SLOPE: A Self Learning Optimization and Prediction Ensembler for Task Scheduling

Lohit Kapoor*, Anish Jindal†, *Student Member, IEEE,* Abderrahim Benslimane‖, *Senior Member, IEEE,*
Gagangeet Singh Aujla‡, *Member, IEEE,* Rajat Chaudhary§, *Student Member, IEEE,*
Neeraj Kumar¶, *Senior Member, IEEE,* Albert Y. Zomaya**, *Fellow, IEEE,*
* Sreenidhi Institute of Science and Technology, Hyderabad, Telangana, India
(e-mail: lohit.kapoor@gmail.com).
†‡§¶ Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala (Punjab), India
(e-mail: anishjindal90@gmail.com, rajatlibran@gmail.com, neeraj.kumar@thapar.edu).
‖ University of Avignon, France
(e-mail: abderrahim.benslimane@univ-avignon.fr).
‡Computer Science & Engineering Department, Chandigarh University, Gharuan (Punjab), India
(gagi_aujla82@yahoo.com)
** School of Information Technologies, J12, University of Sydney, NSW 2006, Australia
(e-mail: albert.zomaya@sydney.edu.au.)

***Abstract*—In a multi-cloud environment, consumers can access multiple cloud services using a single heterogeneous computing architecture. In such an environment, multiple instances of the same cloud service and its component may be geographically dispersed. So, cloud service broker (CSB) exploits the heterogeneity of multi-cloud environment to provide high performance at a low price to its consumers. The consumer tasks are allocated to the geo-dispersed cloud service components for execution of various services. For this purpose, an optimal service components identification and task allocation are major concerns keeping in view of the heterogeneity in multi-cloud environment. For this purpose, a scheduling algorithm, which takes care of location, price, and performance is required. Therefore, in this paper, *SLOPE*: *A Self Learning Optimization and Prediction Ensembler* for Task Scheduling in Multi-cloud Environment is proposed. *SLOPE* works in two phases, 1) In first phase, Bayes theorem is used to design a self-learning algorithm, to compute the conditional probability (strength) of each service component in order to select the probable rule string, and 2) a roulette wheel method is used to select an optimal scheduling policy for a given service request. *SLOPE* helps to identify the best possible service component from the pool of resources on the basis of dynamic factors and then schedule a service request to the selected component. Unlike most of the other existing approaches, *SLOPE* builds an efficient schedule for service selection. Experimental results demonstrate that *SLOPE* performs better in comparison to other competing schemes of its category.**

***Index Terms*—Bayes Theorem, Cloud Service Broker, Multi-cloud Environment, Roulette Wheel Method, Task Scheduling.**

I. Introduction

Cloud computing (CC) provides on-demand service delivery using a pay-as-you-go model, wherein, end user pay for these services, which they use. However, providing low latency and high data rate services without any disruption is a challenging task with the generation of big data from smart devices. Nowadays, multiple cloud services can be provisioned using a single heterogeneous computing architecture in a multi-cloud environment [1]. For example, Amazon, Google and Microsoft provides large-scale big data processing services using geo-located data centers (DCs) [2]. In a multi-cloud environment, a third party entity like, cloud services broker (CSB) acts as an intermediary between different cloud service providers (CSPs) to provide cloud services to the end users by utilizing the infrastructure of multiple clouds [3]. CSB is responsible for distribution of cloud services over several cloud hosting environments and then assisting the end users to select the best available services from a pool of distributed services. For this purpose, it gathers all the necessary information of multiple clouds existing in the market to help the user to deploy its applications in a cost-effective manner. Moreover, CSBs also provides the facility of migration and portability of cloud services [4], which helps the end users to avail the freedom of choosing multiple service options.

A research report by Gartner [5] state that CSB is divided into four categories, i.e., aggregation (for secure portability), arbitrage (best available resource selection), integration (interlinking between vendors) and intermediation (identity and access management). Using these categories, a CSB provides information about the service features, cost breakdowns and service level of agreements (SLAs) to the consumer. Using this information, the consumer can evaluate and select an appropriate service platform from a set of different CSPs. In some case, CSBs may even provide contracts on behalf of the CSPs. For such cases, CSBs are authorized to negotiate the service contracts with the clients. This helps to reduce the operational costs of cloud services. Moreover, CSBs have pre-existing relationships with the vendors, which helps to speed up the vendor acquisition process. This helps to reduce the redundancies and optimize the resource utilization, which in turn allow the organizations to control the operational costs.

