Scheduling in Desktop Grid Systems: Theoretical Evaluation of Policies & Frameworks

Muhammad Khalid Khan
College of Computing & Information Science
Pakistan Air Force – Karachi Institute of Economics & Technology, Karachi, Pakistan

Dr. Tariq Mahmood
Institute of Business Administration, Karachi,
Pakistan

Syed Irfan Hyder
Institute of Business Management, Karachi,
Pakistan

***Abstract*—Desktop grid systems have already established their identity in the area of distributed systems. They are well suited for High Throughput Computing especially for Bag-of-Tasks applications. In desktop grid systems, idle processing cycles and memory of millions of users (connected through internet or through any other communication mechanism) can be utilized but the workers / hosts machines not under any centralized administrative control that result in high volatility. This issue is countered by applying various types of scheduling policies that not only ensure task assignments to better workers but also takes care of fault tolerance through replication and other mechanism. In this paper, we discussed leading desktop grid systems framework and performed a comparative analysis of these frameworks. We also presented a theoretical evaluation of server and client based scheduling policies and identified key performance indicators to evaluate these policies.**

***Keywords*—desktop grid systems; task scheduling policies; work fetch policies**

I. INTRODUCTION

The advancements in the domain of distributed computing have opened up new horizons for high-end computing and storage. Particularly, desktop grid systems have laid down a much cheaper path towards the same. Desktop grid systems utilize idle processing cycles and memory of millions of users connected through Internet, or through any other type of network. This requires decomposition of computationally infeasible problems into smaller problems, distribution of smaller problems to the host / volunteer computers and aggregation of results from these volunteers to from solutions to large-scale problems.

Desktop grid systems can be divided into two categories [48]. When the computers of an enterprise are used to decrease the turnaround time of a compute intensive application, it is called enterprise wide desktop grids or simply desktop grids. The other category is volunteer computing in which home and enterprise computers take part by volunteering idle processing cycles to achieve high throughput.

The desktop grid system infrastructure consists of N number of desktop machines in which one would be termed as master and the others would be known as hosts/workers as shown in Figure 1. Practically a desktop grid system project has several servers to create tasks, distribute them, record the tasks and corresponding results, and finally, aggregate the results of a set of tasks. The tasks and corresponding work units (evaluating data sets) are distributed by the server to the hosts (client installed computer), typically through a software which permits people to participate in the project. Normally, when a host is idle (i.e., the computer's screensaver is running), then it is time to work on the tasks assigned by server. After finishing the tasks, the results are sent to the server. In case the computer that is running a client gets busy again then the client pauses the processing immediately so that the user can executes its own programs. The client continues processing the task as soon as the computer becomes idle again.

Desktop grid system frameworks simplify and automate various functions performed by master and client. Master is responsible for user and job management, client management, tasks management, results verification, security and performance management. Whereas, the client is responsible for collection of hardware statistics from machine, requesting and collecting tasks, task execution, sending back results and allowing users to set preferences. Some of the more popular desktop grid systems frameworks are BOINC [44], XtremWeb [37,45], OurGrid (peer-to-peer) [46], SZTAKI Desktop Grid [74], and HT Condor [47].

Moreover, the phenomena that has started from the PARC Worm (Xerox's initiative to develop worms to enable distributed computing) [28] has resulted in various successful implementations such as SETI@home [29,30], GIMPS [31], Folding@Home [32], FightAidsAtHome [33], Computing Against Cancer [34], Einstein@home [35]. These projects have taken up various scientific problems that include searching for cures of diseases, looking for evidence of extraterrestrial intelligence, finding Mersenne prime numbers, and solving several encryption challenges. Apart from the scientific projects, desktop grid systems have gathered recognition also at corporate level. The business enterprises got inspired with the huge success of desktop grid systems. As there is an abundance of desktop resources in such enterprises, it seems a cost effective solution to utilize the idle processing cycles of such systems and achieve high-end computing. Various such projects were launched by academia [36, 37, 38, 39, 40] and industry [41, 42, 43].

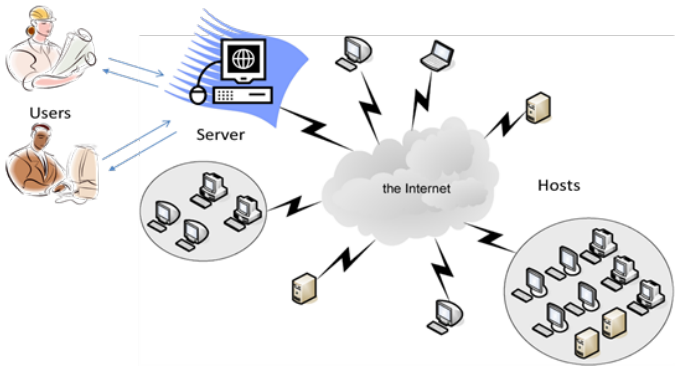

Fig. 1. Infrastructure of Desktop Grid Systems

There is a difference in perspective to scheduling policies as per the needs of scientists and volunteers. Although these perspectives are somewhat contradictory to each other but the scheduling policy should adhere to the needs of both stakeholders. For example the scientist would like to verify the results and would not mind investing processing cycles for it, whereas the volunteer would like to spend more time on actual processing and would count verification as wastage of resources. These requirements of scientists and volunteers are shown in Table 1.

