Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol.4, No.12, 2014

Presentation an Approach for Placement Phase in Mapping Algorithm

Rohollah Esmaeli Manesh[1] Ahmad Zareie[2]
1.Payam_E_ Noor University,Iran
2.Department of Engineering,Songhor Branch, Islamic Azad University, songhor, Kermanshah, Iran

Abstract
The data requirements of both scientific and commercial applications have been increasing drastically in recent years. Just a couple of years ago, the data requirements for an average scientific application were measured in terabytes, whereas today we use petabytes to measure them. Moreover, these data requirements continue to increase rapidly every year, and in less than a decade they are expected to reach the exabyte (1 million terabytes) scale.. In this work, the data duplication technique has not been used by us. That's because of increase in costs and expenses of using a cloud system.In this paper, an approach to mapping workflow tasks and data between cloud system data centers has been presented. This approach encompasses 2 phases: both of which both have been given enough input to appropriately map tasks and data between data centers in such a way that the total time for task execution and data movement becomes minimal. In other words, the goal of mentioned approach is to present a trade-off between these two Goals. Simulations have demonstrated that the said approach can fulfill stated goals effectively.
Keywords: Distributed system, scientific application, application, data requirements

1. Introduction

A large amount of data should be analyzed by researchers in many scientific research fields such as astronomy, physics, geology, etc. In addition, a large amount of data is produced as median results during execution of process similarly. Work flow technology helps execute such scientific applications. It should be mentioned that work flow is extremely complicated and bears many tasks [2] that each requires an assemblage of input and median data to be done. On one hand, executing tasks in a series takes a lot of time, and on the other, because of high amounts of data in these types of applications, they are executed in parallel form on a distributed system. In this condition, all necessary data has to be collected in a system for the start of each task. Hence, execution of work flow application bears some data movement.

Cost of communication is a considerable concern in distributed systems, especially in scientific and commercial workflow applications that require a large amount of data to be stored and transferred in distributed data centers [1].

According to the above-mentioned facts, if scientific workflow is mapped in a distributed system, tasks and data distribution manner affect system efficiency largely.

Globally distributed systems such as cluster computing systems, grid systems, utility computing and cloud computing have been widely extended. Cloud computing is a wide- scale kind of parallel distributed system with virtualization properties, dynamic scalability, and high performance in computing and storage in which services are delivered according to external users' needs. With the advancement of modern society, efforts have been made to generally present computing services such as other basic services (water, gas, telephone) in order to let anyone have access to them easily [3]. In fact, cloud computing systems provide fast access to large numbers of complicated and distributed computers throughout the world via the internet. Users can have access to their data and applications at any time and at any place in different economic, commercial and scientific fields and obviate or decrease costs related to storage and computation in their private sites. A Could computing system is comprised of some geographically distributed data centers where each includes some computing and storage servers and other physical devices for dynamic provision based upon needs [4]. Considering to strength of scalability and high processing, a cloud computing system is an appropriate environment for execution of scientific workflow application. It should be noted that scientific workflow application can widely be used by benefiting from advantages of cloud computing. Nevertheless, this work has some challenges to meet. They are as follows:

1. Memory shortage problems in data centers
2. Movement of large amounts of data and hurtling between data center communication path
3. The need for task distribution mechanisms for the purpose of decreasing total time of workflow
4. The need for permanent storage of output results of workflow in data centers

We have tried to provide an approach for decreasing data movement and total completion time of work flow by using critical path techniques. The fulfillment of these two aims optimally contradicts each other. Therefore, we have attempted to present an algorithm that fulfills a tradeoff between these two aims.

2. Related work

Scheduling of tasks is an important challenge in distributed systems and according to this issue, different approaches have been presented. In 2007, Kamer presented an approach to scheduling independent tasks in a distributed system in which the only relationship between tasks is the need to access common files and there is no dependence between tasks [6]. In [10], a genetic algorithm has been presented for decreasing time of execution of one task graph and load balancing in systems. In [11], an approach to scheduling tasks has been stated for the purpose of fulfillment of task completion deadlines and optimization of execution costs. For scheduling tasks of workflow, approach [12] has been suggested to decrease length of schedule and promoting reliability of application execution in heterogeneously distributed environment. In [13], a method for scheduling tasks of workflow in processors with different capabilities has been provided. Of other mapping approaches, [14, 15] it can be pointed out that allocation of independent tasks in a heterogeneously distributed environment is performed to boost reliability. Different placement strategies have been presented for data placement in distributed systems. In [7], a schedule has been presented for a grid system in which data placement activities can be queued, scheduled, and managed. In work [8], an approach for data placement has been presented in a cloud environment in which the only proof for selecting place of tasks was centered in which most of the required data is in it and length of schedule has been ignored. Also, in [17, 18] strategies for data mapping in distributed environments have been presented. Kosar [9] in his study presented a method for data placement in widely distributed systems. We intend to present an approach for Mapping Scientific Work Flow in Cloud Computing Environment to Minimize Data Movement and Schedule Length of Workflow.

