WaveGrid: a Scalable Fast-turnaround Heterogeneous Peer-based Desktop Grid System [*]

Dayi Zhou and Virginia Lo
University of Oregon
dayizhou|lo@cs.uoregon.edu

Abstract

We propose a novel heterogeneous scalable desktop grid system, WaveGrid, which uses a peer-to-peer architecture and can satisfy the needs of applications with fast-turnaround requirements. In WaveGrid, hosts self-organize into a timezone-aware overlay network, which supports straightforward, quick resource discovery. Scheduling methods in WaveGrid take heterogeneity into account in selecting scheduling and migration targets. WaveGrid then rides the wave of available cycles by migrating jobs to hosts located in idle night-time zones around the globe. We evaluate WaveGrid using a heterogeneous host CPU power profile based on empirical data collected from the global computing project BOINC. The simulation results show that WaveGrid perform consistently well with fast turnaround time and low migration overhead. It performs much better than other systems with respect to turnaround, stability and minimal impacts on hosts.

1 Introduction

A *peer-based desktop grid system* [2, 4, 15, 9, 8] allows cycle donors with similar interests to self-organize into a cycle sharing community by joining an *overlay network*, similar to those used by peer-to-peer file sharing systems. In contrast to the institutional-based Grid systems [7, 20], load sharing systems in local networks [13, 23], and client-server-based global computing projects [1, 6, 3], peer-based desktop grid systems circumvent the performance bottleneck of central servers and are capable of achieving much larger scale. Moreover, a peer-based desktop grid system allows each peer to be a potential donor of idle cycles as well as a potential source of tasks for automatic scheduling in the virtual resource pool. The peer-based model is distinctly different from the client-server-based global computing projects such as BOINC, SETI@home and Stanford Folding. In these systems all the volunteer machines donate idle cycles to a single scientific application but reap no benefits for their own computational needs.

A key challenge in the design of peer-based desktop grid systems is scheduling to meet the requirement of fast turnaround. It is difficult to collect accurate information about host availability in a large scale and dynamic environment. The resources in a peer-based desktop grid are volatile: peers may join and leave, and hosts may withdraw their resources at any time. In order to respect autonomy of the hosts, local jobs should have higher priority than foreign jobs. As a result, the foreign job will make slower progress since it can only access a fraction of the host's CPU availability. Desktop grids are highly heterogeneous, as hosts have different CPU clock rates, different memory sizes and different operating systems. Finally, each node is an autonomous system, so scheduling in peer-based desktop grid systems must be non-intrusive. Scheduling methods relying on heavy performance monitoring are inappropriate as users, especially home machine cycle donors, will find it intrusive to report their CPU usage periodically to some remote clients.

Our solution to the problem, *WaveGrid*, is motivated by the natural distribution pattern of idle cycles. Most users have daily routines with few idle day-time cycles and large chunks of idle night-time cycles. In addition, hosts are geographically distributed in different timezones on the Internet. During the 24 hour daily cycle, the area which contains the most idle hosts changes over time.

Self-organized timezone-aware overlay network. WaveGrid allows hosts to organize themselves by timezone to indicate when they have large blocks of idle time. WaveGrid uses a timezone-aware overlay network built on a structured overlay network such as CAN [16], Chord [19] or Pastry [17].

The timezone-aware overlay network provides a

[*]Research supported in part by NSF ANI 9977524.

mechanism for clients to find idle hosts without the high overhead of traditional blind-search-based resource discovery strategies [10, 22]. It takes constant time for the scheduler to choose the targeted area to search for available hosts based on the node-label, then a limited scope expanding ring search is conducted in that area to discover a group of candidates.

Efficient scheduling and migration. Under WaveGrid, a client initially schedules its job on a host in the current night-time zone. When the host machine is no longer idle, the job is migrated to a new night-time zone. Thus, jobs ride a wave of idle cycles around the world to reduce turnaround time. When migrating, WaveGrid selects the host with the highest performance potential. In this study, we focus on CPU speed, but our model is easily generalized to other criteria. WaveGrid is the first desktop grid system to explore the power of migration strategies for fast turnaround.

