International Journal of Innovative Research in Computer Science and Engineering (IJIRCSE)
www.ioirp.com ISSN: 2394-6364, Volume – 1, Issue – 6. May 2015

Genetic based adaptive task scheduling in cloud computing

[1]Sheena S.Nowshad, [2]N.Ashok Kumar
Department of Computer Science and Engineering, Maharaja College of Engineering, Coimbatore, India
[1]shnnowshad@gmail.com, [2]chmlashok@gmail.com

***Abstract*— Cloud Computing is the utilization of pool of resources for remote users through internet that can be easily accessible, scalable and utilization of resources. This technology compute resources in cloud systems can be partitioned in fine granularity and allocated on demand. It is important to optimize the cloud task's execution performance due to its more constraints like user payment budget and divisible resource demand. For this adaptive algorithm is used. This algorithm allocates the resources at runtime based on task's execution progress and updated resource availability states. This also analyzes the upper bound of cloud task length by considering the host load prediction errors and workload prediction errors. With this the worst case task execution performance is predictable which improves the quality of services. In this algorithm, the resource allocation for each subtask is performed only when its preceding subtask is finished. This can adapt to the load dynamics to a certain extent. This is a mechanism to reallocate the resource vectors for the running subtasks, by leveraging the idle resource fraction released from time to time, further improving resource utilization. But the complexity of this algorithm is greater. The proposed system implements genetic algorithm. This proposes a solution with improved performance and the complexity has been reduced. The genetic algorithm implements the task allocation adaptively.**

Keywords*— *Cloud Computing;Genetic Algorithm;Fault tolerance

I. INTRODUCTION

Cloud computing provides on demand network access to a shared pool of resources. Cloud computing ensures access to virtualized IT resources that are present at the data centre, and are shared by others. The data stored in Cloud are simple to use and are paid for the usage and can be accessed over the internet. These services are provided as a service over a network and are accessible across computing technologies, operations and business models. Cloud enables the consumers of the technology to think of computing as effectively limitless of minimal cost and reliable to need not know about how it is constructed, its working, its operation, or where it is located.

All the resources provisioned by cloud system are supposed to be under a payment model. Each task's workload is likely of multiple dimensions. First, the computer resources in need may be multi attribute (such as CPU, disk reading speed, network bandwidth, etc.)Resulting in multi dimensional execution in nature. Second, even though a task just depends on one resource type like CPU, it may also be split to multiple sequential execution phases, each calling for a different computing ability and various price on demand, also leading to a potentially high dimensional execution scenario.

Scheduling is used for distributing resources among parties which simultaneously and asynchronously request them. For the allocation of task to the resources, several algorithms are used. These algorithms are used mainly to minimize the execution length and to ensure fairness amongst the parties utilizing the resources. This deals with the problem of which resources needed to be allocated for the received task. The allocation of resources can be done through the Task Scheduler enables us to automatically perform routine tasks on a chosen schedule. The Task Scheduler does this by monitoring whatever criteria to choose to initiate the tasks and then execute the task when the criteria are met. The task Allocation problem optimizes the overall performance of the application and provides assurance for the correctness of the result. Traditional job scheduling is often formulated as a kind of combinatorial optimization problem or queue based multiprocessor scheduling problem. That is, most of the existing deadline driven task scheduling solutions from single cluster environment confined in LAN to the Grid computing environment suitable for WAN are also strictly subject to the queuing model under which a single machine's multiple resources cannot be further split to smaller fractions at will. By formulate demand for computing power and other resources as a resource allocation problem with multiplicity, where computations that have to be performed concurrently are represented as tasks and a later task can reuse resources released by an earlier task. The adaptive algorithm is the resource allocation for each subtask (or each particular execution phase) is performed only when its preceding subtask is finished. This can adapt to the load dynamics to a certain extent. In fact, this design an adaptive mechanism to tune/reallocate the resource vectors for the running subtasks, by leveraging the idle resource fraction released from time to time, further improving resource utilization.

Genetic algorithms are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness the more suitable they are the more chances they have to reproduce. To use a genetic algorithm, you must represent a solution to your problem as a genome. The rest of the paper is organized as follows. In Section 2, provides a brief discussion of the most recent relative literatures in the field of cloud resource allocation. In Section 3, we describe the system model followed by basic DODRA algorithm and our proposed genetic algorithm. In Section 4, we perform a comparative analysis of our proposed genetic algorithm against existing DODRA algorithm. Finally in Section 5, we conclude the paper and discuss future research areas.