3. Problem Analysis

Workflow application is a sample of graphs of precedence of task execution that contains a set of input data as well as tasks. Such an application contains many tasks and it works with a large amount of data. A Hybrid Directed Acyclic Graph is used to indicate task priorities and the amount of data that is to be transferred between them. Workflow graph has been shown as follows:

$$G = \{T, D, R, I\}$$

In which:

T=$\{t_1, t_2, t_3, \dots, t_n\}$ and its tasks (activities) collection is shown in the form of square node and related task execution time is mapped in it and shown by w_i.

D is the representative of input data collection and shown as following:

D =$\{d_1, d_2, d_3, \dots, d_m\}$

This component is shown by circle node in graph. Weight of each Data node is data volume, that the d_a node weight is shown by s_a.

R= $\{(t_i, t_j)\}$ is representative of a set of edges which shows a set of dependence and priorities between tasks. $\{(t_i, t_j)\}$ edge means that a datum produced by t_i task is required during execution of t_j. Therefore, t_i is prior to t_j. Weight of the edge is representative of the amount of this data that should be transferred from t_i to t_j and is shown with c (t_i, t_j) .Duration of time which is taken for t_j receives the data that is produced by to t_i as shown with $c(t_i, t_j) \times \rho$. It means, ρ is considered as a network bandwidth delay factor. If $(t_i, t_j) \in$ R and t_i and t_j are mapped in the same center, there is no need for median data movement of these data between centers. Therefore, ρ is considered zero and consequently delay time between end of t_i and start t_j becomes zero. A task node can be commenced to execute when the execution related to parental nodes has been completed and all required data should be gathered or collected in one data center. In this work, node with no parent is called an input node denoted by fn, and a node with no child is called an exit node denoted by ln.

Set *I* also demonstrates that what input data each task requires as well as median data; it is shown as:

I=$\{I_1, I_2, I_3, \dots, I_m\}$

Each D_a contains one respective I_a element in set I, where I_a is representative of tasks that need D_a input data. Number of elements of I_a set is shown as I_{a_len}

Figure.1 shows a sample of workflow graph.

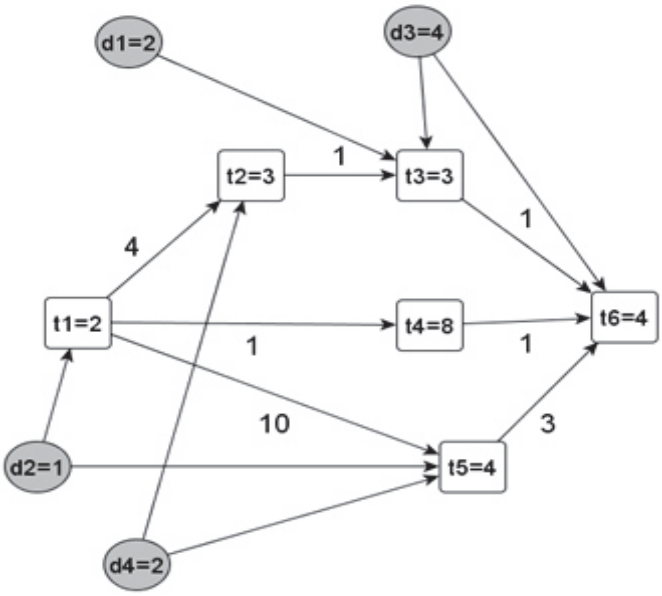

Figure 8- a sample of workflow application graph

As has been mentioned, a cloud computing system bears some distributed data centers. dc_k is applied for showing K'th data center and its storage capacity is shown with cap_k.

If t_i task is selected for mapping on dc_k center, it is expressed as map(t_i, dc_k).

The purpose of the problem is to map tasks in data center so that:

Minimized the amount of transferred date between data center ; it means that:

$$(1) Min\left[\sum_{i,j \text{ where } (t_i,t_j)\in R} c(t_i,t_j) + \sum_{a\ t_i\in i_a \text{ and } t_j\in i_a} s_a\right]$$

Where $[\text{if}(\text{map}(t_i, dc_k) \text{ and map } (t_i, dc_m)) \rightarrow k \neq m]$

And minimized the time for execution of work flow. If task start t_i is shown as st_i and its ending time is shown with et_i, the second purpose of the problem is stated as follows:

$$min\{et_n\} \qquad (2)$$

The fulfillment of these two purposes contradicts with each other to some extent. For example: (if ρ was assumed 1 for simplicity)

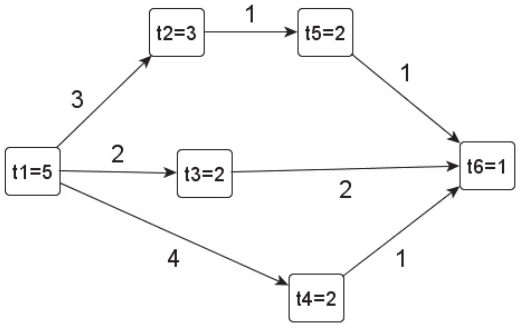

Figure 9 - an example of work flow graph

A mapping of Fig.2 in two data centers has been demonstrated in Fig.3. In this mapping, the amount of transferred data is nine and the time for ending of workflow (schedule length) is 13 (the moment for execution of

tasks is clarified).