Another contribution of our paper is an empirical heterogeneous host profile model for evaluating the performance of Internet-wide desktop grid systems. Previous research projects on desktop grid systems either do not consider heterogeneity of the hosts or use profiles of a small number of lab or office machines. We use statistical data from the BOINC project to generate the host profile, using it to analyze the performance of WaveGrid compared with systems using a range of non-timezone-based migration strategies.

Our simulation results show that: (a) WaveGrid outperforms other systems with respect to turnaround, stability and minimal impacts on hosts; (b) WaveGrid reduces migration delay and minimizes rescheduling attempts; and (c) All systems benefit from scheduling strategies that take host heterogeneity into account.

2 WaveGrid Architecture

The design of WaveGrid springs naturally from the observation that millions of machines are idle for large chunks of time. It is also influenced by the notion of prime time v. non-prime time scheduling regimes used by parallel job schedulers [14], which schedule long jobs at night to improve turnaround time.

There are many motivations for the design of WaveGrid. First, resource information, such as when the host will be idle and how long the host will continue to be idle with high probability, will help the scheduler make much better decisions. WaveGrid builds this information into the overlay network by having hosts organize themselves into the overlay network according to their timezones. Second, efficient use of large

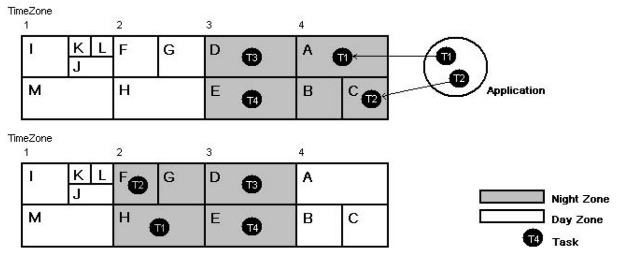

Figure 1. Job initiation and migration in WaveGrid

and relatively stable chunks of idle cycles provides the best performance, in contrast to using sporadic short periods of idle cycles which may be countered by high resource discovery and scheduling overhead. Therefore, WaveGrid proposes to use long idle night-time cycles. Third, the cycle donors are geographically distributed, so that their idle times are well dispersed on the human 24-hour time scale. Machines enter night-time in the order of the timezones around the world, making it well-suited for efficient migration. Fourth, the scheduler should not be intrusive to users' privacy. WaveGrid only needs minimal user input such as timezone information.

2.1 Overlay Construction and Scheduling

WaveGrid builds a timezone-aware, structured overlay network and it migrates jobs from busy hosts to idle hosts. WaveGrid can utilize any structured overlay network such as CAN [16], Pastry [17], and Chord [19]. The algorithm we present here uses a CAN overlay [16] to organize nodes located in different timezones and migration of jobs happens when the current host is no longer available(see Figure 1).

• **Wavezones in the CAN overlay**. We divide the CAN virtual overlay space into several *wavezones*. Each *wavezone* represents several geographical timezones. A straightforward way to divide the CAN space is to select one dimension of the d-dimensional Cartesian space used by CAN and divide the space into several wavezones along that dimension. For example, a 1 x 24 CAN space could be divided into 4 wavezones each containing 6 continuous timezones. Adjustments will be made for timezones with low user population.

• **Host nodes join the overlay:** A host node that wishes to offer its night-time cycles knows which timezone it occupies, say timezone 8. It randomly selects a node label in wavezone 2 containing timezone 8 such as (0.37, 7.12) and sends a join message to that node. According to the CAN protocol, the message will reach the physical node in charge of CAN node (0.37, 7.12) who will split the portion of the CAN space it owns, giving part of it to the new host node.

• **Client selects initial nightzone:** The scheduler for a workpile application knows which timezones are currently nightzones. It selects one of these nightzones (based on some selection criteria) and decides on the number h of hosts it would like to target.