II. SYSTEM MODEL

The system uses the algorithm that allocates the resources at run time based on task's execution progress and updated resource availability states. The objective of the system is to have maximum utilization of resources and to reduce the execution time. The genetic algorithm can be used for the better solution of allocating the task to the correct resource. It overcomes the limitation of the existing system.

A. Existing Model

The existing system proposes algorithm for minimizing the cloud task length with divisible resources and payment budget. The algorithm derives the upper bound of cloud task length by taking into account both the host load and workload prediction. Therefore, the worst-case execution performance is predictable which can improve the quality of service. This design a dynamic version for the algorithm to adapt to the load dynamics over task execution progress, further improving the resource utilization.

This system proposes 2 algorithms.

- *Local Optimal Allocation algorithm*
  This algorithm is used to minimize the cloud task execution length subject to a set of constraints like user's payment budget and host availability states.

- Dynamic optimal divisible resource allocation
  This algorithm is a dynamic version of local optimal allocation algorithm. It is an adaptive mechanism to tune or reallocate the resource vectors for the running subtasks, by leveraging the idle resource fraction released from time to time, further improving the resource utilization.

The system follows a popular data centre model (or server client model) to process cloud user requests. The cloud server is responsible for collecting dynamic availability states of managed nodes and customizing virtual machines based on users various demands.

A task execution can be split into three steps.

1) Select a qualified physical node
2) The task is running in an individual VM instance, whose resources are customized on demand by VMM
3) Send computational results to users

Different task executions are likely of different execution patterns in that they need multiple types of resources . For example, one task execution may be split into multiple steps, each demanding a different CPU rate or I/O bandwidth. Suppose there are n physical nodes .The task's execution is determined by three types of resources including CPU rate, disk input or output rate, and network bandwidth rate. It then has three types of workloads to process an amount of computation to be performed by CPU, an amount of data to read from disk, and an amount of data to transmit through network. That is, The task's workload is a three dimensional vector, and each element of the vector must be no greater than the corresponding available resource rate of the selected execution node.

Many of existing systems (such as Google App Engine leverage PaaS or SaaS architecture) which allows users to customize composite web services . However, the validity of services depends on the availability of their hosting nodes, whose states are likely changed over time. Hence, the host availability vector used in the resource allocation for the remaining subtasks

is best to be dynamically updated over time. On the other hand since the subtasks in a task are can be deemed as different execution dimensions. This devise a dynamic algorithm, to allocate resources for the tasks each of which is made up of multiple subtasks connected in series. All of service bodies are deployed on each host machine and classified based on their functions. DODRA is a dynamic version compared to LOAA as the resource allocation for each subtask (or each particular execution phase) is performed only when its preceding subtask is finished. This can adapt to the load dynamics to a certain extent. In fact, this further design an adaptive mechanism to tune/reallocate the resource vectors for the running subtasks, by leveraging the idle resource fraction released from time to time, further improving resource utilization.

B. Proposed Model

The proposed system can be implemented with genetic algorithm and it overcomes the limitation of the existing system. The limitations of the existing system is:

1) *Higher time complexity*
   The resource allocation for the tasks among different nodes are compared together. The comparison uses the matrix way of operation. Therefore Complexity is greater. And also when there are infinite number of nodes are available, there does not get the optimal solution.
2) *Solution doesnot satisfy with a particular criteria*
   Solution obtained with the DODRA algorithm produces ambiguity. That is there are several constraints such as communication cost, execution cost and execution Time is considered. Therefore, the user will get the sense for this solution.
3) *As the task deadline increases,idle time also increases*
   When we increases the task deadline, the idle time of the hosts to be increases. This degrades the performance of the cloud and the customer have to be wait for infinite period of time. Thus resulting the greater allocation cost.
4) *Fault tolerance cannot be implemented*
   The existing system does not guarantee the availability and reliability of the nodes. That is when the allocated node is failed, the task remains block even if a huge number of jobs are allocated at that time.

To overcome this limitations, Genetic algorithm is to be implemented. Genetic algorithm deals with the natural selection of solution from all possible solutions. The genetic algorithm can be used for the better solution of allocating the task to the correct resource. This is satisfied with a particular criteria i.e rank. The rank will consider the execution cost, execution time and communication cost.