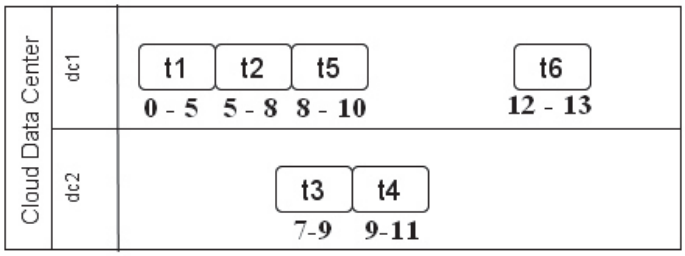

Figure 10 - first mapping for graph of Fig.2

Another mapping of this graph is shown in Fig.4 in which the amount transferred is four and the time of ending of task is 15.

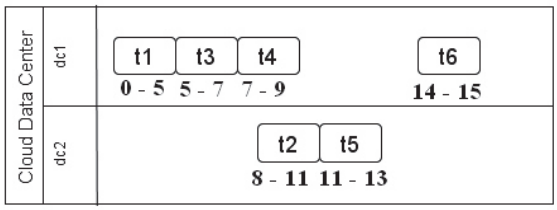

Figure 11 - second mapping for graph of Fig.2

The purpose of mentioned issues is presentation of an algorithm that is capable of fulfilling a tradeoff between data movement and the ending time for workflow execution.

4. Presented Algorithm

Generally, scheduling and tasks mapping can be done both dynamically and statistically. The place of task execution and data storage is determined before commencement of execution of application in static scheduling and the workflow execution end time and load due to execution of application in each data center is determined before commencement of execution [18]. Mapping action is performed during the execution of application in dynamic scheduling and load due to execution of application in each data center cannot be determined before the commencement of execution. In dynamic scheduling, a large amount of necessary data may be placed in one data center but the necessary space for moving other necessary data may not exist. In such condition, the total workflow may be suspended until space becomes free (necessary space is provided). Such case extends both workflow execution end time and storage space that has been occupied by application during execution. In this approach, we have applied the statistic approach because the load due to application is already determined, accepting other applications and programs are performed in data centers in such a way that provision of facilities is assured for application. In addition, we have tried to use only one computational server in each data center according to the fact that some services have been prognosticated in each data center for common (public) usage in order to let other users have available services.

The presented algorithm encompasses two main phase:

The first phase is *converting phase* and the second is *placement phase*. In the former, workflow hybrid graph converts into a simple graph and in the latter (i.e. placement phase), the place for task mapping as well as the order of execution of graph work flow is determined.

Before determining and introducing the presented algorithm, some terms should be introduced:

Earliest Start Time:

Earliest start time implies earliest time (the best possible time) for starting of t_i task that is shown with es_i and is calculated as follows:

$$es_i = \mathrm{Max}_{p_j \in pt}^{pt_{num}} \left(\sum_{k=1}^{p_{jlen}-1} \left(tp_{jk} + \left(c\left(tp_{jk}, tp_{jk+1} \right) + S_a \right) \times \rho \right) \right)$$
$$\text{if } \nexists\ I_a \in I \text{ where } t_j \in I_a \text{ and } t_i \in I_a \text{ then } S_a = 0 \qquad (3)$$

Where Pt contains all paths between t_i and fn input nodes and pt-num is representative of number of elements of this set. In addition, the whole nodes in each p_j path have been determined with tp_j and the

number of elements of this set has also been determined with p_{jlen}.

Float time for each task:

Float time for each task implies duration of time in which a task can be delayed without making any delay in start time of other tasks. It is symbolized with ft_i and is calculated as follows:

$$ft_i = \mathrm{Min}_{j \in succ(t_i)}\left(es_j - \left(c(t_i, t_j) + S_a\right) \times \rho\right) - (es_i + w_i) \quad (4)$$

$$\text{if } \nexists\, I_a \in I \text{ where } t_j \in I_a \text{ and } t_i \in I_a \text{ then } S_a = 0$$

Path:

If it is possible to reach to t_j from t_i through a sequence from one or several edges and vice versa, then it is stated that t_j and t_i are on the same path and they are stated as $is\_path(t_i, t_j)$. Also, the path that bears the total weights of nodes and edges is called the critical path. Algorithm phases will be discussed later.

4.1. Converting phase-For converting hybrid graph to simple graph:

In this phase, work flow hybrid graph is converted to a simple graph in such a way that input data which are required for several tasks are determined to be presented to these tasks.

