A Group based Fault Tolerant Scheduling Mechanism to Improve the Application Turnaround Time on Desktop Grids

Mohammed Khan[1], Irfan Hyder[2], Ghayas Ahmed[2], and Saira Begum[2]
[1]PAF-Karachi Institute of Economics and Technology, Pakistan
[2]Institute of Business Management, Pakistan

Abstract: *Desktop grid is an exciting discipline for high throughput applications but due to inherent resource volatility, desktop grids are not feasible for short lived applications that require rapid turnaround time. Efficient and more knowledgeable resource selection mechanism can make it possible. In this paper, we propose a group based resource scheduling mechanism. The groups are made by using three measures: Collective impact of CPU and RAM, spot checking and task completion history. We evaluated the proposed mechanism over a network of 900 nodes having varied resources and behavior and found that excluding desktop resources on the basis of just clock rates is not a good idea and RAM should also, be considered as a collective parameter besides spot checking and task completion history. We also, show that the appropriate scheduling mechanisms can only be implemented after the grouping of resources on computing strength and behavior. The proposed mechanism ensures that tasks are allocated to hosts with higher probability of tasks completion that reduces tasks failures and improves fault tolerance.*

Keywords: *Scheduling mechanism, fault tolerance, desktop grids.*

Received May 15, 2013; accepted September 19, 2014; published online April 1, 2015

1. Introduction

Gone were the days when super computer was the only option for the high end computing and storage. Desktop grids also known as Volunteer Computing (VC) has laid down a much cheaper path towards the same. It is indeed an era where the abundance of communication bandwidth has open up new horizons. Desktop grid is one of them through which the utilization of the idle processing cycles and memory of millions of users connected on the internet or through any other type of network become possible. The infrastructure of desktop grid system is based on three entities:

- Master is responsible for maintaining the list of hosts/volunteers. Assignment of tasks to the hosts, getting back the results and communicating it back to user is also master's job.
- Volunteer/host is responsible to register with master to show the intent of sharing processing cycles and memory. Its job is to complete the assigned task and send the results to master.
- User is responsible to submit jobs to master which will later sent to hosts.

Various successful projects are already using desktop grid platform such as computing against cancer, GIMPS, FightAidsAthome, Seti@Home [5, 6, 7, 11]. Desktop grid provides an environment in which large scale of computation can be performed without having huge IT infrastructure which ultimately reduces infrastructure cost.

In desktop grid environment hosts are much volatile in terms of host availability and/or CPU availability. Host un-availability refers to a scenario in which host is not connected to desktop grid software and CPU un-availability refers to a scenario in which host is connected to desktop grid software but CPU idle cycles are not available [1]. Desktop grid also suffers from non-reliable or erroneous results submitted by the hosts un-intentionally and/or intentionally. To cater these problems server creates replicas of tasks so that in case of task failure it can be replicated to other workers. Task failure causes tasks to be restarted from scratch on other hosts [1] which in turns cause delay in application completion.

Efficient utilization of resources in desktop grid is a challenging task, which can be achieved through efficient resource management. Volatility is an inherent feature of resources on desktop grids. Applications having large number of tasks as compared to hosts are used in traditional desktop grids. Hence, resource management and application scheduling techniques play a key role in such environment. In this paper, we consider an application having independent and identical tasks where as application performance is measured in terms of application's turnaround time.

Our work focuses on the minimization of turnaround time of a single application instead of multiple applications. Maintaining fairness among jobs/tasks, replication and bandwidth utilization is not

taken into account in our study. The proposed mechanism is primarily based on creating host groups focusing on three aspects that include collective impact of CPU and RAM, spot check and task completion history. To evaluate our proposed mechanism simulation is performed using the traces obtained from educational institution's computer labs.

2. Related Works

In [10] resources are selected and eliminated according to the clock rates and task completion prediction whereas our proposed mechanism doesn't eliminate any worker, rather it places them into groups according to their processing capability i.e., collective impact of CPU and RAM and task completion history. We do not eliminate any worker on the basis of hardware strength because a worker may possess less hardware capability but can provide more processing time and/or its hardware strength may get better.

