International Journal of Engineering & Technology, 7 (3.28) (2018) 284-289

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Level-Based Clustering Approach to Scheduling Workflows in Clouds

Tawfiq Alrawashdeh[1], Zarina Mohamad[2]*, Aznida Hayati Zakaria[2]

[1]Al Husein Bin Talal University, P.O. Box 20 Ma'an, Jordan
[2]Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Besut Campus, Terengganu, Malaysia
**Corresponding author E-mail: zarina@unisza.edu.my*

Abstract

With the rapid increment on the complexity of the workflow, and the resultant demand on the scalability of the environment, executing workflows on traditional environment such as grids and clusters has become challenging task. Generally, schedulers aims to find a trade-off between execution, user requirement, and execution cost. Combine this with the uncertainty in the execution environment results in underlining the importance of designing scalable scheduling algorithm that adopt to the changes in the execution process. Toward this end, we propose the Level-Based Clustering (LBC) algorithm. By considering each level tasks as a single object (cluster), this algorithm aims to establish a relationship between the execution requirement for each cluster, and the number of resources that must be used to execute the entire workflow. We have compared our algorithm with three well-known algorithms from the literature, and the result show that the LBC algorithm achieves 50%, 25%, 50% on average improvement in term of cost, makespan and the number of resources used, respectively.

Keywords: *scheduling, workflows; cluster; divide and conquer.*

1. Introduction

Recently, scientific workflows become increasingly common for compute-intensive and data-intensive scientific applications. Such workflows is normally represented as Direct Acyclic Graph (DAG), where nodes represents tasks with computational requirements, and edges represent the data dependency between the tasks (Figure 1). Workflows is typically executed on distributed environments, where each task is assigned to processing core. Due to the scale of the workflows and the intensity of its processing requirement, and the intensity of its computational requirement, cloud computing (Infrastructure as a Service (IaaS)) has emerge as an efficient environment to execute scientific workflows.

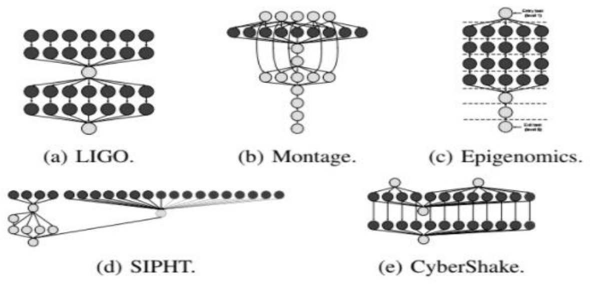


Fig. 1: Scientific Workflows Example

The IaaS provide the user with the ability to access a shared on-demand compute infrastructure, using pay-per-use pricing model. This is often done by leasing virtualized resources, referred to as virtual machines (VMs), with a pre-determine computer hardware, memory, storage, and information measure capability.

Typically, the objective of scientific scheduling workflows is to either reduce the executing cost [16, 17, 24], or to reduce the execution time [3, 5, 14, 15]. Proposals that address reducing the execution cost normally handle the execution time as time-deadline constraint [20, 21, 25]. In term of reducing the execution time, proposals that aim to address this objective normally handle reducing the execution time as a secondary objective [4, 22, 23]. In addition, some proposals have address the bi-objective problem of minimizing the execution time and cost [18, 19, 6].
The problem of scheduling scientific workflows in clouds is NP-Complete in nature [13]. To efficient address this problem, we need to determine the right number of resources to rent. Over-renting is expected to increase the executing cost, since we will pay for unused time slots. Under-renting is expected to increase the total execution time (makespan). In addition, due to the data dependency constraints, we need to take into consideration the structure of the workflows during the scheduling process to efficiently utilize the resources (VMs). This is established, since with the presence of the precedence constraints, the scheduling must respect the data dependency between the tasks to avoid having unused time slots.
Toward this end, we propose the Level-Based Clustering (LBC) algorithm. This algorithm divides the workflow's tasks into clusters, where tasks belong to the same level is allocated to same cluster. This division aim to simplify the resources allocation process, since we treat clusters as isolated objects with computational requirements. We compare the performance of the LBC algorithm against three other well-known algorithms from the literature. The results show that the LBC algorithms achieves 50%,25%,50% improvement compare to the other algorithms, in term of cost, makespan and the number of resources used, respectively.

The rest of the paper is organized as follows. Section 2 presents and discuss the related works. In Section 3, we formally define the scheduling problem. In Section 4, we present our algorithmic solution. The results is discussed in section 5, and the paper is concluded in section 6.

2. Related Works

In this section, we present and discuss the most related proposals to the problem of scheduling scientific workflows in clouds. Many proposals have investigated the problem of scheduling scientific workflows in cloud.

