MINOR PROJECT

"COMPARATIVE ANALYSIS OF WORKFLOW SCHEDULING ALGORITHMS IN CLOUD COMPUTING"

Submitted in partial fulfillment of the
Requirements for the award of

Degree of Bachelor of Technology
In
Computer Engineering (COE)
(Batch: 2014-2018)

Under the supervision of
Mr. JASRAJ MEENA
(Department of COE)

By:
RAHUL CHAUHAN (2K14/CO/090)
RAHUL KUMAR (2K14/CO/091)
RAJAT VERMA (2K14/CO/093)

To :

Department of Computer Engineering

Delhi Technological University

(Formerly Delhi College of Engineering)

Shahbad Daulatpur Bawana Road, Delhi – 110042

Table of Contents

DECLARATION

I hereby certify that the work which is presented in the Minor Project entitled **"COMPARATIVE ANALYSIS OF WORKFLOW SCHEDULING ALGORITHMS IN CLOUD COMPUTING"** in fulfillment of the requirement for the award of the Degree of Bachelor of Technology and submitted to the Department of Computer Engineering , Delhi Technological University (Formerly Delhi College Of Engineering), New Delhi is an authentic record of my own, carried out during a period from January 2017 to May 2017, under the supervision of **Mr. Jasraj Meena (COE Department).**

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

Signature

Rahul Chauhan (2K14/CO/090)

Rahul Kumar (2K14/CO/091)

Rajat Verma (2K14/CO/093)

SUPERVISOR CERTIFICATE

This is to certify that **RAHUL CHAUHAN (2K14/CO/090), RAHUL KUMAR (2K14/CO/091)** and **RAJAT VERMA (2K14/CO/093)** the bonafide students of **Bachelor of Technology in Computer Engineering** of **Delhi Technological University** (Formerly Delhi College Of Engineering), New Delhi of **2014–2018 batch** has completed their minor project entitled "**COMPARATIVE ANALYSIS OF WORKFLOW SCHEDULING ALGORITHMS IN CLOUD COMPUTING**" under the supervision of Mr. Jasraj Meena (COE Department).

It is further certified that the work done in this dissertation is a result of candidate's own efforts.

Date: _____

Mr. Jasraj Meena

COE Department
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad, Daulatpur, Bawana Road, Delhi – 110042

ACKNOWLEDGEMENT

"The successful completion of any task would be incomplete without accomplishing the people who made it all possible and whose constant guidance and encouragement secured us the success."

First of all, we are grateful to the Almighty for establishing us to complete this minor project. We are grateful to **Dr. Rajni Jindal, HOD (Department of Computer Science),** Delhi Technological University (Formerly Delhi College of Engineering), New Delhi and all other faculty members of our department, for their astute guidance, constant encouragement and sincere support for this project work.

We owe a debt of gratitude to our guide, **Mr. Jasraj Meena (COE Department)** for incorporating in us the idea of a creative Minor Project, helping us in undertaking this project and also for being there whenever we needed her assistance.

I also place on record, my sense of gratitude to one and all, who directly or indirectly have lent their helping hand in this venture. We feel proud and privileged in expressing my deep sense of gratitude to all those who have helped me in presenting this project.

Last but never the least, we thank our parents for always being with us, in every sense.

ABSTRACT

Cloud computing is a new terminology which is achieved by Distributed, Parallel and Grid computing. Clouds have progressed towards a service-oriented paradigm that enables a new way of service provisioning based on “pay-as-you-go” model. In such pay-as-you-go models, users are charged for services based on their usage and on the fulfillment of QoS constraints; execution time and cost are two common QoS requirements. Therefore, to produce effective scheduling maps, service pricing must be considered while optimising execution performance. Cloud computing provides different types of resources like hardware and software as services via internet. Users want to make use of these services to execute a workflow application, within a certain deadline or budget. Efficient task scheduling mechanism can meet users’ requirements, and improve the resource utilization, thereby enhancing the overall performance of the cloud computing environment.