TABLE I. CLASSIFICATION OF SCHEDULING PERSPECTIVES IN DESKTOP GRID SYSTEMS

Desktop Grid System Scheduling Perspectives	
Policies driven by Scientist's Perspectives	Policies driven by Volunteer's Perspectives
Maximize Availability of Resources	Minimize Validation Latency
Maximize Turnaround Time	Maximize Utilization
Maximize Reliability	Minimize Resource Wasting
Maximize Throughput	

Moreover, different scheduling policies are implemented in a typical desktop grid system that can be broadly categorized into three categories.

- **Server based task scheduling policy** takes care of tasks assignment to server and is based on clients and tasks preferences (for example size of the job, speed of the host, particular operating system, amount of disk space etc). A scoring-based scheduling policy assigns values to individual parameters to calculate the overall impact.
- **Client based CPU scheduling policy** is related to CPU scheduling of desktop grid application's tasks (works on top of the local operating system's scheduler) and addresses issues such as selection of particular task for execution from the currently runnable tasks, and keeping a particular task in memory from the list of preempted tasks.
- **Client based work fetch policy** determines when a client can ask for more work and the amount of work that can be requested by a client.

Scheduling policies can also be classified as naive or adaptive. The naive scheduling policies do not consider any historical knowledge whereas adaptive scheduling policies use a knowledge-base having historical information and threshold values. These measures are used to perform scheduling decision making. Furthermore, the naive scheduling policies do not consider volunteer's availability and reliability as decision making factor as they do not work on historical information. Hence the task assignments to volunteers remain arbitrary. Although these policies such as First Come First Server (FCFS) are easy to design and implement and are used by many volunteer computing platforms but they do not guarantee results. One the other side the knowledge base / adaptive policies consider history and are capable to adapt to changing circumstances but their decision making criteria is not comprehensive. These policies limit themselves by considering hardware threshold, reliability or availability.

Notwithstanding their use, there are certain limitations to desktop grid systems which include resource management, scheduling, verification of results, computation time, fault tolerance, security breaches, connectivity and bandwidth issues etc. The nodes in a desktop grid system environment are inherently volatile, can be heterogeneous, are slower than high-end machines, and the communication mechanism doesn't guarantee five nine reliability. The fact that nodes may fail at any time arises various design and deployment challenges.

Moreover, the scheduling policies should strive to attain fault tolerance and result verification. This is done through various mechanisms such as replication, voting and spot checking. In replication, similar tasks are assigned to multiple volunteers to counter the problem of volunteer's unavailability that can be categorized into host and CPU unavailability. Replication coupled with voting is used for result verification. In voting, results from multiple volunteers being assigned the same task are compared and the result submitted by the majority of the volunteers is counted as correct. Spot checking is done to assess the reliability of the volunteer. In spot checking, a spot job is submitted to the volunteer whose result is already known to the server. Fault tolerance has its own issues; if not done properly the overhead generated by the fault tolerant mechanism can increase the wastage of processing cycles. Poor scheduling policies cause wastage of precious processing cycles that increases the application's turnaround time as well.

The paper is organized as follows: in section 2, we have discussed leading desktop grid system *Frameworks.* In sections 3 & 4, we have proposed key performance factors to evaluate the server based task scheduling policies and client based work fetch policies respectively. Section 5 concludes the paper.

II. EVALAUATING DESKTOP GRID SYSTEM FRAMEWORKS

The job of the desktop grid system framework is to simplify and automate various functions performed by master and client in a desktop grid system environment. As stated earlier the desktop grid systems can be divided into desktop grids and volunteer computing. For desktop grids BOINC

[44], XtremWeb [45] and OurGrid (peer-to-peer) [46] can be used. In case of volunteer computing, BOINC is a better option especially for applications having large number of tasks. HT Condor [47] can be used equally for both. The desktop grid framework should be able to address following queries:

1) How users submit jobs? Can a user submit more than one job at a time?

2) How tasks are generated of the given job? Will the tasks be dependent or independent?

3) How the granularity of the tasks is decided? Will the tasks be coarse or fine grained?

4) How clients register with server? What hardware parameters are polled from the client?

5) How the tasks are mapped on appropriate clients? How client's and task's preference matched?

6) How many tasks are given to a client at a given time? Can the number be changed?

7) How results are verified and validated?

8) How results from various clients are summed up to give user a consolidated result?

9) How fairness is maintained among various jobs while assigning their tasks to clients?

*10)*How fairness is achieved among the tasks of various jobs at client?

*11)*How fault tolerance is achieved as clients can become unavailable anytime?

*12)*How many replica of a task is generated to achieve fault tolerance?

*13)*How many platforms are supported by client end?

*14)*How the client end users are kept motivated to donate processing cycles?

The above mentioned queries have direct impact on application's turnaround time and throughput. These queries are mostly handled by the server end of the framework. All the desktop grid systems frameworks are capable of handling various jobs, multiple clients, pooling of client statistics and some sort of fault tolerance but most of them decomposes job into independent tasks. Now we will have a brief discussion on some popular desktop grids frameworks and will do a comparative analysis as well.

A. BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) is an open source platform developed at U.C. Berkeley in 2002 [44]. Today approximately 60 projects are using BOINC in a wide range of scientific areas. BOINC server software is used to create volunteer computing projects. Each project has its own server and provides a web site. Volunteer connects to the website to download and install client software. The client software is available on Windows, Linux, and Mac OS X platforms. A BOINC project can have more than one application. BOINC provides flexibility for distributing data and intelligently matches requirements with resources. Having installed the BOINC client, volunteers can attach itself to any project. BOINC client can assign resources to each project. Attaching a project allows it to run arbitrary executables so it is the volunteer's job to assess project's authenticity and its scientific merit. BOINC assigns a numerical value against the volunteer's contribution to a project. BOINC uses volunteer's email to perform cross-project user identification. BOINC client can also attach itself to a web service called an account manager rather than connecting directly to the client. The account manager passes client's credentials to sever to receive a list of projects with which client can connect to.

