Characterizing and Evaluating Desktop Grids: An Empirical Study

Derrick Kondo[1] Michela Taufer[1]
John Karanicolas[3] Charles L. Brooks[3]
Henri Casanova[1,2] Andrew A. Chien[1]

[1]Dept. of Computer Science and Engineering
[2] San Diego Supercomputer Center
[3] The Scripps Research Institute

University of California, San Diego

Abstract

Desktop resources are attractive for running compute-intensive distributed applications. Several systems that aggregate these resources in *desktop grids* have been developed. While these systems have been successfully used for a wide variety of high throughput applications there has been little insight into the detailed temporal structure of CPU availability of desktop grid resources. Yet, this structure is critical to characterize the utility of desktop grid platforms for both task parallel and even data parallel applications.

We address the following questions: (i) What are the temporal characteristics of desktop CPU availability in an enterprise setting? (ii) How do these characteristics affect the utility of desktop grids? (iii) Based on these characteristics, can we construct a model of server "equivalents" for the desktop grids, which can be used to predict application performance? We present measurements of an enterprise desktop grid with over 220 hosts running the Entropia commercial desktop grid software. We utilize these measurements to characterize CPU availability and develop a performance model for desktop grid applications for various task granularities, showing that there is an optimal task size. We then introduce a new metric, cluster equivalence, which we use to quantify the utility of the desktop grid relative to that of a dedicated cluster.

1 Introduction

Cycle stealing systems that harness the idle cycles of desktop PCs and workstations date back to the PARC Worm [20] and have known a widespread success with popular projects such as SETI@home [31, 28], the GIMPS [19], Folding@Home [29], FightAidsAtHome [17], Computing

Against Cancer [9], and others sustaining the throughput of over one million CPU's and 10's of Teraflops/seconds [23]. These successes have inspired similar efforts in the enterprise as a way to maximize return on investment for desktop resources by harnessing their cycles to service large computations. As a result, numerous academic projects have explored developing a global computing infrastructure for the internet [24, 26, 1, 10, 8, 4, 16] and local-area networks [21, 3, 18]. In addition, commercial products have also emerged [15, 33, 32, 25]. We term such computing infrastructures *desktop grids*, and while these systems can be used in a variety of environments, here we focus on the enterprise setting.

While the motivation for using desktop resources is clear, namely the opportunity to perform large computations at low-cost, the main challenge is that the resources are *volatile*. Desktop grid systems have been deployed successfully for a wide variety of high throughput applications, and numerous studies of total available CPU power have been done. But there have been no study of real platform utility because the interactions of resource dynamic behaviors and application structure are complex. In fact, little insight has been gained into the detailed temporal structure of CPU availability, even in the enterprise. Measuring and characterizing this temporal structure is key for quantifying the utility of desktop grids for both task parallel and even data parallel applications. In this paper, we characterize this temporal structure and analyze its impact on desktop grid applications.

Several other studies have measured the percentage of available CPU cycles for large collections of desktop machines [2, 11, 30, 36]. The results of these studies have the following shortcomings for characterizing the utility of desktop grid resources. First, the monitored CPU availability may not be a good model of what an application might receive due to O/S idiosyncrasies, as seen in a number of studies [14, 35]. Second, these studies do not account for keyboard and/or mouse activity which for many systems causes a desktop grid application to be suspended. Third, these studies do not account for possible overhead, limitations, and policies of accessing the underlying resource via a desktop grid infrastructure. Fourth, it is not clear that these results provide enough information to understand the task failure behavior for an application running on the desktop grid. While these last two points could conceivably be added synthetically, doing so in a way that reflects actual desktop grid behavior is not straightforward. By contrast, we conduct measurements directly via a desktop grid infrastructure, allowing us to experience the platform exactly as it would be experienced by an application.

The core focus of our measurement and analysis is to address the following three questions: (i) What are the dynamic characteristics of desktop CPU availability in an enterprise setting? (ii) How do these characteristics affect the utility of desktop grids? (iii) Based on these characteristics, can we construct a model of server "equivalents" for the desktop grids, which can be used to predict application performance?

To answer these questions, we made extensive measurements of an enterprise desktop grid with over 220 hosts running the Entropia commercial desktop grid software. We utilize these measurements to construct a statistical characterization of the temporal structure of CPU availability. Based on this characterization we construct a performance model for the desktop grid for various task granularities. We find that there is an optimal task size, which we compute. We then introduce a new metric, cluster equivalence, which is used to quantify the utility of the desktop grid relative to that of a dedicated cluster. In essence, this metric quantifies the performance penalty for the resource volatility which characterizes desktops, and enables for the first time, the direct comparison of desktop grid and dedicated server performance.

The rest of the paper is organized as follows. Section 2 motivates and describes our measurement methodology. Section 3 describes and analyzes our measurements. Section 4 develops a model for task completion rates and defines the cluster equivalence metrics. Section 5 discusses related work. Section 6 concludes the paper with a summary of results and a perspective on future work.