Genetic algorithm consists of 6 steps:

1) *Initial Population*
   When Genetic Algorithm is used to solve problems, initial population provides solutions to the problem. The set of solutions that are possible is taken as the initial population. These solutions are considered as chromosome. The chromosomes in the initial population are generated randomly and these terminals are chosen to solve the particular problem.
2) *Fitness Function*
   Selecting a suitable fitness function is used to design successful Genetic Algorithm. It evaluates how the selected function meets the objective of the problem. GA evaluates each chromosome by fitness function. Fitness function is used for the measurement of effectiveness of the solution according to the given objective. They help to know which chromosomes retain in the population. Incorrect fitness function may lead to delay in process. When the fitness function is larger it meets the QOS requirements of that task.
3) *Selection*
   Selection operator is used to select among the given chromosomes of current population for inclusion to next population. It is mainly used to find the best fit individual. Each chromosome has equal probability to its score by the sum of all other chromosome probability. These individuals create next generation. Here range selection is used. Every individual is given a rank based on the fitness function. The individual that is most fit gains more preference when using this method.
4) *CrossOver*
   Crossover operator pairs all the chromosomes. It is used to combine two chromosomes to produce next generation chromosomes. It is used for bringing new chromosomes by the mixture of parent chromosomes. Single point crossover is used as only one crossover point is present. In this single crossover point, at the locus, swapping the remaining alleles from parents to others takes place. The operation is performed to select the chromosome. If crossover operation is not performed then the new generation resembles its parents.
5) *Mutation*

Mutation performs the permutation of existing chromosomes. It is used to maintain the genetic diversity from one generation to next generation. It provides new gene values added to the gene pool. Mutation provides small alterations at each individual. It is used for finding new points in search space hence the population variation is maintained. They provide functions on the chromosomes produced by crossover.

6) Termination Algorithm

The selection process is able to copy the chromosome that is having the maximum fit in the given population. This process is repeated for several generations. This helps to obtain the highest fit value. The termination depends on the size of the chromosomes that has been sorted.

Genetic algorithms are stochastic in nature. The complexity depends on the genetic operators, their implementation, the representation of the individuals and the population and on the fitness function. Therefore the complexity of the genetic algorithm is O(gnm) where g is the number of generations is the population size and m is the size of the individuals. In the system, the solution is justified based on the rank.

RANK=EXECUTION TIME+EXECUTION COST+COMMUNICATION COST+IDLE TIME

Therefore,we have to consider only the rank of this algorithm.

The idle time of the hosts implemented with this algorithm decreases and therefore the customers will give the job output without waiting any more even when there is a large number of task is submitted at time.

The fault tolerance technique used is job migration. It can be implemented with a new algorithm.ie. ,Best node algorithm. When a node or more than 1 node is to be failed, the allocated job is to be given to the other nodes. The job allocation is done by using this best node algorithm.

Algorithm: Best Node Algorithm

When a node is failed, the job allocated to that node is to be transferred to the other nodes according to the following steps:

1. Select the nodes having larger idle time.\
2. Evaluate each node with respect to their execution cost, execution time and communication cost.
3. Transfer the job to the node that have the lesser parameters.

III. SIMULATION SETUP AND RESULTS

A. Experimental Settings

We evaluate our Optimal Divisible-Resource Allocation algorithm in a real cluster environment, called Gideon-II, which is the most powerful super computer at Hong Kong. We are assigned eight physical nodes connected with 10 Gbps high-speed intra-network. Each node has 8 2.45 MHz-cores and 16 GB of memory size. We deployed XEN 4.0 on each node. Since there must be one core reserved for XEN hypervisor, we created 56 VM instances (centos 5.2) on the eight physical nodes. Three types of resource attributes (CPU rate, network bandwidth, and I/O disk bandwidth) will be split on demand according to our ODRA algorithm. Specifically, we make use of XEN's credit scheduler to dynamically allocate various CPU rates (or capabilities) to the VM-instances. The network bandwidth and the disk reading/writing rate allocated to each user are both reshaped by linux network traffic controller on demand at runtime. In our experiment, each user task is constructed by multiple subtasks, each corresponding to various web services with heterogeneous workloads to process. The subtasks could also be data transmission via network or data read/ write through disk. So, each subtask could be treated an execution dimension with a particular workload to process (e.g., number of float points and data to transmit) and a resource fraction (or processing ability) to allocate (e.g., CPU rate and network bandwidth).