Among a number of functionalities of CSB, in this paper, a scenario where a CSP uses CSB to execute the user request is considered. To achieve this objective, it may require multiple service components, which may be located in a single or multiple DCs managed by different CSPs [6]. From CSP point of view [7], CSB works as an intermediary entity, which integrates multiple geo-located cloud components and presents them as a single entity to the customers. The major function of the CSB is to reduce the barriers for adopting and managing different service components, i.e., each service component with multiple instances is deployed in a multi-cloud environment. For this purpose, the service components are assumed as a set of resources and each user request is considered as a set of tasks to be executed by these resources, thereby, making it a task scheduling problem. Task scheduling schemes adopted by CSB in multi-cloud environment can be categorized as; functional and non-functional [8]. In functional approaches, CSB deals with the SLA parameters such as-response time, makespan and execution time. On the other hand, in non-functional approaches, parameters like price are considered by the scheduling algorithms.

The methods dealing with functional parameter require an exact matching, but in non-functional methods, the SLA changes with the price. Therefore, a trade-off between quality and price is required in such cases. For example, Fard *et al.* [1], presented an auction-based system, where game theory is used to schedule workflow in a multi-cloud environment. The proposed workflow scheme achieves reductions in cost and makespan. However, the authors have not considered data-locality support in the proposed scheme. To handle this issues, Yuan *et al.* [9] proposed a *k*-means clustering-based strategy to locate and place data, thereby, minimizes the data movement. The proposed strategy increases the efficiency, however, it lacks in cost effectiveness. Authors in [10] compared three dynamic algorithms such as-minimum completion cloud (MCC), MEdian MAX and Cloud Min-Max normalization on the basis of functional parameters. However, the authors do not consider the non-functional parameters. Similarly, Qian and Xia [11] proposed a stochastic theory based scheduling model to serve various functional parameters using Bayesian networks. Furthermore, they provided a technique for scheduling and matching tasks to the service copy. However cost of the service was not included as a parameter. Er-Dum *et al.* [12] presented a genetic algorithm-based data placement and load balancing strategy, which reduces the movement of data across the nodes. The proposed strategy performs better in terms of performance and cost efficiency.

After analysis of the above discussed proposals, *SLOPE*: *A Self Learning Optimization and Prediction Ensembler* for Task Scheduling in Multi-cloud Environment is proposed, which considers functional and non-functional scenarios, wherein, the considered pricing conditions are dynamic.

A. *Contributions*

Following are the major contributions of the paper.

- SLOPE is designed in which Bayes theorem is used to calculate the conditional probability of each service component. This helps to identify the best possible service component from the pool of resources on the basis of dynamic factors such as-latency, flash crowd and SLA.
- After identifying the best suitable service component, it populates the scheduler with multiple scheduling rules. Then, by using the roulette wheel method, an optimal scheduling rule is selected for a given service request.

II. Problem Formulation

A service has many components and each component has multiple instances, which may be deployed on different cloud servers. In order to execute a job (which is a collection of multiple tasks), the CSB chooses a service component among a pool of similar components. However, it is a complicated process and depends on two factors, a) SLA management, i.e., no or minimal SLA violations, and b) Profit maximization, i.e., choosing the cheapest service component. Keeping these factors in mind, let us consider a service (*S*) comprising of different service components *C* = {*C1, C2,..., Cx*}, which are deployed in multiple DCs belonging to multiple cloud vendors. Therefore, in such a multi-cloud environment the services component allocation policy may be defined as a complete directed graph *G*, where *N* represents the services set and *A* represents the arc set between services components. Formally, *G = (N, A)*, where $N = \{0, 1, ..., n\}$ and $A = \{i, j\}|$ s.t. $i \neq j$; $i, j \in N$. Each Job J (service request) from a user U is divided into multiple sub-tasks $T = \{t1, t2, t3, ..., tn\}$, which can be executed by a relevant service component, i.e, $t1 \rightarrow C1, t2 \rightarrow C2, ..., tn \rightarrow Cx$. Therefore, allocation of service components should consider following criteria:

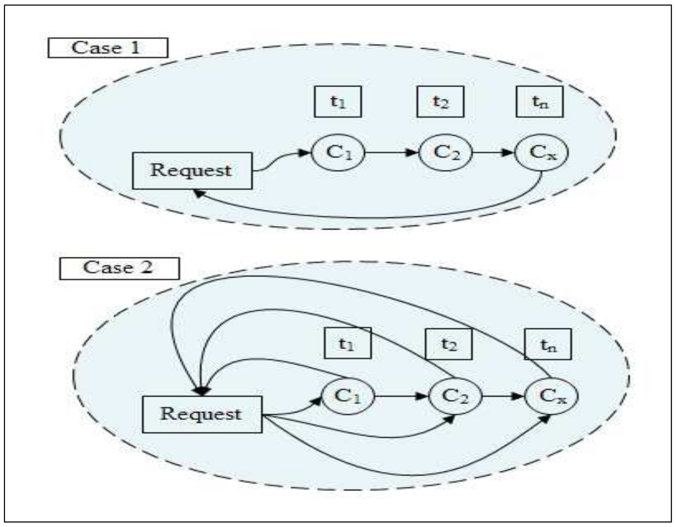


Fig. 1. Task allocation to service components.