A lot of work has been done on this research topic. We have worked on algorithms that minimize the execution time while meeting the budget constraints set by the user. In this report we have conducted a comparative analysis of workflow scheduling algorithms. The algorithms considered for the purpose of comparison are ScaleStar, Budget-constrained Heterogeneous Earliest Finish Time (BHEFT) and Heterogeneous Budget Constrained Scheduling (HBCS).

INTRODUCTION

Cloud creates an infrastructure for enabling users to consume services transparently over a secure, shared, scalable, sustainable and standard worldwide network environment. In addition to this flexibility, Cloud infrastructure services provide the illusion of limitless resource allocation for most users. Cloud Service Providers (CSPs) allow users to either allocate one such machine for many hours or many machines for one hour, while the costs remain the same. Without paying a premium for large scale, this resource elasticity is unprecedented in the history of IT.

The various commercial offers differ not only in pricing but also in the types of machines that can be allocated. For example, for Amazon's EC2, the choice is between "Small", "Large", "Extra Large", "High CPU (Medium)", and "High CPU (Large)" instance types, all coming in two to three flavours. The efficient use of these payment-services is a crucial issue because different type machines may have different rent (pricing). Moreover, different services, belonging to different CSPs, may have different policies for charging. Users would like to pay a price

which is commensurate the budget they can get. However, currently, the problem of allocating the right number of machines, of the right type, for the right time frame, strongly depends on the application program, and is left to the user.

Many complex applications in e-science and e-business can be modelled as workflows[16]. A popular representation of a workflow application is the Directed Acyclic Graph (DAG), in which nodes represent individual application tasks, and the directed edges represent inter-task data dependencies. Many workflow scheduling algorithms have been developed to execute workflow applications.

In general, processing time and economic cost are two typical constraints for MTW (Many-Task Workflow) execution on Cloud services. In this paper, we compared budget conscious schedulers, which minimize MTW execution time within a certain budget. The MTW scheduling approach can be used by both end-users and CSPs. End users can use the approach to orchestrated Cloud services, whereas CSPs can outsource computing resources to meet customers' service-level requirements.

SCHEDULING ALGORITHMS

System Model[3]

A typical workflow application can be represented by a Directed Acyclic Graph (DAG), which is a directed graph with no cycles. In a DAG, an individual task and its dependencies are represented by a node and its edges, respectively. A dependency ensures that a child node cannot be executed before its entire parent tasks finish successfully and transfer the input data required by the child. The task computation times and communication times are modelled by assigning weights to nodes and edges respectively.

A DAG can be modelled by a tuple $G(N,E)$, where N is the set of n nodes, each node $ni \in N$ represents an application task, and E is the set of communication edges between tasks. Each edge $e(i, j) \in E$ represents a task-dependency constraint such that task ni should complete its execution before task nj can start. In a given DAG, a task with no predecessors is called an *entry task*, and a task with no successors is called an *exit task*. It is assumed that the DAG has exactly one entry task n_{entry} and one exit task

n_{exit}. If a DAG has multiple entry or exit tasks, a dummy entry or exit task with zero weight and zero communication edges is added to the graph.

The edges of the DAG represent a communication cost in terms of time, but they are considered to have zero monetary cost because they occur inside a given site. The schedule length of a DAG, also called Makespan, denotes the finish time of the last task in the scheduled DAG, and is defined by:

$makespan = max\{AFT\ (nexit\ )\}$

where *AFT (nexit)* denotes the *Actual Finish Time* of the exit node.

1. ScaleStar Scheduling Algorithm[1]

ScaleStar assigns the selected task to a virtual machine with higher *comparative advantage* which effectively balances the execution time-and-monetary cost goals. In addition, according to the actual charging model, an adjustment policy, refer to as *DeSlack*, is proposed to remove part of slack without adversely affecting the overall makespan and the total monetary cost.

The ScaleStar algorithm is composed of three phases:

A. Task priorities and initial assignment phase

For a given task *ni*, the *CA*1 value of an assignment of VM *vj* (with *pj*) with the best assignment of *v'* (with *p'*) is defined as:

$$CA1(n_i, v_j, v') = \frac{cost(n_i, v') - cost(n_i, v_j)}{cost(n_i, v_j)} + \frac{eft(n_i, v') - eft(n_i, v_j)}{eft(n_i, v_j) - est(n_i, v_j)}$$

B. Task re-assignment phase

CA2 is computed using the following expression given below:

$$CA2(n_i, v_j) = \frac{makespan_{new} - makespan_{old}}{cost_{old} - cost_{new}}$$

The algorithm keeps trying reassignment by considering the smallest or biggest values of the *CA*2 for all tasks and VMs.