In [4] grouping mechanism based on volunteer autonomy failure, availability and service time is proposed whereas groups are managed through mobile agents selected from each group. Whereas, our proposed mechanism emphasizes on volunteer's collective impact and task completion history which ultimately shows liveliness of volunteers. In contrary, instead of volunteers, we left the scheduling responsibility to the server due to volatile nature of volunteers.

In [9] replication mechanism is proposed based on volunteer grouping. Our proposed mechanism is different in several ways i.e., we are focusing on average percentage of cumulative impact of hardware resources for group placement of volunteers, where average CI should be greater than 75% and arranging volunteers in groups is based on number of task completed from assigned tasks.

Anderson and Fedak [2] concluded that evaluation of hardware resources individually is not appropriate i.e., higher disk space will play its role if proper network bandwidth is available to access it, similarly higher processor speed could positively affect processing if more RAM is available to it. To overcome the issues arises while focusing on hardware resources individually. Our proposed framework evaluates collective impact of hardware resources to place volunteers in a group.

Watanabe *et al.* [14] proposed a mechanism to reduce the overhead occur as a result of spot-checking hence to reduce computation time in VC. Our proposed mechanism doesn't cater the spot-check rate rather marks hosts as saboteur if returns erroneous results trice.

Toth and Finkel [13] investigated the effects of task retrieval policies on task completion and concluded that volunteers having download early task retrieval policies completes more tasks as compare to other policies. Similarly, screen saver on mode completes fewer tasks than screen saver off mode. In contrary, our proposed mechanism also focuses on improving tasks completion time but measures of address are hardware collective impact, task completion history. Silaghi *et al.* [12] constructed an emulation based system to evaluate scheduling policies. It is concluded that EDF reduces CPU wastage, devoting all processing cycle to 1 project improves throughput.

Hanandeh *et al.* [8] presented a dynamic replication strategy that considers the storage capacity of the grid node. This work enhances the fast spread mechanism by concentrating on the feasibility of replicating the requested replica on each node among the network.

3. Scheduling Short-lived Applications on Desktop Grids

3.1. Problem Definition

In this study, we are scheduling an application consisting of independent and identical tasks whereas hosts are in order of magnitude of tasks. Hosts are individually managed by their users or owners and takes part in desktop grid computation only when their CPU cycles are idle, due to which hosts are volatile.

Figure 1 shows throughput using FCFS mechanism that is also used in many desktop systems [3, 5, 6, 7, 11] for applications having 450, 900 and 1800 tasks whereas number of hosts are 900. However, each task takes 15 minutes to complete on dedicated 2.0GHz processor. The figure indicates initial edge after which application progresses approximately linearly. It is observed that in case of 450tasks 90% of tasks were completed in 21minutes whereas rest of the tasks took 41minutes that means, last 10% of the tasks took almost equal time to complete as compared to initial 90% of the tasks. Same behavior was observed with application of 900 and 1800 tasks. It can be said that fast host have completed their assigned tasks due to which an initial edge was observed whereas slow hosts caused task failure that resulted in application delay. On the basis of above simulation, we hypothesize that slow hosts due to their clock rate and hosts' having variations in their availability can cause task failure which ultimately results in application delay hence causes increases in application's turnaround time.

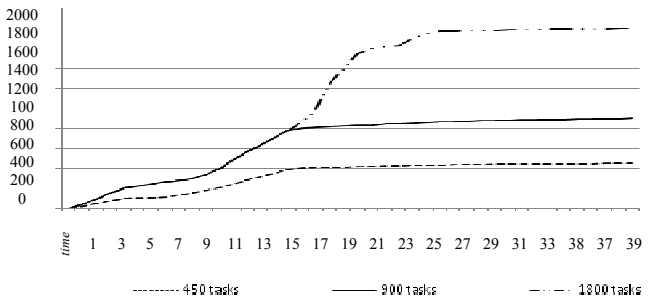

Figure 1. Application completion time of 450, 900 and 1800 tasks applications.