- Consider a case (Case 1), where $C2$ depends on the output of C. Similarly, Cx depends on the output of

$Cx-1$, while in another case (Case 2), $C1, C2, C3$ executes each sub-task in parallel as shown in Fig. 1. In both the cases, identification of best Cx is required to execute an associated task.

- For each service component pair, $(C1, C2) \in C, arc(i,j) \in A$, present in the connection, the latency between them is a time varying function as it depends on the traffic of user requests. Here, the time slots or planning horizon, H is divided into a set of periods $M = \{[b_1, e_1]k_1, [b_2, e_2]k_2, ..., [b_m, e_m]k_m\}$, where b_m and e_m are the starting time and ending time, respectively. Under time-varying conditions, $l_{ijk} \in H$ is the latency between (i,j) during the period $k \in M$ and proposed scheme should identify the right arc, i.e., $min(l_{ijk})$.
- $\forall Cx \in C$, an execution time, y_k is associated, where y is the execution time of a sub-task during period k. Moreover, for each Cx deployed in different DCs having heterogeneous execution time, the proposed scheme should allocate the sub-task to a Cx considering $min(y_k)$.
- Consider $P = \{p_1, p_2, p_3, ..p_x\}$ is the set of all the prices $\forall Cx \in C$ with a price p_x during the period $k \in M$. The identification of any $Cx \rightarrow min(p_x)$ adds non-functionality to develop a scheduling strategy.

Following assumptions are considered in *SLOPE*:

1) For every service request, there is job $|Jj|)$ and all the jobs constitute same number of tasks.
2) There is no user-defined restriction related to the location of the service component.

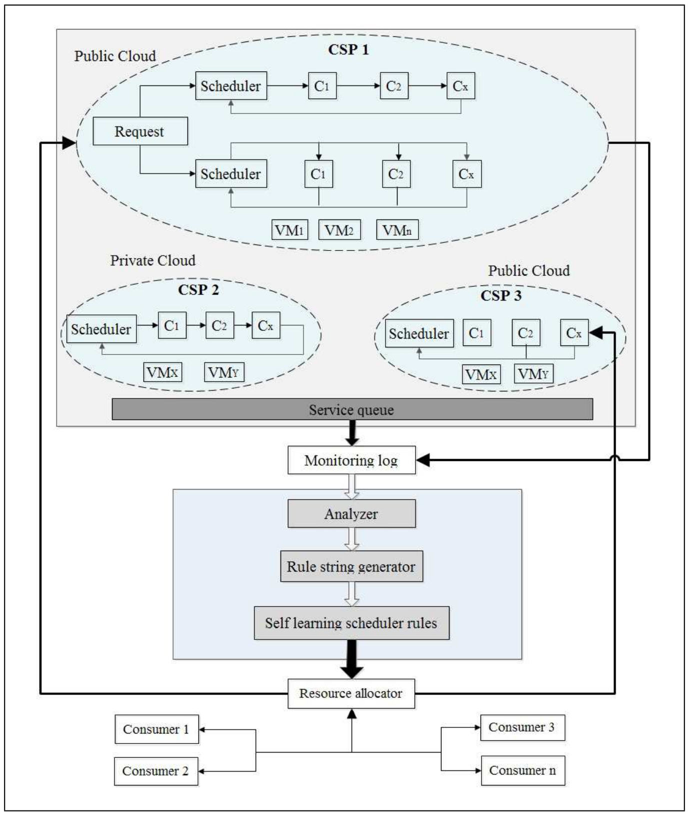


Fig. 2. System model

III. System Model

In the proposed system model, each *C* is associated with a queue, from where all the requests are matched to a *C*, which is selected on the basis of *SLOPE*. The service components allocation scenario in the multi-cloud environment is as shown in Fig. 2. A service request is given to resource allocator, where it is treated as a *task*, which is divided into multiple *sub-tasks*. Each sub-task is allocated to a service component hosted on different resources. The resource allocator uses the allocation policy, which is dynamic in nature. It uses scheduling rules, which are generated by rule string generator to schedule the tasks. Analyzer gathers all the relevant information of all the service components such as-latency, service components, and queue strength from monitoring log. Moreover, it also calculates the current status of each service component and passes the result to rule string generator. This is a continuous process and new rules are continuously generated on the occurrence of any performance degradation. So, this task allocation strategy is considered as an Integer Program as depicted below.