2 Background and Methodology

2.1 Goal

Our goal is to quantitatively characterize the utility of a desktop grid resources for compute-intensive, task-parallel applications with a variety of task sizes. Such a characterization depends in turn on a detailed characterization of the temporal structure of CPU availability. More specifically, our goal is to obtain the following two characterizations: (i) a broad characterization of the desktop grid in terms of how often cycles can be exploited for a desktop grid application, and how these cycles are distributed in the spectrum of available hosts; (ii) a characterization of host availability patterns, that is the distribution of the intervals of time during which a host can be used by a desktop grid application, and how much of the host's CPU is actually exploitable during these intervals. Based on these characterizations, it is possible to reason about the utility of these desktop resources for a spectrum of application configurations, obtain performance models, and define a good utility metric.

A first challenge is to accurately measure the relevant CPU availability information that is needed for obtaining these characterizations. The next two sections presents our measurement methodology and the desktop grid on which we conduct measurements.

2.2 Measurement Procedure

We conduct our measurements via the commercial desktop grid software infrastructure, Entropia [15], by submitting actual tasks to the system. These tasks perform computation and periodically report their computation rates. This method requires that no other desktop grid application be running, allows us to measure exactly what actual compute power a real, compute-bound application would be able to exploit. It differs from a range of other measurement techniques in that it is intrusive and actually consumes the CPU cycles as an application would. An interesting feature of the Entropia system is its use of a Virtual Machine (VM) to insulate application tasks from the resources. While this VM technology is critical for security and protection issues, it also makes it possible for an application task to use only a fraction of a CPU. The design principle is that an application should use as much of the CPU cycles as possible while not interfering with local processes. This allows an application to use from 0% to 100% of the CPUs with all possible values in between. It is this CPU availability that we measure. Note that our measurements could easily be post-processed to evaluate a desktop grid system that only allows application tasks to run on host with, say, more than 90% available CPU.

During each measurement period, we keep the Entropia system fully loaded with requests for our CPU-bound tasks. Each task of fixed time length consist of an infinite loop that performs a mix of integer and floating point operations. A dedicated 1.5GHz Pentium processor can perform 37.5 million such operations per second. Every 10 seconds, a task evaluates how much work it has been able to achieve in the last 10 seconds, and writes this measurement to a file. These files are retrieved by the Entropia system and are then assembled to construct time series of CPU availability in terms of the number of operations that were available to the desktop grid application within every 10 second interval. We will see in Section 2.4 and Section 3.1 that these time series have gaps that are in themselves useful to characterize the performance of the desktop grid.

2.3 Desktop Grid Resources

We used a deployment of the Entropia DCGrid™at the San Diego Supercomputer Center (SDSC) that is installed on over 275 hosts. The hosts are all on the same class C network, with most clients having a 10MB/sec connection and a few having a 100MB/sec connection. All hosts are desktop resources that run different flavors of Windows™. Of the 275 hosts, 30 are used by secretaries, 20 are public hosts that are available in SDSC's conference rooms, 12 are used by system administrators, and the remaining are used by SDSC staff scientists and researchers. The Entropia server was running on a dual-processor XEON 500MHz machine with 1GB of RAM.

During our experiments, about 220 of the 275 hosts were effectively running the Entropia client and we obtained measurements for these hosts. Their clock rates ranged from 179MHz up to 3.0GHz, with an average of 1.19GHz. Figure 1 plots the clock rate values for hosts sorted by increasing clock rates. The curve is not continuous as for instance no host has a clock rate between 1GHz and 1.45GHz, and over 30% of the hosts have clock rates between 797MHz and 863MHz, which represents under 3.5% of the clock rate range.

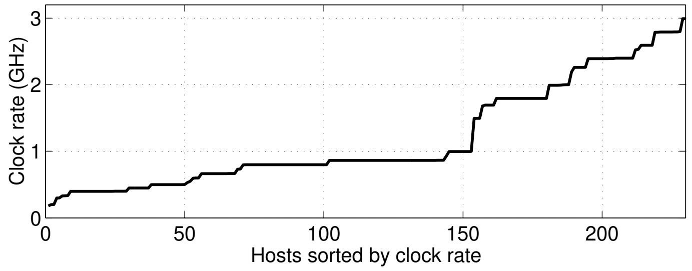


Figure 1: Host clock rate values (GHz) for all hosts sorted by increasing clock rate values.

We conducted measurements during four distinct time periods: from 8/18/03 until 8/22/03, from 9/3/03 until 9/17/03, from 9/23/03 to 9/26/03, and from 10/3/03 and 10/6/03 for a total of approximatively 28 days of measurements.

2.4 Measurement Gaps

The traces that we obtained from our measurements contain gaps. This is expected as desktop resources become unavailable for a variety of reasons: a host is being rebooted or experiences downtime, local processes use 100% of the CPU, the Entropia client detects mouse or keyboard activity, a user actively pauses the Entropia client, etc. However, we observe that a very large fraction (95%) of these gaps are clustered in the 2 minute range. Figure 2 plots the distribution of these small gaps. The average small gap length is 35.9 seconds.

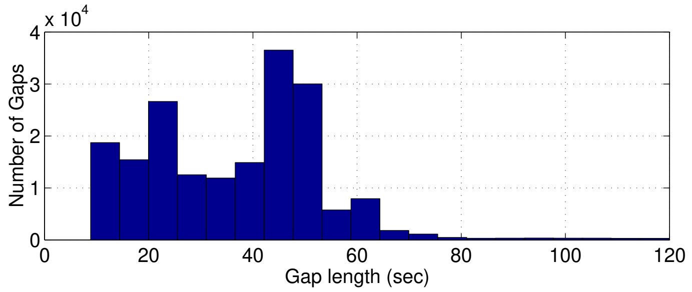


Figure 2: Length Distribution of “small” gaps (<2 min).

