Evaluating the Impact of Client based CPU Scheduling Policies on the Application's Performance in Desktop Grid Systems

Muhammad Khalid Khan and Danish Faiz

College of Computing & Information Sciences, PAF - Karachi Institute of Economics & Technology, Karachi, Pakistan

Summary
Desktop grid systems are distributed computing paradigms which use the idle and underutilized processing cycles and memory of the desktop machines (hosts) to support large scale computations. These systems have inherent uncertainties because the hosts do not work under one administrative domain and can become unavailable at any given point in time. Desktop grid frameworks are based on client server model and employ various scheduling policies at both ends to handle the hostile desktop grid environment. At server end, task scheduling policies are deployed whereas work fetch and CPU scheduling policies are implemented at client end. Task scheduling policy decides which job will be send to client depending upon client and task preferences. Work fetch policy determines when the client can ask for more work from server and CPU scheduling policy selects the job for execution from the jobs available on client. This policy works on top of local operating system's scheduler. In this paper, we evaluated the impact of CPU scheduling policies on the application's performance by using BOINC Client and BOINC Client Emulator (BCE). We analyzed two most widely used CPU scheduling mechanisms by using four scenarios and five performance measures. We found that Early Deadline First (EDF) works better as compared to traditional Round Robin (RR) mechanism in most of the cases.

Keyword:
Desktop Gird Systems; CPU Scheduling Policies; BOINC; BOINC Client Emulator

1. Introduction

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 (Vladoiu 2010). When the computers of an enterprise are used to increase 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 through put.

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. 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 (Anderson 2004), XtremWeb (Fedak et al., 2001), OurGrid (Andrade et al., 2003), SZTAKI (Balaton et al., 2007) and HT Condor (Fajardo et al., 2015).

Scheduling is one the most important issue of desktop grid system because this is only way to handle the inherent uncertainties of desktop grid systems. Different scheduling policies are implemented in a typical desktop grid system that can be broadly categorized into three categories (Kondo 2007):

- **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

Manuscript received December 5, 2016
Manuscript revised December 20, 2016

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.

The impact of server based task scheduling policies (Kondo et al., 2007) and client based work fetch policies (Toth & Finkel 2009) has been studied in detail but to the best of our knowledge, there is almost no work on the impact of CPU scheduling polices at client end. CPU scheduling policies work on top of host's operating system and process the desktop grid tasks on the host. These policies are enforced by the desktop grid framework's client that communicates with the local operating system. CPU scheduling policies answers the questions such as which job to run among the available jobs? Which jobs to keep in memory among the preempted jobs? Poorly performing or incorrectly implemented CPU scheduling policies can reduce system throughput; equally importantly, they can frustrate and de-motivate volunteers, possibly causing them to stop volunteering. In this paper, we are evaluating the impact of CPU scheduling policies on the application's performance.

2. Evaluating CPU Scheduling Policies

To evaluate the impact of CPU scheduling policies, we have used the leading desktop grid framework BOINC which consists of server and client applications. BOINC server is responsible for task scheduling whereas BOINC client is responsible to fetch jobs from the server and get it executed on worker by using various CPU scheduling policies. We have also used BOINC Client Emulator (BCE) to emulate various CPU scheduling policies used by BOINC Client (Anderson 2011). BOINC Client can be connected to one or more projects, each project having one or more Applications. The client runs these applications through host's operating system and hardware resources. All network communication in BOINC is initiated by the client. Getting new jobs from the server and running them on the hosts OS is governed by a set of work fetch and CPU scheduling polices respectively. The structure of the BOINC Client is given in Figure 2.

The client performs CPU scheduling (implemented on top of the local operating system's scheduler; at the OS level, BOINC runs applications at zero priority). It may preempt applications either by suspending them (and leaving them in memory) or by instructing them to quit. All network communication in BOINC is initiated by the client. A client communicates with a project's task server via HTTP. The request is an XML document that includes a description of the host hardware and availability, a list of completed jobs, and a request for a certain amount (expressed in terms of CPU time) of additional work. The reply message includes a list of new jobs (each described by an XML element that lists the application, input and output files, including a set of data servers from which each file can be downloaded).

Historically, BOINC development has relied on a group of volunteers "alpha testers" who monitor the actions of their BOINC clients, communicate problems via email or message boards. This approach,

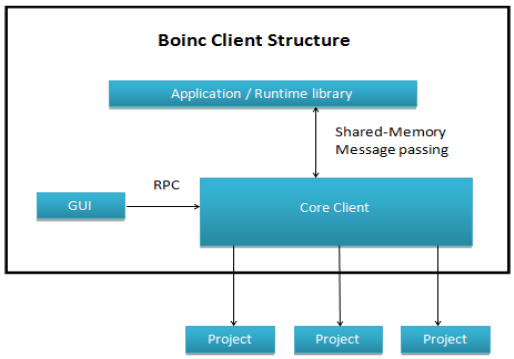

Figure 1: BOINC Client Structure

however, has significant limitations. For example, when an alpha tester reports a scheduling-related problem, it can be difficult to obtain information, such as trace message logs, needed to understand and fix the problem. In addition, the alpha tester approach doesn't help us design scheduling policies for hypothetical situations in which, for example, the GPU/CPU speed disparity is greatly increased, or projects have much tighter latency requirements. Developing and evaluating client policies is made difficult by the unique properties of volunteer computing:

The volunteered computers vary widely on many factors that influence scheduling such as hardware, availability, number and properties of attached projects, and so on. A combination of these factors is called a scenario. Scheduling policies should perform well across the entire population of scenarios.

