A New Approach for Dynamic Job Scheduling in a Volunteer Desktop Grid

Shaik Naseera
Professor, Department of CSE
Sreenivasa Institute of Technology and Management Studies, Chittoor, India

K.V.Madhu Murthy, Ph.D
Professor
Department of CSE
Sri Venkateswara University
Tirupati, India.

ABSTRACT

Nodes in a volunteer desktop grid are based on the volunteer participation of desktop nodes owned by the individual users. The volunteer nodes contribute their idle resources for public execution during their free time and withdraw during their busy time due to high priority node owner's private execution. This property makes desktop grid dynamic in nature. Job scheduling is mainly influenced by two factors: node dynamism and heterogeneity. The job generation rate at each node is different from other nodes in the desktop Grid and hence the load at each node changes with time. This situation leads to increased computational demands at some nodes than from others and makes the grid often to get into unbalanced environment. Job migration to remote nodes involves job transmission latency. Since grid is a dynamic environment, when the job reaches to the remote node for execution the node might become busy and the selected target node may not complete the execution of the job at the expected speed. Therefore, the selection of a target node for job migration plays an important role in improving the overall performance of the desktop Grid. In this paper we present a new approach for dynamic job scheduling that considers node dynamism and job transmission latency into account for making scheduling decisions. The algorithm is compared against the Resource Exclusion and non migration algorithms and the simulation results shows that the proposed algorithm has got considerable improvement over the other two.

General Terms

Grid Computing, Desktop Grid, Algorithms.

Keywords

Volunteer nodes, desktop grid, job scheduling, job migration, node dynamism, average turnaround time.

1. INTRODUCTION

Advances in the computing and networking technologies made possible to manufacture powerful computing devices at lower cost now days, therefore plenty of computing power and resources are available with the user desktops. Desktop grid aims to harvest a number of idle desktop computers owned by individuals to achieve high throughput computing by harvesting idle computing power contributed by the volunteers [1]. The users volunteer their idle resources during their free time and withdraw them during their busy time. Desktop Grid has recently received the rapidly growing interest and attraction because of the success of the most popular examples such as SETI@Home [7] and distributed.net [9].

Nodes connected in the desktop grid are heterogeneous in nature and geographically distributed to different locations. The nodes are connected and communicate with each other over internet. Therefore, transmission latencies are involved for any kind of communications between nodes.

To harvest idle computing power in desktop Grid, the jobs are migrated from source node to remote node there by utilizing the idle CPU time, memory, file system, database, information service and any other sharable resources required for execution from the network. In this view desktop Grid is used for executing a large number of jobs at dispersed resource sites.

Static scheduling algorithms assume that the complete state of the system is known in advance to the scheduler to make the scheduling decisions. Dynamic job scheduling algorithms makes scheduling decision without having the prior knowledge of the system state. Desktop grid is a dynamic environment, the state of the system changes over time. Job schedulers do not possess the prior knowledge about the state of the system. It is not known to the scheduler, in what environment the job will execute. The primary function of a job scheduling strategy is to recommend decision that improves the performance objective. Therefore the scheduler must consider the appropriate parameters for taking job scheduling decisions.

Generally, nodes connected in a desktop grid posses different clock speeds hence the same job will have varying processing times at different nodes (job processing time is calibrated accordingly) and this leads to varying turnaround times. In addition to this, Job transmission to remote nodes in a desktop grid involves transmission latency. When the job reaches to the remote node for execution the state of the system might change and the selected target node may not be the best node for the job to execute. To reflect this dynamism, job scheduling algorithms must consider the change in the load characteristics of the remote node when job is in transit from source node to remote node. Therefore, the choice of parameters that influences the performance objective of job scheduling strategy is the node dynamism and turnaround time offered by the node.

The proposed algorithm reliable node job scheduling strategy (RJSS) in this paper considers the change in load characteristics of the remote node while the job is in transit along with the turnaround time offered by the remote node.