Many proposals have investigated the problem of minimizing the execution time (makespan) [3, 5]. In [3] presented the heterogeneous earliest end time (HEFT) algorithm. This algorithm schedules the tasks using a greedy approach. In each iteration, the process works by assigning the task under consideration to the VM that results in the earliest actual execution time. The efficacy of this approach depends on the available number of resources. However, this approach does not take into consideration the execution cost during the construction of the schedule, and this may result in increasing the execution cost. In [5] presented an improvement version of the shuffled frog leaping algorithm (ASFLA). The results demonstrate significant reduction in the makespan, however the authors did not take into consideration the execution cost for resource.

In [4], the authors studied the problem of minimizing the execution cost under the presence of the time-deadline constraint. To address this problem, the authors proposed the IC-PCP algorithm. Starting from this exit task, this algorithm distribute the time-deadline over the entire branches of the workflow. Then, it schedule the tasks starting from the entre level on the cheapest VM that results in satisfying the tasks deadline. In [8] proposed adaptive algorithm using Mixed Integer Programming (MIP) method that also aims to reduce the cost under the presence of deadline constraint. However, the algorithm does not allow using VMs that already assigned.

Several proposals have investigated the problem of minimizing the makespan under the presence of the budget constraint [16-28]. In [16], the authors proposed heuristic-based solution that aim to reduce the overall execution delay. In each iteration, the main idea of this approach is to improve the current schedule by considering the left budget. In [27] proposed priority based genetic algorithm termed as BCHGA, which address the problem of scheduling the tasks of the workflow under the presence of the budget constraint. In this algorithm, based on the locality of the tasks, each task is assigned either bottom level priority (b-level), or top level priority (t-level). Then, in each round, the algorithm by trying to find better schedule in term of makespan, while minimizing the execution cost. In [26, 28], the authors adopt similar strategy to minimize the execution cost under the presence of the budget and execution time constraints.

Many proposals have addressed the problem of minimizing both the execution time and cost [1-2, 6, 9]. Our work falls into this category. In [1] proposed cluster-based algorithm that aim to determine the priority of each objective using a slack parameter. The value of this parameter ranges from 0 to 1, where assigning 0 to this parameter results in prioritizing the execution time, and assigning 1 to this parameter results in prioritizing the execution cost. This algorithm starts by dividing the workflow tasks into partition in a sequential fashion starting from the entre level tasks. Then, the number of resources assigned to each partition is determined based on the value of the slack parameter. For each partition this number ranges from 1 resource to the number of resources that ensure each task is executed on its earliest starting time. Similarly, the RDAS algorithm [2] uses a fair allocation strategy, which aim to construct schedule with the objective of minimizing the execution time and cost. In [6] proposed algorithmic approach to address the same problem. In this approach, the user has to assign a weight value for each objective.

3. Problem Definition

The input can be represented as Directed Acyclic Graph (DAG) $G = \langle T . E \rangle$, where $T = \{t_1 . t_2 . \dots t_n\}$ is a finite set of tasks, and n denotes the number of tasks in the workflow application.

Each edge connects two tasks representing their precedence constraint or data dependency. A dependency ensures that a child node cannot be run before all its parent tasks finish successfully execution and transfer the required input data. The last task to transfer its data a given task is termed the Most Influential Parent (MIP).

Additionally, we are given a set of resource $R = \{R_1 . R_2 . \dots . R_m\}$. Each set of resources contains restricted range of uniquely identifies resources in terms of memory and storage space. Capacity is the main distinguisher between the resources set for instance. Without losing of generality, we assume that $R_j \ (j > 1)$ is j times faster and more expensive compared to R_1 resources. The execution time for the entire workflow is denoted as makespan or schedule length. For each task, the actual start time (AST) and actual finish time (AFT) is not expected to be similar to the Earliest Start Time (EST) and Earliest Finish Time (EFT). These values depend on the available number of resources.

4. Algorithmic Solution

Due to the data dependency constraints, a task cannot start its execution without receiving all of the required data from its parent. This suggest that the scheduler must ensure that the tasks located at the same level must finishes their execution relatively at the same time in order to reduce the data waiting periods for the tasks located at the next level. In addition, the computational requirements for each level's tasks is typically different. Some level has a single task, and other levels have hundreds of tasks. This suggests that the structure of the workflow must be also taken into consideration during the construction of the execution schedule. To address these issues, in this section, we present the Level-Based Clustering algorithm.

This algorithm starts by assigning tasks located at the same level to a single cluster. This aim to simplify the scheduling process, since tasks located at different level have different computational requirements. Then, starting from the entry level cluster, the resource allocation step starts by identifying each cluster share of the available resources. This is established based on the number of tasks assigned to this cluster, and their computational requirements. In this step, the objective is identifying the number of VMs that must be assigned to each cluster such that all tasks belong to the same cluster relatively finish their execution at the same time. Then, in the tasks scheduling step, we identify the actual starting time for each task.