By leveraging Parallel Colt(a library of matrix computation), we implement 10 matrix computation programs in the form of web services (listed in Table 2), which can be further combined to construct more complex matrix problems. The number of subtasks per task is randomly set in [5, 15], and each subtask is a matrix computation selected from the 10 matrix computations. In our experiment, different tasks have different execution patterns. Second, each task execution involves three types of resources (CPU rate, network bandwidth and I/O disk bandwidth). For a particular task, the first subtask is reading one or more matrix data from disk drive. The second subtask could be some matrix computation like matrix product, decomposition, or others. As a matrix computation is finished, its output (a new matrix) will be transmitted to another VM instance through the network. The data transmission is also a kind subtask whose workload and capacity is data size to transmit and the network bandwidth controlled on demand. The last subtask of one task is storing the final matrix result into the disk drive. The CPU rate assigned to VMs is

controlled by XEN's credit scheduler. Network bandwidth and disk I/O rate are both controlled within [10,300] Mb/s over NFS through Linux network traffic controller. The CPU capacity of each multithreaded program (e.g., matrix product, matrix power) is set to eight-cores (i.e., the maximum processing ability of one physical node), while that of any single-threaded program (such as matrix decomposition) is set to one-core in that more resources cannot get further speedup on it. We predict the workload of each matrix computation based on history for simplicity, where a and b are set to 0.7 and 2 respectively based on our characterization about prediction errors. In practice, one could use more accurate methods like multi-variate polynomial regression , whose margin of prediction errors is 7-10 percent. Each task is associated with a budget, which is a key factor impacting task's resource allocation. The prices of the 10 matrix operations are set to 1; 2; . . . ; 10 with the decrease of their workloads. We evaluate our algorithm with different budgets assigned, which are simulated based on the formula(1) where $b(F_i)$ and Θ refer to the price of the function
F_i and coefficient to tune users' various economic strengths respectively.

$$\check{T}\text{s budget} = \Theta\Sigma b(F_i) \qquad (1)$$

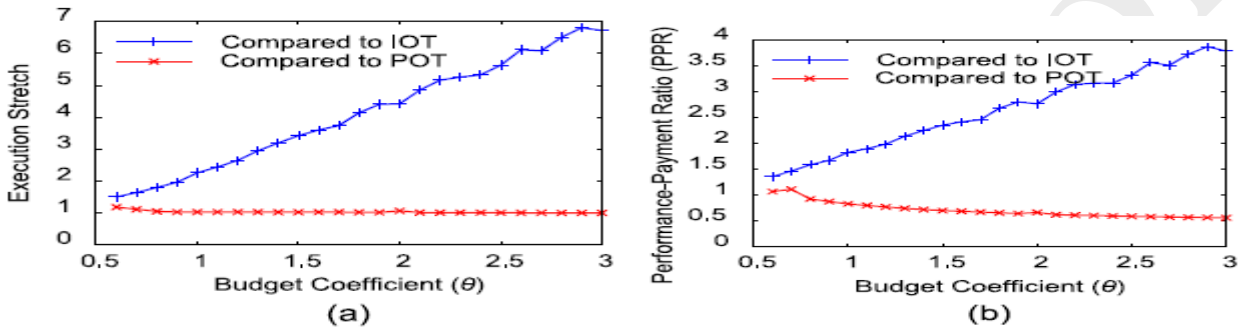

Fig 1.a: Execution Stretch under different payments

Fig 1.b: PPR under different payments

Upon receiving a task made up of multiple consecutive matrix computations, our designed algorithm is triggered to compute the optimal resource vector for it and perform resource isolation (e.g., notifying corresponding VMM to customize the CPU rates by credit scheduler). We adopt three baseline results for comparison. The first one is the execution length based on the ideal convex-optimal resource vector calculated with the assumption that the resource capacities are unbounded. We call it ideal optimal time (IOT). The second one is the execution length based on the practical optimal resource vector under the limited capacities in reality. We call it practical optimal time (POT). The last one is one heuristic algorithm called Load-Price-Ratio based Proportional-Share (LPRPS). This heuristic is designed based on such an intuition: In one task, the subtasks with more workloads should be allocated with more resources, in order to minimize the total execution length. On the other hand, the subtasks with higher prices will cost higher than the ones with lower prices, thus the subtasks with lower prices are better to be allocated with more resources to make the payment more worthy. Hence, LPRPS tries to split the budgets among subtasks proportional to the Load-Price-Ratio.