C. DeSlack *policy*

The key idea of *DeSlack* is to recompute the starting time of each node in the schedule *S* and the slack, in order to make a decision for adjustment. The input of this policy is a schedule *S*. The objective of the policy is to partially eliminate the idle times of the schedule while minimizing the monetary cost.

The ScaleStar Heuristic

Input: A DAG G = (N, E) with task execution time and communication. A set V of *V* VMs with cost of executing tasks and available Budget *B*.
Output: A DAG schedule *S* of G onto V.

1. Compute *b-level* of $\forall ni \in$ N, sort N in decreasing order by *b-level* value
2. Build an array *A*[*N*][*V*], where *N* is the number of tasks, *V* is the number of VMs
3. For $\forall ni \in$ N and $\forall vj \in$ V
    Compute *CA*1 (*ni*, *vj*, *v*') and *CA*1 (*ni*, *v*' , *vj*) value with *v*' use Eq.(9)
    If *CA*1(*ni, vj, v'*) > *CA*1(*ni, v', vj*), Replace *v'* with *vj* , assign *ni* on *v'*
4. Compute *makespanold* and *costold* according to *S*
5. For $\forall ni \in$ N and $\forall vj \in$ V
    If task *ni* is assigned to VM *vj* , *A*[*i*][*j*] = 0
    Else
        Compute *makespannew* assume *ni* is assigned to VM *vj*
        Compute *costnew* assume *ni* is assigned to VM *vj*
        A[*i*][*j*] = *CA*2(*i*, *j*)
6. Assign all the non-zero values of *A* to a set A
7. Sort *A* in increasing order by *CA*2 value
8. *cost* = the current total execution monetary cost
9. While (A is not empty)
    If (*cost* > *B*), get the first element value *A*[*i*][*j*]
        (and remove it) from A
    If (*cost* ≤ *B*), get the last element value *A*[*i*][*j*]
        (and remove it) from A
    Reassign task *i* to VM *j* and calculate new monetary cost of schedule *S*
    If (*cost* > *B*), invalidate previous reassignment for
        task *i* to VM *j*
10. If (*cost* > *B*), use cheapest assignment for *S*
11. Call *DeSlack* policy

DeSlack Policy

1. Except the entry task and the exit task, compute *eft* and *est* for each task and each "idle" according to the schedule *S*;
2. Except the entry task and the exit task, find the first tasks or the last tasks, form a *Candidatetask* set, and sort *Candidatetask* in decreasing order by the length of execution time (*efttask* − *esttask*);
3. Selected all "idle" and form a *Candidateidle* set, and sort *Candidateidle* in increasing order by the length of idle times (*eftidle* − *estidle*);
4. Compute the total monetary cost, *costold*;
5. while (*Candidatetask* is not empty) do *{*
    n ← first task (node) in *Candidatetask*;
    while (*Candidateidle* is not empty) do *{*
        idl ← first "idle" in *Candidateidle*;
        Compute the total monetary cost, *costnew*, assume task *n* isassigned to a same VM (for a idle *idl*);
        if (*estn* ≥ *estidl*, *ωidl* ≥ *ωn*, *ωidl* − *ωn* is the minimum value, and *costnew* ≤ *costold*) *{*
            Task *n* is allocated to the VM where the idle *idl* is, go to (1); *}*
        Eliminate the idle *idl* from the *Candidateidle*; *}*
    Remove task *n* from the *Candidatetask*; *}*

2. BHEFT Scheduling Algorithm[2]

BHEFT is based on the Heterogeneous Earliest Finish Time (HEFT) algorithm[4], which is a well-known list scheduling heuristic aiming at minimizing the overall execution time of a DAG application in a heterogeneous environment. While being effective at optimizing makespan, the HEFT algorithm does not consider the monetary cost and budget constraint when making scheduling decisions. In BHEFT, the HEFT algorithm is extended in order to resolve the BDC(Budget Deadline Constraint)-planning problem and the new algorithm is called the *Budget-constrained Heterogeneous Earliest Finish Time (BHEFT)*[2].