3.2. Proposed Mechanism

The proposed mechanism is based on a grouping through which hosts are categorized. Due to the fact that hosts in desktop grid exhibit wide variety of behaviors, simple scheduling mechanism will not be sufficient. If we can somehow group the hosts that exhibit some common features then relevant scheduling mechanism can be enforced in order to reduce application turnaround time. Before explaining the details of our proposed grouping mechanism, first we would like to elaborate the factors that will be used to form groups. These factors are:

- Collective Impact of CPU and RAM: The work units in a desktop grid environment are assigned to hosts on various basis. One of the most common mechanisms is to check the processor speed and reject the hosts that have low processing power. Our submission is that the rejection of host only on the basis of one parameter is not a good idea and other parameters should also be used to accept or reject a host. This idea is also supported by [10] which conclude that hardware resources may be evaluated in combination i.e., disc space is useful if appropriate network band width is available to access it. Similarity processor speed is useful when substantial RAM is available. To analyze the impact of the CPU and RAM relationship, we developed and executed an application on dot net platform. We executed the application on 2.8GHz. CPU having just 512MB RAM and calculated the execution time. Gradually we lowered the CPU speed by changing the CPU and increased the RAM. The experiment used the CPUs ranging from 1.8GHz. to 3.0GHz and RAM from 512MB to 4GB. As per [10] the execution time of application showed very little change which proves that the RAM should be counted as a collective parameter with CPU. In Table 1, we have summarized and grouped the CPU and RAM available on our infrastructure and assigned a numeric value on the basis of their effectiveness in task execution. These numeric values will be used for calculating the overall impact of the mechanism.

Table 1. Grouping of available CPU and RAM.

CPU-Speed Scoring Scheme	Value
Less than or equal to 1.8 GHz	1
Greater than 1.8GHz and less than or equal to 2.2 GHz	2
Greater than 2.2GHz and less than or equal to 2.6 GHz	3
Greater than 2.6GHz and less than or equal to 3.0 GHz	4
Greater than 3.0GHz	5
RAM-Availability Scoring Scheme	**Value**
Less than or equal to 512 MB	1
Greater than 512 MB and less than or equal to 1 GB	2
Greater than 1GB and less than or equal to 2 GB	3
Greater than 2GB and less than or equal to 4 GB	4
Greater than 4GB	5

- Spot Checking: It is the way to identify a host acting as saboteur. In spot checking, tasks whose result is already to known to server is sent to host and after the submission of result by the host it is crossed checked with server's own result. If result matches the host is marked as non-saboteur otherwise saboteur [6]. The process of spot checking is cyclic.
- Task completion History: Task completion history states what percentage of assigned tasks has been completed before deadline. The deadline is defined by the owner of the application [3]. It is a major factor to place the host in a more reliable group.
- Grouping Mechanism: The overall grouping mechanism is depicted in Figure 2. In the proposed mechanism when a host logins, its hardware strength check is performed. In this process host's processor's speed and RAM size is gathered and on the basis of Table 1 numeric values are assigned. After assigning values, percentage of scored marks are calculated e.g., if a host scores 4 marks for processor and 3 marks for RAM than the overall score will be 7/10 i.e., 70% marks.

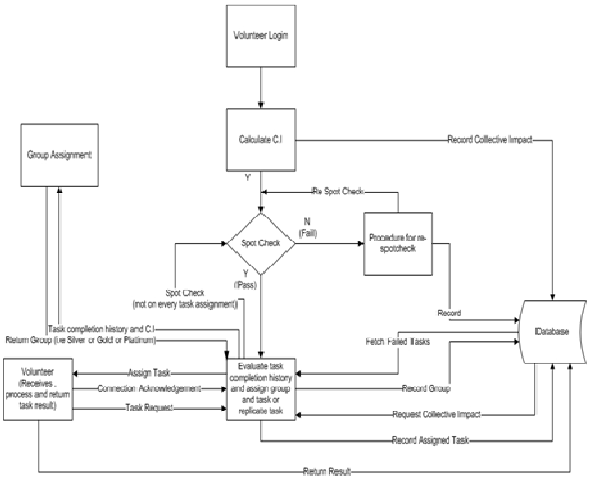

Figure 2. Overall proposed grouping mechanism.