After careful examination of our traces, we found that these short gaps occur exclusively in between the termination of a task and the beginning of a new task on the same host. We thus conclude that these small gaps do not correspond to actual host unavailability, but rather are due to the delay of the Entropia system for starting a new task. This delay is itself due to several overhead sources, as well as an actual built-in limitation that prevents the system from sending tasks to resources too quickly (e.g., to avoid large number of failures in case of application misconfigurations). Therefore, these small availability gaps would not be experienced by the tasks of a real application, but only in between tasks. Consequently, we eliminated all gaps that were under 2 minutes in our traces by performing linear interpolation. A small portion of the gaps larger than 2 minutes may be also attributed to Entropia server delay and thus means that our post-processed traces may be slightly pessimistic. Note that although we use interpolation, we use the average small gap length in our performance models to account for the Entropia server delay (see Section 4). Note that these small gaps are not due to keyboard/mouse activity on the hosts as the Entropia client is suspended for 5 minutes when these happen. Note also that, for a real application, the gaps may be larger due to transfer of input data files necessary for task execution. Such transfer cost could be added to our average small gap length and thus easily included in the performance model developed in Section 4.

3 Platform Measurements

Over our 28 days of measurements, we have obtained traces for over 220 hosts, for a total of 32,435,932 individual CPU availability measurements. The total compute power that was available to our tasks were 12.8 trillions of operations, which is equivalent to 11.2 years of CPU time on a

1.5GHz Pentium processor. The gaps in our traces that correspond to CPU unavailability represent 16% of the total timespan of the measurement period for all hosts. In what follows, we define the term availability precisely and present our characterization of the temporal structure of CPU availability.

3.1 Defining Availability

The term "availability" has different meanings in different contexts and must be clearly defined for the problem at hand [5]. In this work we consider two kinds of unavailability: (i) **host availability**: a binary value that indicates whether a host is reachable and the desktop grid client is up, which corresponds to the definition of availability in [6, 5, 13, 27]; and (ii) **CPU availability**: a percentage value that quantifies the fraction of the CPU that can be exploited by a desktop grid application, which corresponds to the definition in [2, 11, 30, 14, 35]. When a host becomes unavailable (e.g., during a shutdown of the O/S), no new task can be started, and if a desktop grid application task was running on that host it fails. When a CPU becomes unavailable (that is with $<$1% CPU availability) but its host is still available (e.g., when the CPU is used more than 99% by local processes, there is keyboard/mouse activity from the resource owner), then a running desktop grid application task is suspended and can be resumed when the CPU becomes available again (that is with $>$1% CPU availability). Note that host unavailability implies CPU unavailability

We identify these two kinds of unavailability in our measurements as follows. Host unavailability corresponds to measurement gaps in our host traces that occur between executions of our measurement tasks. In other words, either no measurement task was running on the host, or if one was running it failed and never produced trace data. For simplification we assume that the whole gap corresponds to host unavailability, which may be pessimistic given that the task may have failed only at the end of the gap (note however that our measurement tasks are short). CPU unavailability corresponds to gaps in our traces that occur between the beginning and the completion of a measurement task. In this case, the lack of measurement means that the task was suspended but eventually was resumed and completed.

An estimate of the computational power (i.e. number of cycles) that can be delivered to a desktop grid application is given by an aggregate measure of CPU availability. For each data point in our measurements (over all hosts), we computed how often CPU availability is above a given threshold. We present results separately for weekdays and weekends, which we will do throughout this paper, as the desktop grid exhibit significant behavior during and outside of the work week. Figure 3 plots the frequency of CPU availability being over a threshold for threshold values from from 0% to 100%: the data point (x, y) means that $y\%$ of the time CPU availability is over $x\%$. For instance, the graph shows that CPU availability is over 90% about 55% of the time during weekdays, and 70% of the time during weekends. On average CPU are completely unavailable 19% of the time during weekdays and 3% of the cases during weekends. We also note that both curves are relatively flat for CPU availability between 1% and 80%, denoting that hosts rarely exhibit availabilities in that range.

Other studies have obtained similar data about aggregate CPU availability in desktop grids [12, 22]. While such characterizations make it possible to obtain coarse estimates of the power of the desktop grid, it is difficult to related them directly to what a desktop grid application can