HEFT Algorithm[4]

The HEFT algorithm selects the task with the highest upward rank value at each step and assigns the selected task to the processor, which minimize the earliest finish time with an insertion-based approach.

1. Set the computation cost of tasks and communication costs of edges with mean values.

2. Compute upward rank for all tasks by traversing graph upward, starting from exit task.

3. Sort the tasks in scheduling list by non-increasing order of rank values.

4. while there is unscheduled task in list do

5. select the first task ni from the list

6. for each processors do

7. compute EFT(ni,pk)

8. Assign task ni to pk which minimise EFT of task ni.

9. end while

The BHEFT heuristic

Input: DAG G with budget B and deadline D.

Output: A BDC-plan

1. Compute rank for all tasks.

2. Sort all tasks in a planning list in decreasing order of rank.

3. for k=0 to n do

4. select kth task from the planning list

5. compute the spare application budget for task k

6. compute the current task budget for task k

7. construct the set of affordable services for task k

8. for each service which can be used by task k do
9. compute the EFT of mapping k to the service using TSQ
10. endfor
11. select a service for task k according to defined selection rules.
12. endfor

3. HBCS Scheduling Algorithm[3]

A Heterogeneous Budget Constrained Scheduling (HBCS) algorithm that guarantees an execution cost within the user's specified budget and that minimises the execution time of the user's application.

The results showed that algorithm achieves lower makespans, with a guaranteed cost per application and with a lower time complexity than other budget-constrained state-of-the-art algorithms. The improvements are particularly high for more heterogeneous systems, in which a reduction of 30 % in execution time was achieved while maintaining the same budget level.

HBCS consists of two phases, namely a *task selection* phase and a *processor selection* phase.

A. Task Selection

Tasks are selected according to their priorities. To assign a priority to a task in the DAG, the upward rank ($rank_u$) is computed. To prioritize tasks, it is common to consider average values because they must be assigned a priority before the location where they will run is known.

B. Processor Selection

The processor selection phase is guided by the following quantities related to cost. The Remaining Cheapest Budget (*RCB*) as the remaining *CheapestCost* for unscheduled tasks, excluding the current task, and the Remaining Budget (*RB*) as the actual remaining budget are defined. *RCB* is updated at each step before executing the processor selection block for the current task, which represents the lowest possible cost of the remaining tasks:

$$RCB = RCB - Cost_{lowest}$$

where $Cost_{lowest}$ is the lowest cost for the current task among all of the processors.

The HBCS Heuristic

Input: DAG and user defined BUDGET

1: Schedule DAG with HEFT and Cheapest algorithm

2: Set task priorities with $rank_u$

3: **if** *HEFTcost* < *BUDGET*

4: **return** Schedule Map assignment by HEFT

5: **end if**

6: *RB* = *BUDGET* and *RCB* = *CheapestCost*

7: **while** there is an unscheduled task **do**

8: *ni* = the next ready task with highest *ranku* value

9: Update the Remaining Cheapest Budget (*RCB*)

10: **for all** Processor *pi* ∈ *P* **do**

11: calculate *FT(ni, pj)* and *Cost (ni, pj)*

12: **end for**

13: Compute *CostCoeff*

14: **for all** Processor *pi* ∈ *P* **do**

15: calculate *worthiness(ni, pi)*

16: **end for**

17: *Psel* = Processor *pi* with highest *worthiness* value

18: Assign Task *ni* to Processor *Psel*

19: Update the Remaining Budget (*RB*)