Indices:
$a = 1...x$ job index
$t = 1...v$ task index, $t \in a$
$s = 1...y$ task size, $s \in t$

A. Functional parameters

The functional parameters considered in *SLOPE* are described as below.

- **Latency (*l*)**: It is defined as time taken by a request to travel from one service component location to other in a particular time period *k*. It is represented as $l = \{l1, l2, l3, ..lz\}$. Here, every $l \in H$, uses a continuous probability distribution. l can be divided into three consecutive periods: (a) a congestion period (constant) with a steady latency l_c, (b) a transition period, l_t, and (c) a free-flow period (variable) with steady latency l_s. The time to reach from one service component to other is represented by an arc $arc(i \rightarrow j)$ with distance d is calculated by a prediction tree algorithm [13] and is represented as, $L(i,j)$. Based on the results obtained by the prediction tree algorithm, the probability density function (PDF) lies between $b_m, e_m, f(x)$ is used so that

$$P(t1 \leq L \leq t2) = \int_{em}^{dm} f(x)dx \tag{1}$$

The reason to use this predictive approach is to enable non-intrusive estimation, which helps in gathering the information in minimal time and efforts.

- **Queue strength** (β_{qc}): It is defined as the estimated waiting time $\forall\ t_s \in a$, where t_s is a task of size s. Consider t_x is the smallest task and E_x is the executing time taken by a Cx, then:

$$\beta_{qc} = E_x \times \sum_{i=1}^{len} si \in t \tag{2}$$

where, si is the size of the task t and len is the current length of the queue.
Here, the waiting of each task per unit time in each service component queue is calculated by using Poisson distribution, which is represented as below.

$$\beta qc(x) = \begin{matrix} 0, x < 0 \\ \frac{1}{\lambda^x e^{-x/\lambda}} \end{matrix} \tag{3}$$

with average waiting time tasks $\lambda \in M \notin \beta_{qc}$.

- **Execution time (*s*)**: It is the time taken by a service component to execute a task of size s. Here, the total time to execute a job a is represented as below.

$$F(a) = \sum_{t=1}^{v} \tau_{at} H \neq 0 \tag{4}$$

where, *I* is the number of tasks $\in a$.
Based on above mentioned parameters, the estimated time of computing (ETC) $\forall a$ is computed as below.

$$\sum_{t=1}^{v} L(i,j) + \beta_{qc} + F(a) \tag{5}$$

where, *v* is the number of tasks $\forall a$.
Moreover, a sample output of the the ETC matrix is shown in Table I.

TABLE I
SAMPLE ETC MATRIX

	C_1	C_2	C_3	......	C_x
Cloud 1	200	240	400	.	200
Cloud 2	400	320	326	.	430
Cloud 3	546	243	324	.	239
.	.	.	.	.	.
.	.	.	.	.	.
Cloud n	234	432	452	.	321

B. Non-functional parameters

A non-functional parameters, Price (P_C) is considered, which is the amount charged to compute task $t \in a$ by C_x. In cloud spot market, due to price fluctuation, SLA significantly changes with respect to P_C.

C. Decision Variables

$\forall C_x$ is taken by minimizing $T_C \in H$ and is represented as:

$$T_{Cx} = \sum_{i=1}^{n} \sum_{i=1}^{v} ETC(i, i+1) \times (P_{Ci} + P_{Ci+1}) \rightarrow min \tag{6}$$

where, *n* is the number of clouds and *v* is the service components cloud.

So, SLA is the agreement between a user and the cloud service broker for every job request, which is given as below.

$$sla_g = sla_1, sla_2, sla_3..sla_z \forall slag_g \in a \tag{7}$$

SLA is based on the maximum response time offered (R_{off}) by a CSB to execute a *J* under a given price P_c. R_{act} represent an actual/observed response time and is calculated as shown below.

$$R_{act} = \sum_{ij=1}^{t} ETC(i, i+1) \forall a \tag{8}$$

where, $t \in a$.

IV. PROPOSED SCHEME

Fig. 3 depicts the representation of the service components as *nodes*, which are connected to each other forming a Bayesian network. Same type of service component are considered at each *level* and identification of $T_C x \rightarrow min$ is the objective. Assuming *C1* as a parent node and *C2* is its child and so on, z schedule rule is created by identifying an appropriate service component. The rule, which is called most frequently is considered as the best rule.