Proposed mechanism uses three groups i.e., silver, gold and platinum to categorize hosts. If a host scores minimum of 75% marks in hardware strength check then it will be placed in either platinum or gold group based on previous task completion history, otherwise it will be placed in silver group.

Initially, all the hosts are placed in silver group because of the unavailability of task completion history. As the application execution progresses, hosts task completion history will change that will result in the updation of groups [host will be moved from silver to gold or from silver to platinum groups]. If the host had scored >=75% marks in collective impact and was able to complete 80% or more tasks assign to it correctly and before the deadline then the host will be placed in platinum group. Whereas If a host had scored >=75% marks in collective impact and percentage of task completion history is in between 60% to 79.9% than the host will be placed in gold group, Similarly, If a host had scored <75% marks in collective impact than the host will be placed in silver group. These threshold values are kept configurable in the system and optimal values are deduced through

experimentation. Hosts are arranged in a ready queue in order of their groups. Platinum group possess higher priority than gold group. Similarly, gold group possess higher priority than silver group.

Algorithm 1: Host Configuration (input login id and session id).

```
VarProcessorSpeed = Fetch_Processor_Speed()
VarRAMSize = Fetch_RAM_Size().
VarLoginId = Session (" loginID")
Fetch marks for each component
Calculate overallpercentage
SaveInfoToDB(VarProcessorSpeed,VarRAMSize,VarLoginId).
```

First of all host configuration is performed as shown in Algorithm 1. After selecting a host from hardware capability check, test task for the purpose of spot checking is assigned by using Algorithm 2. If a host fails in spot checking, it will be placed in silver group, whereas a host passes spot check will be placed in gold or platinum group according to its task completion history. If a host fails spot check thrice, it will be marked saboteur and will not be assigned tasks in the future. The rejoining of saboteur hosts will be based on configurable heuristics depending on the server workload and user's (Application submitter) preference. These heuristics can be X number of days, X tasks completed etc., in the suggested approach actual tasks are taken as test task due to which hosts in desktop grid will not be able to identify whether assigned task is a test task or actual task. Furthermore, use of actual tasks as test tasks will eliminate the work load of test task creation except for the case when the first host joins in.

Algorithm 2: Spot check (input login id and session id).

```
assign test job to the host
VarNoOfSpotCheckFailed=0
When host return result master will check whether the task is
real or test task.
    If test task true
            Fetch masters own result of that test task
        If masterresult=hostresult
            SaveHostToDB "PASS"
        Else
            VarNoOfSpotCheckFailed =
        VarNoOfSpotCheckFailed +1
            If VarNoOfSpotCheckFailed=3
                    SaveHostToDB "Fail"
    Else
        PerformValidationforActualTask( )
```

In the suggested mechanism, task completion history evaluation is a cyclic process which takes place whenever host requests for task. This process evaluates what percentage of assigned tasks has been completed by particular hosts by using Algorithm 3. If a host scores collective impact >=75% and has completed>=80% of the assigned tasks then host will be placed in platinum group, if task completion percentage in between 60% to 79.9% then it will be placed in gold group. Hosts having collective impact <75% will be placed in silver group.

Algorithm 3: Group placement function (be executed when host request for new task.

```
VarPercentageCompleted, VarCollectiveImpact
VarPercentageCompleted=getTaskCompletedPercent()
VarCollectiveImpact=getCollectiveImpact()
If VarPercentageCompleted  >= 80%  and
VarCollectiveImpact>=75%
  Assign platinum group
elseif  VarPercentageCompleted (in between 60%  to 79.9%)
and VarCollectiveImpact>=75%
   Assign gold group
else
  Assign silver group
MaintainHostReadyQueue(orderbygroup)
```

4. Experimental Methodology

Our experiment is based on simulation driven by traces. Our intention is to evaluate how much improvement can be achieved through our proposed mechanism as compared to the other such mechanisms. We have compared our proposed mechanism with FCFS scheduling which simply schedules jobs on first come first serve basis and used by most VC system in recent time [3, 5, 6, 7, 11]. In addition to FCFS, PRI-CR-Excl is also used for comparison purpose which excludes hosts having clock rates below some defined clock rate threshold value because we claim that RAM should also be considered as a parameter for such decision making [1]. BOINC is the implementation platform [3].