B. XtremWeb

XtremWeb is open source platform developed by INRIA [45]. Its successor XWHEP (XtremWeb- HEP) is currently developed at LAL CNRS. XtremeWeb is a lightweight Desktop Grid with some advance features such as permit multi-users, multi-applications and cross domains deployments. XtremWeb is designed in such a way that it can be used for desktop grids, volunteer computing and Peer to Peer distributed systems. The XWHEP/ XtremWeb architecture consists of servers, clients and workers. Server's job is to host centralized services such as scheduler and result collector. Clients work at user end; users submit applications to the server for processing. The client allows users to manage the platform and interact with the infrastructure as and when required such as job submission, result retrieval etc. Server schedule the jobs submitted by client on workers. Workers are installed at processing node to contributed their computing resources that are aggregated in an XWHEP/ XtremWeb infrastructure. XWHEP improves the security of XtremWeb by the implementation of user accounts and access rights. These features extend user interactions over the platform that includes secure resource usage and application deployment.

C. OurGrid

OurGrid is an open source middleware designed for peer-to-peer computational grids [46]. OurGrid enables the use of idle computing and storage resources over a grid. These resources are shared in such a way that who have contributed the most will get the most required. OurGrid provides a secure platform for the execution of parallel applications having independent tasks also called Bag-of-Tasks (BoT) applications. BoT examples may include parameter sweep simulations, rendering of images and many others. In OurGrid, each grid site corresponds to a peer in the system. The problem of free riders (people who are not contributing their resources but using resources of others) is resolved in OurGrid by using Network of Favours mechanism. This credit mechanism ensures that the computing node sharing its resources will be prioritized over a node that is not sharing the resources. OurGrid Community, a free-to-join cooperative grid is also maintained by OurGrid team.

D. HT Condor

HT Condor referred as condor till 2012 is developed at the University of Wisconsin- Madison to provide high-throughput distributed batch computing [47]. High throughput computing refers to the efficient utilization of available computing resources to provide fault tolerant computational power. Condor is not only capable of managing dedicated resources such as clusters but it can also effectively harness idle

processing cycle of any processing available on the infrastructure. Condor can process a task on a idle node, it is also capable of stopping the execution of a running task, marking a checkpoint and migrating the task to a different processing node. Condor can redirect the task's I/O requests back to the actual machine from where the task is submitted. As a result, Condor can seamlessly combine all te computing power of an organization. Condor architecture is comprised of a single machine serving as the central manager and other machines that are part of the infrastructure. Condor job is assign tasks to the available resources. Condor client programs send periodic updates to the central manager so that the manager can be updated about the status of the resources and can make appropriate task assignments.

Apart from framework like BOINC that are free for use, there are other proprietary frameworks designed for the same. Organizations such as Distributed.net [49], United Devices [50] and Entropia [51] have produced proprietary frameworks (not available for free) for particular industries that can perform specialized tasks such as searching for new drugs at pharmaceutical companies. Bayanihan [39] is another open source framework developed at MIT and is considered as the first web-based desktop grid system framework.

TABLE II. COMPARISON OF DESKTOP GRID SYSTEMS FRAMEWORKS

Frameworks	**BOINC**	**XtremWeb**	**Our Grid**	**HT Condor**
Design Architecture	Client Server	Client Server	Peer to peer	Central Broker
Application Management	Centralized	Centralized	Decentralized	Decentralized
Resource Providers can act as Resource Consumers	No	Yes	Yes	Yes
Task Distribution	Pull	Pull	Push	Push
Deployment / Administration Complexity	Medium / Low (client side)	Low	Low	Medium
Application Development / Porting Complexity	High / Medium (with wrapper)	Low	Low	Medium
Support for Volunteer Desktop Grids	Yes	Yes	Yes	No
Security Features	Code signing, Result validation	Sandbox	Sandbox (Virtual Machine)	Authentication
Web Interface	Yes (Monitoring)	Yes (Monitoring)	Yes (Monitoring, Job Submission)	No
Number of Deployments	~100 (~1M CPUs in big projects)	~10	A few	~100
Programming Language	C/C++	Java	Java	C/C++
Documentation / Help	Good	Good	Good	Very Good

E. Comparison of Desktop Grid Systems Frameworks

We present a comparison between different frameworks in Table 2. Several factors are considered for the comparison such as software design including architecture and applications, project completion and application turnaround time, the potential help available for new user and their security concern. Overall usage of the framework is also an important factor.