B. Experimental Results

There are two key metrics in our evaluation. The first one is called execution stretch, aiming to evaluate task's execution performance. A task's ES is defined as its real execution length (with possibly erroneous workloads predicted) divided by its theoretically optimal execution length calculated based on its real workloads recorded after its execution. Smaller ES implies higher execution efficiency and ES being equal to 1 implies that the task's practical execution
length reaches its theoretically optimal result. The other one is called performance-payment ratio (PPR), which is used to evaluate the effectiveness of user's payment. A task $\check{T}$'s PPR is defined in Formula (2), where t's payment level is equal to $\check{T}$'s final payment/$\check{T}$'s budget .The smaller PPR, the higher performance with lower payment meanwhile, indicating higher satisfactory level

$$PPR(\check{T}) = ES(\check{T}) \times (\check{T}\text{'s } \textit{payment level}) \qquad (2)$$

We first evaluate the impact of different budgets assigned to a single task to its execution performance and user's final payment. In Fig. 1a, we observe that the task's ES compared to its IOT increases linearly with the increase of the budgets assigned. That is, higher budgets assigned will result in larger theoretically optimal resource amounts allocated, leading to a shorter IOT. However, the task has to be run atop the resources with limited capacities in practice, so we also compare task's real

execution length to its theoretically optical value by taking capacity into account (i.e., POT). Fig. 1a shows ES is always very close to 1 in this situation, which confirms that our solution is indeed able to optimize task's performance, based on user requirement and the load dynamics. In addition, the similar observation goes to the PPR metric, as shown in Fig. 1b, confirming the payment should also be satisfied by users.

We also compare the results in the situation with sufficient resources and the one with short supply respectively. Fig. 2a shows the mean/lowest/highest values of the ES compared to IOT and POT respectively. When the number of tasks is small, the mean ES in both situations is always below 1.1, while the highest value of ES (worst situation) is up to 4. This is reasonable based on the following explanation. Note that the computation workloads among some subtasks (i.e., basic matrix operations) are largely different (e.g., between M-M-Multi. and V-V-Multi.), thus the resource amounts derived based on convex-optimization could be quite different. This will make the ideal optimal resource fractions of heavily-loaded subtasks be much bigger than the resource capacities (eight-cores for the multi-threaded programs or 1-core for the single-threaded programs), such that subtasks cannot run in its ideal optimal states. With further increasing number of tasks (from 10 to 40), the ES compared to IOT would increase notably, yet the ES compared to the POT still keeps pretty close to 1 (as observed in Fig. 2a). In absolute terms, in comparison to POT, the mean value of ES can be limited down to about 1.1. This is attributed to the fact that more and more tasks cannot be assigned with the ideal optimal resource vectors while it can be assigned with the practical optimal resource vectors with regard to the limitation of the resource capacities. In fact, in such a situation with relatively short resource supply, task's practical optimal performance would also be degraded correspondingly, and ES =PO the tasks under our resource allocation run as efficiently as the practical optimal state with the capacity limitation. The similar observation goes with the PPR metric, as shown in Fig. 2b.