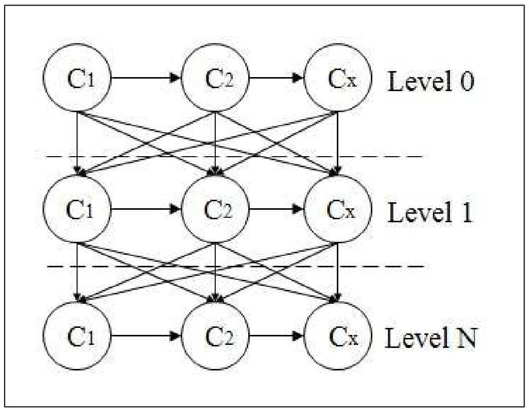


Fig. 3. A directed graph for component allocation.

The conditional probability for each node is calculated. So, for node C_{x+1} and its parent C_x, the conditional probability is given as below.

$$P(C_{x+1|C_x}) = \frac{P(C_{x+1,C_x})}{C_x}$$
$$= \frac{\sum(C_{x+1} = TR, C_x = TR)}{\sum(C_{x+1} = TR, C_x = TR) + \sum(C_{x+1} = TR, C_x = FA)} \tag{9}$$

where, TR and FA denotes TRUE and FALSE, respectively.

Now, the probability of the nodes with no parent, like, C_1, is given as below.

$$P(C_1) = \frac{\sum(C_1 = TR}{\sum(C_1 = TR + \sum(C_1 = FA} \tag{10}$$

The probability values obtained are used as rule strings and the component with the higher probability is selected. Since the first service component in a solution has no parents, so it is selected from nodes C_x with the normal probability $\frac{1}{m}$, where *m* is the total number of C_x service components. The next service component is selected from level *1*, i.e., C_{x+1}, by considering the conditional probabilities based on the parent node, i.e., $l \in H$. This process is repeated till the last level. Therefore, in a such Bayesian network, $\forall C_x$, conditional probabilities are computed using Bayes theorem.

Hence, $\forall \beta qc \in C_x$,

$$P(C_x|\beta qc) = \frac{\sum(\beta qc \in C_x|C_x P(C_x)}{P(\beta qc)} \tag{11}$$

and $\forall \tau \in C_x$

$$P(C_x|\tau) = \frac{\sum(\tau \in C_x|C_x P(C_x)}{P(\tau)} \tag{12}$$

where, βqc is Queue strength and τ is the execution time calculated by using Eq. 3.

Hence, by multiplying Eqs. 10 and 11 with P_C, the strength of each service component $Cx \in C$ is generated, which is calculated as below.

$$S(C_x) = P(C_x|\beta qc) \times P(C_x|\tau) \times P_{cx} \tag{13}$$

After calculating the strength of each service component, the strength of its connection $arc(i, j)$ with each of its child node is calculated, i.e., $(C_{x+1} \rightarrow C_x)$, which is given as:

$$P(C_x)|P(C_x \leq L \leq C_{x+1}) = \frac{P(C_x \leq L \leq C_{x+1})|C_x)P(C_x)}{P(C_x \leq L \leq C_{x+1})} \tag{14}$$

where, C_x is the parent node, C_{x+1} is its child node and $P(C_x \leq L \leq C_{x+1})$ is the probability of latency between C_x and $C_x + 1$.

Consider $R = \{R1, R2, R3 \ldots Rz\}|R \in G$, is the set of the all the probable rule strings, where *R1* is represented as $C_1(1)C_2(1)C_x(1)$, *R2* as $C_1(2)C_2(2)C_x(2)$,...., so on.

Therefore, $\forall R_z$ strength.

$$S_T(R_Z) = \sum_{x=1}^{N} \big(S(C_x) \in R_z + P(C_x \leq L \leq C_{x+1}) \in R_z\big) \tag{15}$$

A. Sequence of operations

To schedule a set of tasks among multiple service components in multi-cloud environment, a decision needs to be taken to identify the best rules among others. SLOPE is a self-learning system which identifies Rule in a particular time slot. This learning procedure consists of following steps:

1) Execute MCC [10] to find the task-node pair and store results in the form of rule strings with respect to *H*.
2) Strength calculation of nodes belonging to the graph *G* using Bayes Theorem and based on results obtained in step 1.
3) Populate all the probable rules obtained by calculating strength using Eq. 15.
4) Identify the best rule for a given job using roulette wheel method, i.e., higher strength rule is selected.
5) Monitor the outcome of the rule and in case of SLA violation go to next step.
6) In the event of a new rule, the strength of each of the node is increased evenly and for the old rule the strength is decreased, go to step 3.

V. Results and Discussions

A. Performance Metrics

The following performance metrics are considered for evaluation of *SLOPE*.