This phase algorithm is stated as below:

1- D Input data is divided into two categories:

Data that is required for tasks -which have been located in one graph path -are placed in the first category. In other words, d_a data is placed in the first category so that:

$$\{ d_a \mid if\ i_{a_{len}} > 1\ then\ \ \forall t_j \in I_a\ \forall t_f \in I_a\ \rightarrow is\_path(t_f, t_j)\ \ \} \quad (5)$$

Other data needed to be located in different paths are placed in the second category. Then, each category data is ordered based on their amount (volume) in ascending order. For example, d_1, d_2, d_3 data is placed in the first category and d_4 data is placed in the second category for the graph as shown in Figure 1.

2- Movement trend of first category data is determined as below at first:

For each d_a data which I_a=$\{t_k, t_j, t_l, \ldots, t_f\}$, $\{t_k, t_j, t_l, \ldots, t_f\}$ tasks are ordered ascending based on es and d_a data is given to these tasks in this manner so that S_a is added to edge weight between each task to the next task based upon this ordered list.

3- Movement trend of second category data is determined as below:

For the first d_b data of this category, which I_b=$\{t_k, t_j, t_l, \ldots, t_f\}$, $\{t_k, t_j, t_l, \ldots, t_f\}$ tasks are ordered ascending based on es +ft and d_b data is given to these tasks in this manner so that S_a is added to edge weight between of each task to the next task based upon this ordered list.

Pseudo-code of this phase of algorithms is shown in Fig.5:

Converting phase of mapping algorithm	
1	cat_2 and cat_1 are empty set
2	For each d_a in set D do
2-1	If $(\text{for each } t_j \text{ in set } I_a \text{ All } t_f \text{ in set } I_a \quad \text{is\_path}(t_f, t_j))$
2-1-1	Add I_a to cat_1
2-2	Else
2-1-1	Add I_a to cat_2
3	For each I_a in set cat_1 do
3-1	Sort tasks in set I_a by earliest start time
3-2	For k=1 to I_a.count-1
3-2-1	$c(I_a.t_k, I_a.t_{k+1}) \leftarrow c(I_a.t_k, I_a.t_{k+1}) + s_a$
4	For α=1 to cat_2.count do
4-1	Sort tasks in set $cat_2.I_a$ by (earliest start time+ float time)
4-2	For k=1 to I_a.count-1
4-2-1	$c(I_a.t_k, I_a.t_{k+1}) \leftarrow c(I_a.t_k, I_a.t_{k+1}) + s_a$
4-3	calculate earliest start time and float time for all tasks

Figure 12 - Converting phase of mapping algorithm

In line 2, input data are divided into two categories. In line 3, scheduling determination for use of first category data is performed. Then, scheduling determination for use of second category data is performed. For example:

The result of converting phase execution for presented graph in Fig.1 is shown as Fig.6:

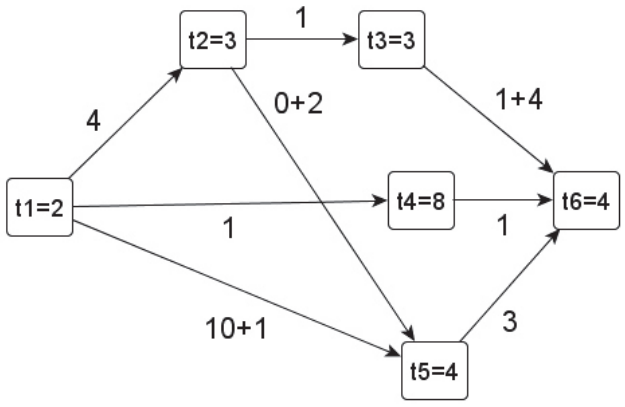

Figure 13 - result of converting phase execution for presented graph in Fig.1

It demonstrates that d_3 data is given to t_3 at first and then it will be given to t_6; d_4 data is given to t_2 at first and then it will be given to t_5; d_2 data is given to t_1 at first and then it will be given to t_5; and finally d_1 data will be given to t_3.

4.2. Placement phase- for mapping data and scheduling tasks:

In this phase, tasks are divided into some groups in which each group task performance order has been determined. Then, the time for data movement and necessary storage space for execution of each of the group tasks will be defined. According to the most necessary space required for the group, it is mapped in one of the

available data centers. If the most required group space amount amounts to what cannot be mapped in any of the data centers, there is obligation for moving some stored data from one group to another.

Phase levels of such algorithm can be stated as follows:

1- Earliest start time for all graph tasks has to be calculated for start to execution.

2- Critical path between fn node and ln node should be found. Nodes through the path should be placed in $group_k$.

3- Es for all tasks should be computed again since they have produced group.