III. EVALAUATING SERVER BASED TASK SCHEDULING POLICIES

The desktop grid system server can comprise of many complex scheduling policies. There are numerous criteria for job assignment, based on host and job diversity (for example size of the job and speed of the host relative to an estimated statistical distribution, disk and memory requirements for the job to be completed, homogeneous redundancy and host error rate). A scoring-based scheduling policy uses a linear combination of these terms to select the best set of jobs that can be assigned to a given host. We have made two categories of the task scheduling mechanism proposed earlier. The first category is *Using Tradition Techniques*, we have grouped papers in this category that have proposed scheduling framework / algorithms based on computing strengths, behavior or makespan analysis of the host [1, 2, 3, 6, 7, 8, 9, 13, 57, 58, 60, 62, 64, 67, 68, 69, 70]. These papers have also talked about grouping similar hosts and proposed improved replication methods [14, 15, 16, 17, 18, 19, 20, 21]. Papers that incorporated fault tolerance mechanisms [22, 23, 24, 25, 27, 53, 56] are also made part of this category. By using experimental methodology, these papers suggested improved results in various contexts however they have only used traditional problem solving techniques. Our second category is about *Using Predictive Analytics*. Papers which have implemented some sort of statistical [4, 5, 10, 66, 72, 73], probabilistic [55, 59, 61, 65] or machines learning algorithms / mechanisms [11, 12, 63, 71] are made part of this category. Even for fault tolerance, analytical methods are used. These papers have gathered data from real desktop grid systems or established test beds to gather data, implemented aforementioned techniques and presented promising results.

A. Key Performance Factors

We have identified the following key performance factors for evaluating the performance of task scheduling mechanisms. Scheduling mechanism that performs most of below mentioned points is taken as better mechanism. Though none of these factors are considered collectively in the literature but few of them can be found in [2, 3, 5, 6, 7, 14, 15, 22, 25, 26].

- Resource Availability
- Makespan Evaluation
- Replication
- Resource Capability
- Sabotage Tolerance
- Group based Design

Resource Availability

Availability of host is a critical factor for scheduling in a desktop grid system. As hosts are not managed centrally, they can become unavailable at any time. Scheduling mechanism must check host availability before assigning a task to any host. Host unavailability refers to hosts being powered off whereas CPU unavailability refers to a situation where host is connected to the server but its CPU is busy in performing host's local tasks. The configuration of desktop grid client is done in such a way that the host's CPU is only available to desktop grid when it not executing any local task i.e. when the CPU is idle. If host is available but the CPU is not available for processing, the task is suspended and can be resumed on the same host at a later time.

Makespan Evaluation

Makespan is a life time of a task during its execution from start to finish. The job of any scheduling mechanism is to minimize the makespan by assigning tasks to better hosts. Once a task is assigned to host, its makespan is estimated and if the actual makespan of the task matches the estimated makespan than the task assignment to that particular host is justified. This can also be taken as the "on-time task completion" and the scheduling mechanism should assign tasks to hosts having better "on-time task completion" history.

Replication

As the resources are not under centralized administrative domain, there is a chance that they may become unavailable at any point in time. The solution to this problem is replication in which a replica of the assigned task is assigned to some other host as well. Replication helps is countering volatility but excessive replication also cause wastage of processing cycles.

Resource Capability

Consideration of host clock rate or memory size to exclude or prioritize hosts at the time of task scheduling is a common way of resource allocations. However, only focusing on resource capabilities and not considering availability and reliability may result in poor decision making. Resources with low capabilities may be more reliable and can be available for more time.

Sabotage Tolerance

There may be hosts in desktop grid systems that try to submit erroneous results. To identify the saboteurs, spot checking is performed in which master assigns a task to hosts whose result is already known to master. Hosts that do not give correct result are counted as saboteurs and should not be considered for task assignments. There is also a need to verify the results computed by these hosts. Voting is one of the mechanisms and has couple of variants. In majority voting, results from the majority of the hosts are considered as correct whereas in n-first voting, results from the n hosts is considered as correct. Scheduling mechanism should consider this aspect of fault tolerance.

Group based Design

It has been observed that grouping similar host helps is scheduling while keeping the cost low. This also facilitate in establishing various replication strategies. The idea is not to make decision making for each host but to establish same policies for similar host arranged in a group. The parameters of assigning hosts to different groups may vary and may include availability, reliability, computing strength etc.

Now, we present the comparative evaluation of the task scheduling mechanisms discussed earlier on the basis of the key performance factors. Table 3 presents predictive analytics papers whereas table 4 lists the papers that use traditional techniques. A better scheduling mechanism will have "Y" in most of the fields. It is also evident from the evaluation that considering task dependencies as well as task granularity for scheduling in desktop grid systems are still open issues.

TABLE III. PERFORMANCE EVALUATION OF SCHEDULING MECHANISMS BASED ON PREDICTIVE ANALYTICS

Key Performance Factors / Reference No.	Resource Availability	Makespan Evaluation	Replication	Resource Capability	Sabotage Tolerance	Group based Design
[4]	Y	Y	N	Y	N	N
[5]	N	N	N	N	Y	N
[10]	N	Y	Y	Y	N	N
[11]	N	Y	Y	Y	N	N
[12]	Y	Y	N	Y	N	N
[54]	Y	Y	N	Y	N	N
[55]	Y	N	N	Y	Y	Y
[59]	Y	Y	Y	Y	N	N
[61]	Y	N	N	N	Y	Y
[63]	N	Y	N	N	Y	N
[65]	Y	N	N	N	Y	N
[66]	Y	N	N	Y	Y	N
[71]	Y	N	N	Y	N	N
[72]	Y	Y	N	Y	N	N
[73]	Y	N	N	Y	N	N