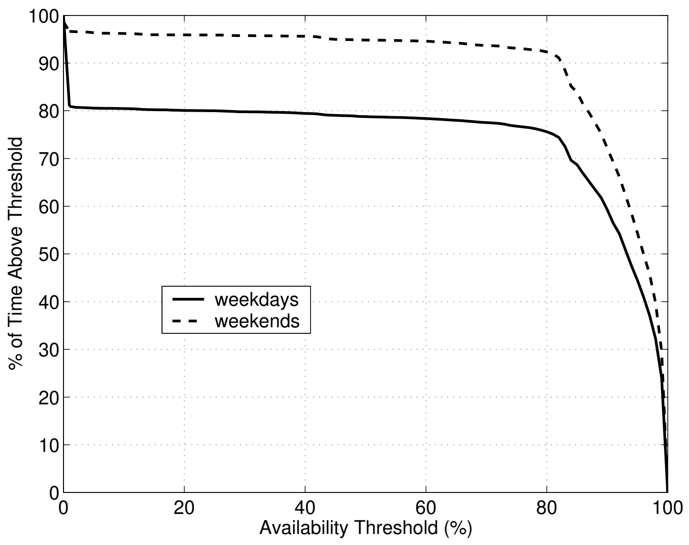


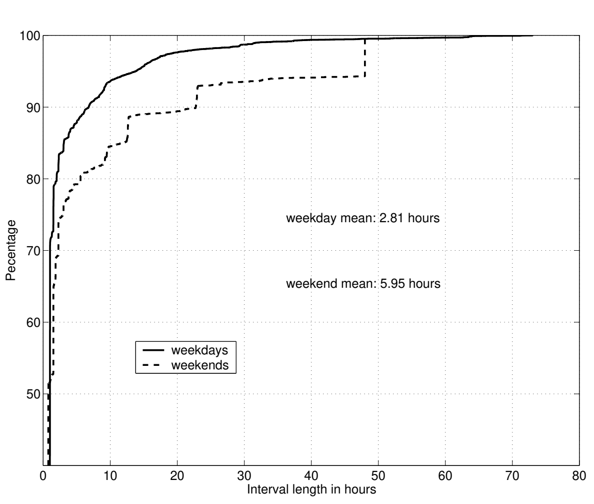
Figure 3: Percentage of times when CPU availability, in percentage, is above a given threshold, over all hosts, for weekdays and weekends.

hope to achieve. In particular, the understanding of host availability patterns, that is the statistical properties of the duration of time intervals during which an application can use a host, and a characterization of how much power that host delivers during these time intervals, are key to obtaining quantitative measures of the utility of a platform to an applications. We develop such characterization in the next section.

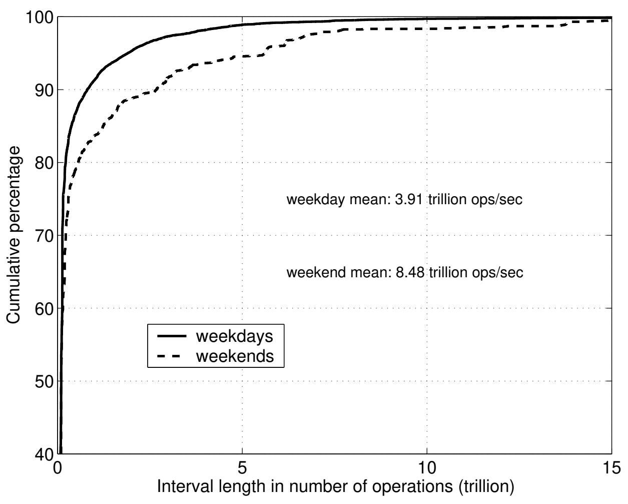
3.2 Host Availability Patterns

Figure 4(a) plots the cumulative distribution of the duration of **host availability** intervals, both for weekdays and weekends. We see that intervals are significantly longer during weekends, with an average of close to 6 hours, versus under 3 hours during weekdays. Further, for the weekends, several intuitive features in the availability intervals distributions are clearly visible (e.g. 24 hours, 48 hours, etc.). While this data is interesting for applications that require hosts to be reachable for given period of time (i.e. content distribution) and could be used to confirm and extend some of the work in [6, 5, 13, 27], it is less relevant to our problem. Indeed, from the perspective of a compute-intensive application, a 1GHz host that is available for 2 hours with average 80% CPU availability is less attractive than, say, a 2GHz host that is available for 1 hour with average 100% CPU availability.

By contrast, Figure 4(b) plots the cumulative distribution of the host availability intervals, both for weekdays and weekends, but in terms of the number of operations performed. In other words, these are intervals during which a desktop grid application task may be suspended, but does not fail. And instead of showing availability interval durations, the x-axis shows the number of operations that can be performed during the interval, which is computed using our measured CPU availability. This quantifies directly the performance that an application, factoring in heterogeneity



(a) Interval lengths in number of hours.



(b) Interval length in number of operations.

Figure 4: Cumulative distribution of the "length" of host availability intervals in between task failures, for weekdays and weekends.

in hosts. Other major trends in the data are as expected with hosts and CPUs are more frequently available during weekends than weekdays. The empirical data enables us to quantify task failure rates and develop a performance model (see in Section 4).

4 Performance Models and Utility Metrics

4.1 Expected Task Failure and Work Rates

Based on our characterization of the temporal structure of resource availability, it is possible to derive the expected task failure rate, that is the probability that a host will become unavailable before a task completes. This expectation is strongly dependent on the task lengths, as shown in Figure 5. The expected task failure rate on this figure are computed from the distribution of number of operations performed in between failures (from the data shown in Figure 4(b)) based on random incidence, and is shown for weekdays and for weekends. For illustration purposes, the x axis shows task sizes not in number of operations, but in execution time on a dedicated 1.5GHz host, from 15 minutes up to 12 hours. The two dominant features are first that for a given task size the failure rate on weekends is lower than on weekdays, and second that the failure rate increases with task size.

Based on the data we have presented, we can now propose a model for an application's expected work rate (that is the number of useful operations performed by time units), $W(s)$, given a uniform task size s (number of operations per task), as follows.