4.1. Trace Dataset and Dataset Gathering Method

Traces were collected by submitting tasks of fixed length i.e., 15minutes, which performs integer and floating point operations under an infinite loop. Submitted tasks perform computations and write their computation rate periodically to a file. Through this file, server can evaluate host availability and CPU availability. Host availability indicates the time interval in which host system is connected to desktop grid software, whereas CPU availability shows that the host is connected to desktop grid software and also performs the assigned tasks [1]. Instead of host availability and CPU availability, our proposed mechanism primarily focuses on number of tasks completed by hosts which ultimately aggregates both types of availabilities.

Traces were gathered from a university named PAF-KIET where computer labs can be categorized into two types depending on their usage and types of processing units they possess. General purpose labs have an aggregate of 300PCs with an average f 2.0GHz processor speed whereas special purpose labs (used for practical classes) have an aggregate of 600PCs with an average to 2.8GHz processor speed. The average size of RAM in general labs is 2GB whereas special labs possess an average of 3GB RAM. All the three

campuses have a total of 15labs out of which 5 labs are for general use whereas 10labs are for practical classes. Host available in general labs are often busy due to which availability interval to process tasks are not so long however hosts available in practical labs provide longer availability of processing cycles. The clock rates are not widely dispersed which means majority of the hosts possesses nearly the same clock rate i.e., in between 1.8 to 3.2GHz.

4.2. Simulated Applications

Applications varying in number of tasks and task size were taken into account for simulation. Task size has direct relation with task failure i.e., increase in task size increases task failure in linear fashion, as concluded in [1] we used applications consisting 450, 900 and 1800 independent tasks for simulation. Furthermore each application has three instances consisting 5, 15 and 35 minutes of tasks respectively. Experiments for varied number of tasks and task sizes are performed separately using proposed mechanism, FCFS and PRI-CR-Excl. Available number of hosts are 900, so the selection of 450, 900, 1800 tasks is only due to symmetric evaluation of results when tasks are half, equal and double than available number of hosts. The rationale behind this approach is based on evaluation, when the number of tasks are half as compared to the available number of hosts than the master has little chance to assign task to slow host whereas when tasks are equal to number of hosts then the chance of task assignment to slow host increases. In the case of twice as many tasks compared to hosts, master will be left with no option but to assign tasks to slow hosts.

4.3. Performance Metrics

We hypothesize that slow host and host which returns erroneous results causes delay in application completion. As our overall aim is to reduce the applications turnaround time in desktop grid environment that is why our performance metrics is tasks makespan, if the individual task's make span is reduced than the overall application's turnaround time will automatically be optimized. We compare the proposed mechanism with FCFS and PRI-CR-Excl on the basis of the above mentioned performance metrics. We did stepwise comparative analysis to analyze the impact of each step i.e., hosts selection based on collective impact of CPU and RAM without grouping section 4, host selection based on grouping mechanism section 5.

5. Host Selection based on Collective Impact of CPU and RAM

In this method we focus on host's hardware strength in terms of CPU and RAM. Both components are assigned a numeric value according to the ranges defined in Table 1, then aggregate of those assigned values are calculated. If a resource obtains 75% or higher marks then it will be allowed to serve as volunteer. Key idea behind the approach is to eliminate slow hosts which might cause task failure and application delay. Simulations were performed to observe the difference between FCFS, PRI-CR-Excl and resource selection based on collective impact of CPU and RAM.

Figure 3 shows make span achieved for applications having 450, 900, 1800 tasks through FCFS, PRI-CR-Excl and collective impact. It is observed that the host selection based on collective impact shows better makespan as compared to FCFS and PRI-CR-Excl in all cases. The reasons are FCFS must have selected slow hosts as it performs scheduling on first come first serve basis whereas PRI-CR-Excl must have rejected hosts on the basis of low CPU speed. On the other hand the collective impact would neither have chosen slow hosts nor have rejected hosts just on the basis of CPU because it considers RAM as well in decision making.