TABLE IV. PERFORMANCE EVALUATION OF SCHEDULING MECHANISMS BASED ON TRADITIONAL TECHNIQUES

Key Performance Factors / Reference No.	Resource Availability	Makespan Evaluation	Replication	Resource Capability	Sabotage Tolerance	Group based Design
[1]	Y	Y	N	Y	N	Y
[2]	Y	Y	N	Y	N	N
[3]	Y	Y	Y	Y	N	N
[6]	N	N	N	N	N	N
[7]	Y	Y	Y	Y	N	Y
[8]	N	Y	N	N	N	Y
[9]	N	N	Y	Y	Y	N
[13]	Y	N	N	N	N	Y
[14]	Y	Y	Y	Y	N	N
[15]	Y	N	N	Y	N	N
[16]	N	Y	Y	Y	N	Y
[17]	Y	N	Y	N	N	Y
[18]	Y	Y	Y	Y	N	Y
[19]	Y	N	Y	Y	N	Y
[20]	Y	Y	N	Y	N	N
[21]	N	Y	Y	N	Y	N
[22]	N	N	Y	N	Y	N
[23]	Y	N	N	Y	N	N
[24]	N	Y	Y	N	Y	Y
[25]	N	N	N	N	Y	N
[26]	Y	N	Y	Y	N	N
[27]	Y	Y	N	Y	N	N
[53]	Y	Y	Y	Y	N	N
[56]	Y	Y	Y	Y	Y	Y
[57]	Y	Y	N	Y	N	N
[58]	Y	Y	N	Y	N	N
[60]	Y	N	Y	N	N	N
[62]	N	Y	Y	N	N	N
[64]	Y	N	N	Y	N	N
[67]	Y	Y	N	N	Y	Y
[68]	N	N	N	Y	Y	Y
[69]	Y	N	N	Y	N	N
[70]	Y	Y	N	Y	N	N

IV. EVALAUATING CLIENT BASED WORK FETCH POLICIES

Work fetch policies should be designed to fetch a balanced amount of work for the client according to the clients shared resources ensuring their optimum utilization. Any imbalance in the amount of work fetched would either result in wasted CPU cycles and other resources (RAM, disk) caused by missed deadlines or less than optimal utilization of the already scarce shared resources. BOINC Client uses two work fetch policies **buffer none** and **buffer multiple tasks**, also a number of other variations have been suggested in [7,9]. As stated earlier, work fetch policies addresses the issues of when to ask for more work, which project to ask work for and how much work to ask for. We have discussed the variations of work fetch policies below:

Buffer None [7]

The policy does not buffer any tasks. It only downloads a task after returning the result of the previous task.

Download Early [9]

The policy downloads a new task when the client is 95% done with the task it is processing.

Buffer One Task [9]

The policy buffers one task so the client always has a task to process, even while it is downloading a new task.

Buffer Multiple Tasks [7]

The policy buffers task for number of days. The amount of tasks is limited to a number that can possibly be completed before the tasks' deadlines.

Hysteresis Work Fetch

Uses hysteresis (making decisions based on past behavior) and it asks a single project for the entire shortfall rather than dividing it among projects.

A. Key Performance Factors

We have identified the following key performance factors for evaluating the performance of various fork fetch policies. The policy which is aligned to most of the given KPIs is counted as better policy.

- Tasks buffered
- Continuous internet connectivity
- Chance of having wasted fractions
- Round robin simulation
- Hysteresis
- Utilization Of GPUs
- Utilization Of Multiple Cores

Tasks Buffered

This refers to amount of work that can be buffered by the client. Clients normally use both buffer multiple tasks and buffer none policies each having their own pros and cons. Buffer none ensures the maximum amount of CPU time to the current task yielding very low missed deadlines but results in wasted CPU cycles when downloading new tasks or when that client is available for computation but disconnected from internet, Buffer Multiple tasks does not keep the shared resources idle while upload and download operations but may result in wasted fractions if deadlines for buffered tasks are not met, missed deadlines is also an undesired effect from server scheduling point of view which results in poor reliability of a particular host.

Continuous Internet Connectivity

Buffering no or little amount of work requires continuous connection with internet as the hosts needs to download new work as soon as it completes on hand work, hence internet connectivity is required all the time. This fact becomes a serious bottleneck with the increase in mobile computing devices (Laptops, cell phones, tablets) which can available for computing but may or may not be connected to the internet during that interval.

Missed Deadlines

Missed deadline occur when a client is not able to complete the task within its deadline, this results in wasted fractions and also has a negative impact on hosts reliability. While buffering multiple tasks the optimum amount of work to be fetched depends on the future CPU availability which is unknown but can be measured using traces and other mechanisms with some degree of accuracy. The influx of the Green Movement (when computer goes into power saving mode by disabling all unnecessary programs while the screen saver is on) has made this task even more difficult.

Round Robin Simulation

The round-robin simulation predicts the amount of time each processor will be kept busy with the current workload. This helps in measuring the shortfall of idle instance-seconds which is a critical factor in deciding the amount of work to be fetched from attached projects for buffer multiple policies.

Hysteresis

This refers to technique that relies not only on the current client state but also on the past behavior in making work fetch decisions.

Utilization of GPUs

With the advent of GPU (Graphics Processing Units), a new class of volunteers is now available [52]. The GPU based clients have different architecture as compared to clients based on CPU. The work fetch policies must also consider the architecture and limitation of GPUs.

Utilization of Multiple Cores

In a multicore CPU environment, it is important to utilize all the available cores for computing. Policy executing only one task at a time work fine as long as the task is multithreaded and able to run on multiple cores but in the case of single threaded tasks it becomes a serious drawback.