The volunteer computers are not directly accessible to BOINC software developers. We are not able to deploy new software on these computers, or log in to them.

BCE addresses these issues by providing a new way of studying BOINC scheduling policies. It takes as an input a description of a usage scenario, emulates (using the actual BOINC client code) the behavior of the client over some period of time, and calculates various performance metrics. In addition Volunteers can run BCE by pasting their BOINC client state files into a web form. Hence, when an alpha tester notices a bug or anomaly, they can, in many cases, reproduce it using BCE, and report it (together with

their state files) to BOINC developers, who can then examine the problem under a debugger and fix it easily. BCE uses a mix of emulation and simulation as shown in Figure 2. The implementation of job scheduling, job fetch and preference enforcement uses the same source code as of the BOINC client. In terms of scheduling, BCE reproduces the exact behavior of the client; hence it "emulates" the client whereas the other components of the system are simulated:

- job execution is simulated, and run times are normally distributed;
- host availability is modeled as a random process in which available and unavailable periods have exponentially distributed lengths.
- BOINC schedulers are simulated with a simplified model.

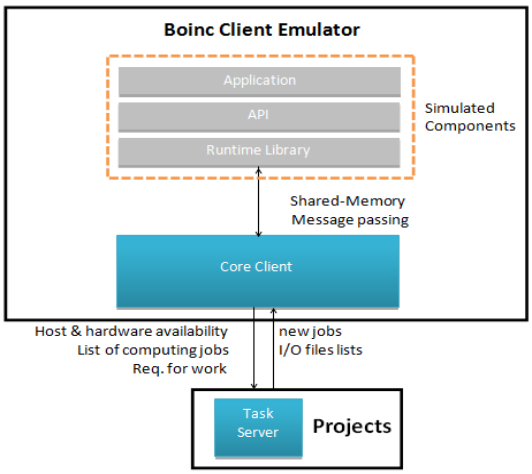

Figure 1: BOINC Client Emulator (BCE)

3. Experimental Methodology

To evaluate that the affect of CPU scheduling policies on the application's performance (turnaround time and throughput), we designed four scenarios by using combination of various task scheduling policies as shown in Table 1:

Table 1: Experiment Scenarios

	Round Robin Only	Scheduler EDF Simulation	Hysteresis Work Fetch
Scenario 1	Yes	No	Yes
Scenario 2	No	No	Yes
Scenario 3	Yes	Yes	Yes
Scenario 4	No	Yes	Yes

We used the most commonly used CPU scheduling policies i.e. Round Robin and Early Deadline First (EDF). Round robin can be harmful to jobs having short deadlines. To overcome this impact, we have also used EDF. We have used the similar work fetch policy for all scenarios i.e. work fetch hysteresis which relies not only on the current client state but also on the past behavior in making work fetch decisions. Our goal is to analyze the different figures of merits (Anderson 2011) and study the differences in the throughputs of attached applications for each case. The figures of merits are:

- **Idle fraction**: the fraction of processing capacity (as measured by peak FLOPS of all processor types) that was idle during the emulation period.
- **Wasted fraction**: the fraction of processing capacity (as measured by peak FLOPS) that was used for jobs that did not complete by their deadline.
- **Resource share violation**: the RMS over projects P of the difference between P's share of processing resources and the amount it actually received.
- **Monotony**: a measure of the extent to which the system ran jobs of a single project for long periods (such behavior is undesirable for many volunteers).
- **RPCs per job**: the average number of RPCs per job. The lower this is, the less load is placed on project servers.

We have used two different applications; first is a a locally running project with title Cplan1 which is configured on our development BOINC server machine and runs a sample application named UpperCase which consists of jobs with homogenous work units. The second project is a live project with title PrimaBoinca. This project is concerned with estimations for the identification of prime number and reducing the time of a deterministic prime test. The resource share settings and applications specs are given below:

Project 1: Cplan1 (locally running project)

- Resource share: 50%
- Application and version: app example_app
- Job params: fpops_est 1000G fpops mean 1000G std_dev 0G
- Latency: 85794.00 weight 1.00
- App version: 24253 ()
- CPU: 1 with 2 GFLOPS
- App version: 22489 ()
- CPU: 1 with 3 GFLOPS

Project 2: PrimaBoinca (live project)

- Resource share: 50%
- Application and version: app primaboinca
- Job params: fpops_est 1000G fpops mean 1000G std_dev 0G
- Latency: 604754.54 weight 1.00
- App version: 705 ()
- CPU: 1 with 3 GFLOPS

We run simulations for four different scenarios keeping the Hardware configuration same, Hardware configuration used for simulation is: 4 CPUs, 2.5 GFLOPS. The Client is simulated to run for 10 days, host availability is modeled as a random process in which available and unavailable periods have exponentially distributed lengths, the mean of the activity periods can be controlled with on_lambda parameter in client_state.xml, in our case we have kept it to a default value of 1 hour.