20: **end while**

21: **return** Schedule Map

Experimental Results

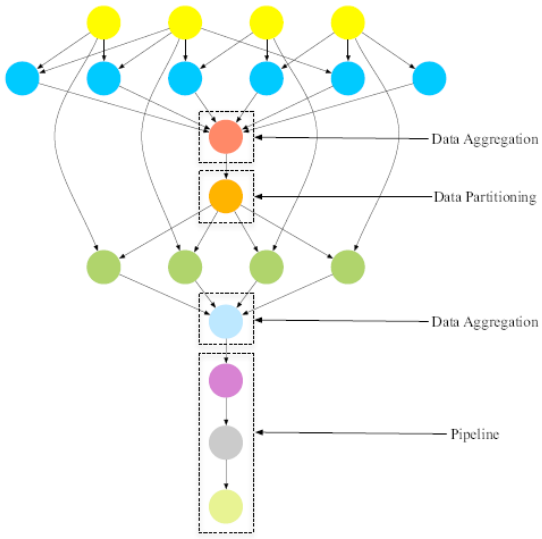
In this section we do the comparison of above Workflow Scheduling Algorithms, namely ScaleStar, BHEFT and HBCS.

The following datasets were used for comparison of the above scheduling algorithms:

A. Montage

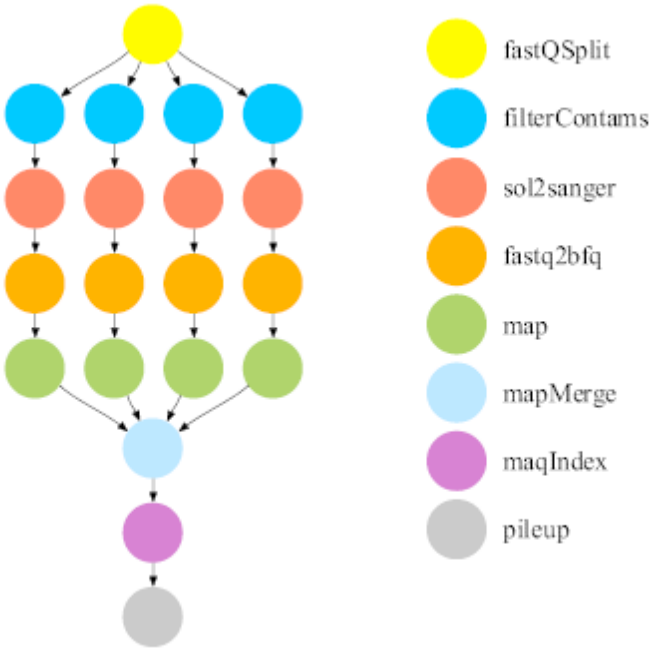
Montage[5] was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to generate custom mosaics of the sky using input images in the Flexible Image Transport System (FITS) format.

The Montage application has been represented as a workflow that can be executed in Grid environments such as the TeraGrid[6].



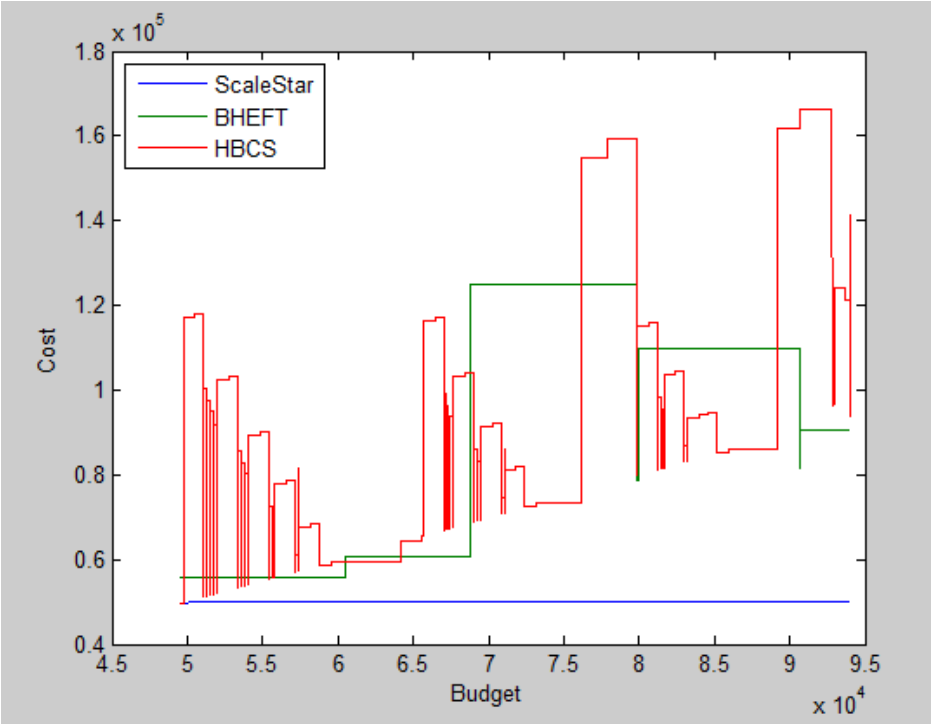
B. Epigenomics[16]