B. Discussion

The evaluation of work fetch policies on the basis of key performance factors is given in Table 5 that lists the work fetch policies on x-axis and key performance factors on y-axis and summarizes their dependencies (internet connectivity), degree of efficiency in respective areas (chance of missing deadlines, round robin simulation, hysteresis, utilization of GPUs, utilization of Multiple Cores). It can be observed that variations of work fetch polices that buffer no or one tasks get excellent scores for meeting deadlines but suffer in other areas such as handling single threaded tasks on multi core CPUs, GPU Utilization and their dependency on a continuous internet connection which proves to be the major drawback specially now when the number of mobile devices on which the internet connectivity is sporadic are increasing rapidly. Buffering multiple tasks perform better in utilizing multicore CPUs and GPUs, their major advantage being the ability of work without continuously being connected to the internet. However misjudged amount of buffered work can lead to poor utilization of resources by underestimating the amount of work or missed deadlines by overestimated work fetch.

TABLE V. EVALUATION OF WORK FETCH POLICIES USING KEY PERFORMANCE FACTORS

	Work Buffered	Continuous Internet Connectivity	Chance of Missing Deadlines	Round Robin Simulation	Hysteresis	Utilization of GPUs	Utilization of Multiple Cores
Buffer None	None	Yes	Negligible	N/A	N/A	Poor	Good
Download Early	None	Yes	Negligible	N/A	N/A	Poor	Good
Buffer One Task	Single Work units	Yes	Little (only in case of large tasks)	N/A	N/A	Poor	Good
Buffer Multiple Tasks	Multiple Work units	No	Huge	Yes	No	Good	Excellent
Hysteresis Work Fetch	Multiple Work units	No	Huge	Yes	Yes	Good	Excellent

Overall picture suggests the hysteresis work fetch gets good scores comparatively in all evaluation criteria with the prospect of reducing the chances of missed deadlines as we continue to find improved methods and heuristics for predicting the CPU availability for a period of time.

V. CONCLUSION

We have discussed leading desktop grid systems frameworks and performed a comparative evaluation. We have also conducted a thorough theoretical and experimental evaluation of the task scheduling, CPU scheduling and work fetch policies in desktop grid systems. We have identified that task scheduling can only be improved by grouping the similar workers so that relevant resource allocation and replication policies can be applied. Task dependence and granularity are still unaddressed areas in task scheduling. We have analyzed that work fetch policies has direct impact on the task completion and performance of hysteresis work fetch was found better on majority of the evaluation parameter as compared to buffer-one or buffer-none that performs well only on limited scale.

REFERENCES

[1] Heien, E. M., Anderson, D. P., & Hagihara, K. (2009). Computing low latency batches with unreliable workers in volunteer computing environments.*Journal of Grid Computing*, *7*(4), 501-518.

[2] Lee, Y. C., Zomaya, A. Y., & Siegel, H. J. (2010). Robust task scheduling for volunteer computing systems. *The Journal of Supercomputing*, *53*(1), 163-181.

[3] Kondo, D., Chien, A. A., & Casanova, H. (2007). Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. *Journal of Grid Computing*, *5*(4), 379-405.

[4] Estrada, T., Fuentes, O., & Taufer, M. (2008). A distributed evolutionary method to design scheduling policies for volunteer computing. *ACM SIGMETRICS Performance Evaluation Review*, *36*(3), 40-49.

[5] Gao, L., & Malewicz, G. (2007). Toward maximizing the quality of results of dependent tasks computed unreliably. *Theory of Computing Systems*, *41*(4), 731-752.

[6] Krawczyk, S., & Bubendorfer, K. (2008, January). Grid resource allocation: allocation mechanisms and utilisation patterns. In *Proceedings of the sixth Australasian workshop on Grid computing and e-research-Volume 82* (pp. 73-81). Australian Computer Society, Inc..

[7] Choi, S., Baik, M., Gil, J., Jung, S., & Hwang, C. (2006). Adaptive group scheduling mechanism using mobile agents in peer-to-peer grid computing environment. *Applied Intelligence*, *25*(2), 199-221.

[8] Villela, D. (2010). Minimizing the average completion time for concurrent Grid applications. *Journal of Grid Computing*, *8*(1), 47-59.

[9] Toth, D., & Finkel, D. (2009). Improving the productivity of volunteer computing by using the most effective task retrieval policies. *Journal of Grid Computing*, *7*(4), 519-535.

[10] Rood, B., & Lewis, M. J. (2010, May). Availability prediction based replication strategies for grid environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* (pp. 25-33). IEEE.

[11] Estrada, T., Taufer, M., & Reed, K. (2009, May). Modeling job lifespan delays in volunteer computing projects. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 331-338). IEEE Computer Society.

[12] Zhang, J., & Phillips, C. (2009). Job-Scheduling with Resource Availability Prediction for Volunteer-Based Grid Computing. In *London Communications Symposium, LCS*.

[13] Daniel Lazaro, Derrick Kondo and Joan Manuel Marques, "Long-term availability prediction for groups of volunteer resources," J. Parallel Distributed. Computing 72 (2012) 281–296

[14] Kondo, D., Fedak, G., Cappello, F., Chien, A. A., & Casanova, H. (2006, December). On Resource Volatility in Enterprise Desktop Grids. In *e-Science*(p. 78).

[15] Huu, T. T., Koslovski, G., Anhalt, F., Montagnat, J., & Primet, P. V. B. (2011). Joint elastic cloud and virtual network framework for application performance-cost optimization. *Journal of Grid Computing*, *9*(1), 27-47.