4. Results

The output given in Table 2(a) & (b) shows that the scenarios using EDFsimulation depicted slight increase in share violation. The high level of monotony found in all scenarios is due to the fact that project "Cplan1" and project "PRIMABOINCA" consists of jobs with considerably smaller work units. Wasted fraction values for all scenarios are considerably less counter intuitive to the fact that scenario 1 and scenario 2 have a very high ratio of deadlines missed which should result in high levels of wasted fraction. This may be due to the fact that deadlines are only missed for project Cpan1 and for Project "Primaboinca" deadlines missed in all scenarios remains zero. So the combined effect may have resulted low wasted fraction values in scenario 1 and scenario 2. Apart from these differences levels of wasted fraction, idle fraction, share violation and monotony do not vary by very large margins for all scenarios, but a visible difference is observed in application throughputs as shown in Table 2(b). It is observed that disabling EDF (Early Deadline First) results in larger no of missed deadlines (Scenario 1 and 2) as compared to (Scenario 3 and 4) having considerably less missed deadlines. Scenario 3 and 4 show that using weighted round robin increased throughput by 30% and 8% for Cpan1 and Primaboinca respectively.

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
Projects	1	2	1	2	1	2	1	2
Deadline Met	672	171	1191	246	1590	357	2076	389
Deadline Missed	633	0	514	0	419	0	422	0

Figure 2(a): Output of given scenarios

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Wasted fraction	0.01099	0.008924	0.007274	0.007326
Idle fraction	0.953993	0.939583	0.922587	0.90691
Share violation	0.077736	0.070115	0.136578	0.099217
Monotony	0.89127	0.916768	0.890234	0.88445
RPCs per job	0.125243	0.117314	0.108092	0.114858

Figure 2(b): Application Throughput

5. Conclusion

We presented a thorough evaluation of the two most commonly used CPU scheduling policies i.e. round Robin and Early Deadline First. We kept the work fetch policy same for all the scenarios i.e. Work Fetch Hysteresis. We designed four scenarios, used two different types of applications and performed evaluations on five measures. It is concluded that EDF in most cases results in less wasted fractions and low missed deadlines but may cause high share violation and monotony in favor of some projects leaving others starving for resources yielding low throughput. However, EDF is only optimal for uniprocessor environment. As a future work, it is desirable to study multiprocessor scheduling policies and compare the results with EDF.

References

[1] Vladoiu, M. (2010). Has Open Source Prevailed in Desktop Grid and Volunteer Computing?. Petroleum-Gas University of Ploiesti Bulletin, Mathemat-ics-Informatics-Physics Series, 62(2).

[2] Anderson, D. P. (2004, November). Boinc: A system for public-resource computing and storage. In Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on (pp. 4-10). IEEE.

[3] 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.

[4] Andrade, N., Cirne, W., Brasileiro, F., & Roisenberg, P. (2003, June). OurGrid: An approach to easily assemble grids with equitable resource sharing. In Workshop on Job Scheduling Strategies for Parallel Processing (pp. 61-86). Springer Berlin Heidelberg.

[5] Balaton, Z., Gombás, G., Kacsuk, P., Kornafeld, A., Kovács, J., Marosi, A. C., ... & Kiss, T. (2007, March). Sztaki desktop grid: a modular and scalable way of building large computing grids. In 2007 IEEE International Parallel and Distributed Processing Symposium (pp. 1-8). IEEE.

[6] Fajardo, E. M., Dost, J. M., Holzman, B., Tannenbaum, T., Letts, J., Tiradani, A., ... & Mason, D. (2015). How much higher can HTCondor fly?. In Journal of Physics: Conference Series (Vol. 664, No. 6, p. 062014). IOP Publishing.

[7] 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.

[8] 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.

[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] Anderson, D. P. (2011, May). Emulating volunteer computing scheduling policies. In Parallel and Distributed

Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on (pp. 1839-1846). IEEE.

Muhammad Khalid Khan received the M.S. degree in Computer Science from SZABIST, Karachi in 2005, MBA degree from PAF-KIET in 2008. He is currently associated with the college of computing and information science at PAF-KIET, Karachi as a PhD candidate. He also look after BSCS program at PAF-KIET. He has more than 10 years experience in academia and close to 5 year in software industry. He has more than 20 publications on his accord.