4- If there is gap between ending of t_i task and start of next t_j task, an optimum node should be selected for placement in the gap ranging from succeed (t_i) and predecessor(t_j) tasks that its execution time is equal t0 or less than gap length. Optimum node is determined as follows:

$$Max_h(\sum(c(t_h,t_l)\times\rho)-\sum(c(t_h,t_m)\times\rho)) \quad (6)$$

where $[(t_l \in group_k) \text{ and } (t_m \notin group_k)]$ And $[t_h \in pred(t_j) or\ t_h \in succ(t_i)]$

According to the above, shown relation optimum node is selected and it is placed in considered empty space (gap).

5- A Sub-graph should be created for each category of residual connected nodes (if sub-graph includes several fn and ln nodes; nodes with the most critical path are stretched between them should be determined as fn and ln). Then, step 1 to 5 of algorithm placement should be performed in each sub-graph and the next group should be determined. This cycle continues until all nodes in workflow graph are grouped.

6- In the next level, the time for moving data between created groups is determined as in the following relationship:

$$\text{Transfer\_time}(t_i,t_j)_{if\ c(t_i,t_j)\neq 0} = \begin{cases} es_i + w_i, & (t_i \in group_k) \text{and} (t_j \in group_m) \rightarrow k < m \\ es_j - c(t_i,t_j)\times\rho, & (t_i \in group_k) \text{and} (t_j \in group_m) \rightarrow k > m \\ -1, & (t_i \in group_k) \text{and} (t_j \in group_m) \rightarrow k = m \end{cases} \quad (7)$$

The procedure of groups' determination is in the manner in which each group has more load than the next group since each graph converts to a smaller sub-graph at any level. Therefore, we have tried to make the situation in such a way that if data is to be transferred from one group to another group with higher index, such transfer is performed as fast as possible on one hand, and if data is to be transferred to a group with a lower index, such transfer is delayed as much as possible. Under this condition, load balance is restored to some extent. If t_i and t_j task exist in one group, its median data should not be transferred. Therefore, transfer time has been shown with -1.

7- The amount of required storage space should be determined during the execution of each of the group tasks in this level. The amount of storage load of $group_k$ during execution of t_i task is computed as below:

$$\begin{aligned} &= s_{load(t_i)} \\ &\sum_{j\in pred(t_i)} c(t_j,t_i) + \sum_{before(t_i,t_k) and\ after(t_i,t_w)} c(t_k,t_w) + \\ &\sum_{j\in succ(t_i)} c(t_i,t_j) + \sum_{stt(d_f)=t_e\ and\ after(t_i,t_e)} S_f + \\ &\sum_{ent(d_f)=t_e\ and\ before(t_i,t_e)} S_f + \\ &\sum_{after(t_i,t_j)} \sum_{t_e\in pred(t_j) and\ transfer\_time(t_e,t_j)+c(t_i,t_j)\times\rho<es_i} c(t_e,t_j) + \\ &\sum_{before(t_i,t_j)} \sum_{t_e\in succ(t_j) and\ transfer\_time(t_j,t_e)>es_i} c(t_j,t_e) \end{aligned} \quad (8$$

In the above shown relation, $\sum_{j\in pred(t_i)} c(t_j,t_i)$ refers to data that are necessary during execution of t_i.

If one t_j task group is placed before t_i task it is shown with $before(t_i,t_j)$. If also t_j task is placed

after t_i task, it is shown with $after(t_i,t_j)$. Therefore, $\sum_{before(t_i,t_k)and\ after(t_i,t_w)} c(t_k,t_w)$ is due to median data that is produced in group tasks before t_i and they are required for following tasks after t_i in the group.

According to the fact that primary input data should be stored in the system until the ending of workflow execution, therefore, there is some load in this data. If d_f input data is given to t_e before all tasks, it is shown as $stt(d_f) = t_e$ and they should be placed in the group in which t_e task exists. Also if d_f input data is given to t_e as the last task it is shown as $ent(d_f) = t_e$ in which after executing t_e task, d_f data should be stored in the same group in which t_e exists. Therefore, $\sum_{stt(d_f)=t_e\ and\ after(t_i,t_e)} S_f$ and $\sum_{ent(d_f)=t_e\ and\ before(t_i,t_e)} S_f$ should be added to $s\_load(t_i)$.

$\sum_{j\in succ(t_i)} c(t_i,t_j)$ implies to median data that are produced with t_i itself and they should be stored for the next tasks.

$\sum_{after(t_i,t_j)} \sum_{e\in pred(t_j)and\ transf.er\_time(t_e,t_j)+c(t_i,t_j)\times\rho<es_i} c(t_e,t_j)$ is the result of median data that were produced by other group tasks and they are needed for tasks after t_i in a group that transferred to another group *before execution* of t_i task.

Also $\sum_{after(t_i,t_j)} \sum_{e\in pred(t_j)and\ transfer\_time(t_e,t_j)+c(t_i,t_j)\times\rho<es_i} c(t_e,t_j)$ results from data which has been produced by tasks before t_i in group, that transferred to another group *after execution* of t_i task.