4.1. Clustering Step

In this step (Algorithm 1), we partition the workflow into set of clusters such that tasks belong the same level located at the same partitions. This algorithm starts by calculating the number of required clusters based on the height of the workflow (line 5). Then, the algorithm iterates to assign each task to its level cluster (line 8-17). By performing level based clustering, we aim in to handle each cluster as an isolated object in term of computation requirements.

Algorithm 1: Clustering Step

```
1 : procedure PARTITIONING ($G . L$)
2 : ▶Input: $G = < V . E >$, Levels $< L >$
3 : ▶Output: C = c1,cp2, …. cm.
4 :     p1 ← vi ∈ V in L1
5 :     i ← 1
6 :     while we have unassigned node do
7 :         j ← 2
8 :         for each l in L do
9 :             if |v| ∈ V in Li equal to 1 then
10 :                ▶ the entry tasks located at level 1
11 :                Cj-1 ← vi ∈ V in Li
12 :            else
13 :                Cj ← vi ∈ V in Li
14 :                j ← j+1
15 :            end if
16 :            i ← i+1
17 :        end for
18 :    end while
19 :    return C
20 : end procedure
```

4.2. Resource Allocation Step

The objective of the resource allocation step is to determine the maximum number of VMs that will be allocated to each cluster. In this step, we ensure that the selected VMs run relatively for the same amount of time, and this establish an upper bound on the execution time for each clusters. Such bounding reduces the expected delay, since we will have deadline on the cluster execution time, and this reduces the impact of the data dependency constraints. The main idea of this step is to determine the size of the execution time-slot for each clusters. For each cluster, this time-slot represents the maximum amount of time a VM can use to execute this cluster tasks.

Algorithm 2 describe the process of the resource allocation step. For each cluster (for loop line 3), to calculate the time-slot for cluster (A), we start by calculating the total running time for this cluster's tasks (r_A) (line 4). Then, we calculate the average running time (av_A) for this cluster tasks (r_A / n_A), where n_A is the number of tasks assigned to cluster A (line 5). Now to determine the size of the time-slot, we calculate the average running time for the available VMs on this cluster tasks. The time-slot (ts) for cluster A can be calculated as follows (line 6):

$$ts_A = \frac{r_A}{avm} + av_A \qquad (1)$$

For each cluster, the time-slot represents the upper-bound for the VMs running time. Thus, we establish latest execution time for the tasks belong this cluster. This helps in term of maximization the utilization of the VMs, since there is no dependency between the same cluster tasks, and thus they can be executed simultaneously. In addition, this helps in reducing the number of used VMs, because we assign tasks to the VMs in sequential order, and a task will not be assign to a new VM until the current VM reaches its limit. Once the time-slot for the current considered cluster is determined, we start the tasks allocation process for this cluster tasks (line 9).

Algorithm 2: Resource Allocation

```
1 :  procedure R_allocation  ($R . C$)
2 :  ▶ Output: S = s1, s2, …, sm.
3 :  for each $c_i$ in C do
4 :      calculate $(r_A) \leftarrow \sum_{T=0}^{n}$ runtime
5 :      calculate $(av_A) \leftarrow r_A / n_A$
6 :      calculate $ts_A = \frac{r_A}{avm} + av_A$
7 :      calculate  S  ← $ts_A$
8 :      for each task ∈ C do
9 :          $S_i$ ← task_schuduling ($ts_A$,$c_i$)
10 :     End for
11 :  end for
12 :         return S
13 : end procedure
```

4.3. Tasks Scheduling Step

In this step (Algorithm 3), the actual finishing time for each task in the current considered cluster is determined. Starting from the entry level cluster, we order the tasks based on their EST (line 5). Then, we start the process of assigning the tasks to the selected VMs (lines 6-15). A task will be assigned to the current considered VM if this does not result in exceeding this VM time-slot. Otherwise, a new VM will be used. Once a task is assigned to VM, this task AST and AFT are determined based on this task order of execution. This process stops once all considered tasks is assigned to VM. Next, we calculate the actual finishing time for this cluster, and that represent the latest actual finishing time for the tasks that belong to this cluster (C). Different scheduling strategy can be used in this step. For instance, we can use the best fit strategy from the pin-packing literature [29]. However, in our work, the location of the tasks on the VMs does not play an important role in determining the quality of the final schedule, since the algorithm behavior is controlled by the size of the time-slot.