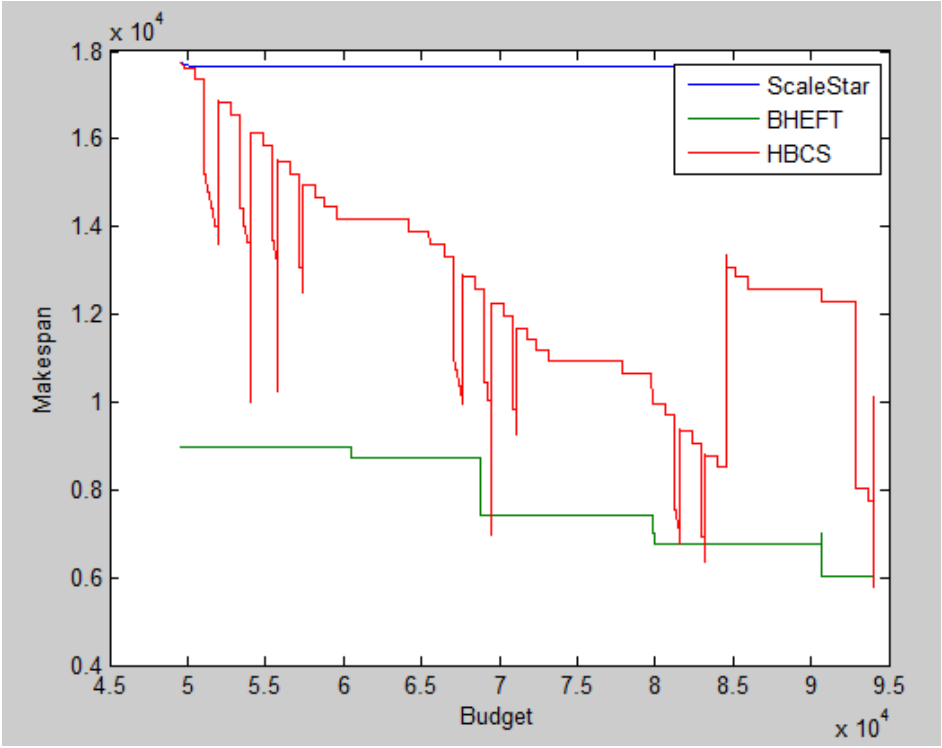
The USC Epigenome Center is currently involved in the mapping of the epigenetic state of human cells on a genome-wide scale. This workflow is being used by the Epigenome Center in the processing of production DNA methylation and histone modification data.