8- According to maximum load in each group, if any data center that can supply such storage space exists, the group should be mapped in it, but if such data center does not exist, some data which are more than the available capacity of the data center should be transferred to groups with lowest storage load and when they are needed, they should be returned to their group. Selection data for transferring should be done according to the following:

If t_i has highest storage load in group:

8-1- if $stt(d_f) = t_e$ and $after(t_i,t_e)$ condition has been established for d_f input data; at the starting time of work flow execution, this data is placed in a group with lowest amount of load and then will be transferred to its group at the time of work flow execution when necessary; or if $before(t_i,t_e)\ and\ ent(d_f) = t_e$ condition exists, this data is transferred to a group with lowest amount of load after execution of t_e.

8-2- if t_i task load does not reach to the lower amount of available data center, the following data should be selected for transferring to group with the lowest amount of load and repetitive return to its group.

$$max_{j,k}(es_j + w_j + 2\times c(t_j,t_k)\times\rho) < es_k \quad where\ before(t_i,t_j)and\ after(t_i,t_k) \tag{9}$$

The above shown equation was selected because data which has been produced with t_j and is required by t_k should be transferred to another group and returned repetitively without making any delay in t_k task start time.

In the above equation, max has been used because this transfer decreases t_i load as well as other task loads between t_j and t_k.

8-3- if thus, t_i task load does not reach the lower amount of available data center, equation obligation in relation (9) should be deleted and other data should be selected for transfer. pseudo of this phase of algorithm is

as following:

Placement phase of mapping algorithm

1 Insert all tasks of DAG in to set T_0 and k=0

2 While task is in T_k that does not grouping do

2-1 calculate earliest start time and float time for all in T_k

2-2 find critical path between fn and ln nodes

2-3 place tasks in critical path in the $group_k$

2-4 calculate earliest start time and float time for all in T_k

2-5 for r=1 to $group_k$.count-1 do

2-5-1 if$(es_{r+1} > es_r + w_r)$

2-5-1-1 find h that $Max_h(\sum(c(t_h,t_l)\times\rho) - \sum(c(t_h,t_m)\times\rho))$
where $[(t_l \in group_k) \text{and} (t_m \notin group_k)]$And
$[t_h \in pred(t_{r+1}) or\ t_h \in succ(t_r)]$

2-5-1-2 place t_h in $group_k$ between t_r and t_{r+1}

2-6 for connected tasks that don't grouping create sub graph[s] and done step 2 to 11 for k=k+1

3 for each t_i and t_j that t_i is in $group_k$ and t_j is in $group_m$ and $c(t_i,t_j) > 0$ do

3-1 if$(k < m)$

3-1-1 transfer_time$(t_i,t_j) = es_i + w_i$

3-2 else if$(k > m)$

3-2-1 transfer_time$(t_i,t_j) = es_j - c(t_i,t_j)\times\rho$

3-3 else

3-3-1 transfer_time$(t_i,t_j) = 0$

4 for each task t_i in DAG calculate $s\_load(t_i)$

5 for each group find max(s_load)

6 for each group do

6-1 b=true

6-2 if$(is\ data\ center\ that\ cap > \max(s_load))$

6-2-1 map(group, Best data center)

6-3 else

6-3-1 while $is\ n't\ data\ center\ with\ cap > \max(s\_load)$ and B Do

6-3-1-1 if $[stt(d_f) = t_s \text{ and } after(t_i,t_s)]$ or $[ent(d_f) = t_s \text{ and } before(t_i,t_s)]$

6-3-1-1-1 transfer d_f to group with min load

6-3-2 while $is\ n't\ data\ center\ with\ cap > \max(s\_load)$ and B Do

6-3-2-1 find

$max_{j,k}(es_j + w_j + 2\times c(t_j,t_k)\times\rho) < es_k \quad where\ before(t_i,t_j) and\ after(t_i,t_k)$

And transfer $c(t_j,t_k)$ to group with min load that max$(s_load) = s_load(t_i)$

6-3-2-2 if $(max_{j,k} = 0)$

6-3-2-2-1 B=false;

6-3-3 if(!B)

6-3-3-1 while $is\ n't\ data\ center\ with\ cap > \max(s\_load)$ Do

6-3-3-1-1 find

$max_{j,k}(es_j + w_j + 2\times c(t_j,t_k)\times\rho) \quad where\ before(t_i,t_j) and\ after(t_i,t_k)$

And transfer $c(t_j,t_k)$ to group with min load that max$(s_load) = s_load(t_i)$

6-3-4 map(group, best data center)

Figure 14- pseudo of Placement phase of mapping algorithm

In lines related to part two of this phase, graph tasks are placed in groups and the order of each group tasks is determined. In lines related to part three of this phase, data movement time that should be transferred

between groups are determined. In line 5, it is determined what group task has the highest amount of storage load at the In lines related to part 6 of phase, each group is mapped to an available data center. In this part, BEST DATA CENTER implies to the data center which has the smallest storage space for storage in a manner that supplies maximum need for group (i.e. max(s_load)).

Only a percentage of such storing space of data center should be taken into consideration for execution of workflow application according to possibility of results production that require being permanently stored in data center or other applications that require to be execution this data center.