Algorithm 3: Tasks Scheduling

```
1 : procedure TASK_scheduling ( $ts_A . C$)
2 : ▶Output: scheduled tasks.
3:   for each T ∈ C do
4 :      for $c_j$ ∈ C do
5 :          T ← Asc.Sort (EST)
6 :          for $t_r$ ∈ T  do
7 :              if $ts_A$ >=j   then
8 :                  Create a new $vm_j$
9 :                  J = 0
10 :             end if
11 :             $v_r \leftarrow t_r$
12 :             $AST \leftarrow \begin{cases} AFT \\ 0 \end{cases}$
13 :             $AFT \leftarrow AST + \text{Runtime of } t_r$
14 :             j ← j + Runtime of $t_r$
15 :         End for
16 :     End for
18 : End for
```

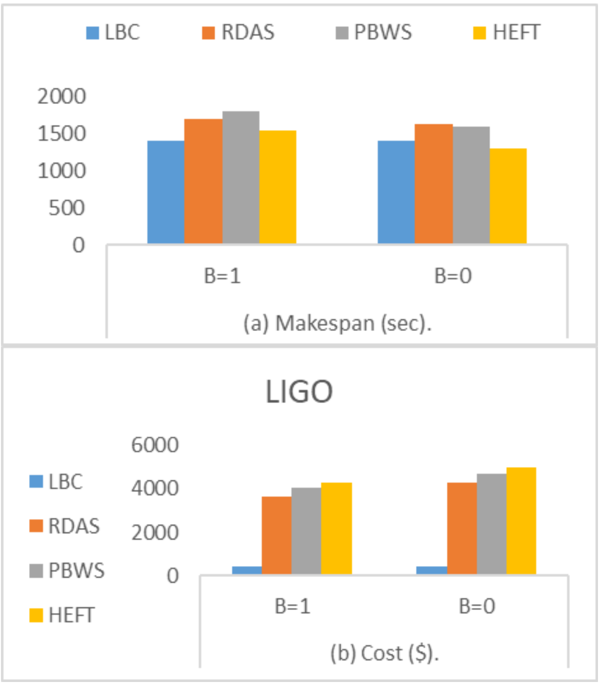
5. Results and Discussion

To evaluate the performance of the presented algorithm, we have conduct an extensive set of experiments. As am input, we have used five types of well-known real scientific workflows: Epigenomics, LIGO, SIPHT, Montage and CyberShake (Figure 1). These files are obtained from the Pegasus workflow repository (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator). We used three type of resource sets R = R1, R2 and R3. We assume that VMs from resource set R3 is three times faster

than R1 resources. The hourly rates used in our experiments are \$10, \$20 and \$30 for R1, R2 and R3 respectively.
To benchmark the performance of our algorithm, we compare its performance against the PBWS [1] RDAS [2] and HEFT [3] algorithms. The objective of the HEFT algorithm is to determine the fastest schedule to execute the tasks of the workflow. In each iteration, the algorithm allocates the task under consideration to the VM such that execution time of this task is minimized. In the RDAS, the algorithm starts by partitioning the workflow tasks based on the execution time for the critical path tasks. Then, it uses fair share strategy to determine the number of tasks allocated to each partition. The PBWS algorithm uses slack parameter (B) to determine the priority of each objective function. The value of this parameter range from 0 or 1. When this value is equal to zero, the algorithm focus on reducing the execution time. Assigning one to this parameter results in shifting this algorithm behavior toward reducing the cost.
For the fairness of the comparison, we start by running PBWS for B = 0 and 1. Then use the obtained number of VMs by PBWS as an input to the HEFT, RDAS, and LBC algorithms.

Next, we present and discuss the results for each input scenario.

LIGO: Figure 2 shows the results for the LIGO workflow experiments. From the figure we can see that the LBC algorithm outperform the other algorithms in term of cost. In addition, we can see that the LBC algorithm obtains relatively the same execution time, regardless of B value. The LIGO workflow has well organized structure. In the LBC algorithm this results in establishing clusters with relatively the same running time. Such structure is expected to results in using relatively the same number of VMs to execute each clusters. This reduces the number of idle time-slots, and thus reduces the cost of the obtained schedule by the LBC algorithm. The HEFT algorithm uses greedy strategy that focus on reducing the makespan. This is expected to increase the cost of the schedule obtained by this algorithm, since resource utilization was not takin into consideration during the construction of the schedule. In the PBWS and the RDAS algorithms having partitions that cross several levels creates a dependency relationship between the structure of theses partitions and the performance of the obtained schedules. This is established since having partitions with unorganized structure results in increasing the expected cost and makespan.



