IJICTR International Journal of Information & Communication Technology Research ITRC

Volume 5- Number 4- Autumn 2013 (59-69)

Research Note

Increasing the Total Efficiently Executed Work on Volunteer Computing Environment Based on Work Send Policy

T. Kianpishe
Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
t_kianpishe@aut.ac.ir

Mohammad Kazem Akbari
Computer Engineering and Information Technology
Amirkabir University of Technology
Tehran, Iran
akbarif@aut.ac.ir

Received: February 7, 2012- Accepted: July 7, 2013

Abstract— **Volunteer computing is a type of parallel computing systems that is defined by a large number of nodes and autonomous changeable resources. Task scheduling is one of the most crucial issues on volunteer computing. A well-defined scheduling attracts the versatile guest applications to submit their jobs to this environment and motivates internet-users to contribute more. Hence, optimization of the server-side policies will stimulate volunteers' eagerness to contribute more of their resources. The existent methods cannot use the capacity of the host system efficiently. In this paper, we propose a new scheduling mechanism, which tries to decrease the idleness factor of the host system and keeps down its waste factor. In our method, we schedule some of the jobs in one RPC round earlier than usual. The experimental results show that the total efficiently executed work increases up to 70% and the average increases by 35%.**

Keywords-volunteer cmputing; scheduling mechanism; remote procedure call; EDF algorithm

I. Introduction

Volunteer computing is a type of distributed computing environments in which computer owners share their resources to run one or more projects on a voluntary basis. This method differs from grid computing (with shared and managed resources within or among organizations) [1]. In fact, volunteer computing – also is called Peer-to-Peer - uses a volunteer computer, which is connected to the Internet, as a source to increase computing and storage speed of distributed scientific projects. This computing method is used in molecular biology, pharmacology, astrophysics, meteorology, etc. For example, SETI@home has achieved 668 TeraFLOPS speed by this method.

The main objective in volunteer computing is consuming the idle time of worldwide-distributed computers that are connected to internet, and using them to perform the large-scale distributed applications. Indeed, the volunteer internet-users offer part of the idle time of their computers to perform part of an application. Therefore, volunteer computing is an expansion of the cycle stealing method on the Internet.

Demand of the commercial center managers for using volunteer computing, pervasive property of Internet and growing number of internet-connected devices reveal the performance of volunteer computing. It is not easy to handle non-active resources for creating a Very Large Parallel Computer

(VLPC). Issues in achieving these objectives include: size of the components (server, network, and host) in volunteer computing systems based on the characteristics of applications; high performance and secure execution; management of resources and workload as inputs of scheduling algorithms, and the effect of application characteristics (CPU/data-bound).

Volunteer computing administrators must use appropriate policies to enable guest jobs to run in parallel with local jobs of the volunteer computer. These policies are discussed in four areas: processor scheduling, work fetch, work send and work completion estimation. Indeed, an administrator that wants to have a proper scheduling must know the number of available CPU cycles of user's system for an application, the highest priority and the order of the tasks, an appropriate project to run on user's system, and the validity and corresponding credits of returned results.

As the other types of distributed systems, volunteer computing systems must consider several issues, and solve them. These systems must have a high scalability to both tolerate hundreds of thousands of nodes and increase the efficiency. The owner of a computational resource must be able to define a policy that considers availability of the host. The administrator of the volunteer computing environment must decide based on that policy. The Mean Time Between Failure (MTBF) of internet-connected workstations is another important topic. The introduced architecture of volunteer computing system must be able to tolerate the intermittent failures and to maintain the efficiency in an acceptable value. All the participated computers must be protected against malicious or mistake manipulations. Results of Projects must be protected against malicious hosts' interference and the host/application security must be provided. The system must be able to both tolerate environment dynamism and adapt itself with the types of configuration, communication delays and throughput.

Volunteer computing has several issues that need to be addressed to obtain an efficient system. Most volunteer systems are in a private network or have limited connectivity. So, it is impossible for the server to contact clients directly. Some clients have a low connectivity. Therefore, the client should ensure the resource would not run out of work. Another important issue is that the results that are received are not always correct. Volunteers do not have a continuous availability, so the server should maintain the task in its database for a long period. If the scheduling policy of the project is prone to waste resources, volunteers may stop supporting the project. There are several ways to detect and analyze of these anomalies in large distributed systems [16].

Currently, one of the most important issues is the scheduling; i.e., how jobs must be organized on the host.

II. BOINC MIDDLEWARE

The encouraging users to continue their participation is the administrators' goal, so it is suggested to use a middleware that performs coordination without interference of the user so that the user only should establish the first session and determine the conditions and afterward he will not involve with the issues of the execution of volunteer computing projects. One of the well-known middlewares is BOINC that runs on both the host and the server's systems. BOINC system is one of the most popular systems. This system has the universal code (LGPL) and is made from development of the NSF-funded research project at UC Berkeley Space Sciences Laboratory. This system provides the host software for Windows, Mac x OS, Linux and different types of UNIX [4]. Currently, there are about 40 BOINC-based projects and hundreds of thousands of volunteer computers perform about 8.425 PetaFLOPS of these projects per day [9].

Goals of using the BOINC are new issues of volunteers to enter the network and to get started, sharing resources among fully autonomous and independent projects, supporting different applications (it is clear that each issue needs its own requirements).

These systems have two types of servers: a scheduling server and a data server. The scheduling server uses the Remote Procedure Call (RPC). It distributes tasks and aggregates the results of tasks. The data server selects files to transmit and handles file transfer. This selection is based on the mechanisms of authentication (to block malicious requests). In addition, there is a database that stores description of each application, volunteer, scheduling, etc. The BOINC software includes server-side and host-side components. The host-side component runs BOINC-based projects in the application level and is responsible to CPU scheduling method, implemented on the higher level of a local OS scheduler. The host will initialize all the communications; the host requests jobs from servers of attached projects alternately. Some hosts are connected to the physical network periodically (e.g., portable or dial-up computers). These computers may have an off-and-on connection. In similar situations, BOINC tries to provide enough jobs for these systems to keep them busy until the next connection [13].

One of the most important issues for servers is the local scheduling. The purpose of the scheduling is to maximize the use of resources (keeping users busy), successfully completing the jobs in their deadlines, allocating resources among user-accepted projects fairly, and satisfying the user by executing variable projects.

BOINC-based projects are autonomous. Each project has a server that includes several components: a web interface, a task server and a data server. A task server can be configured by any policy of the scheduling policies. In all policies, when a job is going to be transmitted to a host, which has sufficient disk and memory, and if the host is able to complete it in its deadline, the job may be elected to transmit. A task server is responsible for generating of the tasks, transmitting them to the hosts and processing the returned tasks.

In volunteer computing environment, a host requests a (collection of jobs) from an application server through a workRequest call. The middleware of volunteer system describes its run-time environment (e.g., OS and architecture) and a list of received and stored items in its local cache directory. Based on this information, the server selects a collection of tasks, and returns a description of each task, inputs of task, host-suitable parts of the application and a server address, that host can save the results on it. At the end of the computing, the host uses a workResult call to return results to the specified address. This call will be mirrored to reach the server provided the work to announce the end of it [6].

As compared with other types of the high performance computing, volunteer computing has a very high diversity. Volunteer computers differ in software and hardware, speed, availability, reliability, network connection and other features. Similarly, tasks and applications vary in hardware requirements and completion time constraints [2, 3].

These differences introduce a new set of issues in volunteer computing environment. The first issue is the work selection problem: Whenever a host connects to a task server, the server should select the best work set from a database with millions of jobs based on a complex set of criteria. Furthermore, the server has to service hundreds of these requests per second.

Since the server-side scheduling and an appropriate task selection make conditions so variable, optimization of used policies in this area has an effective role in increasing volunteers' tendency to offer their resources.

The scheduler consists of two parts:

- **Dispatcher**. This part selects a group of tasks from task database and gives it to the scheduler. Task selection procedure follows a simple pattern. First, the dispatcher chooses the best application to determine the task. Priorities are configured based on the minimum execution ratio of the task for each application. Then, the dispatcher selects the last published task for the application.
- **Scheduler**. This part is responsible for receiving the tasks. Tasks are scheduled with First-In-First-Out (FIFO) algorithm. When a host requests a work set, the scheduler offers a host-compatible job set. Two examples of a job never transmitted to the same participate. In addition, the server marks incomplete jobs with a workAlive signal and schedules them again if they had not completed in their earlier execution.

Policies of the dispatcher and the scheduler can be configured dynamically.

In the basic job selection policy, job cache is parsed in a random time. For each job, parsing results that do not need the database are marked. For example, it is checked if the host memory and/or the disk space is enough or not, and if the host is able to complete the job in its deadline or not? After this step, the job is locked. Jobs, that do not need to get access the database, are marked. The questions like this: Is there a sample of job that is assigned to a specific host? This procedure continues until appropriate jobs, which satisfy job request of the host, are selected.

III. Scheduling Policies

The Scheduling mechanism of a volunteer system is as follow:

The server divides the computational work into some tasks and allocates each task to a volunteer system. When a task execution has completed, volunteer system sends the results back to the server. Due to received results, the server raises or disrates credit of the volunteer.

In [14], authors mention that burst projects, with fewer tasks and interested in response time, have emerged. Many works have proposed new scheduling algorithms to optimize individual response time but their use may be problematic in presence of other projects. In that article, they show that the commonly used BOINC scheduling algorithms are unable to enforce fairness and project isolation. They believe that burst projects and considering different scheduling parameters may dramatically affect the performance of all other projects (burst or non-burst). They declare that the non-cooperative optimization may result in inefficient and unfair share of the resources. Also, see [15].

Scheduling policies have different inputs. The first input is the host's hardware characteristics, including the number of CPUs and benchmarks. The BOINC client uses different characteristics like active-fraction (the fraction of time that BOINC middleware is running, and is permitted to do the computing), and statistics for network connections.

Second input is the user's priority. The BOINC client enables the users to determine their priorities and the pattern of accessibility of volunteer computing projects to their system. These priorities, applied to all the connected projects, are as follow:

- Priority of tasks; giving top priority to the user's application;
- Share of the source for each project; users can divide resources among projects so that the system is prevented from bottleneck conditions.
- CPU usage limitation; when the host system is running assigned jobs, the maximum number of usable CPU cycles, and the maximum ratio of time that the project can use the CPU (e.g., to minimize CPU heat of the user) can be specified.
- Communication time; the range of the time that the BOINC middleware is allowed to communicate with server-side middleware can be determined.
- Confirmation before a connection; the user should allow the BOINC client before establishing a communication. This priority is appropriate for dial-up connected users and applications with low delay.
- The minimum connection interval; the minimum time between network activity periods. This priority allows the users to offer a hint that if a host (e.g., a laptop) is connected to the network periodically, how long the client middleware can connect to the network. The server fetches enough

jobs for the host to occupy the system. On the other hand, this priority allows the dial-up connected users to tell the BOINC middleware about the scheduled intervals of network connection. In fact, it allows the dial-up connected users (that pay the cost of their connection) to manage the communications in a way that middleware do most of required displacements to continue jobs in one connection interval.

- Schedule intervals; the policy can specify parts of the time that the host's CPU Scheduler must consider to schedule tasks (default value is one hour).
- The number of jobs that can be saved on the host;
- The number of additional jobs that can be saved on the host;
- Disk access intervals; the minimum time between the disk accesses can be determined. This priority seems appropriate for laptops with low power.
- The maximum usage of disk; the maximum amount of host's disk that the BOINC client is allowed to use.
- The minimum disk spaces; the minimum amount of host's disk that is free [10].

Finally, each job (which is work unit) has the specific parameters, including an estimated number of floating-point operations (FPOPS) and a deadline to report the results. Most of the BOINC projects use the redundancy computing, where two or more pieces of work set are executed on different hosts (with the almost similar systems). If the host system does not return the work set in its deadline, the user will not receive credit.

As mentioned in the previous session, BOINC is the most famous and widely used software in the volunteer computing environment. Process of receiving and executing jobs by the BOINC middleware follows the four dependent policies:

- CPU scheduling; which of the currently executable jobs is the best choice to run?
- Work fetch; when the host decides to request more jobs for a project, which project is the best choice and how many jobs should be requested?
- Work completion estimation; how is the remaining of the CPU time estimated for a job?
- Work send; when the server receives a project request from a host system, which jobs are the best choices to send?

These policies have a great effect on the efficiency of BOINC-based projects. For evaluations, parameters such as the job size deviation, the deadline, and the number of attached projects are investigated.

To evaluate the scheduling policies, following metrics are used:

- *Idleness*. Idleness duration in an interval of availability of the CPU is the fraction of CPU cycles in which jobs are not executed. The range of this factor is (0, 1) and ideal value of it is zero.
- *Waste*. Waste duration in an interval of availability of the CPU is the fraction of CPU cycles that are used by jobs that miss their deadlines.
- *Share Violation*. A Criterion emphasizes the user's resource share. In the best case, its value is "0" and in the worst case is "1" (satisfactory case for guest applications is assigning higher importance to them than user's jobs).
- *Monotony*. Reverse value of the number of switching of the project on the host; "0" means that the number of switches is the maximum (based on the scheduling intervals and other factors) and "1" means that no switch action has occurred. This factor represents to assign equal share of resources to all projects.

A. Cpu Scheduling Policies

Using an appropriate policy, the host-side BOINC middleware manages projects in a way that avoids the interference of BOINC projects with jobs of the host system and organizes jobs in a way that maximizes the resource utilization. Two basic policies exist. In the Round-Robin scheduling policy, the time division among projects is based on the weight (share) of each project from available resources of volunteer host. In other words, the host assigns weight to each project based on the portion of each project from a resource and runs jobs according to their weights. Alternate method is the weighted Round-Robin scheduling policy, where jobs that may miss their deadlines are determined. Scheduling plan of these jobs is running the job with earlier deadline (Earliest Deadline First, EDF). The weighted Round-Robin scheduling policy has the least waste value due to considering deadlines. Although growing the number of projects increases the waste - because deadlines become similar -, but still results are better than that of the Round-Robin scheduling policy.

In a simulation of workload of the real projects, it must be ensured that the combination of jobs with different sizes and deadline exists. Thus, "mix" is defined. Each workload has a mix with size M, where M is the number of projects. The host system accepts a project called P_i, where $1<=i<=M$. Each project (P_i) has following characteristics:

(1) mean value of FPOPS for each job = i * basic size of FPOPS

(2) latency bound = i * base latency bound

(3) resource share of P_i = i * base resource share

The basic values are input values.

The projects in the similar categories with similar characteristics are stored (e.g., chemical computing) and the user accepts the projects with similar groups. Therefore, the user can accept a large number of projects.

B. Work Fetch Policies

Work fetch has two basic policies. In the first policy, the host requests at least one job for all the projects either running on the system or in its queue. This new queue is divided among projects based on their resource share. Since this policy pays no attention to the power of the host system to implement projects effectively, it wastes the system's processing capacity. To solve this problem, another policy is suggested. It considers the amount of executed jobs for each project and maintains that. This policy uses a simulation of weighted Round-Robin scheduling policy to estimate CPU shortfall. In other words, the scheduler selects any project that has both the highest total amount of run jobs and the CPU shortfall to fetch. However, jobs with scant deadlines will be avoided. Both policies are done on the host side. Later policy considers the amount of executed jobs for each project. Therefore, its waste factor is less than earlier.

C. Work Completion Estimation Policies

Both host and server should have an estimation of job completion time. This estimation is used when the host is going to request more jobs from the server, and the server is going to transmit more jobs to the host. Whatever this estimation is close to the actual amount, the probability of efficiently execution of projects will be increase.

The BOINC application reports the fraction of the work that is executed. Result of estimating confidence is increased by execution of tasks. One method to estimate completion time of currently executing jobs is using following estimation:

$$[F * A + (1 - F) B],$$

where "F" is a priority coefficient, "A" is an estimation based on the elapsed time of CPU cycles and the executed fraction of the job, and "B" is based on benchmark estimations, the number of FPOPS, user priorities and CPU efficiency (the ratio of average CPU cycle used to maximize project's execution duration).

There is an alternative method to estimate the job completion time. This method uses the Duration Correction Factor (DCF) for each project; i.e., it estimates the ratio of time that the job is executed on the host's CPU to the estimated CPU usage of the job. In this corrective method, there are sudden increases, but exponential decreases. Evaluations demonstrate that using the DCF policy corrects estimated value faster than that of the earlier method.

D. Work Send Policies

Two methods have been proposed to implement work send policy.

In the first method (which is shown in results with X label), for a host request with the size of "x" seconds, the server selects jobs that their total estimated execution time is at least "x" seconds and sends them. So, if the host disconnects, there will be enough jobs to keep its system busy until the next connection presents. This method provides jobs due to the host request, so it can decrease the idleness metric based on the requested size of jobs. Instead, it will increase the waste criterion; because jobs are selected regardless of deadlines of existence jobs on the host system (either ready-to-run or running jobs). Therefore, influence of adding a new job to previous jobs is not considered, previous jobs may be corrupted, and their deadline restrictions may not be met. This issue reduces the amount of efficiently executed jobs done by the host system.

To solve the problems of previous methods, an alternative policy has been suggested, where the server uses EDF simulation. As noted, the BOINC middleware uses EDF algorithm (or its derivations) to implement the scheduling of the host's CPU cycles. Thus, when a workRequest is received from a host, the server implements the EDF algorithm. This algorithm is used in the host system to select new jobs. Work set request message from a host includes a list of all the in-progress jobs on the host system, with their deadlines and completion times. The server selects new jobs which are compatible with the host from its database (as a job set) and runs EDF simulation using information of this new job set and that of not-completed jobs' (which are received from the host). These information includes their deadlines and estimated completion times. In fact, this method simulates the host CPU scheduling algorithm (EDF method) for existing jobs on the host system and the new elected job. If the results of this simulation shows no interference in jobs' normal execution or does not force a deadline miss for this new job or running-jobs on the host (i.e., all jobs will be able to complete within their deadlines), the new job will be selected for transmission. Using this policy, the server collects an appropriate work set and sends it to the host system. In this policy, jobs will be evaluated before transmission to consider the deadline limitations. So, it is able to improve the utilization of the host system and reduce the waste criterion. Election procedure in this policy is bound by deadline restrictions. Therefore, it cannot use all the capacity of the host system to implement the applications; parts of the donated CPU cycles of the host system may not be used whereupon the host system power may be wasted. This issue is against the volunteer computing nature that tries to use CPU cycles of the volunteer system efficiently to perform volunteer applications sooner and faster.

In this paper, purpose was reducing the idleness factor of the host system and simultaneously keeping the waste factor at the low rate, whereupon the number of efficiently executed work (i.e., a set of jobs that have successfully completed within their deadlines) would be increased.

IV. The Proposed Approach

Since a good scheduling and an appropriate task selection on the server-side middleware can greatly alter the conditions, optimization of the used policies will have an effective role in expanding the volunteers' tendency to offer their resources to this environment.

As noted, none of the currently used policies of the work send issue can maximize the usage of CPU capacity the host system (reasons were mentioned earlier). Therefore, we proposed a new mechanism here, called XEDF simulation policy. This policy uses two steps for selecting and transmitting the jobs.

In the first step, similar to the EDF simulation policy, new jobs are selected as many as possible. The server puts the selected jobs in an initial queue, named **initial_work** queue. As noted before, the choice is made according to deadline restrictions.

In the next step, the server provides a new set of jobs for the idle time of the host system; called **extra_work** queue. These new jobs are selected from the jobs that have greater deadlines. This distance can be at least equal to the sum of the total size of the remained execution time of the existence jobs on the host and the new jobs that are selected in the first step. In fact, we use a post-processing method. The server selects the jobs that can be possibly sent to the host during the next work request. Similar to the first step, the server selects the new jobs according to the mentioned condition in the EDF simulator. Now, we have two sets of jobs; one set includes the jobs selected by the host in the first step and another includes the jobs that existed on the host before the last request. In the second step, priority of these sets are assumed to be higher than the extra_work set and all jobs after completion of these jobs are new (the reasons of this assumption will be explained later).

Then, the selected jobs in the first step (*initial_work* set) and the selected jobs in the second step (*extra_work* set) are sent to the host. First, the host runs the available jobs in its system along the new selected jobs in the first step (*initial_work*). If the host experiences an idle cycle on its CPU due to the completion of the jobs of the first queue, it moves the jobs selected in the second step (*extra_work* set) to the ready queue in order to schedule and execute them. In addition, the host enables the flag of the *extra_work* set usage.

In the next work request of the host from the server, the host reports the *extra_work* usage flag to the server besides the information of the existing jobs. If this flag is not activated, the server finds that the jobs of the *extra_work* set are not used. Therefore, the server protects this collection in *extra_work* set. Then, in the second step, it chooses the new jobs based on these jobs. During this procedure, the jobs with missed deadlines or the jobs with useless execution (the jobs that their deadlines are before the next connection) are removed from the queue.

The results showed that this work send policy reduced idleness, had no tangible increase in waste factor and ultimately increased the total efficiently executed work in the host system.

V. EVALUATING OF THE PROPOSED APPROACH

A. Scheduling Scenarios

The simulator used for evaluating the proposed policy was *ClientSim* simulator [9]. This tool simulates the logic of the CPU scheduling policy, the work fetch policy, the job completion time estimation using DCF and the work send policy in each host. Core of the simulator is based on BOINC. It was possible to separate the code of scheduling from the code of network and memory access. Therefore, with a few changes, the simulator is connected to the common scheduling code of BOINC. In fact, only the part of the BOINC code, which is used to access the network (RPC for communication between servers and projects), is replaced with the stubs of simulator (stub is a part of the work that executes the middleware role). Therefore, the implemented scheduling mechanism and most of the main code of the simulated host are known for the BOINC source code [10].

The simulator accepts and uses some basic characteristics for each host, middleware and each project.

Parameters for each host are:

- Number and speed of CPUs,
- Fraction of time that the host is available,
- Availability intervals as an exponential distribution with λ_{avail} parameter,
- Connection intervals of the host system.

Parameters for the BOINC middleware on the host side are:

- CPU scheduling intervals,
- Amount of queued jobs,
- Amount of additional jobs that can be stored,
- Maximum number of CPUs, memory and disk that can be used by BOINC projects,
- Maximum resource share of BOINC projects from the CPU of the host,

Parameters for each project are:

- A resource share of each project,
- A latency bound (i.e., a job deadline),
- An estimated FPOPS for each job,
- Normal distribution of FPOPS for each job,

Any combination of these parameters is called a scenario.

We defined two projects to evaluate the proposed XEDF simulation policy and compared it with other policies. Each project had a different latency bound and a different job size (FPOPS). A standard deviation was defined for the execution duration of each job in each project. This means that specific processing time of each job can be calculated from the assumed standard deviation of job size of a project as usual.

Simulation duration was 27 days and each CPU scheduling cycle was assumed 60 seconds. The rate of the availability of the host was considered 0.8 [10], [11] and the range of the availability was calculated using an exponential distribution with λ=1000. User access to the internet was given 0.55 per day and the host buffer size was considered 0.55 per day, too (simulation needs a large enough job buffer because the common EDF simulation policy only offers a good response in this mode [10]). The host used two CPU cycles to perform BOINC jobs (the most common number of CPU cycles for the regular Internet users) and the speed of each CPU was set to 1e9 [10]. Resource share of both projects was considered equal. The basic job size in one of projects was twice as big as that of another project and the amount of buffer size was 13.2 times larger than the largest project (thus, changes in standard deviation of the rate of job size in each project ultimate various amounts of job sizes). It should be noted that values of mentioned parameters are based on real estimations.

The studies were based on two criteria: waste and idleness (which were discussed in part 3). Consequently, the total amount of efficiently executed work that can be run by the host system was considered as the third criterion.

B. Evaluation Results

Except the EDF algorithm, the processor can be scheduled with the Least Slack Time (LST) algorithm (also called Least Laxity First, LLF), too. In this algorithm, the lowest priority is assigned to the job with the lowest **laxity**. The *laxity* is the maximum time that a job can wait for CPU usage (with basic speed) without missing its deadline.

The value of laxity was calculated from the following formula:

$$T_{rem} = E - (t - R),$$

$$T_{slack} = D - (t + T_{rem}),$$

where "E" is the execution duration of the job, "R" is the arrival time of the job to the system, "D" is the deadline of job, "T_{rem}" is the remaining time of the start of the job execution, "T_{slack}" is the amount of laxity of the job at the time of $t<D$.

Since the laxity of a job varies in the time, its priority will change, too. Of course, there are different versions of this algorithm, but all of them follow the above pattern for laxity [12].

We considered the LST algorithm, alongwith the EDF algorithm in the present evaluation, because they are two of known algorithms in distributed computing and volunteer computing environments. Indeed, the CPU scheduling of the host system was implemented in two methods using the EDF algorithm and the LST algorithm whereupon the work send policies were divided into four categories including:

- A policy based on "x" seconds work request (mentioned as X_Sched)
- EDF simulation policy (mentioned as EDF_Sched)
- LST simulation policy (mentioned as LST_Sched. It is similar to the EDF simulation policy. Their difference is in the job selection method. The earlier method uses deadline restriction, whereas the later one uses laxities)
- XEDF simulation policy (mentioned as XEDF_Sched)

First, the total amount of efficiently executed work that can be performed by the host (the number of executed jobs that were completed successfully), the idleness factor and the waste factor of the volunteer system were studied. The evaluations were based on various amounts of latency bounds for each job. The amount of latency bounds was set based on the ratio of the size of the host's system buffer and this ratio varied between 0.3 and 4.5 times larger than the buffer size.

According to **Fig. 1**, it can be determined that, for each of the four policies mentioned above, when the latency bound increased, the total efficiently executed work increased, too. This increase is because the pressure on the CPU of the host system to perform tasks faster decreased while the latency bound increased. However, when we used the XEDF simulation policy, while the latency bound increased, the total amount of efficiently executed work that was performed by the host was more than that of other three policies. The reason was that the host used the *extra_work* set for the idle time of its system. In fact, when the host system experimented an idle cycle, it could transfer the *extra_work* set to its ready queue to be run thus it can prevent the CPU from remaining idle.

Fig. 2 shows the idleness fraction of the host system when it used each of the implemented policies with different latency bounds. As can be seen, with increasing the size of the latency bound for each policy, the idleness fraction was reduced. This is because the server could choose more jobs than before (based on the deadline restriction in two of the policies and the laxity restriction in the other one). However, it can be seen that the percentage of the idleness factor in the XEDF simulation policy was less than that of other policies. This is because our proposed policy used both the *extra_work* set and a prediction of the idle time of the host system. It may sound that the reduction was not significant, but it is better to note that the value of the idleness fraction should be considered along with the waste fraction and the total amount of the efficiently executed work that was performed by the host. In other words, the denominator used to calculate the idleness factor had different values in each of the four policies (it was equal to the total number of jobs available in the host system). This reduction in conjunction with the total amount of efficiently executed work and the waste factor should be considered in evaluation. In the XEDF simulation policy, the idleness factor incremented for higher latency bounds in comparison with other policies. This is because the number of jobs that could be sent by the server to the host system was limited by the buffer size of the host system and the server could not transmit jobs more than the buffer size of the volunteer system.

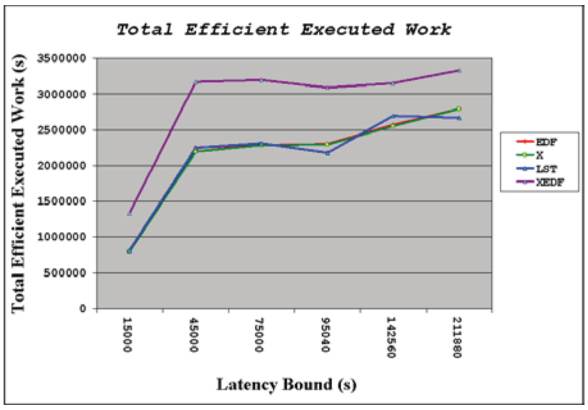

Fig. 1 The comparison of the total efficiently executed work had done by the host system

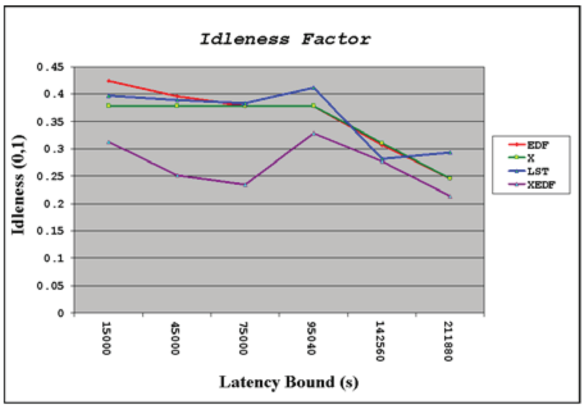

Fig. 2 The comparison of the idleness factor of the host system

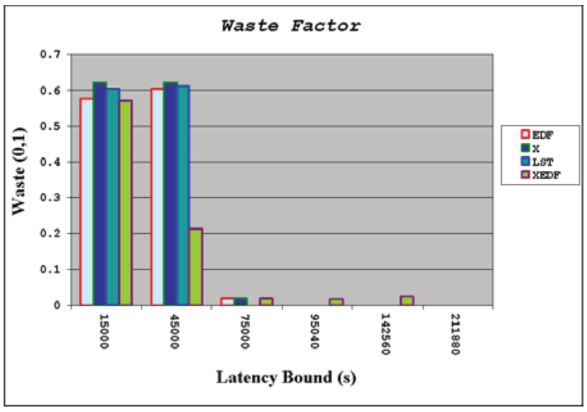

Fig. 3 The comparison of the waste fraction of the host system

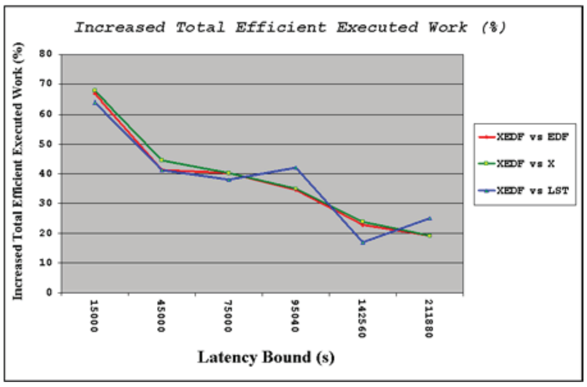

Fig. 4 The increased percentage of the total efficiently executed work had done by the host system (utilization)

Fig. 3 demonstrates the waste factor of the host system. As can be observed, when the latency bound of jobs was smaller than the buffer size of the host system, the X policy should cause higher waste fraction due to its lack of attention to the deadline restriction when it elected work. The XEDF simulation policy seemed better than the EDF simulation policy and the LST simulation policy; because it received more jobs than others and also it selected jobs based on the server limitation of deadlines (which was also affected the laxity). The XEDF simulation policy selected all jobs according to this restriction. It caused less idleness fraction beside higher executed work and less waste fraction. When the latency bound was larger than the buffer size of the host, all four policies had very little waste fraction. In this case, waste fraction of the XEDF simulation policy increased slightly. This (slightly) increase could be omitted when we considers the increase in the total efficiently executed work.

Finally, **Fig. 4** presents the increased percentage of the total amount of efficiently executed work that was performed by the host using the XEDF simulation policy in comparison with other policies. It can be seen that the total amount of efficiently executed work increased up to 70% and the average increased by 38%.

In the next scenario, the effect of increasing the standard deviation of each job (the job's size in each project) was studied. This study was considered because of evaluating the effect of variance of jobs' size among three discussed criteria. In this case, different sizes of jobs were presented when the host received the jobs from different projects; so each job had a different size.

Two different modes were considered for evaluation. These cases were based on the ratio of the latency bound to the host system's buffer size. In the first case, a situation was considered in which the latency bound of projects was larger than the host system's buffer size. In the second case, the latency bound of projects was less than the buffer size. This may put the host under pressure because it should complete the jobs in a short period.

Fig. 5 gives the amount of total efficiently executed work done by the host for different standard deviations in the first case. The standard deviation of "10" meant that the jobs' sizes of projects were getting close to one another and the standard deviation of "1000" meant that the jobs' sizes had much different values. It can be seen that when the standard deviation increased, the total efficiently executed work decreased; because the server was limited in work selection since slots were larger and may not fit in the remained suggested time in the X policy or may cause missing deadlines in other simulation policies (EDF, LST and XEDF). However, the results indicated that the XEDF simulation policy improved the total efficiently executed work more than other policies. In addition, unlike other policies, with increasing the standard deviation, the total efficiently executed work had less decrease in the XEDF simulation policy. Because, when the number of jobs increased, the probability of ascending the size of jobs in the *extra_work* and *initial_work* sets and allowed work selection range increased. Therefore, increasing the standard deviation had less effect.

Fig. 6 shows the idleness fraction of the host system with increasing standard deviation in the first case. It can be seen that, in all four policies, with increasing the standard deviation, the idleness factor increased (the cause of this event was mentioned before in the total efficiently executed work). Nevertheless, the idleness fraction of the host system in the XEDF simulation policy was less than those of other policies; because it predicted the idle time of the host and provided extra jobs for these times. Consequently, the amount of total efficiently executed work for the project increased.

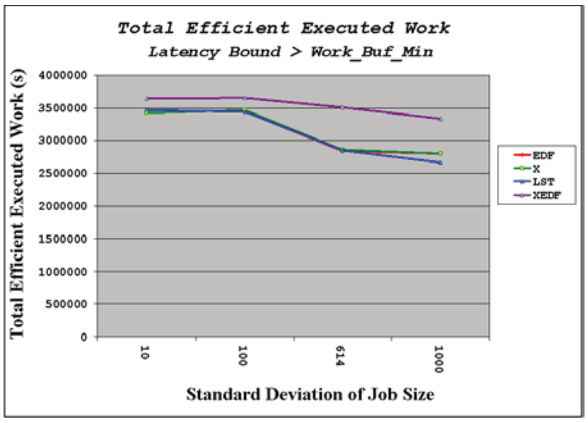

Fig. 5 The comparison of the total efficiently executed work had done by the host system when the latency bound is higher than the host system's buffer size.

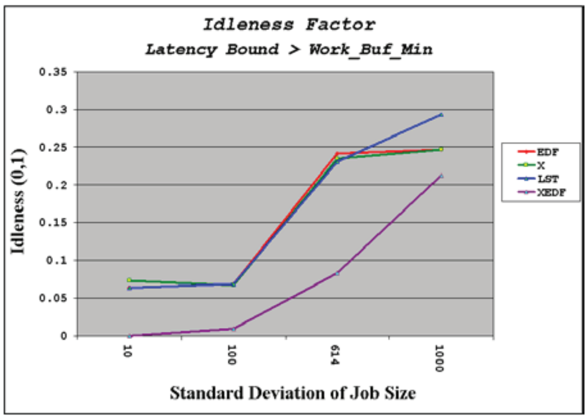

Fig. 6 The comparison of the idleness factor of the host system when the latency bound is higher than the host system's buffer size.

Fig. 7 presents the waste factor of the host system based on the standard deviation increment. While increasing the standard deviation, the waste factor remained low for four policies because the number of jobs downloaded by the host was few. Of course, the XEDF simulation policy had higher (very partial) waste fraction than other policies, but it was close to others when the standard deviation increased. The reason of this was due to downloading more jobs by the host system in the XEDF simulation policy compared with other policies. So, the proposed policy in this paper can offset this partial increase by the increase in the total efficiently executed work which was done by the host system.

Fig. 8 shows the amount of total efficiently executed work performed by the host in the second case, i.e., the case in which the latency bound was smaller than the buffer size. While the standard deviation increased, the total efficiently executed work increased using the XEDF simulation policy. Of course, the amount of executed work was less than the case in which the latency bound was larger than the buffer size; because the work selection was limited. However, considering the existence of a secondary queue for idle times of the host system in the XEDF simulation policy, this method was able to perform more efficiently jobs relative to other methods. In addition, it can be seen that, the less the diversity of work size based on FPOPS, the better the results provided by both the EDF and LST simulation policies, compared with the X and proposed policies.

Fig. 9 demonstrates the idleness factor of the host system with the standard deviation increment in the second case. As can be seen, with the standard deviation increase, the idleness factor increased for all the four policies (the reason was mentioned before). Nevertheless, it was always less for the X and XEDF simulation policies. The reason is that in the EDF and LST simulation policies, the server could not select and send a large amount of work for the host. These two policies had a less amount of latency bound than the buffer size of the host. If the number of jobs increased, the host would not be able to complete the jobs in their deadlines. However, the X policy that selected jobs without considering the tasks' deadlines sent them according to the host's demand; therefore, its idleness factor was less. In addition, in the XEDF simulation policy due to the existence of the secondary queue, the previous case would be true. In fact, this amount of idleness must be considered with the total available jobs in the host system; consider **Fig. 8** and 10.

Fig. 10 presents the waste factor of the host system in the second case with different values of standard deviation. As can be seen, with increasing standard deviation, the waste factor always decreased; because the number of downloaded jobs decreases. Nevertheless, this reduction was faster for the XEDF simulation policy; because the number of downloaded jobs was more than that of other policies and the fast reduction of waste factor, lower value of idleness factor and higher amount of totally downloaded jobs resulted in an increase in the amount of total efficiently executed work.

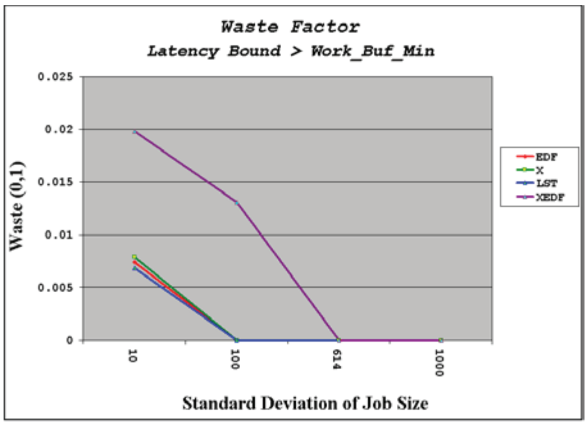

Fig. 7 The comparison of the waste factor of the host system when the latency bound is higher than the host system's buffer size.

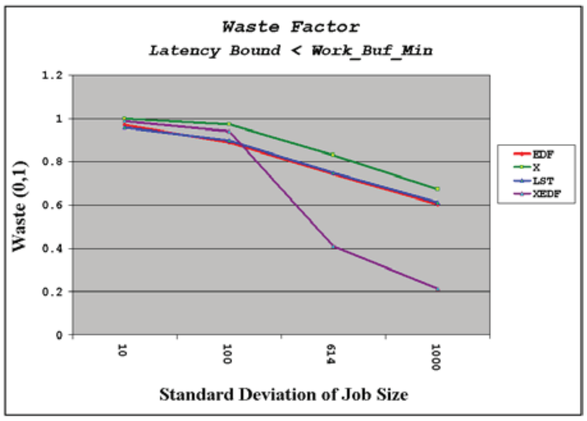

Fig. 10 The comparison of the waste factor of the host system when the latency bound is higher than the host system's buffer size.

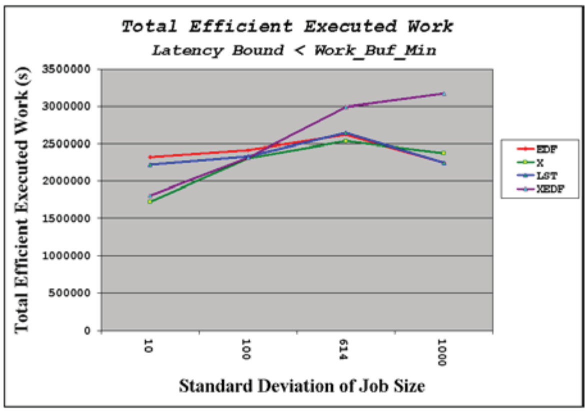

Fig. 8 The comparison of total efficiently executed work had done by the host system when the latency bound is smaller than the host system's buffer size.

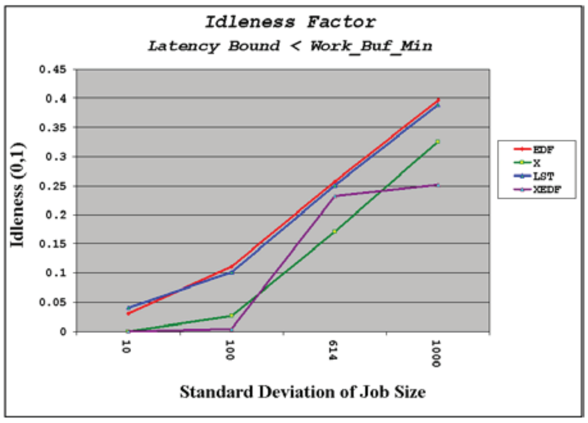

Fig. 9 The comparison of the idleness factor of the host system when the latency bound is smaller than the host system's buffer size.

VI. CONCLUSION

Human beings are developing in the science world and along this progress, the need for new and, of course, cheaper resources increases. Therefore, using the introduced computational method can effectively help in solving scientific computational problems (along the storage issues).

Currently, companies find earlier computational solutions or tools very expensive. Therefore, they can benefit from the volunteer computing environment. For example, manufacturers can use these features in computational analysis and processing simulations of complex production lines. This will fire the bottleneck problem and weaknesses in the effects of processing and prediction of changes and help to make jobs cheaper and more efficient.

One of the very important issues in volunteer computing is the middleware; what is the structure of the middleware, how much complexity must be tolerated or what architecture should be followed and issues like these. The volunteer computing network administrators are careful, but they have not full control on their resources. Therefore, the scheduling issue is important. Most executed work is done according to frequent estimations. Server scheduling is a part of this scheduling issue and optimizing it will help in increasing the number of jobs for the projects.

In this article, using the XEDF simulation policy, there was an attempt to upload more jobs on the host system and increase the total efficiently executed work done by the host system. It can be seen that, the total amount of efficiently executed work increased up to 70% and the average increased by 35%. This improvement can increase the speed of the project's execution; thus, the scientific goals can be improved – by efficiently using the host system.

REFERENCES

[1] L. F. G. Sarmenta, Volunteer Computing, Ph.D. Dissertation, Massachusetts Institute of Technology, Supervisor: Stephen A. Ward, June 2001.

[2] G. Fedak, C. Germain, V. N´eri, F. Cappello, "XtremWeb: A Generic Global Computing System", Laboratories de Recherché en Informatique, Universities Paris Sud, December 13, 2000.

[3] L.F.G. Sarmenta, "Bayanihan: Web-Based Volunteer Computing Using Java", Lecture Notes in Computer Science 1368, Springer-Verlag, 1998, pp. 444-461, Proc. of the 2nd International Conference on World-Wide Computing and its Applications (WWCA'98), Tsukuba, Japan, March 3-4, 1998.

[4] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage", Space Sciences Laboratory, University of California at Berkeley, 2004.

[5] D. P. Anderson, C. Christensen, B. Allen, "Designing a Runtime System for Volunteer Computing", IEEE, SC2006, 0-7695-2700-0/06, November 2006.

[6] D. P. Anderson, E. Korpela, R. Walton, "High-Performance Task Distribution for Volunteer Computing", Space Sciences Laboratory University of California, Berkeley, 2005.

[7] D. Kondo, D. P. Anderson, J. McLeod VII, "Performance Evaluation of Scheduling Policies for Volunteer Computing", 3rd IEEE International Conference on e-Science and Grid Computing e-Science'07, 2007.

[8] D. P. Anderson, G. Fedak, "The Computational and Storage Potential of Volunteer Computing", Technical Report, 2006.

[9] URL. http://boinc.berkeley.edu/.

[10] D. Kondo, M. Taufer, C. Brooks, H. Casanova, A. Chien, "Characterizing and Evaluating Desktop Grids: An Empirical Study", In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04), April 2004.

[11] XtremLab. http://xtremlab.lri.fr/.

[12] H. Jin, H. A. Wang, Q. Wang, G.Z. Dai, "An Improved Least-Slack-First Scheduling Algorithm", Journal of Software, 2004.

[13] G. M. Kumar, "A Survey of Desktop Grid System Scheduling", International Journal of Computer Science & Engineering Technology (IJCSET), Mar 2013.

[14] B. Donassolo, A.Legrand, C. Geyer, "Non-Cooperative Scheduling Considered Harmful in Collaborative Volunteer Computing Environments", Proceesings of the 11th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'11), IEEE Computer Society Press, May 2011.

[15] L. M. Schnorr, A. Legrand, J. Vincent, "Multi-scale Analaysis of Large Distributed Computing Systems", Proceedings of the third international workshop on Large-scale system and application performance, LSAP'11, 2011.

[16] L. M. Schnorr, A. Legrand, J. Vincent, "Detection and Analysis of Resource Usage Anomalies in Large Distributed Systems Through Multi-scale Visualization", Wiley Interscience, 2011.

Mohammad Kazem Akbari received his B.Sc. degree from the National University, and the M.Sc. and Ph.D. degrees from the Case Western Reserve University. He is currently a faculty member with the Department of Computer Engineering and IT at Amirkabir University of Technology and the Chair of the IHPCRC.

Tayebe Kianpishe received her B.Sc. degree in computer engineering from the Bahonar University, Kerman, Iran, in 2006, and her M.Sc. degree in computer engineering from Amirkabir University of Technology, Tehran, Iran, in 2010. Her research interests include parallel processing, cloud, grid and cluster computing systems, as well as volunteer computing.