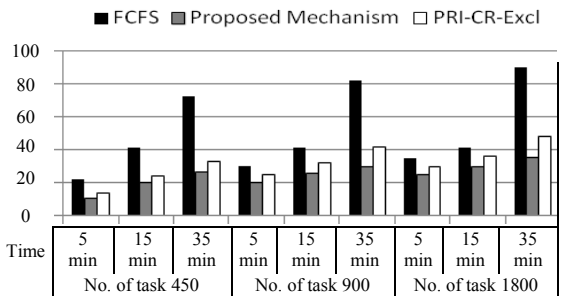

Figure 3. Application completion time using FCFS, Collective Impact and PRI-CR-Excl.

Furthermore, applications having tasks with length of 5 min show least makespan than the applications having 15 or 35 min length. The reason behind this is the granularity of tasks due to which even the host whose availability interval is short, can complete the task. In addition to this, 450 tasks are less than the available number of hosts which allows scheduler to assign tasks to hosts whose collective impact is relatively high. In case of 900 and 1800 tasks makespan is increased as compared to 450 tasks which shows that scheduler has assigned tasks to such hosts that were not considered in case of 450 tasks. As the number of tasks would increase scheduler will have to assign tasks to weak hosts.

In the coming section we will present the impact of overall grouping mechanism that not only include collective impact of CPU and RAM but also spot checking and task completion history. Addition of these mechanisms have improved the results because these methods have eliminated the saboteurs (intentional or un-intentional) and have considered the host past performance.

6. Host Selection based on Grouping Mechanism

Here we examine the application turnaround time of FCFS, PRI-CR-Excl and proposed grouping mechanism based on collective impact of CPU and RAM, spot check and task completion history).

Comparison of Figures 3 and 4 shows that the grouping mechanism not only out performs the FCFS and PRI-CR-Excl but also improves the results of collective impact of CPU and RAM (when used independently). Task assignment to slow host in FCFS must have caused task failure that has enforced scheduler to assign those tasks to other hosts for execution form the scratch. PRI-CR-Excl performed better than FCFS but consumed little more time than proposed grouping mechanism. PRI-CR-Excl excludes resources which possess slow clock rate but because of enhanced RAM size, may have completed the task. Due to this factor PRI-CR-Excl causes wastage of CPU cycles.

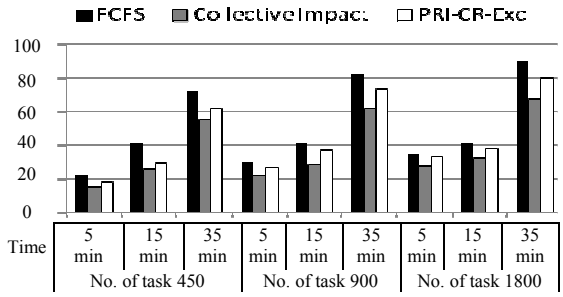

Figure 4. Shows applications completion time using FCFS, PRI-CR-Excl and proposed mechanism.

Proposed grouping mechanism performed better because it provides a facility in which resource's internal processing capability is measured cumulatively due to which some slow clock rate hosts having higher RAM can join desktop grid which eliminates the aspect of CPU wastage, so by the utilization of such CPU cycles proposed grouping mechanism improves application latency.

Furthermore, grouping mechanism selects resources from ready queue which contains volunteers arranged according to their groups. Resources of Platinum group possess better collective impact and better task completion history due to which these resources are arranged on top in ready queue whereas gold group members are placed after platinum and silver group members are placed after gold group. Hence, at the time of task assignment proposed mechanism gets better available resource from all the available resources which in turn cause improvement in application turnaround time.

7. Conclusions

The proposed grouping mechanism is based on collective impact for CPU and RAM, spot-checking and task completion history that categorize the hosts in platinum, Gold and Silver groups and has proved its worth as compared to FCFS and PRI-CR-Excl mechanisms. The results have shown significant improvement in application turnaround time that is why it is safe to conclude that arranging hosts into groups according to their computational strength and task completion history leads towards the improvement in application turnaround time.