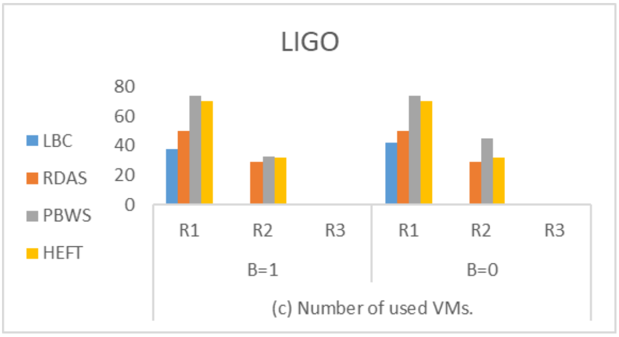


Fig. 2: LIGO workflows

CyberShake: Figure 3 present the results for the CyberShake workflow experiments. From the figure we can see the LBC algorithm outperforms the other algorithm in term of makespan. In term of cost, we can see that when B = 1, the LBC algorithm outperform the other algorithm. Reducing the value of B to zero, results in LBC and PBWS obtains relatively the same cost. The CyberShake workflow has wide-structure, with significantly large number of tasks in each level. In the LBC algorithm, the mechanism of constructing the clusters based on the level of the tasks has a great advantage in this situation. This occurs because the LBC algorithm aims to reduce the average execution time for level tasks, and increasing the number of tasks in each level highlight the benefits of such strategy. This is the main reason behind the seen performance in term of makespan. Regarding the execution cost, having workflow levels with relatively high number of tasks results in increasing the execution overhead. This is established since structure increases the probability of having unused time-slots, due to the presence of data dependency and the large number of tasks in each level.

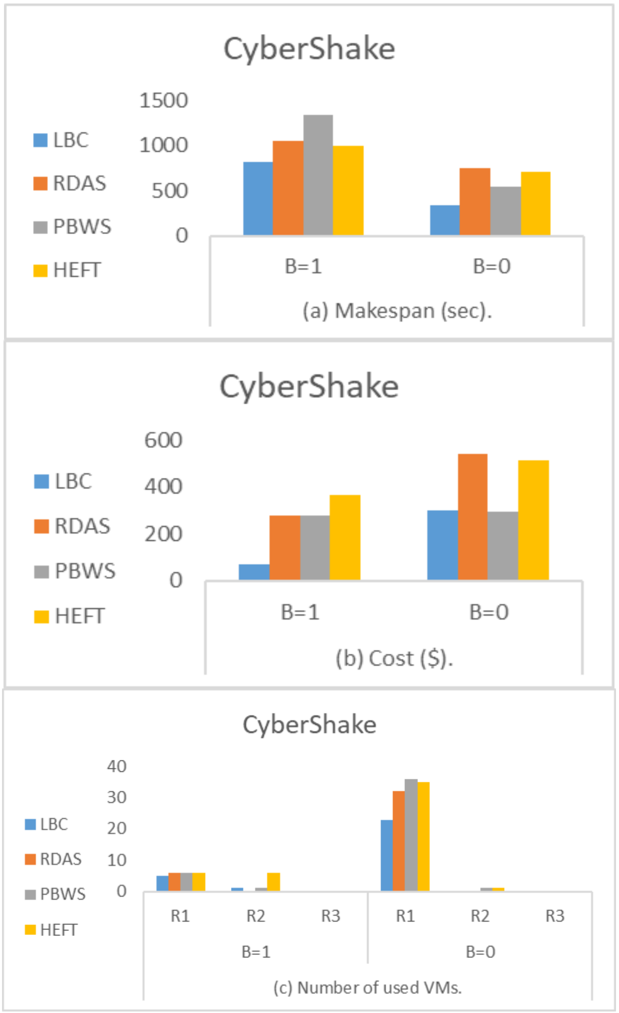


Fig. 3: CyberShake workflows

Epigenomics: The results for this workflow set experiments in shown in Figure 4. From the figure we can see that the LBC algorithm constantly outperforms the other algorithms in term of cost. In addition, we can see that the makespan of the schedule obtained by the LBC algorithm is relatively similar to the PBWS. The Epigenomes workflows have balances structure with high demand computational tasks. In these settings, the mechanism of clustering the workflow tasks based on their level reduces the expected performance of the LBC algorithm. This occurs because in order to schedule the tasks for a given, the tasks of the pervious cluster need to finish their execution. Combine this with the presence of the computationally demanding tasks introduce execution delay.

