **26**, 1200-1214.

[5] https://scec.usc.edu/CyberShake/. [Jan. 14, 2019].

[6] https://pegasus.isi.edu/. [Fev. 11, 2019].

[7] https://cloud.netapp.com/blog/ebs-efs-amazons3-best-cloud-storage-system/. [Sep. 27, 2018].

[8] https://aws.amazon.com/ebs/. [Oct. 15, 2018].

[9] https://www.ibm.com/cloud-computing/bluemix/fr/block-storage/. [Oct. 24, 2018].

[10] J. M. Pierson, Large-scale Distributed Systems and Energy Efficiency. A Holistic View. *John Wiley and Sons*, 2015.

[11] T. Li, W. Yang and A. Y. Zomaya, An energy-efficient virtual machine placement and route scheduling scheme in data center networks. *Future Generation Computer Systems*, **77**, 1-11, 2017.

[12] N. Cordeschi, M. Shojafar and E. Baccarelli, Energy-saving self-configuring networked data centers. *Computer Networks*, **57**, 3479-3491, 2013.

[13] Q. Li, K. Wang, S. Wei, L. Xu and M. Gao, A data placement strategy based on clustering and consistent hashing algorithm in Cloud Computing. *9th International Conference on Communications and Networking*, pp. 478-483, 2014.

[14] M. Ebrahimi, A. Mohan, A. Kashlev and S. Lu, BDAP. A Big Data Placement Strategy for Cloud-Based Scientific Workflows. *IEEE First International Conference on Big Data Computing Service and Applications*, pp. 105-114, 2015.

[15] Z. Brahmi, S. Mili and R. Derouiche, Data Placement Strategy for Massive Data Applications base on FCA Approach. *IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1-8, 2016.

[16] Q. Xu, Z. Xu and T. Wang, A Data-Placement Strategy Based on Genetic Algorithm in Cloud Computing. *International Journal of Intelligence Science*, **05**, 145-157, 2015.

[17] J. Taheri and A.Y. Zomaya, Genetic algorithm in finding Pareto frontier of optimizing data transfer versus job execution in grids. *IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum*, pp. 2130-2139, 2012.

[18] L. Cui, J. Zhang, L. Yue, Y. Shi and al. A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds. *IEEE Transactions on Services Computing*, **11**, 727-739, 2018.

[19] Z. Er-Dun, Q. Yong-Qiang, X. Xing-Xing and C. Yi, A Data Placement Strategy Based on Genetic Algorithm for Scientific Workflows. *Eighth International Conference on Computational Intelligence and Security*, pp. 146-149, 2012.

[20] Q. Zhao, C. Xiong, K. Zhang, Y. Yue and J. Yang, A Data Placement Algorithm for Data Intensive Application in Cloud. *International Journal of Grid and Distributed Computing*, **9**, 145-156, 2016.

[21] Z. Xiang, T. Liu, B. Lin and al. A Data Placement Strategy for Scientific Workflow in Hybrid Cloud. *IEEE 11th International Conference on Cloud Computing*, pp. 556-563, 2018.

[22] X. Li, L. Zhang, Y. Wu, and al. A Novel Workow-Level Data Placement Strategy for Data-Sharing Scientic Cloud Workows. *IEEE Transactions on Services Computing*, 1-1, 2016.

[23] B. Lin, F. Zhu, J. Chen, X. Chen, N. N. Xiong and L. J. Mauri, A Time-driven Data Placement Strategy for a Scientific Workflow Combining Edge Computing and Cloud Computing. *CoRR*, 2019.

[24] W. Lei, S. Peng, W. Du, W. Wang, and G. S. Zeng, Security-aware intermediate data placement strategy in scientific cloud workflows. *Knowledge and Information Systems*, **41**, 423-447, 2014.

[25] T. Wang, S. Yao, Z. Xu and S. Jia, DCCP: an effective data placement strategy for data-intensive computations in distributed cloud computing systems. *Journal of Supercomputing*, **72**, 2537-2564, 2016.

[26] L. Liu, J. Song, H. Wang, and P. Lv, BRPS: A Big Data Placement Strategy for Data Intensive Applications. *IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pp.813-820, 2016.

[27] Q. Zhao, C. Xiong, and P. Wang, Heuristic Data Placement for Data Intensive Applications in Heterogeneous Cloud. *Journal of Electrical and Computer Engineering*, **2016**, 8, 2016.

[28] J. Zhang, J. Chen, J. Luo, and A. Song , Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. *Tsinghua Science and Technology*, **21**, 47181, 2016.

[29] H. Kim, Y. and Kim, An adaptive data placement strategy in scientific workflows over cloud computing environments. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1-5, 2018.

[30] X. Qiang, Xu. Zhengquan and W. Tao A Data-Placement Strategy Based on Genetic Algorithm in Cloud Computing. *International Journal of Intelligence Science*. **05**, 145-157, 2015.