- **Makespan (M):** It is defined as the completion time needed to execute all the jobs using the available rule *R*. Let $M(S_k)$ be the makespan, of service k, $1 \leq k \leq m$, $|Ai|$ be the total number of tasks of the job A_i, $1 \leq i \leq n$, $ST(ij)$ be the start time of task j of job i , $CTL(k)$ be the completion time of last task on service k, $CTL(k)$ is initially assumed as 0. Then, the individual makespan, of each service Ck is calculated as below.
$$M(S_k) = \sum_{j=1}^{x} ETC(j) \tag{16}$$
Similarly, the total makespan of all the selected *R* is given as below.
$$TM(S_k) = \sum_{j=1}^{r} M(S_k) \tag{17}$$
where r is a selected rule.
- **Matching proximity** (MP_{sch}): It indicates the degree of proximity of a given response time produced by a schedule with regard to the response time of a schedule agreed during SLA. A large value of matching proximity means that a large number of jobs are assigned to the machine that executes them faster. Formally, this parameter is defined as below.
$$MP_{sch} = \frac{\sum_{j=1}^{x} ETC(j)/Agreed\_SLA(RT)}{x}, \tag{18}$$
Therefore,
$$f(x) = \begin{cases} 0 < MP_{sch} < 1, & x > 0 \\ 0, & otherwise \end{cases} \tag{19}$$
- **Resource Utilization**: It is defined as the utilization of resources with low price. Consider TS_c is the total price of the service components used to execute a job j, then the total price T_P charged to a CSB using a scheduling algorithm is given as below.
$$T_P = \sum_{j=1}^{x} TS_c(j) \tag{20}$$
- **SLA violation**: It is calculated as below.
$$SLA = R_{off} - R_{act}; \;\; R_{off}, R_{act} > 0 \tag{21}$$
where, R_{off} is the offered response time and R_{act} is the actual response time.
In order to evaluate the proposed scheme, the SLA success and penalty is calculated by a function $y(SLA)$, given as below.
$$y(SLA) = \begin{cases} penality(\rho), & SLA < 0 \\ sucsess(\dot{s}), & SLA \geq 0 \end{cases} \tag{22}$$
$\forall \rho \in y(SLA)$, $\dot{s}$ becomes ρ called request drop $(R_d)|\forall R_{act} = 0$. Now, the total number of violations is calculated by observing the total number of $penalties$.

B. System Implementation

SLOPE is evaluated using benchmark datasets used in [14]–[16]. However, these datasets are customized to add price as a new parameter. A random function in Python is used to set a range of different prices. In the benchmark datasets, the followings parameters are used.

- Distribution *u*: (Uniform).
- Type of consistency *x*: highly consistent (*c*), inconsistent (*i*) and semi consistent (*s*).
- Job heterogeneity *yy*: High (*hi*) or Low(*lo*).
- Resource heterogeneity *zz*: High (*hi*) or Low(*lo*).
- Price offered *pp*: High (*hi*), Medium (*md*) or Low (*lo*).

The details of the datasets are presented in Table II.

TABLE II
DATA SETS AND PARAMETERS

Dataset	Instances	Service requests	Jobs	Tasks	Clouds	DCs
D1	12	1000	10	5	10	5
D2	12	1000	20	10	20	10

In the first dataset (*D1*), 1000 service requests are considered and in each service request, there are 10 jobs and for each job, there are 10 tasks. Moroever, 10 clouds are considered, where each cloud is having 5 DCs at different locations. Similarly, in *D2* for 1000 service requests, there are 20 jobs per request and 10 tasks per job. A total of 20 different clouds are considered, wherein each cloud has 10 DCs deployed at different geographical locations. Therefore, each instance in the considered scenario can be labeled with different combinations of $u\_x\_yyzz\_pp$. In order to provide the different combination, cloudsim [17] is used to simulate the environment. Table III shows different configurations of DCs used to evaluate *SLOPE*. It is assumed that each virtual machine (VM) acts as a service component.

TABLE III
CONFIGURATIONS OF DCS IN CLOUDSIM 3.0.1

Number of hosts per DC	6
Number of VMs per host	5-10
VM (RAM)	1-2 GB
VM bandwidth	500-1000 mbps
VM queue length (buffer)	10-20
PEs per VM	1-2
Job Length	10,000-100,000 Million Instructions
Price	1-10 $ per task execution

In the first experiment (Fig. 4), relationship between the response times, planning horizon (lean time, transition time and peak time) and server rates are depicted. It is evident from the figure that with an increase in the arrival rate, the response time rises exponentially. However, the performance of server decreases. This result provides the base to observe and predict arrival rate and perform further experiments. Now, the proposed SLOPE is compared with the round robin (RR) algorithm (a static scheduling algorithm), MCC (a dynamic algorithm) and Bayesian optimization (BO) algorithm (an adaptive classifier).