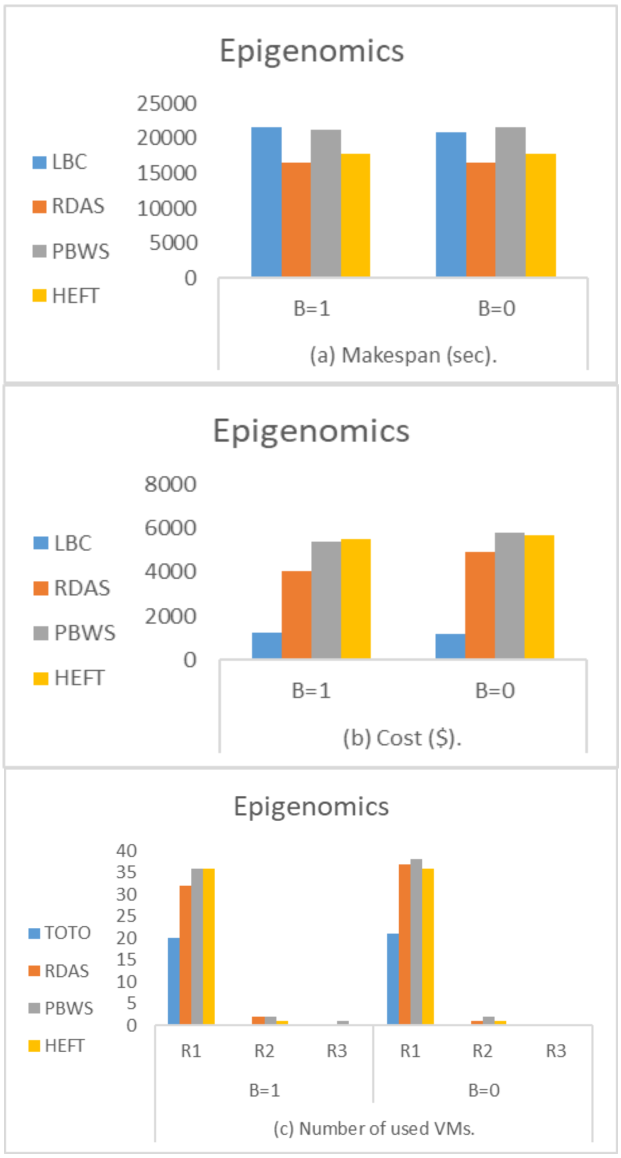


Fig. 4: Epigenomics workflows

SIPHT: The results for the Sipht workflows experiments is shown in Figure 5. The results show that the LBC algorithm significantly outperforms the other algorithms in terms of cost and makespan. This behavior is also due to the structure of the workflow. Sipht workflow has unbalanced structure where most of the levels have different number of tasks. This results in having clusters with different computational requirements. Such structure underlines the benefits of the LBC algorithm that treat the clusters as isolated objects.

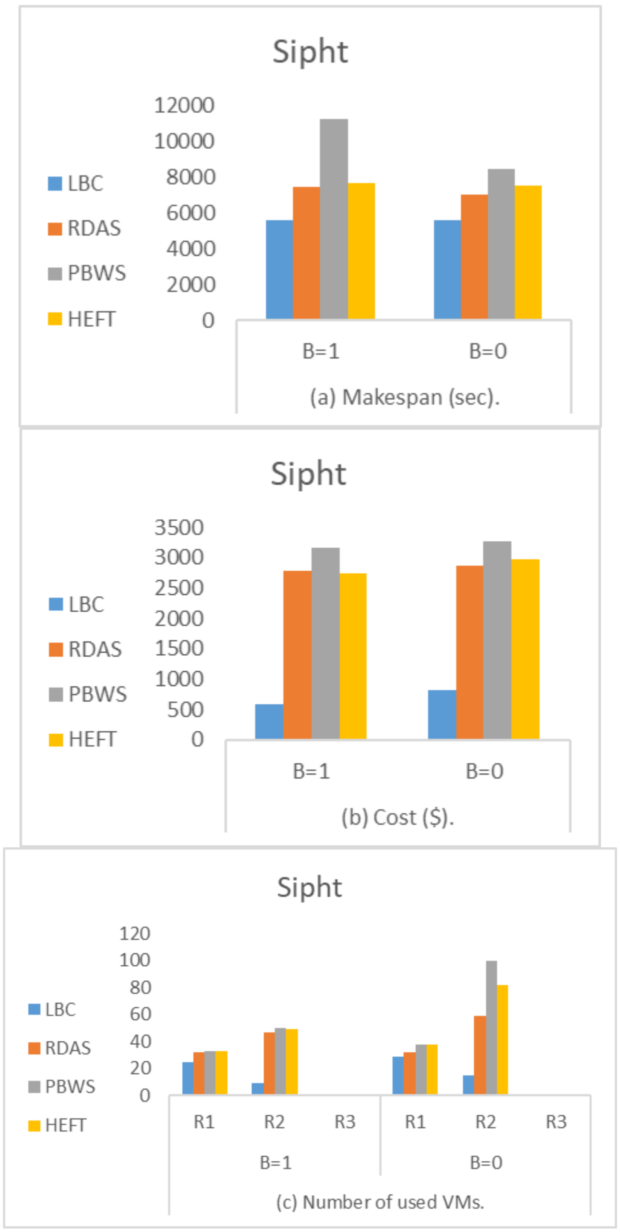
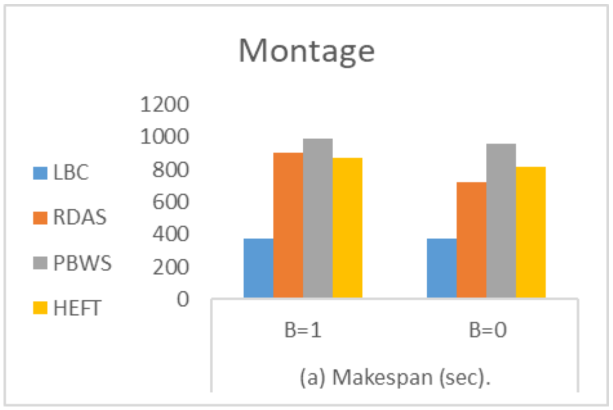