References

[1] Anderson P., "Emulating Volunteer Computing Scheduling Policies," *in Proceedings of IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, Shanghai, pp. 1839-1846, 2011.

[2] Anderson D. and Fedak G., "The Computational and Storage Potential of Volunteer Computing," *in Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*, pp. 73-80, 2006.

[3] Berkeley Open Infrastructure for Network Computing, available at: https://boinc.berkeley.edu, last visited 2013.

[4] Choi S., Baik M., Hwang C., Gil J., and Yu H., "Mobile Agent based Adaptive Scheduling Mechanism in Peer to Peer Grid Computing," *in Proceedings of the International Conference on Computational Science and its Applications*, Singapore, pp. 936-947, 2005.

[5] Compute Against Cancer, available at: http://www.computeagainstcancer.org, last visited 2013.

[6] Fight Aids at Home, available at: http://www.fightaidsathome.org, last visited 2013.

[7] Great Internet Mersenne Prime Search, available at: http://www.mersenne.org, last visited 2013.

[8] Hanandeh F., Khazaaleh M., Ibrahim H., and Latip R., "CFS: A New Dynamic Replication Strategy for Data Grids," *the International Arab Journal of Information Technology*, vol. 9, no. 1, pp. 94-99, 2012.

[9] Khan K., Hyder I, Chowdhry B., Shafiq F., and Ali H., "A Novel Fault Tolerant Volunteer Selection Mechanism for Volunteer Computing," *Sindh University Research Journal-Science Series*, vol. 44, no. 3, pp. 501-506, 2012.

[10] Kondo D., Chien A., and Casanova H., "Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids," *Journal of Grid Computing*, vol. 5, no. 4, pp. 379-405, 2007.

[11] Search for Extraterrestrial Intelligence., available at: http://setiathome.ssl.berkeley.edu, last visited 2013.

[12] Silaghi G., Domingues P., Araujo F., Silva L., and Arenas A., "Defeating Colluding Nodes in Desktop Grid Computing Platforms," *in Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, Florida, USA, pp. 1-8, 2008.
[13] Toth D. and Finkel D., "Improving the Productivity of Volunteer Computing by Using the Most Effective Task Retrieval Policies," J*ournal of Grid Computing*, vol. 7, no. 4, 2009.
[14] Watanabe K., Fukushi M., and Horiguchi S., "Optimal Spot-checking for Computation Time Minimization in Volunteer Computing," *Journal of Grid Computing*, vol. 7, no. 4 , 2009.

Mohammed Khan is the Director College of Computing and Information Sciences at PAF-Karachi Institute of Economic and Technology, Pakistan. He holds Post Graduation in Computer Science as well as in management science. He is experienced in academic leadership and has launched several bachelors programs at PAF-KIET. He has also completed several consultancy and training assignment at leading organizations. His research interests include distributed systems, intelligent and multi agent systems and performance evaluation.

Irfan Hyder is working as the Dean College of Business Management and College Engineering and Sciences at Institute of Business Management, Pakistan. He holds a PhD and MS in computer science from University of Texas at Austin, USA. He has a wide experience in academic leadership, consultancy, entrepreneurial ventures, trainings, teaching, research, design and implementation of innovative programs and as a motivational speaker. In his career spanning over 18 years, he has worked in leadership positions at various organizations. He was deputy director at IBA and Dean and VP at PAF-KIET. He has extensive industry consultancy experience for private and public sector organizations.

Ghayas Ahmed is serving as Deputy Controller in Federal Urdu University, Pakistan. He holds MS Computer Science from PAF-Karachi Institute of Economic and Technology. He also, possesses close to 11 years of software development experience. His research interests include software engineering development patterns and distributed systems.

Saira Begum is serving as Lecturer in Computer Science Department at Jinnah University for Women, Pakistan. Due to more than 6 years if software development experience, she is also heading the final year project committee. She holds MS Computer Science from PAF-Karachi Institute of Economic and Technology. Her research interests include programming paradigms and databases.