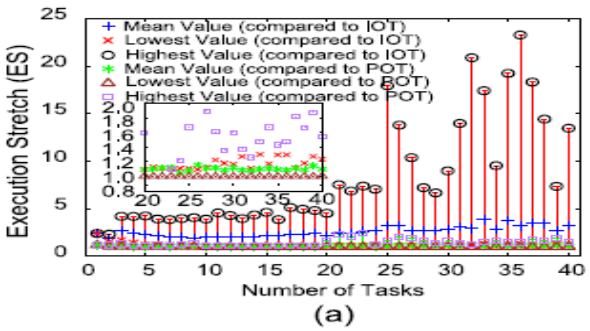

Fig 2.a: Execution Stretch under different number of tasks

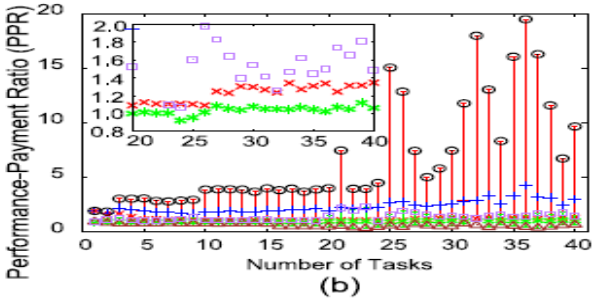

Fig 2.b:PPR under different payments

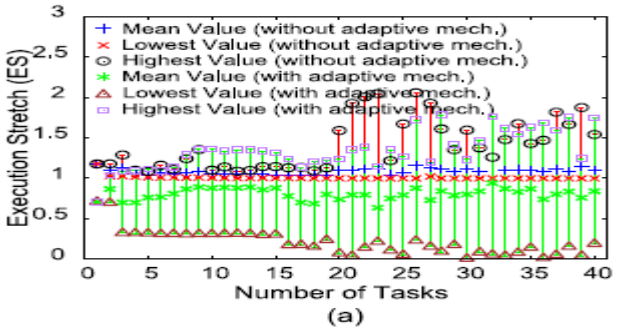

Fig 3.a: Execution Stretch under adaptive mechanism

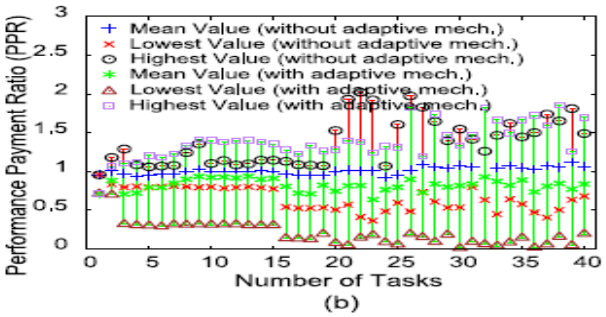

Fig 3.b:PPR under adaptive mechanism

When comparing to IOT, the PPR enhances with increasing number of tasks to process, yet it increases more slowly than that of ES shown in Fig. 2a, because users' payments are correspondingly reduced with the smaller resource amounts allocated. We also evaluate the performance of the Algorithm 3 with the adaptive mechanism, which aims to further refine subtasks' resource allocation to adapt to the resource state changes at runtime. With the adaptive support, as some resources previously occupied by some tasks are released due to their completion, the optimal resource vectors of the other running tasks on the same node will be performed again. In Fig. 3a, we observe that the adaptive version of DODRA significantly outperforms the one without the adaptive support, via the mean ES. The mean ES with and without the adaptive mechanism is about 0.7 and 1.1 respectively. This confirms that task's execution would be impacted by the serious resource competition, while our adaptive

mechanism could effectively solve the problem by adaptively reallocating the released resources at runtime. In addition, Fig. 3b shows the PPR under the adaptive version of our algorithm. When there are 15+ tasks submitted (competitive situation), the mean value of PPR is much lower than that of the non-adaptive version by about 35 percent in general.
We also analyze the fairness of task's resource allocation based on Jain's fairness index , where xi is either ES or PPR

$$F(x) = (\Sigma x_i)^2 / \Sigma x_i^2 \qquad (3)$$

The fairness of ES (PPR) with respect to POT is much higher than the one with respect to IOT, as shown in Fig. 4a. This is because the basic matrix operations in our experiment are with largely different workloads (Table 2), which could easily make the resource fractions assigned at different dimensions are quite uneven. That is, the resource amounts expected at some dimensions may be extremely huge, finally succeeding the corresponding resource capacities to different degrees. Then, the degradation of the practical execution compared to IOT could be very arbitrary.