The rest of the paper is organized as follows: Section 2 describes the related work in this aspect, section 3 describes the modeling issues and common assumptions considered, section 4 describes the problem statement, section 5 describes the mathematical model, section 6 describes the algorithm and section 7 describes the simulation results.

2. PROBLEM DESCRIPTION

Nodes connected in a desktop grid migrates their jobs from source node to target node for better turnaround time. The nodes are dynamic in nature and hence the load changes continuously with time at each node. Static scheduling algorithms migrates jobs based on the prior knowledge of the system state and dynamic scheduling algorithms do not possess the prior knowledge of the system state. Therefore, when a job is migrated from source node to destination node, the load conditions at the remote node might change while job is in transit and hence the selected target node for job migration may not be the best node for the job to execute further. This situation demands the need for the consideration of change in load characteristics of the remote node while job is in transit. We propose a new job scheduling strategy RJSS that address the following properties of the desktop Grid.

Heterogeneity: Nodes connected in the desktop grid are heterogeneous in nature and RJSS considers the processing capability of each node while making the scheduling decisions.

Dynamic: Desktop Grid user generates jobs at each node randomly and hence the load changes dynamically at each node.

Distributed: Each node in the desktop Grid takes scheduling decisions for the locally generated jobs. Therefore, RJSS is distributed in nature and has no central coordinator.

To make the model simple and realistic to the real desktop grid environment, the following modeling issues and common assumptions are considered. These considerations are made by taking into the account of field study conducted and found in the literature for the real grid environments.

2.1 Modeling Issues

- Each node *i* generates jobs on Poisson distribution with mean λ_i.
- The byte size of the generated jobs at each node *i* follows exponential distribution with mean μ_i.
- The time taken to complete/processing time of the job at each node i is chosen from normal distribution with mean φ_i and variance $\sigma^2{}_{\varphi_i}$.

2.2 List of Common Assumptions

- We assume that the nodes connected in a desktop Grid are reliable and does not undergo node failures and volunteer inferences.
- There is a fixed bandwidth between two nodes (B is constant).
- The Congestion towards a node is chosen fixed (C is constant).
- The nodes communicate via ISP server located at equidistant to all the nodes, therefore the logical distance between nodes becomes constant (D is constant).
- The size of the load packet that contains the remote node load information is assumed to be fixed for all the nodes. Therefore, p is constant.
- The jobs are independent. Each job has different size and processing time.
- When a node decides to send a job for remote execution, it is dispatched completely to the remote node in its entirety.
- The queuing discipline followed by the scheduler is FCFS.
- The size of the result for each job is same as the instruction/data size of the job and the result is sent to the source node after completing the execution.
- The mean job arrival rate (λ_i), mean instruction/data size of the job (μ_i), mean job processing time (φ_i), mean clock speed (c_i) for nodes in a desktop grid are chosen from sample space of normal distribution with appropriate mean and variances (Ex:- λ_i is chosen from a sample space of normal distribution with mean $\bar{\lambda}$ and variance σ_{λ}^2. Similarly same technique is followed for other parameters)

2.3 Problem Statement

"Let G be the desktop Grid consisting of N volunteer nodes. The nodes are spread in a geographically different locations. The nodes connected in the G are heterogeneous and posses different clock speed c_i. The nodes are connected over internet and communicate with each other via Internet Service Provider (ISP). The ISP is located at equidistance to all the nodes in the desktop Grid. The Desktop Grid user generates jobs at each node on Poisson distribution with mean λ_i. The mean instruction/data size of a job (bytes) for each node i follows exponential distribution with mean μ_i. The processing time of a job at node i follows normal distribution with mean φ_i and variance $\sigma^2{}_{\varphi_i}$. The objective is to find a target node for scheduling each job generated in the grid that minimizes the average turnaround time per job in the Grid."

3. RELATED WORK

This section describes some of the existing job scheduling algorithms in the field of study.