From our measurements we know that there is an average overhead, g, characterized to be 36.9

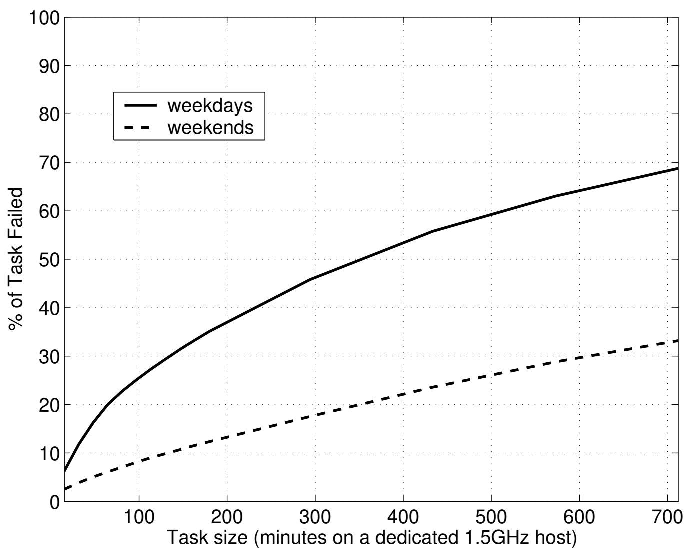


Figure 5: Task failure rate versus task size (in number of minutes of dedicated CPU time on a 1.5GHz host).

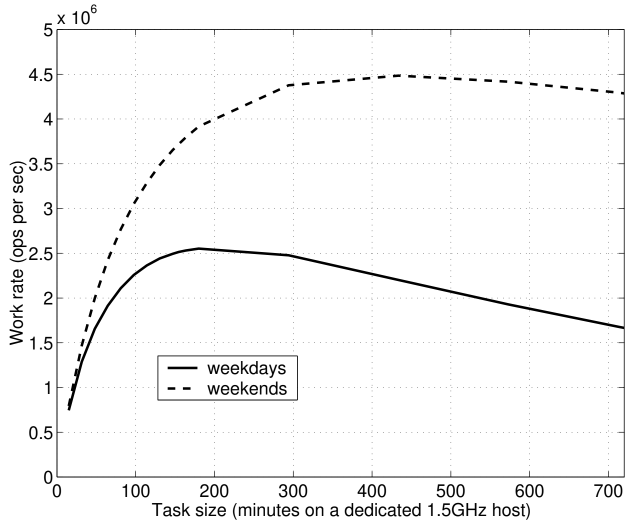


Figure 6: Model of application work rate for entire desktop grid, in number of operations per seconds versus task size,in number of minutes of dedicated CPU time on a 1.5GHz host.

seconds in between each task on the same resource due to the Entropia server (see Section 2.2). Using the data from Figure 5, we can compute the task failure rate, $f(s)$, as a function of the task size. We can also estimate the average compute rate in operations per second for a host in the desktop grid, r, by computing the average delivered operation per second host availabilities from our traces, and taking an average over all host. $W(s)$ in number of operations per seconds for an application using N hosts in the desktop grid is then computed as:

$$W(s) = N \times \frac{r(1 - f(s))}{1 + \frac{r}{s}g}, \tag{1}$$

where $\frac{r}{s}$ is the number of tasks that would be completed by time unit if there were no failures. Note that one could fit the data shown in Figure 5 to an analytical model to obtain a closed-form expression for $W(s)$.

Figure 6 plots $W(s)$ rate for the same task sizes as in Figure 5, for both weekdays and weekends. For weekdays, for task sizes below 200 minutes per-host progress increases rapidly as the task size compensates for the fixed overhead. However, as task size increases further, the per-host progress decreases as the penalty of additional task failures wastes some of the CPU cycles. This trend is also exhibited for weekends, but the longer availability intervals enable compute rates to improve up to task sizes around 450 minutes. Thus, for both weekdays and weekends, the trade-off between overhead and failures produces an optimal task size, which is 250 and 450 minutes respectively. Note that these are the number of minutes that a task execution would require on a dedicated 1.5GHz host, so the effective execution times experienced on the SDSC Entropia grid range from approximately five times longer to two times shorter.

4.2 Cluster Equivalence

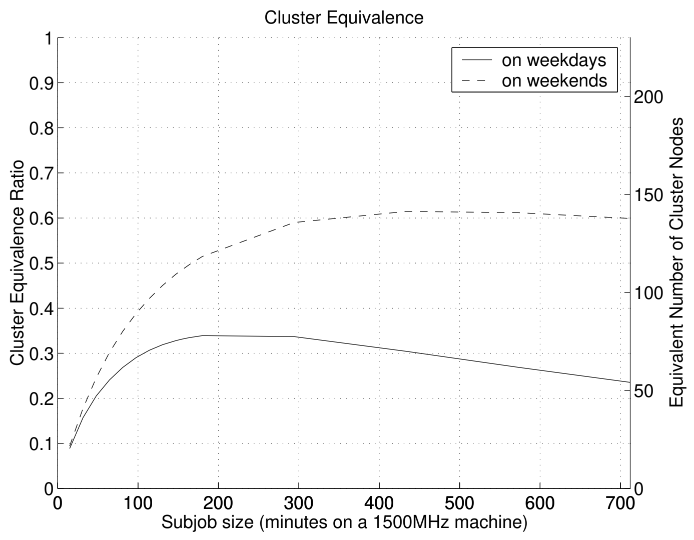
To characterize the impact of resource volatility in a desktop grid on usable performance, we have derived a new utility metric called cluster equivalence. That is, for a given desktop environment (and corresponding temporal CPU availability), what fraction of a dedicated cluster CPU is each desktop CPU worth to an application? With this information, we can establish for a desktop grid the size of a dedicated cluster to which its performance is equivalent. Because the objective is to quantify the performance impact of resource volatility, we normalize assuming that the CPU clock rate of each cluster is equal to the mean CPU clock rate in the desktop grid [1].