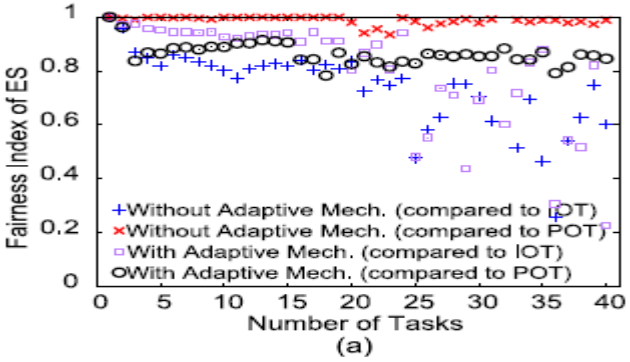

Fig 4.a:Fairness index of ES

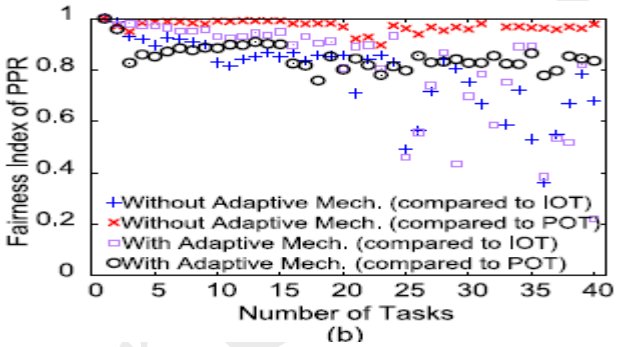

Fig 4.b:Fairness index of PPR

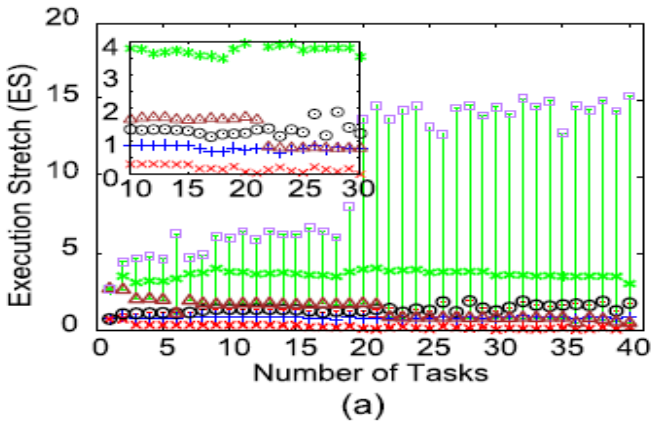

Fig 5.a:ES between ODRA and LPRPS

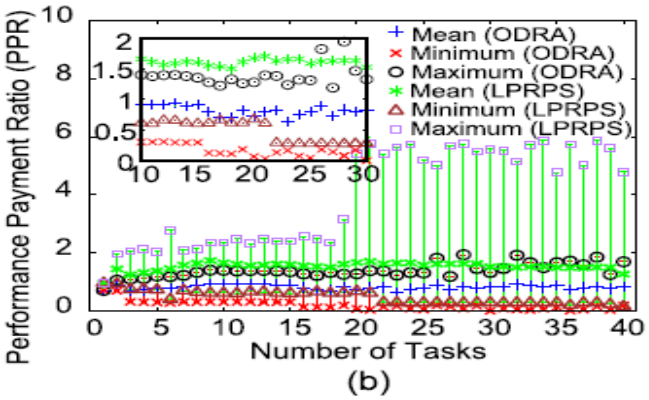

Fig 5.b:PPR between ODRA and LPRPS

However, we could still observe quite stable and highly fair treatment on task's resource allocation w.r.t. POT, i.e., when comparing to the execution length to the practical optimal state considering the capacity limitation. Finally, we compare the execution performance of our algorithm to that of the Load-Price-Ratio based Proportional-m Share heuristic, as shown in Fig. 5 (Both are tested under the dynamic/adaptive resource allocation). We could clearly observe that LPRPS performs with much higher mean value of ES and PPR than that of our solution. Our ODRA algorithm can also effectively adapt to the competitive situation, i.e., the mean value and maximum value of the ES and PPR under ODRA stay quite stable even through there were over-many tasks submitted. In absolute terms, the mean values of ES and PPR under ODRA are about 1/4 and 1/2 of those under the LPRPSheuristic, which means ODRA outperforms the LPRPS heuristic significantly.

IV. RELATED WORK

Cloud resource allocation problem has been extensively studied for years and most of the existing work is strongly subject to the assumption with precise workload/host load prediction.Usiao[1] proposed a distributed load rebalancing method for distributed file systems in clouds.Unlike the file system where data size is relatively easy to predict precisely, we have to deal with erroneous prediction issue in our computational cloud platform with multiple execution dimensions.

PACORA[2] is a performance aware convex optimization model for resource allocation problem, assuming workload information could be known in advance. Jalaparti.[3] aims to optimize the resource allocation utilities between any two clients or client and provider. Their solutions have a strong assumption that the resource capacities are always large enough,while in our model, limited resource capacity is a key constraint, leading to a huge challenge especially in the bound analysis with prediction errors. Meng et al.[4] explicitly leads to maximize resource utilization by analyzing VM-pairs compatibility in terms of the workload and estimated VM sizes. Their solution is able to approximately identify the compatibility of any pair of two VMs but cannot resolve the situation with more than two VMs on the same machine.Wei et al.[5] formulated the cloud resource allocation to be a binary integer programming problem and solved it using an evolutionary method. A strong assumption in their work is the precise prediction of task's workloads on multiple execution dimensions.

In order to provide guaranteed service-level agreements, it is crucial to analyze the possible situation wuth inaccurately predicted information. Maos's auto scaling method and Di's approach took into account load prediction issue in cloud systems, whereas they both handled a different objective that aims to minimize user payment with guaranteed task deadlines. Cloud Scale[6] is a cloud system that automates fine-grained elastic resource scaling for multi-tenant cloud infrastructures.It employs online demand prediction and handles prediction errors to achieve adaptive resource allocation. In contrast to this work,we theoretically derived the upper bound of the task execution length with different margins of prediction errors. Moreover, we improved the algorithm by making it adapt to dynamic changes of resource availability states, significantly enhancing the robustness.