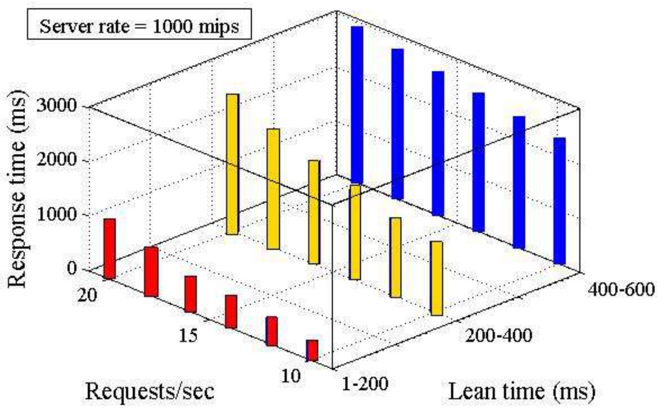


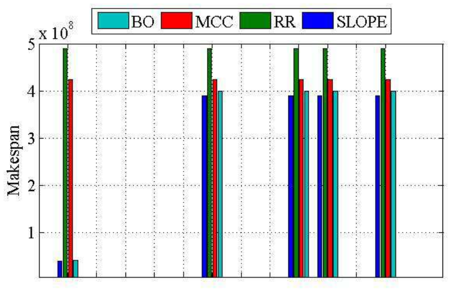
Fig. 4. Relation between the response times, planning horizon, server rates

The makespan of the proposed SLOPE is calculated by using Eq. 17 for both *D1* and *D2*. Each algorithm is executed 100 times. The average makespan considering all 12 instances for *D1* and *D2* is shown in Figs. 5(a) and 5(b), respectively. For simplicity, the results are scaled in logarithm to the base 10. From the above result, it is clear that average makespan of the SLOPE is comparatively less to other algorithms. Comparing it with BO for peak time, lean time and transition time, the makespan is reduced to 10%, 6%, and 12% respectively. SLOPE and BO achieves the best makespan values for almost all the instances. Now, in the scenario where resources are highly heterogeneous, the performance of MCC is as good as SLOPE. This may be due to the fact that for small ETC matrices MCC performance is exceptionally well. Moreover, in sub-small matrices, MCC outperforms BO in few cases. In Fig. 5(c), the results for cheap resource utilization are presented. SLOPE, BO and MCC are looking for cheaper resources but their outcomes depend on the type of instance. However, RR is the only one, which is not affected from the type of instances.

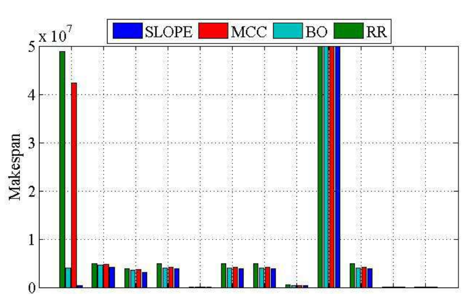
Fig. 5(d) indicates that the SLOPE has achieved high rate of matching proximity for all the 12 instances on the average. It is evident from the obtained results that SLOPE performs exceptionally well in almost all the performance metrics. Moreover, Figs. 5(e) (Training) and 5(f) (Test) provides a deep inspection of SLOPE and validate its performance with respect to SLA violations. RED area presents the area of SLA *failure* and GREEN representing SLA *success*. Test set indicates that after identifying best rule *R*, considering all above-mentioned parameters SLA failure for SLOPE is very low.

VI. CONCLUSION

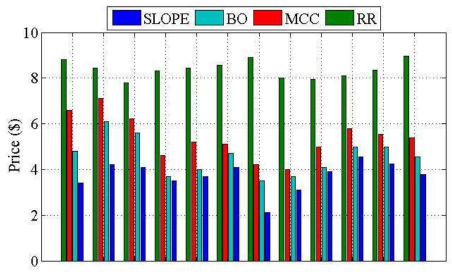
In this paper, Bayes theorem and Roulette wheel method are used to design a *self-learning optimization and prediction ensembler* (*SLOPE*) is designed for handling CSB scheduling in multi-cloud environment. *SLOPE* is compared with three existing methods (BO, MCC and RR). All these methods were implemented and evaluated for both functional and nonfunctional parameters on benchmark datasets. The computational results show that once the scheduling rules are identified,