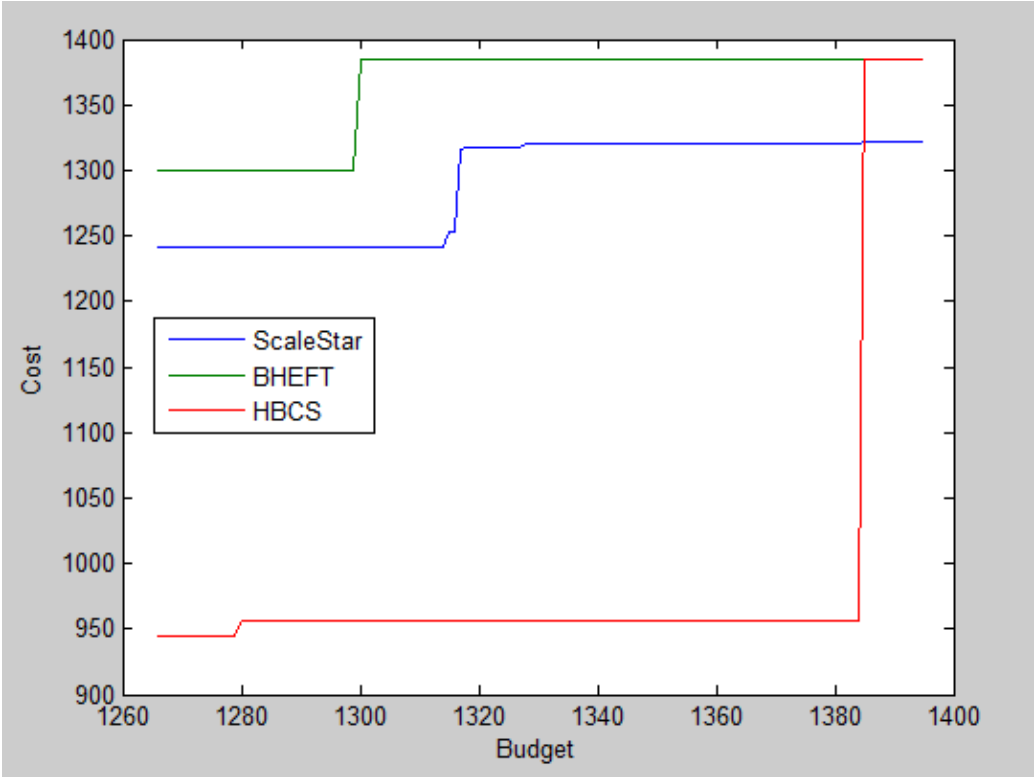
Graphical Analysis

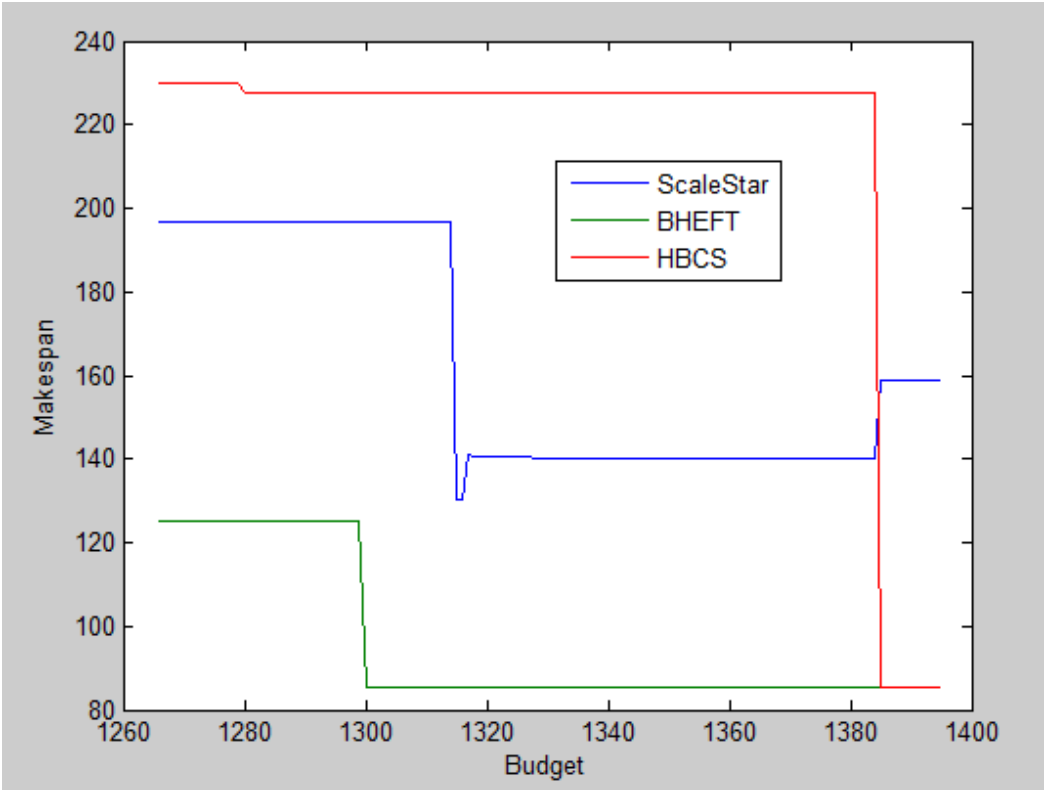
For Epigenomics:





For montage:


1400
1350
1300
1250
1200
1150
1100
1050
1000
950
900
Cost
ScaleStar
BHEFT
HBCS
1260
1280
1300
1320
1340
1360
1380
1400
Budget


240
220
200
180
160
140
120
100
80
Makespan
ScaleStar
BHEFT
HBCS
1260
1280
1300
1320
1340
1360
1380
1400
Budget

CONCLUSION

In this report we performed a comparative analysis of three well known algorithms namely ScaleStar, BHEFT and HBCS for workflow scheduling in the cloud environment. Our study makes an important contribution in timely and useful handling of scientific workflows on clouds. Our extensive simulation shows the comparison of the algorithms on two QoS parameters namely makespan and cost of scheduling.

According to our analysis we see that HBCS algorithm provides the maximum variation in cost and makespan when the budget is varied. Whereas BHEFT produces a moderate variation in results and ScaleStar produces almost no variation in results. For the epigenomics dataset, BHEFT produces schedules that have lowest makespans. Also in case of montage dataset we see that BHEFT produces schedules that have lowest makespans. Hence we can say that BHEFT outperforms the other two scheduling algorithms.

References

[1] Lingfang Zeng, Bhardway Veeravalli "ScaleStar: Budget Conscious Scheduling Precedence-Constrained Many-task Workflow Applications in Cloud"

[2] Wei Zheng, Rizos Sakellariou "Budget-Deadline Constrained Workflow Planning for Admission Control"

[3] Hamid Arabnejad, Jorge G. Barbosa "A Budget Constrained Scheduling Algorithm for Workflow Applications"

[4] Topcuoglu, H., Hariri, S., Wu, M. "Performance effective and low-complexity task scheduling for heterogeneous computing."

[5] "Montage: An astronomical image engine." [Online]. Available: http://montage.ipac.caltech.edu

[6] G. B. Berriman, E. Deelman, J. Good, J. Jacob, D. S. Katz, C. Kesselman, A. Laity, T. A. Prince, G. Singh, and M. Su, "Montage: A grid enabled engine for delivering custom science-grade mosaics on demand," in SPIE, 2004.

[7] Bittencourt, L.F., Madeira, E.R.M.: Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. J. Internet Serv. Appl. **2**(3), 207–227 (2011)

[8] Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: First International Conference on e-Science and Grid Computing, 2005, pp. 8–pp. IEEE (2005)

[9] Chen, W.-N., Zhang, J.: An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **39**(1), 29–43 (2009)

[10] Prodan, R., Wieczorek, M.: Bi-criteria scheduling of scientific grid workflows. IEEE Trans. Autom. Sci. Eng. **7**(2), 364–376 (2010)

[11] Wieczorek, M., Podlipnig, S., Prodan, R., Fahringer, T.:Bi-criteria scheduling of scientific workflows for the grid. In: 8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID'08, pp. 9–16. IEEE (2008)

[12] Yu, J., Buyya, R.: A budget constrained scheduling of workflow applications on utility grids using genetic

algorithms. In: Workshop on Workflows in Support of Large-Scale Science, 2006. WORKS'06, pp. 1–10. IEEE (2006)

[13] Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**(3), 217–230 (2006)

[14] Byun, E.-K., Kee, Y.-S., Kim, J.-S., Deelman, E., Maeng, S.: Bts: Resource capacity estimate for time-targeted science workflows. J. Parallel Dist. Comput. **71**(6), 848–862 (2011)

[15] Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.: Scheduling workflows with budget constraints. Integr. Res. Grid Comput., 189–202 (2007)

[16] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, Karan Vahi "Characterizing and Profiling Scientific Workflows".