Beyond the scope of cloud computing, there are some existing works that discussed robust convex optimization problem, which is similar to our resource allocation issue with erroneous predictions .chaisiri [7] formulated the robust cloud resource provisioning as a stochastic programming problem with uncertainty. In comparison to the above work, the resource allocation formulated in this paper is particular and largely different ,due to the no-linear target function with multiple execution dimensions. We designed LOAA algorithm to solve it with provably optimal output. Since LOAA is a specific algorithm, it needs particular analysis about its output with uncertain information.

Scheduling performed using several algorithms are studied in the literature recently. In, the author proposes an image caching mechanism to reduce the overhead of loading disk image in virtual machines. The author inpresents a dynamic approach to create virtual clusters to deal with the conflict between parallel and serial jobs. In this approach, the job load is adjusted automatically without running time prediction. In, the author describes a suspend/resume mechanism is used to improve utilization of physical resource. The overhead of suspending/resume is modelled and scheduled explicitly. In present a planner guided strategy for multiple workflows. It ranks all ready tasks and decides which task should be scheduled first. However if there are new lower rank workflows coming continuously, the higher rank task will not be scheduled to execute. In a massive scalable cloud, this situation will become true. This algorithm only considers the execution time. The author in describes the Minimum Execution Time (Met) assigns each job in arbitrary order to the nodes on which it is expected to be executed fastest, regardless of the current load on that node. Met tries to find good job node pairing, but because it does not consider the current load on a node it will often cause load imbalance between the nodes and not adapt application in the heterogeneity computer systems. On scheduling in grid computing environment, genetic algorithm could be adopted to solve the NP-hard problem. The author in represents an extensive study on the usefulness of GAs for designing efficient Grid schedulers when makespan and flow time are minimized. A first-come-first-served and a Genetic algorithm are used for the load balancing strategy. It gives GA based task scheduling for optimum decision. The author in introduces the Task and Virtual Machine (TVM) scheduler, which schedules VMs in cloud and task on VMs. By scheduling both, task and VMs, it is possible to define virtual computing systems that go beyond the limitation imposed by the availability of resources. The Virtual machines are instantiated in every host in the network prior to the execution of the application. Tasks composing application have requirements that can limit the choice of resources for execution.

Acknowledgment

This research is supported by the laboratory in Maharaja Engineering College.

References

[1] E. Ayday, H. Lee, and F. Fekri, "An iterative algorithm for trust management and adversary detection for delay tolerant networks," IEEE Trans. Mobile Computing, vol. 11, no. 9, pp. 1514-1531, Sept. 2012.

[2] N. Li, and S. K. Das, "RADON: Reputation-Assisted Data Forwarding in Opportunistic Networks," in 2nd ACM International Workshop on Mobile Opportunistic Networking, Pisa, Italy, Nov.2010, pp. 8-14.

[3] F.Li, J.Wu, and A. Srinivasan, "Thwarting blackhole attacks in disruption-tolerant networks using encounter tickets," in IEEE Conference on Computer Communications, 2009, pp. 2428- 2436.

[4] Psaras, L. Wood, and R. Tafazolli, "Delay / disruption tolerant networking : State of the art and future challenges, technical report, Dept. of Electrical Eng., Univ. of Surrey, 2009.

[5] S. Trifunovic, F. Legendre, and C. Anastasiades, "Social trust in opportunistic networks," Proc. IEEE INFOCOM, pp. 1-6, Mar.2010.

[6] Haojin Zhu, Member, Suguo Du, ZhaoyuGao, Mianxiong Dong, Member, IEEE, and Zhenfu Cao "A probabilistic misbehavior detection scheme towards efficient trust establishment in delay-tolerant networks",IEEE,2013.

[7] Baoqun Yin, Dong Guo and Shan Lu, "Analysis Of Admission Control In P2P based Media Delivery Network," International Journal of Innovative Computing, Information and Control, University of Science and Technology of China, ISSN 1349- 4198,Volume **7**, Number 7(B), July 2011.

Authors Short Profile:

Sheena S.Nowshad was born in Kerala,India in 1989.She received her B-Tech degree in Information Technology from Caarmel engineering College,Ranni in 2011.She is currently doing her M.E,Computer Science in Maharaja Engineering College, Coimbatore.

N.Ashok Kumar was born in Tamil Nadu. He is currently working as an assistant professor in the department of computer Science in Maharaja Engineering College.