[16] Schulz, S., Blochinger, W., & Hannak, H. (2009). Capability-aware information aggregation in peer-to-peer Grids. *Journal of Grid Computing*,*7*(2), 135-167.

[17] Choi, S., Baik, M., Hwang, C., Gil, J., & Yu, H. (2005). Mobile agent based adaptive scheduling mechanism in peer to peer grid computing. In *Computational Science and Its Applications–ICCSA 2005* (pp. 936-947). Springer Berlin Heidelberg.

[18] Khan, M. K., Hyder, I., Chowdhry, B. S., Shafiq, F., & Ali, H. M. (2012). A novel fault tolerant volunteer selection mechanism for volunteer computing.*Sindh University Research Journal—Science Series*, *44*(3), 138-143.

[19] Kondo, D., Casanova, H., Wing, E., & Berman, F. (1993, October). Models and scheduling mechanisms for global computing applications. In *Vehicle Navigation and Information Systems Conference, 1993., Proceedings of the IEEE-IEE* (pp. 8-pp). IEEE.

[20] Anderson, D. P., & Fedak, G. (2006, May). The computational and storage potential of volunteer computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on* (Vol. 1, pp. 73-80). IEEE.

[21] Sarmenta, L. F. (2002). Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, *18*(4), 561-572.

[22] Watanabe, K., Fukushi, M., & Horiguchi, S. (2009). Optimal spot-checking for computation time minimization in volunteer computing. *Journal of Grid Computing*, *7*(4), 575-600.

[23] Kondo, D., Anderson, D. P., & McLeod, J. (2007, December). Performance evaluation of scheduling policies for volunteer computing. In *e-Science and Grid Computing, IEEE International Conference on* (pp. 415-422). IEEE.

[24] Kondo, D., Taufer, M., Brooks III, C. L., Casanova, H., & Chien, A. (2004, April). Characterizing and evaluating desktop grids: An empirical study. In*Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International* (p. 26). IEEE.

[25] Silaghi, G. C., Araujo, F., Silva, L. M., Domingues, P., & Arenas, A. E. (2009). Defeating colluding nodes in desktop grid computing platforms.*Journal of Grid Computing*, *7*(4), 555-573.

[26] Kondo, D., Araujo, F., Malecot, P., Domingues, P., Silva, L. M., Fedak, G., & Cappello, F. (2007). Characterizing result errors in internet desktop grids. In *Euro-Par 2007 Parallel Processing* (pp. 361-371). Springer Berlin Heidelberg.

[27] Morrison, J. P., Kennedy, J. J., & Power, D. A. (2001). Webcom: A web based volunteer computer. *The Journal of supercomputing*, *18*(1), 47-61.

[28] Shoch, J. F., & Hupp, J. A. (1982). The "worm" programs—early experience with a distributed computation. *Communications of the ACM*, *25*(3), 172-180.

[29] SETI@home. The SETI@home project. http:// setiathome.ssl.berkeley.edu/

[30] Sullivan III, W. T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., & Anderson, D. (1997, January). A new major SETI project based on Project Serendip data and 100,000 personal computers. In *IAU Colloq. 161: Astronomical and Biochemical Origins and the Search for Life in the Universe* (Vol. 1, p. 729).

[31] GIMPS. The Great Internet Mersene Prime Search accessible from http://www.mersenne.org/

[32] Folding@home accessible from http://folding.stanford.edu/

[33] FIGHTAIDS. The Fight Aids At Home project accessible from http://www.fightaidsathome.org/

[34] CANCER. The Compute Against Cancer project accessible from http://www.computeagainstcancer.org/

[35] Einstein@Home accessible from http://einstein.phys.uwm.edu/

[36] Camiel, N., London, S., Nisan, N., & Regev, O. (1997, April). The popcorn project: Distributed computation over the internet in java. In *6th International World Wide Web Conference*.

[37] Fedak, G., Germain, C., Neri, V., & Cappello, F. (2001). Xtremweb: A generic global computing system. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on* (pp. 582-587). IEEE.

[38] Pedroso, H., Silva, L. M., & Silva, J. G. (1997). Web-based metacomputing with JET. *Concurrency: Practice and Experience*, *9*(11), 1169-1173.

[39] Sarmenta, L. F., & Hirano, S. (1999). Bayanihan: Building and studying web-based volunteer computing systems using Java. *Future Generation Computer Systems*, *15*(5), 675-686.

[40] Ghormley, D. P., Petrou, D., Rodrigues, S. H., Vahdat, A. M., & Anderson, T. E. (1998). GLUnix: A Global Layer Unix for a network of workstations.*Software Practice and Experience*, *28*(9), 929-961.

[41] Entropia, Inc. accessible from http://www.entropia.com

[42] Platform Computing Inc. accessible from http://www. platform.com/

[43] Data Synapse Inc. accessible from http://www.datasynapse.com/

[44] BOINC accessible from http://boinc.berkeley.edu/

[45] XtremeWeb accessible from http://www.xtremweb.net/

[46] OurGrid accessible from http://www.ourgrid.org/

[47] HTCondor accessible from http://research.cs.wisc.edu/htcondor/

[48] Vlădoiu, M. (2010). Has Open Source Prevailed in Desktop Grid and Volunteer Computing?. *Petroleum-Gas University of Ploiesti Bulletin, Mathematics-Informatics-Physics Series*, *62*(2).