Authors in [2] evaluated scalable search methods for the selection of candidate host for the scheduling of job. The search methods are namely expanding ring search, random walk search, advertisement based search and rendezvous point search and CCOF.

In expanding ring search, when a client node needs idle cycles, it sends request to its immediate neighbors. If neighbors are busy or no enough candidates are available, it then sends the request to nodes on one hop farther. In random walk search, the client node sends the request to k random neighbors. In advertisement based search, when a node joins the system, it sends out its profile to its neighbors in the limited scope. The neighbors cache this profile along with the nodeID for future use. In rendezvous point search, a group of dynamically selected rendezvous points are used for information gathering. Hosts advertise their profiles to the nearest rendezvous point(s) and the client contact the rendezvous point(s) to locate available hosts. The experimental evaluation shows that the rendezvous point search performs better than other methods under light workload.

Nodes in a geographically wide spread grid environment contains day zone and night zone nodes. The nodes generally enter into day-time or night-time in the order of the time zones around the world. So the idle computing times are available on the human time scale. Authors D. Zhou and Virginia Lo in [3] took the advantage of the night-zone and day-zone nodes and presented wave scheduler algorithms

namely migration immediate, wave immediate, migration linger and wave linger algorithms.

In migration immediate, the client schedules the task on to a machine that is immediately available. When node fails the task is immediately migrated to a randomly available host. In wave immediate, the task is migrated to a night-zone machine. In migration linger, the task is allowed to stick onto the same host even after its failure for a random amount of time. If the host is still not available even after the specified time interval, the task then migrates. Fixing the amount of lingering time will influence the turnaround time of the job.

X. He et al. have presented Min-min algorithm [5] and Max-min algorithm [6]. In Min-min heuristic the shortest job is considered first for mapping onto a machine that offers earliest completion time. In Max-min heuristic the longest job is assigned to a node that offers minimum earliest completion time is chosen first.

Some research projects [10,11] have taken profit into account and applied economic models in grid resource scheduling. Incentive based scheduling presented in [8] is to build a global computational grid in which every participant has incentive to stay and pay in it. If the resource provider causes the job to miss its deadline, some penalty is imposed on the resource provider.

Maheswaran et. al [4] presented on-line and batch-mode heuristics for mapping independent tasks onto heterogeneous computing systems. In on-line mode, each task is considered once for matching and scheduling. The task need not wait for the next mapping event to occur for scheduling. As soon as the task arrives, the scheduler maps the tasks onto a machine, the selection of a machine is done based on the heuristics: machine that offers minimum completion time (MCT), minimum execution time (MET), switching algorithm, k-percent best and opportunistic load balancing.

All these scheduling heuristics are based on the greedy choices that depend on the transitory completion times of the jobs. These methods do not consider the information about the changing environmental variables like node dynamism and load changes when the job is in transit.

4. PROPOSED METHOD: RJSS

Each node in G has a potential job generator through which jobs are generated dynamically. The jobs are executed either locally or at the remote node based on the recommendations of RJSS. In RJSS, the selection of a candidate node for job scheduling is done based on the job turnaround time at the remote node and the change in load characteristics of the remote node when the job is in transit.

The scheduling strategy followed by RJSS is described below.

When a job *j* is generated at node *i* (J_{ij}), the node *i* collects the current load information (load packet *p)* from the remote nodes using *k*-random walk search. The load packet *p* possess the following information.

- The turnaround time for local job *j* at the remote node *l* along with transmission latency for J_{ij}.
- Probability that a new local job is generated at the remote node *l* when job J_{ij} is in transit to node *l.*
- Remote jobs traffic intensity at node *l* when J_{ij} is in transit to node *l.*

RJSS schedules the J_{ij} at remote node *l* when the following performance criterion is met in *k* attempts.

- The turnaround time for J_{ij} at node *l* is less than the turnaround time at node *i*.
- The probability of generating a new local job at node l while J_{ij} is in transit is below the user defined threshold value and
- The traffic intensity due to new remote jobs at node l while J_{ij} is in transit is below the user defined threshold value.