More precisely: "Given an N-host desktop grid, how many nodes of dedicated cluster, M, with comparable CPU clock rates, are required such that the two platforms have equal utility?" We define M/N as the *cluster equivalence* ratio of the desktop grid.

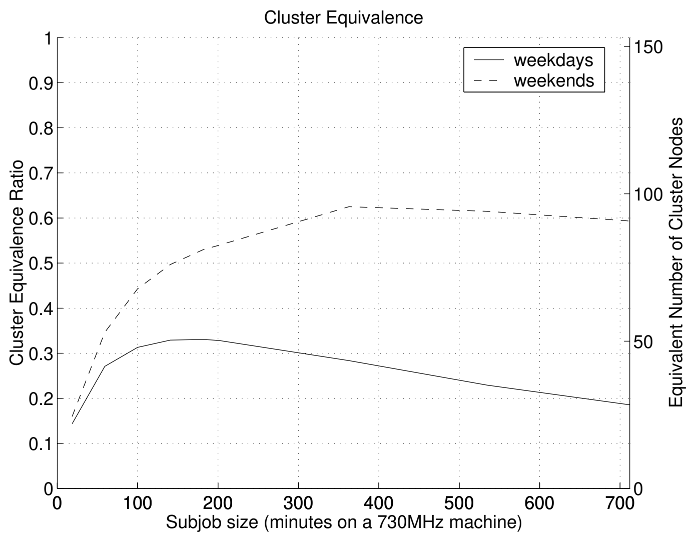
It is clear from our desktop grid measurements that the cluster equivalence ratio depends on the application's structure and characteristics. Here we consider only task parallel applications with various task sizes as in Section 4. The higher the task size the lower the cluster equivalence ratio since the application becomes more subject to failures (see Figure 6).

We compute the cluster equivalence for a range of application task sizes, as shown in Figure 7(a). Thus curves are essentially scaled versions of those in Figure 6. The data points on this

[1]Numerous industrial interactions by one of the authors suggest that this is true in many companies.



(a) Whole desktop grid



(b) Desktop grid as of 2001

Figure 7: Cluster equivalence of a desktop grid CPU as a function of the application task size. Two lines are shown, one for the the resources on weekdays and weekends.

graph can be used to determine the effective cluster CPU's that the SDSC desktop grid delivers. For example, for a 0.256 trillion operation task (approximately two hours on a 1.5 GHz CPU), the performance of the 220-node SDSC Entropia desktop grid is equivalent to a 97-node cluster on weekends, and to a 70-node cluster on weekdays. For longer tasks with 0.673 trillion operations (approximately five hours on a 1.5GHz CPU), the 220-node SDSC Entropia desktop grid is equivalent to a 136-node cluster on weekends, and to a 78-node cluster on weekdays.

For comparison, Figure 7(b) shows the cluster equivalence metric computed for a subset of the desktop grid that excludes the most recent machines (in this case machines with a clock rate higher than 1GHz, which corresponds to 153 hosts). The mean clock of this subset of hosts was approximately 730MHz. We observe that the trends are similar to that seen in Figure 7(a). In fact, the average relative difference between the cluster equivalence ratios for the entire desktop grid and the subset, over all task sizes, is approximately 10%.

The fact that our cluster equivalence metric is relatively consistent for different subsets of the desktop grid is explained by Figure 8. This figure plots the cumulative percentage of operations delivered by a subset of the entire platform, corresponding to an increasing percentage of the sorted hosts. In other words, data point (x, y) on the graph means that $x\%$ of the hosts (taking the most "useful" hosts first) deliver $y\%$ of the compute operations of the the entire platform. Hosts are sorted either by number of delivered operations per seconds (as computed from our measurements) or by clock rate, as seen in the two curved in Figure 8. We can see that the two curves are strikingly similar. This indicates that the average availability patterns of the hosts in our platform over our measurement period are uncorrelated with host clock rates. This in turn explains why our cluster equivalence metric is consistent for the whole platform and a subset containing

