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].
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. EVALUATING 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 an 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 its 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

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

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. EVALUATING 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 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 is 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.

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>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[5]</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>[10]</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[11]</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[12]</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[54]</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>[55]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>[59]</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[61]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>[63]</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>[65]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>[66]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>[71]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[72]</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>[75]</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
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.

#### 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 policies 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 | NA | NA | Poor | Greed |
| Download Early | None | Yes | Negligible | NA | NA | Poor | Greed |
| Buffer One Task | Single Work units | Yes | Little (only in case of large tasks) | NA | NA | Poor | Greed |
| 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


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


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


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


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


SZTAKI Desktop Grid accessible from http://doc.desktopgrid.hu/doku.php