When RJSS does not encounter a candidate node in k attempts, the job J_{ij} is scheduled at local node for execution.

6. ALGORITHM: RJSS

6.1. Algorithm RJSS

begin

Generate N nodes and perform the following steps concurrently;

for each node *i* in the grid G ***do***

generate job *j* following the arrival distribution of the node *i*;

for each unscheduled job *j* ***do***

Perform random walk and choose a node *l* from the Grid;

for each chosen node *l* from *k*-random walks ***do***

Compute waiting time w_{ij}^{lt} for job *j* at node *l;*

Compute job transmission latency L_{ij} to node *l*;

Let $t' := L_{ij}$;

Compute turnaround time T_{ij}^{l} for job *j* at node *l*;

$$T_{ij}^{l} := w_{ij}^{lt} + L_{ij}$$

Compute probability of new local job $P_l^{t'}$ during t';

$$P_l^{t'} := (\lambda_l t')e^{-\lambda_l t'}$$

Compute average number of remote job arrivals per unit time at node *l*;

$$Q_l^t := \frac{1}{t}\sum_{j=1}^{m_r^t} \begin{cases} 1 & if\ J_{rj}\ is\ scheduled\ at\ node\ l \\ 0 & otherwise \end{cases}$$

where $1 \le r \le N$

if $(T_{ij}^{l} < T_{ij}^{i})$ ***and*** ($P_l^{t'} < 0.5$ and $t' Q_l^t < 1$) ***then***

Schedule job *j at* node *l;*
Mark job *j* of node *i* as scheduled;

Break;

endif

endfor

if (job *j* is unmarked) ***then***

Schedule job *j* at local node *i*;

Mark job *j* as scheduled;

endif

endfor

endfor

end;

6.2. Accuracy of Prediction

An interesting question is whether our proposed model is applicable in the real situations since the derived model is based on the probabilistic estimation of jobs generated at the remote node when the job is in transit. To verify the correctness and effectiveness of the proposed method, we have measured the prediction accuracy of the proposed method with the actual load value realized during the simulation.

The performance metric used to evaluate the accuracy of the prediction in the proposed model is defined as

$$\left|\frac{predicted\ load - actual\ load}{actual\ load}\right|$$

In our simulation, the mean arrival rate of the jobs ($\bar{\lambda}$) in the grid is varied in the range from 30 to 50 jobs per 1000Sec, mean processing time of the job ($\bar{\psi}$) in the grid ranges from 20Sec to 40Sec, mean byte size of the job ($\bar{\mu}$) ranges from 5KB to 40KB and mean clock speed of the node ($\bar{c}$) in the grid ranges from 200MHz to 500MHz.

Fig. 1 gives the mean and standard deviation of prediction error with different nodes ranging from 100 to 500. As we can see, the prediction error is less than 70% for most of the cases. As the number of nodes increases, we find that the prediction error decreases and becomes stable. Therefore, we conclude that our model is stable with the increase in the number of nodes.

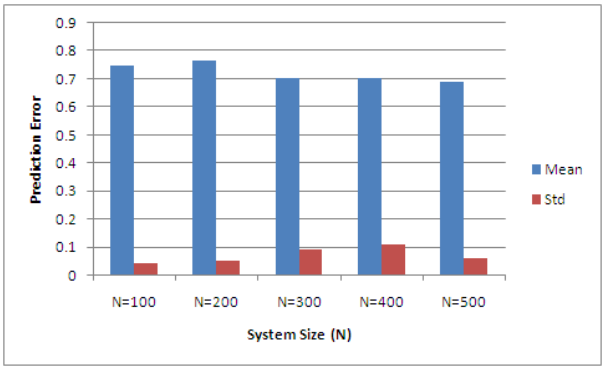

Fig 1. Mean and STD of prediction error with different nodes

6.3. Choosing 'k' in a k-random walk

As it is described in the section 3.1 RJSS performs a *k*-random walk to select a target node for the scheduling of job. The value of *k* chosen in the k-random walk is 10% of the number of nodes (N). This is determined by analyzing the performance of RJSS for different values of *k*. *k*- value is varied between 5% to 100% of the number of nodes (N). Fig. 2 shows the change in $\bar{G}^t$ for different values of *k* and it is shown that the best results are obtained when *k* is taken between 10% to 20% of N nodes in the grid. Therefore, we have chosen 10% of nodes as *k*-value in our simulation.