Fig. 5: Sipht workflows

Montage: From the results (Figure 6), we can see that in terms of makespan, the LBC algorithm achieves the best behavior. However, as we also can see, the LBC algorithm achieves the highest cost. In Montage workflow, most of the levels has small number of tasks. Combine this to the mechanism of isolating the level, results in increasing the expected delay in execution, and thus increasing the cost.



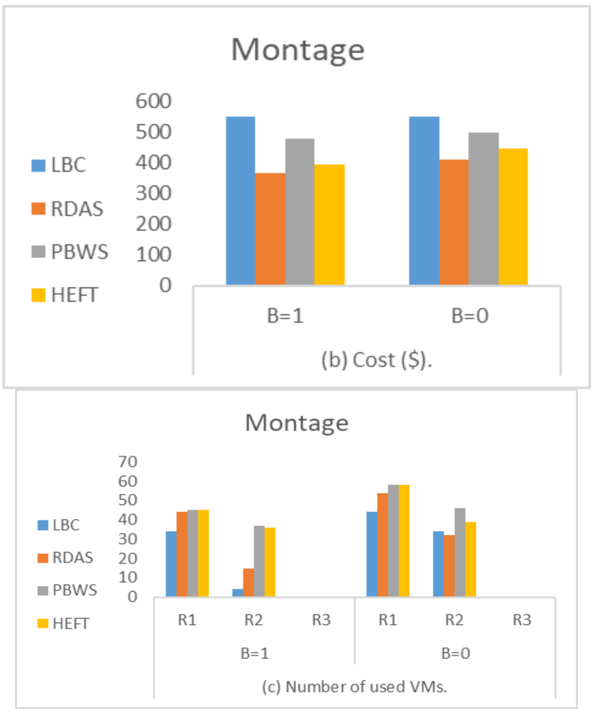


Fig. 6: Montage workflows

6. Conclusion

In this paper, we address the problem of scheduling scientific workflow on cloud such that the utilization of resources is maximized. to address this problem, we presented anew algorithm known as a level-based clustering algorithm. the proposed algorithm uses divide-and-conquer approach that treat each level task as an isolated object. we compared the performance of our algorithm against three well-known algorithms from the literature. the results show that in most situations, our approach significantly outperforms the other algorithms in term of cost, makespan and the number of resources used.

References

[1] K. Almi'Ani & Y. C. Lee, "Partitioning-Based Workflow Scheduling in Clouds", Proceedings of the IEEE International Conference Advanced Information Networking and Application, (2016), pp. 645–652.

[2] K. Almi'Ani, Y. C. Lee, & B. Mans, "Resource Demand Aware Scheduling for Workflows in Clouds", Proceedings of the IEEE 16th International Symposium on Network Computing and Applications, (2017), pp. 1-5.

[3] H. Topcuoglu, S. Hariri, & M. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing", IEEE Transactions on Parallel Distributed System, 13(30, (2002), 260–274.

[4] S. Abrishami, M. Naghibzadeh, & D. H. Epema. "Deadline-Constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds", Future Generation Computer Systems, 29(1), (2013), 158-169.

[5] P. Kaur & S. Mehta, "Resource Provisioning and Work Flow Scheduling in Clouds Using Augmented Shuffled Frog Leaping Algorithm", Journal of Parallel and Distributed Computing, 101, (2016), 41-50.

[6] D. Poola, S. K. Garg, R. Buyya, Y. Yang, & K. Ramamohanarao, "Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds", Proceedings of the IEEE 28th International Conference on Advanced Information Networking and Applications, (2014), pp. 858- 865.

[7] V. Singh, I. Gupta, & P. K. Jana, "A Novel Cost-Efficient Approach for Deadline-Constrained Workflow Scheduling by Dynamic Provisioning of Resources", Future Generation Computer Systems, 79, (2018), 95–110.