[49] Distributed.Net, accessed from http://www.distributed.net

[50] De Roure, D., Baker, M. A., Jennings, N. R., & Shadbolt, N. R. (2003). The evolution of the grid. *Grid computing: making the global infrastructure a reality*, *13*, 14-15.

[51] Chien, A., Calder, B., Elbert, S., & Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, *63*(5), 597-610.

[52] Toth, D. (2007, February). Volunteer computing with video game consoles. InProc. 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Computing and Systems.

[53] Conejero, J., Caminero, B., Carrión, C., & Tomás, L. (2014). From volunteer to trustable computing: Providing QoS-aware scheduling mechanisms for multi-grid computing environments. *Future Generation Computer Systems*,*34*, 76-93.

[54] Tchernykh, A., Pecero, J. E., Barrondo, A., & Schaeffer, E. (2014). Adaptive energy efficient scheduling in peer-to-peer desktop grids. *Future Generation Computer Systems*, *36*, 209-220.

[55] Gil, J. M., & Jeong, Y. S. (2014). Task scheduling scheme by checkpoint sharing and task duplication in P2P-based desktop grids. *Journal of Central South University*, *21*(10), 3864-3872.

[56] Klejnowski, L., Niemann, S., Bernard, Y., & Müller-Schloer, C. (2014). Using Trusted Communities to improve the speedup of agents in a Desktop Grid System. In *Intelligent Distributed Computing VII* (pp. 189-198). Springer International Publishing.

[57] Canon, L. C., Essafi, A., & Trystram, D. (2014). A Proactive Approach for Coping with Uncertain Resource Availabilities on Desktop Grids. *FEMTO-ST, Tech. Rep. RRDISC2014-1*.

[58] Reddy, K. H. K., Roy, D. S., & Patra, M. R. (2014). A Comprehensive Performance Tuning Scheduling Framework for Computational Desktop Grid.*International Journal of Grid and Distributed Computing*, *7*(1), 149-168.

[59] Lerida, J. L., Solsona, F., Hernandez, P., Gine, F., Hanzich, M., & Conde, J. (2013). State-based predictions with self-correction on Enterprise Desktop Grid environments. *Journal of Parallel and Distributed Computing*, *73*(6), 777-789.

[60] Kang, W., Huang, H. H., & Grimshaw, A. (2013). Achieving high job execution reliability using underutilized resources in a computational economy. *Future Generation Computer Systems*, *29*(3), 763-775.

[61] Gil, J. M., Kim, S., & Lee, J. (2014). Task scheduling scheme based on resource clustering in desktop grids. *International Journal of Communication Systems*, *27*(6), 918-930.

[62] Xavier, E. C., Peixoto, R. R., & da Silveira, J. L. (2013). Scheduling with task replication on desktop grids: theoretical and experimental analysis.*Journal of Combinatorial Optimization*, 1-25.

[63] Naseera, S., & Murthy, K. M. (2013). Prediction Based Job Scheduling Strategy for a Volunteer Desktop Grid. In *Advances in Computing, Communication, and Control* (pp. 25-38). Springer Berlin Heidelberg.

[64] Zhao, Y., Chen, L., Li, Y., Liu, P., Li, X., & Zhu, C. (2013). RAS: A Task Scheduling Algorithm Based on Resource Attribute Selection in a Task Scheduling Framework. In *Internet and Distributed Computing Systems* (pp. 106-119). Springer Berlin Heidelberg.

[65] Gil, J. M., Kim, S., & Lee, J. (2013). Task Replication and Scheduling Based on Nearest Neighbor Classification in Desktop Grids. In *Ubiquitous Information Technologies and Applications* (pp. 889-895). Springer Netherlands.

[66] Salinas, S. A., Garino, C. G., & Zunino, A. (2012). An architecture for resource behavior prediction to improve scheduling systems performance on enterprise desktop grids. In *Advances in New Technologies, Interactive Interfaces and Communicability* (pp. 186-196). Springer Berlin Heidelberg.

[67] Choi, S., & Buyya, R. (2010). Group-based adaptive result certification mechanism in Desktop Grids. *Future Generation Computer Systems*, *26*(5), 776-786.

[68] Durrani, M. N., & Shamsi, J. A. (2014). Volunteer computing: requirements, challenges, and solutions. *Journal of Network and Computer Applications*,*39*, 369-380.

[69] Yang, C. T., Leu, F. Y., & Chen, S. Y. (2010). Network Bandwidth-aware job scheduling with dynamic information model for Grid resource brokers. *The Journal of Supercomputing*, *52*(3), 199-223.

[70] Peyvandi, S., Ahmad, R., & Zakaria, M. N. (2014). Scoring Model for Availability of Volatile Hosts in Volunteer Computing Environment. *Journal of Theoretical & Applied Information Technology*,70(2).

[71] Brevik, J., Nurmi, D., & Wolski, R. (2004, April). Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. In*Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on* (pp. 190-199). IEEE.

[72] Finger, M., Bezerra, G. C., & Conde, D. R. (2010). Resource use pattern analysis for predicting resource availability in opportunistic grids.*Concurrency and Computation: Practice and Experience*, *22*(3), 295-313.

[73] Anjos, J. C., Carrera, I., Kolberg, W., Tibola, A. L., Arantes, L. B., & Geyer, C. R. (2015). MRA++: Scheduling and data placement on MapReduce for heterogeneous environments. *Future Generation Computer Systems*, *42*, 22-35.

[74] SZTAKI Desktop Gird accessible from http://doc.desktopgrid.hu/doku.php