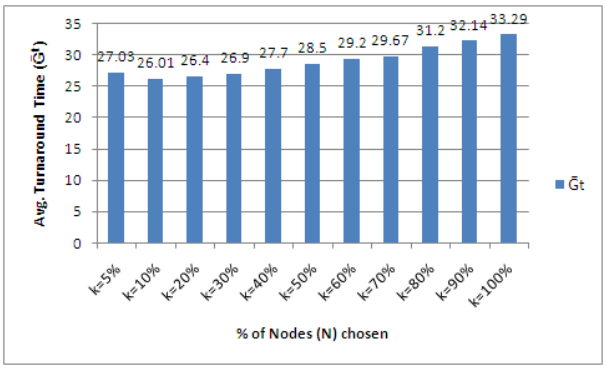

Fig 2. Change in $\bar{G}^t$ for varying *k*-value

7. SIMULATION SETUP

To conduct the experiment and analyze the performance of scheduling strategies, a simulated desktop grid is designed using object oriented system design. An instance of a node that simulates the behavior of a node in a desktop grid environment is shown in Fig 1. The node is designed with entities like a job generxator, a job dispatcher, a decision policy, and a job scheduler to generate events for simulation based on their predefined parameters. An instance of a node that shows the interconnection among these entities is shown in Fig 1.

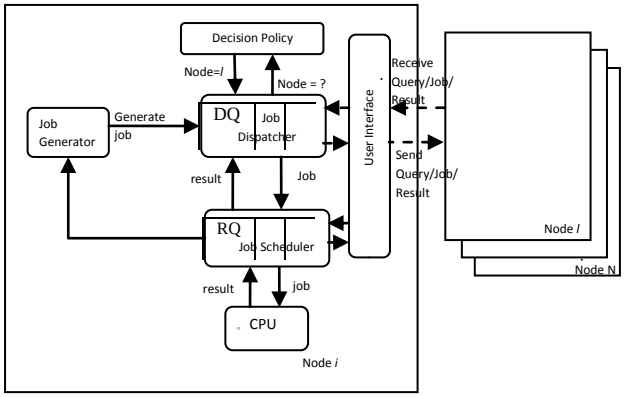

Fig 3: An instance of a node *i* in the Grid

7.1. Description of entities in the node:

$UserInterface_i()$:- The nodes communicate with each other by passing messages through user interface. User interface forward messages to the other nodes and receive messages from other nodes via central ISP server located at equidistant to all the nodes.

$Job\_Generator_i()$:- Each node has a local job generator. It generates jobs for each node i on Poisson distribution with mean λ_i. When a job is generated, it is sent to the local job dispatcher queue (DQ).

$Job\_Dispatcher_i()$:- Job dispatcher possess a Dispatcher Queue (DQ_i). It maintains the queue of locally generated jobs to be scheduled and results of the completed jobs to be dispatched. For each unscheduled job, it makes a scheduling decision based on the recommendations of decision policy. Based on the recommendations of decision policy, the job is marked scheduled and is either sent to the local scheduler or migrated to the remote node.

Decision_Policy$_i$() :- It is a part of the job dispatcher and it makes scheduling decision for each unscheduled job about the node at which the job is to be executed. To make scheduling decision, it computes the turnaround time for local job at the remote node and traffic intensity probability at the remote node while job is in transit by collecting control (load) information from remote nodes using *k*-random walk search.