[8] T. Dziok, K. Figiela, & M. Malawski, "Adaptive Multi-Level Workflow Scheduling With Uncertain Task Estimates", Lecture Notes in Computer Science: Parallel Processing and Applied Mathematics, 9574, (2016), 90–100.

[9] V. Arabnejad, K. Bubendorfer, & B. Ng, "Budget and Deadline Aware e-Science Workflow Scheduling in Clouds", IEEE Transactions on Parallel and Distributed Systems, 2018, (2018), 1-10.

[10] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, & K. Vahi, "Characterizing and Profiling Scientific Workflows", Future Generation Computer Systems, 29, (2013), 682–692.

[11] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, & K. Vahi, "Characterization of Scientific Workflows", Proceedings of the IEEE Third Workshop on Workflows in Support of Large-Scale Science, (2008), pp. 1-10.

[12] J. D. Ullman, "NP-Complete Scheduling Problems", Journal of Compute and System Sciences, 10, (1975), 384-393.

[13] S. Liu, K. Ren, K. Deng, & J. Song, "A Task Backfill Based Scientific Workflow Scheduling Strategy on Cloud Platform," Proceedings of the IEEE Sixth International Conference on Information Science and Technology, (2016), pp. 105-110.

[14] Y. C. Lee & A. Y. Zomaya, "Stretch Out and Compact Workflow Scheduling with Resource Abundance", Proceedings of the 13th IEEE/ACM International Symposium on Cluster Cloud and Grid Computing, (2013), pp. 219-226.

[15] C. Q. Wu, X. Lin, D. Yu, W. Xu, & L. Li, "End-to-End Delay Minimization for Scientific Workflows in Clouds under Budget Constraint", IEEE Transactions on Cloud Computing, 3(2), (2015), 169-181.

[16] M. A. Rodriguez & R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds", IEEE Transactions on Cloud Computing, 2(2), (2014), 222 – 235.

[17] J. J. Durillo, H. M. Fard, & R. Prodan, "MOHEFT: A Multi-Objective List-Based Method for Workflow Scheduling", Proceedings of the IEEE 4th International Conference Cloud Computing Technology and Science, (2012), pp. 185-192.

[18] R. Prodan & M. Wieczorek, "Bi-Criteria Scheduling of Scientific Grid Workflows", IEEE Transactions on Automation Science and Engineering, 7, (2010), 7(2), 364-376.

[19] M. Malawski, K. Figiela, M. Bubak, E. Deelman, & J. Nabrzyski, "Cost Optimization of Execution of Multi-level Deadline-Constrained Scientific Workflows on Clouds", Proceedings of the IEEE International Conference on Parallel Processing and Applied Mathematics, (2013), pp. .

[20] M. Mao & M. Humphrey, "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows", Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, (2011), pp. 251-260.

[21] H. Arabnejad & J. G. Barbosa, "A Budget Constrained Scheduling Algorithm for Workflow Applications", Journal of Grid Computing, 12(4), (2014), 665-679.

[22] Q. Jiang, Y. C. Lee & A. Y. Zomaya, "Executing Large Scale Scientific Workflow Ensembles in Public Clouds", Proceedings of the IEEE 44th International Conference on Parallel Processing, (2015), pp. 520-529.

[23] L. Zeng, B. Veeravalli, & X. Li, "Scalestar: Budget Conscious Scheduling Precedence-Constrained Many-Task Workflow Applications in Cloud", Proceedings of the IEEE 26th International Conference on Advanced Information Networking and Applications, (2012), pp. 534-541.

[24] J. Yu, R. Buyya, & C. K. Tham, "Cost-Based Scheduling of Scientific Workflow Applications on Utility Grids", Proceedings of the IEEE International Conference on e-Science and Grid Computing, (2005), pp. 1-9.

[25] A. Verma & S. Kaushal, "Cost Minimized PSO Based Workflow Scheduling Plan for Cloud Computing", I.J. Information Technology and Computer Science, 7, (2015), 37-43.

[26] A. Verma & S. Kaushal, "Budget Constrained Priority Based Genetic Algorithm for Workflow Scheduling in Cloud", Proceedings of the IET International Conference on Recent Trends in Information, Telecommunication and Computing, (2013), pp. 8-14.

[27] V. Arabnejad, K. Bubendorfer, & B. Ng, "Scheduling Deadline Constrained Scientific Workflows on Dynamically Provisioned Cloud Resources", Future Generation Computer Systems, 75, (2017), 348-364.

[28] M. R. Garey, R. L. Graham, & J. D. Ullman, "Worst-Case Analysis of Memory Allocation Algorithms", Proceedings of the 4th Annual ACM Symposium on the Theory of Computing, (1972), pp. 143-150.