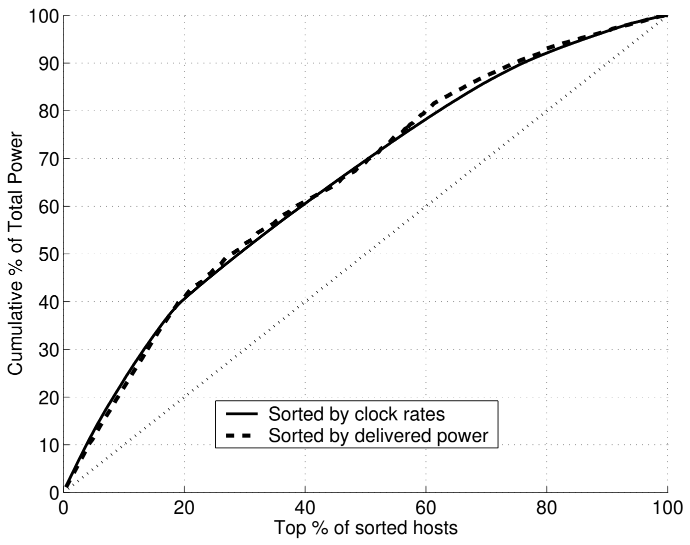


Figure 8: Cumulative percentage of total platform computational power for hosts sorted by decreasing effectively delivered computational power and for hosts sorted by decreasing clock rates.

only older machines.

Interestingly, we also find that the curves in Figure 8 while not linear, are only moderately skewed (as compared to the dotted line in the figure). For instance, the 30% most useful hosts deliver 50% of the overall compute power. Similarly the 30% least useful hosts deliver approximately 14% of the overall compute power. Note that this skew is not as high as to justify using only a small fraction of the resources.

5 Related Work

Several efforts have been underway to advance the state-of-the-art of computing on desktop grids are are thus broadly related to this work. From a practical standpoint, several commercial projects platform [15, 33, 32, 25] have delivered full-fledge desktop grid infrastructures that can be deployed securely, precisely managed, and professionally supported within enterprises. In terms of enhancing the range of application structures that can run on desktop grids, the MPICH-V project [7] provides a fault-tolerant MPI programming model for volatile resources that are part of the XtremWeb [16] desktop grid infrastructure. From the application perspective, works such as the one in [34] have demonstrated that applications can be tuned at the algorithmic level to be more susceptible to benefit from desktop grids. Our work could have impact for all the above projects as it provides characterization of the temporal structure of host and CPU availability in desktop grid, which is fundamental for advanced resource management, resource selection, and application scheduling.

The works in [6, 5, 13, 27] have presented measurement and analysis of host availability in enterprise systems and in large peer-to-peer networks (where host availability is defined as the host being reachable). While these results were meaningful for the considered application domain (i.e., distributed storage), their relevance for our purpose is dubious. Indeed, it is not clear how to relate up-times to actual CPU cycles that could be effectively exploited by a compute-intensive desktop grid applications. Therefore, while our results contain host availability data (see for instance Figure 4(b)), our main focus is on quantifying and characterizing CPU availability during periods of host availability.

A few other studies have obtained percentages of CPU cycles available for large collections of machines as part of research projects [2, 11, 30], or have made available such measurement data through a monitoring infrastructure [36]. While the results from these studies could be used to quantify the utility of a desktop grid for a compute-intensive application, we have highlighted their shortcomings in Section 1. Essentially, it is difficult to related the values obtained by lightweight CPU availability sensing to the CPU availability that would be experienced by a desktop grid application. Our work differs in that, rather than using lightweight sensing, we conducted intrusive measurements to experience the desktop grid exactly as a real application would.

A few authors have measured the performance of desktop grids for specific applications. The work in [12] studies the performance of and Entropia deployment for four different applications with different granularities and compares some of the results to runs performed on a dedicated cluster. In this work we go further by conducting measurements of effective host and CPU availability, proposing a model for the performance of the desktop grid, and defining a cluster equivalence metric to quantify the platform utility. It would be interesting to apply our results to those obtained in [12]. Similarly, work in [22] reports on several applications that were deployed on XtremWeb desktop grids [16]. Here also our work goes further as we make a fundamental connection between host and CPU availability characterizations on expected application performance.

6 Conclusion

Desktop grids are attractive platforms for running compute-bound distributed applications as they can provide low-cost, otherwise unused CPU cycles, and make it possible for enterprises to maximize return-on-investment for desktop resources. However, due to the inherent volatility of these resources, usage scenarios applications have been mostly confined to the execution of a single long-lived, embarrassingly parallel application, with the goal of achieving high throughput. More advanced usage, namely strategies for running such application more efficiently, and the ability to deploy more diverse applications (e.g. short-lived applications with a number of tasks significantly lower than the number of available resources, applications that require some task synchronization, multiple concurrent applications), is impeded by: (i) the lack of understanding of resource availability characteristics; (ii) the lack of quantitative models and metrics of the utility of both individual resources and of the entire platform for different applications. In this paper we address these two concerns by conducting measurements of CPU availability and by running a real-world application on a production Entropia [15] desktop grid.

We obtained detailed characterizations of the temporal structure of CPU availability, with which we developed a model for the task completion rate of embarrassingly parallel applications for various task sizes. We developed a general "cluster equivalence" metric that quantifies the utility of a desktop grid by comparing its utility to that of a dedicated cluster. This metric can be computed for any desktop grid, and we found it to be consistent over subsets of our desktop grid. We found that host volatility reduced the power of a 220-host desktop grid to be equivalent to an 136-hosts cluster, and to a 78-host cluster during weekdays, where cluster hosts have a clock rate equal to the mean clock rate of the desktop grid hosts.