Job_Scheduler$_i$() :- The Job scheduler of each node possess a ready Queue(RQ$_i$). It maintains the queue of jobs that are scheduled locally. The queuing discipline followed by the scheduler is first come first serve (FCFS) without preemption basis. Scheduler schedules the jobs from RQ to the local CPU$_i$. Upon the completion of the currently executing job, the scheduler sends the result to the DQ. Local dispatcher dispatches the result to the source node (local/remote).

CPU$_i$():- The CPU is the processing unit for each node. It executes the currently assigned job without pre-emption. The status of the CPU will be busy during the execution of a job. The status of the CPU becomes idle as soon as the completion of the currently executing job.

7.2. Data/Control Flow among Entities

The sequence of data and control flow among the entities shown in Fig. 3 is described below.

1. Job_Generator$_i$() generates the job *j* for each node *i.*
2. Job_Dispatcher$_i$() interact with the decision policy for each unscheduled job *j* for making scheduling decision.
3. Decision policy performs a *k-random walk* search in association with the job dispatcher and interacts with the user interface to fetch control information from randomly chosen remote node *l*.
4. The control information from remote node *l* obtained by user interface is forwarded to the decision policy.
5. The decision policy computes the turnaround time for local job at remote node along with the probabilistic estimate of local/remote jobs at the remote node while job is in transit.
6. Decision policy recommends the local node *i* or remote node *l* based on turnaround time offered and probabilistic estimate of the traffic intensity.
7. The job dispatcher schedules the job at local scheduler *i* or migrates to the remote node *l* based on the recommendations of decision policy.
8. The job scheduler submits each locally scheduled job for CPU execution and collects the results upon the completion of the execution of the same.
9. Job dispatcher dispatches the results to the source node.

8. SIMULATION RESULTS

The simulation is done using object oriented programming through java and the experiment is conducted by varying the parameters like number of nodes (N), mean arrival rate of the job in the grid ($\bar{\lambda}$), mean size of the job in the grid ($\bar{\mu}$), mean processing time of the job in the grid ($\bar{\psi}$), and mean clock speed of the node in the grid ($\bar{c}$). The performance of RJSS is compared against the algorithm Resource Exclusion (RE) by plotting the graphs for average turnaround time per job in the Grid $\bar{G}^t$ with respect to the number of jobs finished execution u^t.

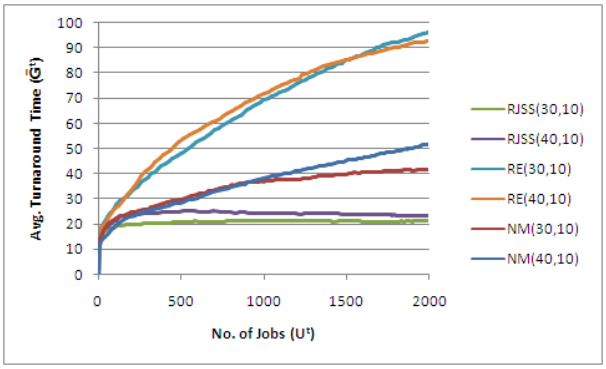

Fig 4. The effect of varying $\bar{\lambda}$ on $\bar{G}^t$ with respect to u^t

Fig. 4 is drawn to study the effect of varying $\bar{\lambda}$ on $\bar{G}^t$ with respect to u^t for the fixed parameters N=100, D=50Km, C=0.1Sec, $\bar{\mu} = 5KB$, $\bar{\psi}$=30Sec, $\bar{c} = 200MHz$. The mean ($\bar{\lambda}$) and variance (σ_{λ}^2) of the job arrival rate in the grid is shown in the parenthesis in Fig. 4. As $\bar{\lambda}$ increases at each node, the system becomes much busy with their local jobs and has less idle computing cycles. The property of RJSS in estimating the futuristic load characteristics of remote node makes RJSS to take better scheduling decisions. It is observed in Fig. 4, that $\bar{G}^t$ for RJSS is less compared to RE and NM for changing values of mean arrival rate of jobs in the Grid. RJSS achieves better turnaround time from the early stages of scheduling process compared to RE and NM algorithm. It is also observed that the changing $\bar{\lambda}$ causes variation that is indirectly proportional to the variation in $\bar{G}^t$.