5. Experiments and results

5.1. Simulation strategies

Evaluation of presented algorithm was made by using C# programming language. Then some graphs were produced by using a function that has been presented in task [8] and algorithms were executed in such graphs; results were then shown in graphs by using Matlab. After that, for evaluating algorithm efficiency and data movement amount, it was compared with data placement strategy that has been presented in [8].

Normal schedule length shown below has been applied for comparing schedule length [15]:

$$NSL=\frac{et_n}{\sum_{t_i \in Cp} W(t_i)} \quad (10)$$

Where;

et_n is total time for execution of work flow

$W(t_i)$ is duration of t_i task execution(completion)

According to the fact that $\sum_{t_i \in Cp} W(t_i)$ demonstrates total time of execution of critical path nodes and such amount is representative of low limit for schedule length, NSL is always greater than or equal to 1.

Normal data movement is defined as following for comparing movement amount:

$$NDM=\frac{Data\ Movement}{\sum_{t_i,t_j \in T} C(t_i,t_j)+\sum_{a=1}^{m}[s_a \times (I_{a\_len}-1)]} \quad (11)$$

Where,

Data movement shows data movement amount due to execution of algorithm.

$\sum_{t_i,t_j \in T} C(t_i,t_j)$ Shows total amount of median data in workflow application

$\sum_{a=1}^{m}[s_a \times (I_{a\_len}-1)]$ is due to movement of input data. In other words, $\sum_{t_i,t_j \in T} C(t_i,t_j)+\sum_{a=1}^{m}[s_a \times (I_{a\_len}-1)]$ shows maximum amount of movement. Therefore, always NDM ≤ 1≤0

In shown results, all numbers in graphs are the average results of algorithm execution in 10 different graphs with the same parameters.

5.2 Simulation results:

In this experiment, algorithms were executed in graphs with the same parameters and different task numbers. NDM and NSL produced from the said two algorithms are shown in Fig.8:

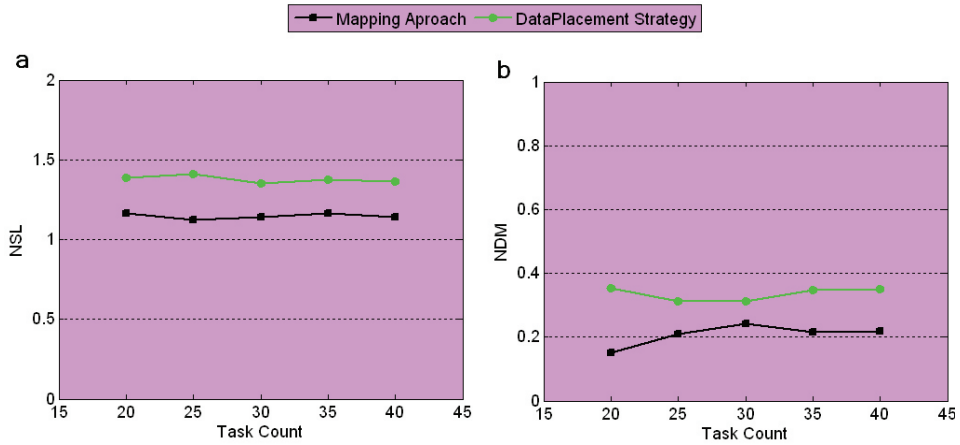

Figure 8 - Comparison between two approaches based on change in number of workflow tasks

The above figure shows that both approaches demonstrate stable behavior when number of tasks increases. The cause of this stability is that both NDM and NSL parameters change by impact of input data amount and via coefficient of median branch between tasks. Therefore, NDM and NSL amounts of both approaches are stable to some degree when these parameters are the same and they have no subtle differences. However, our algorithm provides better results.

In the next experiment, efficiency of the said two algorithms was evaluated against change in input data number in graphs with the same parameters.

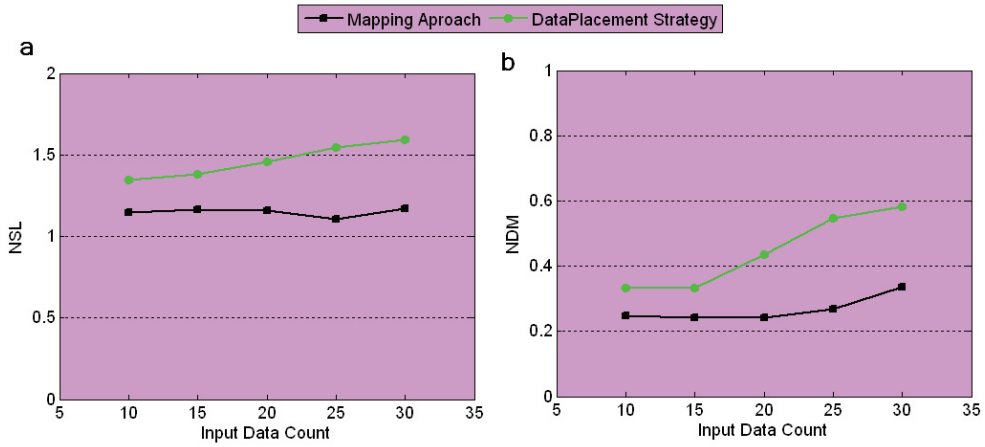

Figure 9 - evaluation of efficiency of two approaches according to change in number of input data