In future work we will we will perform measurements and compute the cluster equivalence of other platforms with other software than Entropia. In particular, we are planning to work with an XtremWeb [16] deployment. We are currently engaged in running real-world applications from the domain of protein folding on the SDSC Entropia desktop grid. In an upcoming paper we will report on results from these experiments and how they related to the characterizations and models presented in this paper.

Our broader, longer-term goal, is to exploit the availability measurements and characterizations obtained in this paper to study the deployment of different applications on desktop grid. In particular, we believe that characterizing the correlation of host availability is key for deploying short-lived parallel applications with a number of tasks significantly lower than the number of available hosts, or applications that require task synchronization. The key to enabling such applications will be intelligent resource management and application scheduling strategies (e.g. resource selection, task duplication) that take into account availability and availability correlation. Already, works such as MPICH-V [7] provide mechanisms for advanced applications on desktop grids. Our work will provide the foundations for making intelligent scheduling decisions for MPICH-V applications.

Acknowledgements

The authors wish to thank Entropia, Inc. and the San Diego Supercomputer Center for providing the testbed has made this work possible. We also wish to thank Ricky Chang, Shawn Marlin, Matt Clark and Matt Campbell, whose help with using and understanding the testbed has been invaluable.

References

[1] A. D. Alexandrov, M. Ibel, K. E. Schauser, and C.J. Scheiman. SuperWeb: Towards a Global Web-Based Parallel Computing Infrastructure. In *Proc. of the 11th IEEE International Parallel Processing Symposium (IPPS)*, April 1997.

[2] R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson, and D.A." Patterson. The Interaction of Parallel and Sequential Workloads on a Network of Workstations. In *Proceedings of SIGMETRICS'95*, pages 267–278, May 1995.

[3] A. Barak, S. Guday, and Wheeler R. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, volume 672 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[4] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the Web. In *Proc. of the 9th International Conference on Parallel and Distributed Computing Systems (PDCS-96)*, 1996.

[5] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *In Proceedings of IPTPS'03*, 2003.

[6] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed file System Deployed on an Existing Set of Desktop PCs. In *Proceedings of SIGMETRICS*, 2000.

[7] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes. In *Proceedings of SC'02*, 2002.

[8] N. Camiel, S. London, N. Nisan, and O. Regev. The PopCorn Project: Distributed Computation over the Internet in Java. In *Proc. of the 6th International World Wide Web Conference*, April 1997.

[9] The Compute Against Cancer project. `http://www.computeagainstcancer.org/`.

[10] P. Cappello, B. Christiansen, M. Ionescu, M. Neary, K. Schauser, and D. Wu. Javelin: Internet-Based Parallel Computing Using Java. In *Proceedings of the Sixth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1997.

[11] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.

[12] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel and Distributed Computing*, 63:597–610, 2003.

[13] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, July 2003.

[14] P. Dinda. The Statistical Properties of Host Load. *Scientific Programming*, 7(3–4), 1999.

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

[16] G. Fedak, C. Germain, V. N'eri, and F. Cappello. XtremWeb: A Generic Global Computing System. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'01)*, May 2001.

[17] The Fight Aids At Home project. `http://www.fightaidsathome.org/`.

[18] D. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat, and T. Anderson. GLUnix: a Global Layer Unix for a Network of Workstations. *Software-Practice and Experience*, 28(9), July 1998.

[19] The great internet mersene prime search (gimps). `http://www.mersenne.org/`.

[20] S.A. Hupp. The "Worm" Programs – Early Experience with Distributed Computation. *Communications of the ACM*, 3(25), 1982.

[21] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, 1988.

[22] O. Lodygensky, G. Fedak, V. Néri, A. Cordier, and F. Cappello. Auger & XtremWeb : Monte Carlo computation on a global computing platform. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP2003)*, March 2003.

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

[24] J. Pedroso, L.M. Silva, and J.G. Silva. Web-based metacomputing with JET. In *Proc. of the ACM PPoPP Workshop on Java for Science and Engineering Computation*, June 1997.

[25] Platform Computing Inc. `http://www.platform.com/`.

[26] L. Sarmenta and S. Hirano. Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java. *Future Generation Computer Systems*, 15(5-6):675–686, 1999.

[27] S. Saroiu, P.K. Gummadi, and S.D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedinsg of MMCN*, January 2002.

[28] The seti@home project. `http://setiathome.ssl.berkeley.edu/`.

[29] M.R. Shirts and V.S. Pande. Screen Savers of the World, Unite! *Science*, 290:1903–1904, 2000.

[30] S. Smallen, H. Casanova, and F. Berman. Tunable On-line Parallel Tomography. In *Proceedings of SuperComputing'01, Denver, Colorado*, Nov. 2001.

[31] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, G. Gedye, and D. Anderson. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proc. of the Fifth Intl. Conf. on Bioastronomy*, 1997.

[32] DataSynapse Inc. `http://www.datasynapse.com/`.

[33] United Devices Inc. `http://www.ud.com/`.

[34] B. Uk, M. Taufer, T. Stricker, G. Settanni, A. Cavalli, and A. Caflisch. Combining Task- and Data Parallelism to Speed up Protein Folding on a Desktop Grid Platform - Is efficient protein folding possible with CHARMM on the United Devices MetaProcessor? In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.

[35] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU Availability of Time-shared Unix Systems. In *Peoceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8)*, August 1999.

[36] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5–6):757–768, 1999.