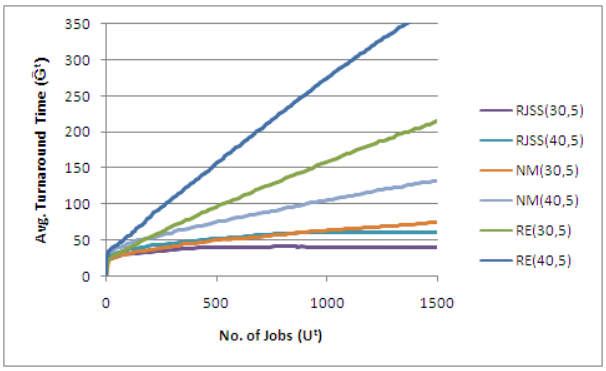

Fig 5.effect of varying $\bar{\psi}$ on $\bar{G}^t$ with respect to U^t

Fig. 5 is drawn to study the effect of increasing mean processing time of the job $\bar{\psi}$ on $\bar{G}^t$ and U^t with fixed N=100, D=50Km, C=0.1Sec, $\bar{\lambda}$=20Sec, $\bar{\mu} = 5$KB, $\bar{c} = 20$MHz, $\bar{f} = 0$ and $\bar{r} = 0$. The increasing $\bar{\psi}$ generates jobs with higher processing time and hence jobs require longer time to finish execution. This property makes nodes busy and influence on the average turnaround time of the system. However RJSS for its effective scheduling decisions, makes it to perform better than RE and NM. From Fig. 4, it is observed that the increasing values of $\bar{\psi}$ has little effect on RJSS than RE and NM.

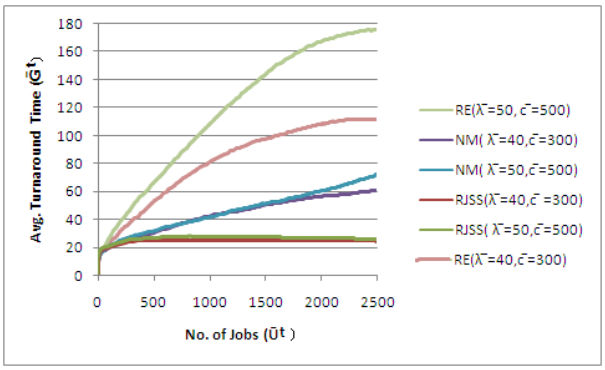

Fig 6. effect of varying $\bar{\lambda}$ and $\bar{c}$ on $\bar{G}^t$ with respect to U^t

Fig. 6 is drawn to study the effect of increasing $\bar{\lambda}$ and $\bar{c}$ on $\bar{G}^t$ and u^t with fixed N=100, D=50Km, C=0.1Sec, $\bar{\lambda}$=20Sec, $\bar{\mu} = 5KB$, $\bar{\psi} = 30Sec$. The increasing $\bar{\lambda}$ increases frequency of job arrivals and $\bar{c}$ makes the availability of large number of high speed processing nodes in the Grid. When a job is migrated to a high clock speed node it could complete at much faster rate compared to a node with lower clock speed machine. Thus jobs get completed at faster rate and nodes become idle soon. This makes lot of idle CPU cycles available in the Grid and hence executes the jobs at faster rate. The values chosen for $\bar{\lambda}$ and $\bar{c}$ is shown in parenthesis in Fig. 6. It is observed that the increasing $\bar{\lambda}$ does not influence the performance of RJSS as the increasing $\bar{c}$ compensates that. But, increase in $\bar{\lambda}$ increases the queue length of the jobs in NM and choosing the high speed nodes in RE under utilizes the slower nodes in the Grid, thus NM and RE increases $\bar{G}^t$with U^t. From this, it is observed that RJSS performs better than RE and NM with varying $\bar{\lambda}$ and $\bar{c}$.