[31] A. K. Kayyoor, A. Deshpande and S. Khuller, Data placement and replica selection for improving co-location in distributed environments. 1302-4168, 2013.

[32] mit. V. çatalyürek, K. Kaya and B. Uçar, Integrated data placement and task assignment for scientific workflows in clouds. *In: Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing*, ACM, 2011.

[33] Q. Zhao, C. Xiong, X. Zhao, C. Yu and J. Xiao, A Data Placement Strategy for Data-Intensive Scientific Workflows in Cloud. *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 928-934, 2015.

[34] https://aws.amazon.com/s3/fags/?nc1=h-ls/. [Nov. 19, 2018].

[35] https://www.cisco.com/. [Nov. 25, 2018]
[36] X. Meng, V. Pappas and L. Zhang, Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. *2010 Proceedings IEEE INFOCOM*, pp. 1-9, 2010.
[37] R.L. Graham, D.E. Knuth and O. Patashnik, Integer Functions, Ch. 3 in *Concrete Mathematics: A Foundation for Computer Science*, MA: Addison-Wesley, 1994.
[38] F. P. Tso, K. Oikonomou, E. Kavvadia and al. , S-CORE: Scalable Communication Cost Reduction in Data Center Environments. *School of Computing Science*, University of Glasgow, 2013.
[39] G. Stumme, R. Taouil, Y. Bastide and al., Fast computation of concept lattices using data mining techniques. *In Proceedings, 7th Int. Workshop on Knowledge Representation Meets Databases (KRDB 2000)*, 2000.
[40] L. Lakhal and G. Stumme , Efficient mining of association rules based on formal concept analysis. *Formal Concept Analysis Foundations and Applications*, 2005.
[41] C. Carpineto and C. Romano, A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, **24**, 95122, 1996.
[42] B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations. 1999.
[43] R. Godin, R. Missaoui and H. Alaoui, Incremental concept formation algorithms based on Galois lattices. *Computation Intelligence*, **11**, 246267, 1995.
[44] E. M. Norris, An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathmatiques Pures et Appliques*, **23**, 243250, 1978.
[45] G.Bernhard and W. Rudolf, Formal concept analysis. *Mathematical foundations*, 2012.
[46] S. O. Kuznetsov, A fast algorithm for computing all intersections of objects in a nite semilattice. *Automatic Documentation and Mathematical Linguistics*, **27**, 1121, 1993.
[47] J. P. Bordat, Calcul pratique du treillis de Galois dune correspondance. *Math. Sci. Hum.*, **96**, 31 47, 1986.
[48] Pegasus workow gallery: https://pegasus.isi.edu/ workow gallery/. [Fev. 21, 2019]
[49] https://scec.usc.edu/scecpedia/ CyberShake/. [Fev. 21, 2019]
[50] http://montage.ipac.caltech.edu/. [Fev. 21, 2019]
[51] http://epigenome.usc.edu/. [Fev. 21, 2019]
[52] L. Nourine, and O. Raynaud, Fast algorithm for building Lattices. *Information Processing Letters*, **71**, 199-204, 1999.
[53] https://github.com/manoelcampos/cloudsim-plus. [Fev. 11, 2018]
[54] I. T. Cotes-Ruiz, R. P. Prado, S. García-Galán, J. E. Muñoz-Expósito and N. Ruiz-Reyes, Dynamic Voltage Frequency Scaling Simulator for Real Workflows Energy-Aware Management. *in Green Cloud Computing*, 2017.
[55] M. Shojafar, C. Canali and R. Lancellott, Adaptive Computing-plus-Communication Optimization Framework for Multimedia Processing in Cloud Systems. *IEEE Transactions on Cloud Computing*, 2016.
[56] C. Fiandrino, D. Kliazovich, P. Bouvry and al., Performance Metrics for Data Center Communication Systems. *2015 IEEE 8th International Conference on Cloud Computing*, 2015.
[57] T. Guérout, T. Monteil, G. D. Costa and al., Energy-aware simulation with DVFS. 2013.
[58] https://azure.microsoft.com/en-us/pricing/details/bandwidth/. [Mar. 15, 2019]
[59] https://searchaws.techtarget.com/definition/availability-zones. [Apr. 3, 2019]
[60] M. Dias. Sérgio and José. Vieira Newton, Reducing the Size of Concept Lattices: The JBOS Approach. 2010.
[61] S. Kuznetsov and S. Obiedkov, Comparing performance of algorithms for generating concept lattices. *J. Exp. Theor. Artif. Intell*, **14**, 189-216, 2002.
[62] AK. Jain, Data clustering: 50 years beyond K-means. *Pattern recognition letters*, **31(8)**, 651-666, 2010.
[63] K. Bousselmi, Z. Brahmi and M. M. Gammoudi, Energy Efficient Partitioning and Scheduling Approach for Scientific Workflows in the Cloud.*IEEE International Conference on Services Computing (SCC)*, 2016.

Edited by: Dana Petcu
Received: April 22, 2019
Accepted: June 18, 2019