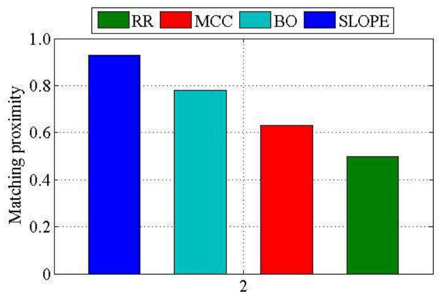
(a) Makespan for D1



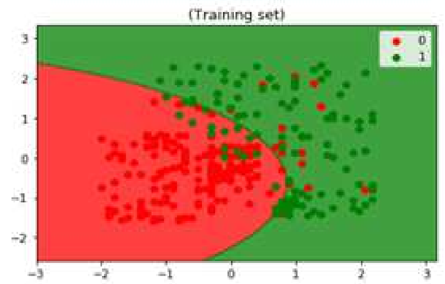
(b) Makespan for D2



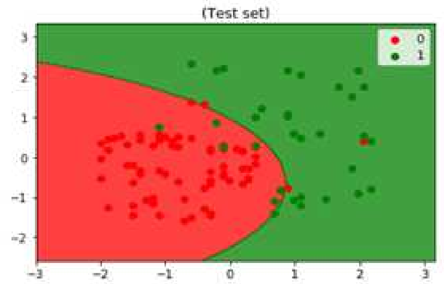
(c) Resource utilization (in terms of price)



(d) Matching Proximity



(e) Training for SLOPE



(f) Testing for SLOPE

Fig. 5. Performance evaluation of SLOPE with respect to RR, MCC and BO

the performance of the CSB is optimized with respect to the selected parameters such as-makespan, price, and SLA violations. The experimental study reveals the benefits to know the characteristics of scheduling in advance, which are obtained by applying dynamic scheduling algorithm. Such type of inputs helps self-learning schedulers to achieve an efficient allocation of resources depending on the desired system metrics (makespan, resource utilization, matching proximity and SLA management). In future, the proposed system would be evaluated on the basis of location of the service component. Further, deep learning algorithms would be inspected to manage high heterogeneity of task, resources or user behavior.

References

[1] H. M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments," *IEEE Transactions on Parallel and Distributed systems*, vol. 24, no. 6, pp. 1203–1212, 2013.

[2] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, "Optimal decision making for big data processing at edge-cloud environment: An sdn perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778–789, Feb 2018.

[3] R. Chaudhary, G. S. Aujla, N. Kumar, and J. J. P. C. Rodrigues, "Optimized big data management across multi-cloud data centers: Software-defined-network-based analysis," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 118–126, Feb 2018.

[4] M. Ali, R. Dhamotharan, E. Khan, S. U. Khan, A. V. Vasilakos, K. Li, and A. Y. Zomaya, "Sedasc: secure data sharing in clouds," *IEEE Systems Journal*, vol. 11, no. 2, pp. 395–404, 2017.

[5] C. Pettey and R. van der Meulen, "Gartner says cloud consumers need brokerages to unlock the potential of cloud services," Gartner Report, 2009. [Online]. Available: https://www.gartner.com/it-glossary/cloud-services-brokerage-csb/

[6] D. Alsadie, Z. Tari, E. J. Alzahrani, and A. Y. Zomaya, "Dynamic resource allocation for an energy efficient vm architecture for cloud computing," in *The Australasian Computer Science Week Multiconference*. ACM, 2018, p. 16.

[7] H. Wu, S. Deng, W. Li, J. Yin, Q. Yang, Z. Wu, and A. Y. Zomaya, "Revenue-driven service provisioning for resource sharing in mobile cloud computing," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 625–640.

[8] V. Stantchev and C. Schröpfer, "Negotiating and enforcing qos and slas in grid and cloud computing," in *International Conference on Grid and Pervasive Computing*. Springer, 2009, pp. 25–35.

[9] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, 2010.

[10] S. K. Panda and P. K. Jana, "Efficient task scheduling algorithms for heterogeneous multi-cloud environment," *The Journal of Supercomputing*, vol. 71, no. 4, pp. 1505–1533, 2015.

[11] Z. Qian and H. Xia, "Cloud computing system scheduling model based on dynamic bayesian network," *Journal of Residuals Science & Technology*, vol. 13, no. 7, 2016.

[12] Z. Er-Dun, Q. Yong-Qiang, X. Xing-Xing, and C. Yi, "A data placement strategy based on genetic algorithm for scientific workflows," in *Eighth International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2012, pp. 146–149.

[13] L. B. Ping, C. P. Kit, and E. K. Karuppiah, "Network latency prediction using high accuracy prediction tree," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*. ACM, 2013, p. 42.

[14] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[15] F. Xhafa, J. Carretero, L. Barolli, and A. Durresi, "Immediate mode scheduling in grid systems," *International Journal of Web and Grid Services*, vol. 3, no. 2, pp. 219–236, 2007.

[16] F. Xhafa, L. Barolli, and A. Durresi, "Batch mode scheduling in grid systems," *International Journal of Web and Grid Services*, vol. 3, no. 1, pp. 19–37, 2007.

[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.