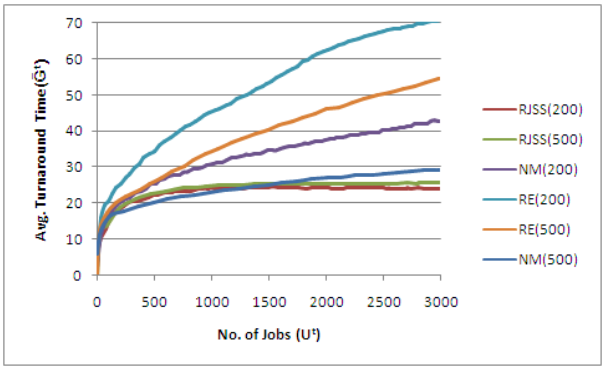

Fig 7. Effect of varying N on $\bar{G}^t$ with respect to U^t

Fig. 7 is drawn to study the effect of increasing number of nodes N on $\bar{G}^t$ and u^t with fixed D=50Km, C=0.1Sec, $\bar{\lambda}$=20Sec, $\bar{c} = 20MHz$, $\bar{\psi} = 30Sec$, $\bar{\mu} = 5KB$. The N value chosen is shown in the parenthesis in Fig. 7. The increasing N makes more number of nodes to qualify in the *k*-random walk search and hence increases the decision making delay. This influence on $\bar{G}^t$and hence a slight variation is observed in the performance of the RJSS for increase in the number of nodes. However, RJSS performs comparatively better than RE and NM algorithms.

9. CONCLUSIONS AND FUTURE SCOPE

The performance of RJSS is discussed in the previous section for varying the parameters of the Grid. For all the cases it is observed that RJSS performed better than RE and NM algorithms.

The nodes considered in this paper are assumed to be dedicatedly available for public execution. But, in a real desktop grid environment, the volunteer nodes undergo inferences due to high priority node owner's jobs. Volunteer autonomy makes the volunteer nodes join or leave the grid at any instant of time and hence volunteer nodes will be available in the desktop grid with different volunteer times. We want to consider this factor into account to extend the proposed algorithm further and study its performance.

10. REFERENCES

[1] SungJin Choi, Rajkumar Buyya, " Group-based adaptive result certification mechanism in Desktop Grids", Future Generation Computer Systems 26 (2010) 776_786, Science Direct.

[2] V. Lo, D. Zhou, D. Zappala, Y. Liu, and S. Zhao, "Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet," The 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), LNCS 3279, pp.227-236, Feb. 2004.

[3] Zhou and V. Lo, "Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-to-peer Desktop Grid Systems," 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05), LNCS 3834, pp. 194-218, Jun. 2005.

[4] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems, *The 8th Heterogeneous Computing Workshop (HCW'99), pp. 30-44, Apr. 1999.*

[5] He, X., X-He Sun and G.V. Laszewski, "QoS guided Min-min heuristic for grid task scheduling. Journal of computer science and Technology", Vol. 18, pp. 442-451, 2003.

[6] Estiminani, K. and M. Naghibzadeh. A Min-min and Max-min selective algorithm for Grid task scheduling. The third IEEE/IFIP International conference on Internet, 2007, Uzbekistan.

[7] SETI@home, http://setiathome.ssl.berkeley.edu

[8] Y. Zhu, L. Xiao, Z. Xu, L. M. Ni, "Incentive-based scheduling in Grid computing," Concurrency and Computation: Practice and Experience, vol. 18, issue 14, pp. 1729-1746, Dec. 2006.

[9] Distributed.net, http://distributed.net

[10] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing," *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, pp. 1507-1542, 2002.

[11] S. Shetty, P. Padala, and M. Frank, "A Survey of Market Based Approaches in Distributed Computing," Technical Report TR03-13, 2003.