As deducted from diagrams shown in Figure 9, in comparing of data placement strategy, our algorithm presents better NDM and NSL with increase in number of input data, especially, increase in number of input data and data placement strategy ends increase in NDM amount. According to the fact that our algorithm is determined in the first phase, it demonstrates how data necessary for several tasks are given to them, and how they are combined in the main graph. Therefore, critical path is affected and data with greater size (volume) are placed in the critical path. Finally, the critical path data and task are mapped in one data center. Such approach that manipulates or makes data with greater volume requires lower movement.

Conclusion and further works

In addition, task execution should be scheduled in such a way that (as far as possible) the time for total completion of the application becomes minimal. It tends to decrease in use of computational services and storage which induces users and client satisfaction. Therefore, we present an approach that fulfills two purposes. Experimental results are indicative of success of the presented approach. In this work, the data duplication technique has not been used by us. That's because of increase in costs and expenses of using a cloud system.

References

[1] E. Deelman, A. Chervenak, Data management challenges of data-intensive scientific workflows, *in: IEEE International Symposium on Cluster Computing and the Grid*, (2008), pp. 687–692.

[2] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-Science: An overview of workflow system features and capabilities, *Future Generation Computer Systems* 25(5)(2009), pp. 528-540.
[3] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, I. Brandic,_Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility , *Future Generation Computer Systems* 25(6)(2009),pp. 599-616.
[4] R. Maggiani, Cloud computing is changing how we communicate, IEEE International Professional Communication Conference 19(2009).
[5] K. Kaya, B. UÇar, C. Aykanat, Heuristics for Scheduling file-sharing tasks on heterogeneous systems with distributed respositories, J. Parallel Distrib. Coput 67(2007),pp. 271-285.
[6] K. Kaya, B. Uçar and C. Aykanat, Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories, J. Parallel Distrib. Comput. 67 (2007), pp. 271 – 285.
[7] T. Kosar, M. Livny, Stork: Making data placement a first class citizen in the grid, in: Proceedings of 24th International Conference on Distributed Computing Systems, ICDCS (2004), pp. 342–349.
[8]] D. Yuan, Y. Yang, X. Liu, J. Chen, A Data Placement Strategy in Cloud Scientific Workflows , Future Generation Computer Systems (FGCS), 26(8)(2010), pp. 1200-1214.
[9]T. Kosar, S. Son,G. Kola,M. Livny, Data placement in widely distributed environments , *Advances in Parallel Computing* 14(2005), pp. 105-128.[10] Fatma A. Omara,Mona M. Arafa, Genetic algorithms for task scheduling problem , *Journal of Parallel and Distributed Computing* 70(1)(2010), pp. 13-22.
[11] Y. Yuan, X.Li,Q.Wang,X. Zhu, Deadline division-based heuristic for cost optimization in workflow scheduling , *Information Sciences* 179(15) (2009),pp. 2562-2575 .
[12] X.Tang, K.Li, R. Li,B.Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems , *Journal of Parallel and Distributed Computing* 70(9)(2010),pp. 941-952 .[13] Z. Shi,Jack J. Dongarra, Scheduling workflow applications on processors with different capabilities , *Future Generation Computer Systems* 22(6)(2006),pp. 665-675.
[14] Q. Kang, H. He, H. Song and R. Deng, Task allocation for maximizing reliability of distributed computing systems using honeybee mating optimization, Journal of Systems and Software 83(11) (2010), pp. 2165-2174.
[15] C. Chiu, Y. Yeh,J.Chou, A fast algorithm for reliability-oriented task assignment in a distributed system, *Computer Communications* 25(17)(2002),pp. 1622-1630 .
[16] I. Ahmad and Y. Kwok, Benchmarking and comparison of the task graph scheduling algorithms, *J. Parallel Distrib. Comput.* **95** (1999), pp. 381–422.
[17] A. Chervenak, E. Deelman, M. Livny, M.-H. Su, R. Schuler, S. Bharathi, G. Mehta, K. Vahi, Data placement for scientific applications in distributed environments, in: 8th Grid Computing Conference, (2007), pp. 267–274.
[18] G. Singh, K. Vahi, A. Ramakrishnan, G. Mehta, E. Deelman, H. Zhao, R. Sakellariou, K. Blackburn, D. Brown, S. Fairhurst, D. Meyers, G.B. Berriman, J. Good and D.S. Katz, Optimizing workflow data footprint, *Scientific Programming* **15** (2007), pp. 249–268.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar