Models and Scheduling Mechanisms for Global Computing Applications

Derrick Kondo[⋆] Henri Casanova[⋆†] Eric Wing[⋆] Francine Berman[⋆†]

[⋆]Computer Science and Engineering Department
University of California, San Diego
9500 Gilman Dr, LA JOLLA CA 92093-0114, USA

[†]San Diego Supercomputer Center
9500 Gilman Dr, LA JOLLA CA 92093-0505, USA

[dkondo,casanova,berman]@cs.ucsd.edu, ewing@gridlab.ucsd.edu

Abstract

Projects like SETI@home have demonstrated the tremendous capabilities of Internet-connected commodity resources. The rapid improvement of commodity components makes the global computing platform increasingly viable for other large-scale data and compute-intensive applications. In this paper, we study how global computing can accommodate new types of applications. We describe a global computing model that captures resource characteristics and instantiate this model with data from several surveys and studies. We propose performance metrics for global computing applications and evaluate two scheduling mechanisms in simulation. We then draw conclusions concerning the development and enhancement of global computing systems.

1. Introduction

1.1. Background

As commodity technology improves and becomes more pervasive, Internet commodity resources are increasingly attractive for large-scale applications. Peer-to-peer (P2P) platforms consist of networked machines that can play the role of both client and server in order to aggregate resources without a centralized point of control. Much of the popularity of P2P is due to the overwhelming success of the SETI@home project [1]. To participate, users download and install the SETI@home client software, which repeatedly downloads work from a central server and reports any signals found in radio telescope data. The project boasts over 500,000 active users, most of which run the SETI@home client on home PC's, and over a cumulative 20 TeraFlop/sec. In comparison, the fastest supercomputer at the time of writing this paper, IBM's ASCI White, has a peak performance of 12.3 TeraFlop/sec. Moreover, the cost of setting up and maintaining SETI@home is less than 1% of the $110 million required for the ASCI White [2].

We describe applications like SETI@home as *global computing* (versus P2P) because they are are at best *a special instance of P2P where clients have the server as their only peer*. Currently, the majority of global computing projects run embarrassingly parallel applications whose tasks can be scheduled independently of one another. This is the best-case application scenario for a platform with heterogeneous and volatile resources.

1.2. Motivation and Approach

We wish to investigate the potential of global computing for applications and usage scenarios that may depart from the traditional SETI@home application model. To this end, we first specify a global computing platform model that takes into account the volatile aspect of resources. We then instantiate the model using data gathered from Internet surveys and studies. We define metrics that quantify the performance of global computing applications. Using a simulator, we study the performance impact of two scheduling mechanisms. Simulation allows us to explore a large number of scenarios in a reproducible and timely fashion.

We use two metrics for application performance on global computing systems namely *throughput*, which is the amount of useful work performed per time unit, and *turnaround time*, which is the time to complete computation and data transfer for a task or a batch of tasks. On one extreme, global computing applications demand solely high throughput, possibly at the expense of turnaround time (e.g. SETI@home). On the other extreme, applications may wish to minimize turnaround time, sacrificing throughput (e.g. a short-lived application). However, many applications lie in the middle of this spectrum and they seek a compromise between low turnaround time and high throughput. For instance, one can envision a compute service where users submit individual jobs and expect them to be completed quickly. The goal is to service a large number of users and to have each user serviced in a timely fashion.

We will explain in Section 2.2.1 that several factors prevent the use of sophisticated scheduling algorithms on the global computing platform. Instead, in Section 3.3, we focus on simple scheduling mechanisms that can be easily deployed and implemented, such as task *duplication* and *timeouts*.

2. Models For Global Computing

2.1. Application Model

We consider applications consisting of input that is divisible into *work units*, corresponding output called *result units*, and an algorithm used to process the work units. Work units are downloaded by clients for processing, and after being computed, the result units are uploaded to a server. Work units *can* have interdependencies, i.e., the processing of one work unit can depend on the result of another. Without loss of generality, we assume work units are of fixed size, and clients can download multiple work units from the server.

In order to provide context for our work, we use an application from computational biology as a case-study [3]. The application searches for significant patterns in the mass of genomic and proteomic data. It is based on a biological string comparison algorithm that uses dynamic programming. The algorithm, derived from Smith-Waterman [4], finds regions of similarity between DNA, RNA, or protein strings. Global computing systems are ideal platforms for such analyses because they scale to thousands of nodes. Work units are formed from data gathered in the various genome or proteome databases. Specifically, each work unit consists of two biological strings to be compared against one another. Due to the completeness of the genome sequencing projects, these strings are often too long to be compared by a single client. As such, the strings can be divided into substrings, from which a group of *dependent* work units are formed.

2.2. Global Computing Platform Model

Our goal is to model a set of Internet-connected machines such as the ones available in current global computing projects. Unlike traditional Grid environments [5], these machines are mostly home PC's running the Windows OS with various network connections. Machines can be powered on and off by the users, and a significant percentage of machines have intermittent network connections (e.g. dial-up or laptop connections) .

The computing platform consists of a central server and a set of many clients that all communicate with the server. Specifically, the server handles all client requests to download work units and upload result units, as our model currently disallows inter-client communication. The server mediates possible data dependencies and work unit synchronizations when applicable. We leave "true" P2P computing for future work.

A client is defined as a host with disk space, CPU, and a network link to the server, and clients exhibit various CPU speeds, network bandwidths, amounts of disk space, and availabilities. A client is allowed to compute work units only during its CPU's idle time.

We assume that each client communicates to the central server via a dedicated network link. This simplifies real network topologies and ignores network contention either at links emanating from the server or at links shared by clients. The performance cost of contention becomes significant when the application's computation to communication ratio is relatively low, i.e., when clients compute work units at a fast pace. For the applications we are considering, the computation to communication ratio is high enough that the application does not become too intrusive with respect to network usage. Usually client computation is on the order of days or weeks and communication is on the order of minutes. Furthermore, possible server-side contention can be reduced in actual implementations by using a server farm for instance.

2.2.1. Programming for a Global Computing Platform

Application development for global computing must account for autonomous clients. Due to client firewalls, dynamic IP addresses, and transient network connections, clients must initiate communication with the server. Therefore, the server does not control when tasks will be distributed to clients. In addition, the server has no means of controlling remote jobs, making job assignment decisions irrevocable. Finally, the server only has partial information about available resources and their availability. These factors make it difficult to use sophisticated scheduling algorithms such as the ones used for parallel or Grid computing. Consequently, we choose to focus on simple scheduling mechanisms that can be easily implemented and deployed as part of global computing systems.

3. Experimental Design and Methods

3.1. Instantiation of Application and Global Computing Model

We define two classes of clients: *conservative* and *extreme*. Each class is described by the characteristics shown in Table 1; Table 2 shows the distributions of network capabilities for each class. Conservative clients model home PC's and have relatively slow and sporadic network bandwidth and are sparsely used during after-work hours. In contrast, extreme clients model PC's at the work place, for example, and as such, are continuously powered on and many have constant and relatively fast broadband network connections.

In Table 1, *average on-time* per day is the time per day that the client is powered on. *Percent idle* is the percentage of the on-time that the client is idle, during which the client can compute work units. *On-range* is the period during the day that computer can be on. *Bursts* are continuous periods when the client is idle and powered on during a day. The size of each burst, b, is defined below:

$$b = \frac{(average\_on\_time) * \%idle}{number\_of\_bursts}$$

The time at which each burst occurs is randomly chosen within the on-range.

One of our contributions in this paper is that we instantiate our model with real statistical data. Specifically, we set the parameter values and standard deviations based on data gathered from various surveys and studies of Internet [1, 6, 7, 8, 9, 10, 11, 12]. Most of the machines participating in global computing projects are PC's running the Windows OS [1]. About one-third of these PC's have processor speeds between 333-449MHz. One fifth of home PC's are between 450MHz-649MHz, and another fifth are between 233-332 MHz. The remaining small percentage of machines are below 232MHz and above 650MHz [9]. We model the CPU speed distribution as normal with a mean of 333MHz and a standard deviation of 200 MHz. About half of home PC's have a 5 GB or higher hard drive [9]. Since disks are becoming cheaper and larger, we assume clients can donate 60 MB of disk space to the project, which is less than 2% of a 5 GB drive. About 12% have high speed connections, such as DSL or cable, and the remainder have 56Kbps speeds or less [9]. As such, 12% of the conservative clients have DSL or cable connections. Average monthly use of a home PC is about 23 hours [8], which averages to 47 minutes per day, and we define on-time for a conservative client accordingly. Most PC's are idle 90% of the time [13], and so, clients spend most of their time processing work units.

In our simulations, independent work units are 2.5 KB in size, and dependent work units are 1.0 KB. Work unit sizes are relatively small so that clients with slow

Client Profiles		
Attribute	*Conservative*	*Extreme*
on-time / day	47 min	24 hours
# of bursts / day	5	1
on-range in day	6:00PM -11:59PM	All day
% idle	90	80
% idle std dev	10	20
CPU speed	333 MHz	333 MHz
CPU std dev	200 MHz	200 MHz
Disk space	60 MB	60 MB

Table 1. Characteristics of conservative and extreme clients

Client Population		
% with network connectivity	*Conservative*	*Extreme*
56kbps modem	88%	50%
Cable, DSL	12%	25%
Ethernet 10BaseT	0%	25%

Table 2. Breakdown of clients connected to different types of networks

network connections or processors can participate in the project (clients with high bandwidth can download many work units at once). We wish to measure application performance on an environment that has reached steady state. As a result we assume that no client joins or leaves the system during a run of an application. Our simulator is implemented in object-oriented PERL, which facilitates substitution of new schedulers, specification of different client classes, and changes in client behavior. The simulator takes as input the values of Tables 1 and 2, the distribution of client types, a description of the application to be executed, and the scheduling mechanisms to use. The simulator outputs all simulation events and the performance metrics defined in the following section.

3.2. Metrics

We define three ways to measure application performance: *work unit turnaround time*, *set turnaround time*, and *work unit throughput*. Work unit turnaround time is the time between the "first" download of a work unit and the "first" upload of the corresponding result unit, ignoring duplication. If the scheduler duplicates work units frequently, then clients tend to download the same work unit, which delays the download of other work units. To account for the negative effect of replication accrued from one work unit to another, we use set turnaround time, which measures the time to complete a fixed-size group, or *set*, of work units. Specifically, set turnaround time measures the interval between the download of the first work unit in the set and upload of the last result unit in the set. As replication increases set turnaround time should also increase. Note that turnaround time includes work unit execution time, but execution time is not a metric that we consider directly. Especially for clients with sporadic on-times, the majority of the turnaround time may be due to the client being powered off instead of actual program execution.

Work unit throughput measures the amount of useful work completed per time unit (excluding duplicated work).

3.3. Scheduling Mechanisms

The server's scheduler determines how many work units and which ones (in the case of dependent work units) are sent to clients. To determine the number of work units to send to a client, the scheduler finds the most limiting constraint of that client. In particular, the scheduler determines the maximum number of work units that can be sent to a client given its processor speed (Max_{cpu}), given its network bandwidth (Max_{net}), or given its disk space (Max_{disk}). The minimum of those three numbers gives the number of work units that will be sent to the client.

We define Max_{cpu} as the number of work units that will keep the average conservative client busy for a week and the average extreme client busy for more than six hours. Estimates for the execution time of a work unit are based on benchmarks of the application's algorithm [4, 3] on several machines. Max_{net} is the maximum number of work or result units that can be downloaded or uploaded respectively in five minutes. We chose a period of five minutes in order to prevent network use by the client software from becoming intrusive. Max_{disk} is the number of work units and corresponding output that can fit on the donated portion of the client's disk when the algorithm checkpoints.

The scheduler uses work unit duplication and timeouts. Duplication is the ability to replicate work units and is specified by the maximum number of times a work unit can be replicated and sent to clients, although the maximum number will not necessarily be reached. Duplication can reduce work unit turnaround time since multiple clients compute the same work unit and the server uses the result that is returned soonest. However, because multiple clients with duplicated work units produce the same results, duplication wastes resources. Timeouts can also cause replication. Timeouts are used to specify the time that must elapse before a work unit can be sent again to a client. In other words, when a work unit times out, the server assumes it will not be returned (e.g. because the client went off-line) within the lifetime of the application. In contrast to duplication, which caps the number of work unit copies to be distributed to clients, work units that timeout are replicated until the server receives the corresponding result units. The information needed about clients and work units is easily obtainable (e.g. client hardware profiles can be collected when a user registers for the project), and so these two mechanisms can be implemented with simple database bookkeeping.

Our simulation experiments measure application performance for different client groups as either duplication or timeouts are varied over a simulated time of 4.5 months. We used five client groups with the following conservative/extreme percentages: 100%/0%, 75%/25%; 50%/50%; 25%/75%; and 0%/100%. We used 1024 clients per group. Each work unit set consisted of 30 work units.

Duplication ranges from 1 to 60, and timeouts range from 90 minutes to 2700 minutes in 90 minute intervals. We also simulate an infinite timeout. The timeout ranges are defined with respect to the average work

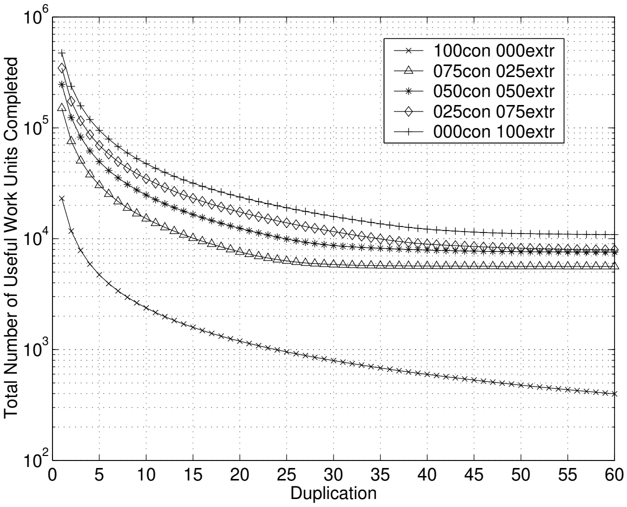

Figure 1. Work unit throughput vs. duplication for independent work units. The trend is the same for set throughput.

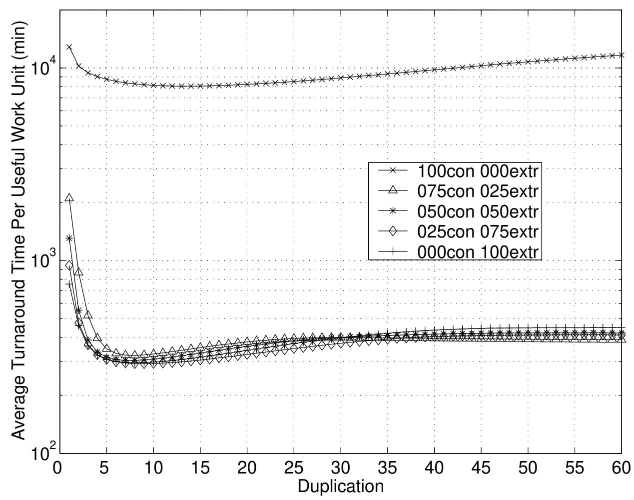

Figure 2. Work unit performance vs. duplication for independent work units.

unit turnaround time. Timeouts far below the average turnaround time cause unnecessary replication whereas timeouts far above the average turnaround time are ineffectual.

Each simulation experiment was repeated 10 times using parameters generated randomly to fit specific distributions (see Section 3.1). A total of 9,250 simulation experiments including repetitions were run on the Meteor cluster at the San Diego Supercomputer Center and the Prospero cluster at the Tokyo Institute of Technology, using the APST software [14].

4. Results and Discussion

All our results are presented with a log scale on the y axis. We computed coefficient of variations (c.v.) for repetitions in our simulations. All c.v. values are under 5% which indicates that we have performed a statistically significant number of repetitions.

4.1. Duplication

As duplication increases, the number of independent work units and work unit sets completed drops exponentially (see Figure 1). The client groups with more extreme clients are obviously more productive. For all client groups, throughput peaks without duplication, i.e., when only one copy of each work unit is sent. At duplication of 20, work unit throughput levels off, and thereafter decreases little.

Similarly, the turnaround time per work unit or set initially drops at a near exponential rate (see Figure 2). However, after duplication of 5, turnaround time begins to rise. The rise is the most evident via the set turnaround time metric (see Figure 3).

For dependent computations, the trends are similar to our observations for independent computations. However, in contrast to the independent case, work unit turnaround times do not noticeably rise as duplication increases. The increase in turnaround time at high levels of duplication appears only with set turnaround time.

Duplication lowers throughput because clients perform redundant computation that wastes computational resources. However, turnaround time also decreases with duplication because it raises the probability that a work unit will be handed to a productive client. For low duplication amounts, turnaround time depends on the the performance of a few clients. As duplication increases, the same work unit is handed out to more clients, and the chance that the replicated work unit is computed by "fast" clients usually becomes higher. However, relatively high levels of duplication can be counterproductive and decrease performance. After the server sends a work unit to an extreme client, sending a replicate to another extreme client with a similar profile might not improve the turnaround time, and instead, might delay the latter extreme client from doing other non-duplicated work.

Turnaround time per work unit is not sensitive enough to efficiently detect a decrease in application performance resulting from excessive duplica-

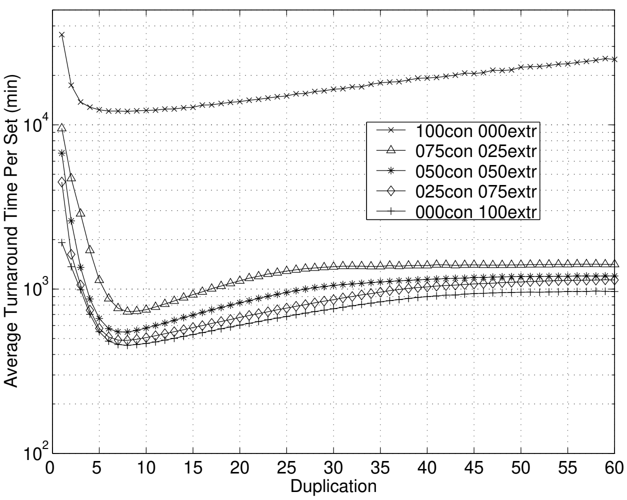

Figure 3. Set performance vs. duplication for independent work units.

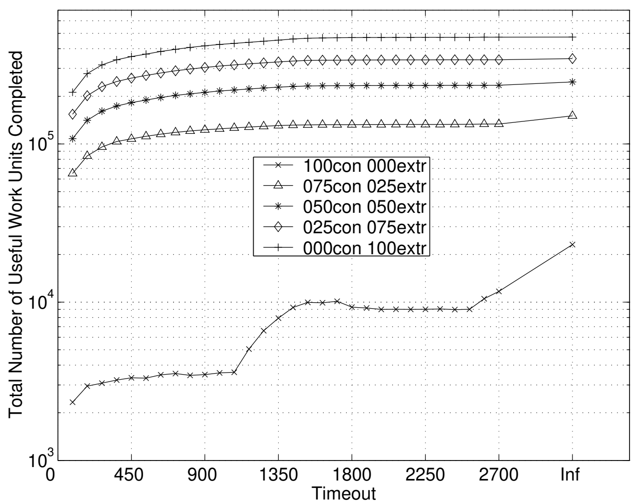

Figure 4. Work unit throughput vs. timeout for independent work units. The trend is the same for set throughput.

tion. In contrast, *set* turnaround time increases noticeably when duplication begins to waste resources. Turnaround time per set is a sensitive metric because it accounts for the negative effects of duplication accrued from one work unit to another.

4.2. Timeouts

Independent work unit and set throughput increase with timeout values (see Figure 4). Also, note that throughput for 2700 minute timeouts is only slightly less than throughput for infinite timeouts. Similarly, work unit and set turnaround times increase with larger timeouts (see Figures 5 and 6). For most client groups, the turnaround time at infinite timeouts is much higher than the time at 2700 minutes. The trend is especially noticeable with set turnaround times. The exception occurs with the 100% extreme client group. The trends in throughput for dependent work units match those for independent work units

The effects of timeouts on throughput are similar to those of duplication. As timeouts become relatively small, work units are replicated more often, which lowers throughput. However, timeouts can result in significant performance gains without necessarily decreasing throughput as seen on Figures 4 and 6.

In addition to lowering set turnaround times, timeouts prevent high levels of replication that are counterproductive with respect to application performance. For the reasons described in Section 3.3, one might expect the turnaround time to eventually increase for extremely small timeouts, which might cause excessive replication. This trend does not occur in our results because timeouts effectively delay work unit replication such that a work unit is usually completed before replication becomes counterproductive. For example, if the average work unit turnaround time is four hours, and a timeout of one hour is used, a work unit will expire and be resent on average three times, which is not enough to lower client productivity. Note that, the turnaround times due to timeouts are higher than the ones resulting from duplication.

4.3. Duplication Versus Timeouts

Duplication can result in better turnaround time but potentially wastes resources, which causes lower throughput. For both independent and dependent work units, the minimum set turnaround time occurs near duplication of 7. However, work unit throughput drops significantly near that duplication amount because many clients compute identical work units.

Timeouts can also improve application turnaround time but not as much as duplication. The minimum set turnaround times due to duplication are considerably lower than the minimum times due to timeouts (compare Figures 3 and 6). Throughput at timeouts of 2700 minutes remains high and very close to the levels when no timeouts are used. To achieve the equivalent set turnaround times via duplication, a duplication of about 2 is needed (see Figures 3 and 6), but the throughput at duplication of 2 is much less than the throughput at 2700 for independent and dependent

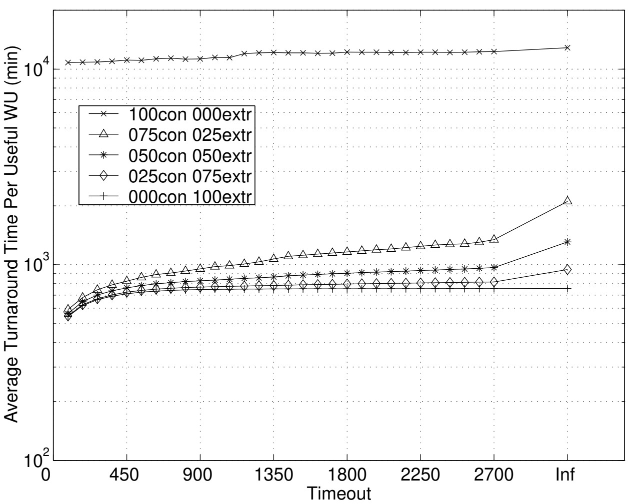

Figure 5. Work unit performance vs. timeout for independent work units.

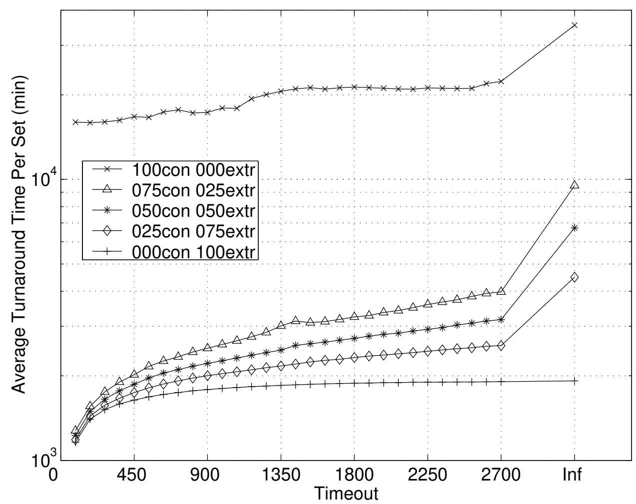

Figure 6. Set performance vs. timeout for independent work units.

work units (see Figures 1 and 4).

In summary, timeouts are a means of compromising by lowering turnaround time without decreasing throughput. Duplication on the other hand allows for more aggressive decreases of turnaround time at the expense of throughput.

4.4. Performance of Different Client Groups

Duplication and timeouts improve application performance the most when clients are heterogeneous. When resources perform similarly, duplication and timeout are less effective because sending identical work units to clients with similar profiles will not improve turnaround times. The 100% extreme client group is relatively homogeneous in the sense that clients are constantly powered on and many have "always-on" network connections. Thus, the performance gains of duplication or timeouts for the 100% extreme group are less then the gains for the 50% conservative and 50% extreme group for instance. Performance gains for the 100% conservative group are significant because client performance is quite variable within this class. That is, the conservative clients are powered on infrequently and many have dial-up connections. Clients that have modems are restricted to a download or upload frequency of once per day, and perform far worse than those with faster connections.

4.5. Conclusions

If an application requires high throughput but has an excess of input in comparison to the clients' aggregate computing power, then the application should use no duplication and long timeouts. For example, SETI@home could use timeouts of 2700 minutes to reduce turnaround time while maintaining high throughput (see Figures 4 and 6). On the other hand, if an application has only a small data set and a mass of clients available, then high duplication and short timeouts could result in low turnaround time.

Most often, though, an application seeks to optimize both throughput and turnaround time of tasks. In this case, the scheduler should measure application performance according to sets of work units because it is a more sensitive metric than work unit turnaround time. Our results show that there is an optimal point at which application performance peaks with respect to set turnaround time. A compute service where users submit multiple jobs for client processing should minimize set turnaround time to satisfy the users' desire for minimum turnaround time and also the service's goal to maximize job throughput. According to Figure 3 a compute service with applications similar to our genomics application should use duplication of about 7 to optimize throughput and turnaround time, independently of the mix of available clients.

5. Related Work

Our work is related to other scheduling efforts in the areas of distributed and Grid computing [15, 16, 17]. However, as was explained in Section 2.2.1, limited job control and characteristics of the platform itself prevents the use of most previously used scheduling strategies. We opted for the investigation of simple scheduling mechanisms that can be readily implemented as part of existing systems.

To the best of our knowledge, our work is one of the first to instantiate a model of the global computing platform. In addition, we analyze scheduling strategies for applications with both dependent and independent tasks. Projects such as SETI@home [1] or Entropia [13] focus on the implementation of a software infrastructure for global computing and only support applications without task dependencies.

6. Future Work

We will extend our scheduling methods to use predicted turnaround times derived from client download and upload history for better client selection for task assignment and duplication. As part of this work we are developing a software prototype in order to deploy the genomics applications of Section 2.1. That software, **BIO@home**, allows for independent or dependent work units, as well as work unit duplication and timeout. We will report on BIO@home in a future paper.

Experts in computer and information technology believe that networks will support higher bandwidths and become more pervasive. As such, global computing and P2P computing are becoming viable and cost-effective for an increasingly broad class of applications. Our work in that context provides initial directions for quantifying and improving the performance of these new applications.

Acknowledgments

We thank the cluster group at the San Diego Supercomputer Center and the Matsuoka laboratory at the Tokyo Institute of Technology for providing computational resources. We acknowledge Matthew Farrellee for his implementation of the BIO@home screensaver, and Shava Smallen for her insightful comments.

References

[1] SETI@home. `http://setiathome.ssl.berkeley.edu`, March 2001.

[2] Andy Oram, editor. Peer-To-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[3] A. Arslan, O. Egecioglu, and P.A. Pevzner. A new approach to sequence comparison: Normalized sequence alignment. Bioinformatics, 17(4):327–337, April 2001.

[4] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. Journal of Molecular Biology, 147:195–197, 1981.

[5] Ian Foster and Carl Kesselman, editors. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.

[6] Mediamark Research Inc. Statistical Abstract of the United States, page 658. Communications and Information Technology: Internet Access and Usage and Online Service Usage. U.S. Census Bureau, 2000.

[7] U.S. Energy Information Administration. Statistical Abstract of the United States, page 726. Construction and Housing: Appliances and Office Equipment Used by Households by Region and Household Income: 1997. U.S. Census Bureau, 2000.

[8] Mitchell-Kernan, Claudia I., editor. Science & engineering indicators. National Science Board, Washington, D.C. USA, 2000.

[9] Baker, S. and Lanctot, R. and Koenig, S. and Wargo, S. Home PC Portrait. PC Data, Inc., Reston, VA USA, 2000.

[10] Nielsen/NetRatings Inc. January 2001 Internet Usage Stats. `http://cyberatlas.internet.com/`, January 2001.

[11] M. Pastore. The Role of the PC: Communication and Leisure. PC Data, `http://cyberatlas.internet.com/`, August 2000.

[12] M. Pastore. Residential High-Speed Access Takes Big Step in 2000. Nielsen/NetRatings, `http://cyberatlas.internet.com/`, February 2001.

[13] Entropia Inc. `http://www.entropia.com`, 2001.

[14] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In Proceedings of SuperComputing 2000 (SC'00), Nov. 2000.

[15] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In Proc. of Supercomputing'96, Pittsburgh, 1996.

[16] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In Proceesings of the 8th International Conference of Distributed Computing Systems, pages 104–111, June 1988.

[17] D. Abramson, J. Giddy, I. Foster, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid ? In Proceedings of the International Parallel and Distributed Processing Symposium, May 2000.