There are a variety of nightzone selection criteria for selecting the initial wavezone and the wavezone to migrate to, including (a) schedule the task to the wavezone whose earliest timezone just entered night-time, (b) schedule the task to a random night-time zone, and (c) schedule the task to a wavezone that currently contains the most night-time zones. The first option performs better than the others, since it provides the maximal length of night-time cycles. However, it may be better to randomly select a nightzone if many jobs simultaneously require scheduling to avoid collisions.

• **Host Discovery:** The scheduler randomly generates h node labels in the wavezone containing the target nightzone and sends request messages to the target node labels using CAN routing. Each contacted host does an expanding ring search in a limited scope to discover more candidates.

• **Host Selection:** The application scheduler chooses the best host from the candidate group to schedule the foreign job on. The primary selection criteria in a desktop grid system is the availability of the host.

To address the concern that foreign jobs will disturb a user's local work, WaveGrid uses a strict host availability model, where CPU cycle sharing is limited to the time when owners are away from their machines and the CPU load from local applications is light. Figure 2 illustrates a sample host profile of available idle cycles under a strict user local policy in WaveGrid: The host is available only when the CPU load is less than 75% and there is no mouse or keyboard activity for 15 minutes. In reality, many cycle sharing systems use a conservative CPU availability model. Condor supports strict owner policies: users can specify a minimum CPU load threshold for cycle sharing, or specify specific time slots when foreign jobs are allowed on that host. SETI@home uses a screensaver model: it runs when no mouse or keyboard activities have been detected for a pre-configured time; otherwise it sleeps.

Secondary selection criteria includes the CPU power, memory size and type of operating system, etc. In this study, we focus on CPU power which is directly related to the execution time of the foreign job. When a group of candidates is selected based on CPU availability, the host with the best CPU power is chosen.

After negotiations, the application scheduler ships code to the chosen hosts.

• **Migration to next timezone:** When morning comes to a host node and the host is no longer available, it selects a new target nightzone, randomly selects a host node in that nightzone, and after negotiating with that host, migrates the unfinished job to the new host.

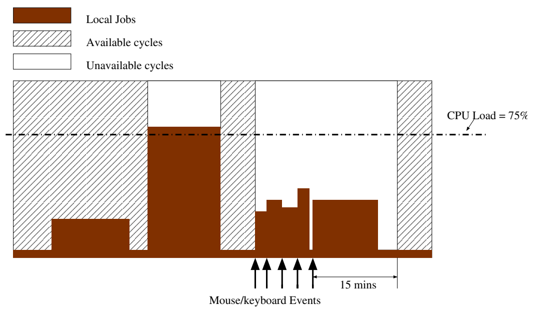

Figure 2. Sample host profile of available idle cycles

2.2 Migration

The type of applications which are suitable to schedule and migrate in WaveGrid are large *Workpile (Bag-of-tasks)* jobs. Each job consists of a number of independent tasks requiring large amounts of CPU cycles but little if any data communication. Examples of workpile applications include state-space search algorithms, ray-tracing programs, gene sequencing and long-running simulations. Often these applications have higher performance requirements such as faster turnaround time and higher throughput . Some of the above applications may have real time constraints, such as weather forecasting and medical diagnosis for a patient, or scientific simulations that must be completed in time for a scheduled reporting deadline.

The migration cost is higher in global peer-based cycle sharing systems than in local area networks because the code and data are transferred on the Internet. If a short running job is migrated many times in its life span, the accumulated migration cost may well counter the migration benefit. Long jobs which run for hours or even for months receive maximal benefit from migration. For such jobs, the cost of migration, which includes resource discovery overhead to find a migration target, checkpointing, and cost to transfer the code and data is negligible compared to the total runtime.

Many long running applications satisfy the migration criteria: small size and minimal data communication. For example, the average data moved per CPU hour by users of SETI@home is only 21.25 KB, which is feasible even for users with slow dial-up connections. With respect to program size, Stanford Folding is only about 371KB, and SETI@home is around 791KB. These applications run for a long time. The average computation time of each SETI@home job is about 6 hours.

3 Simulated Peer-based Desktop Grid Systems

To evaluate the performance of WaveGrid, we compare it with a *no-migration system* and a *random-migration system.*

3.1 Components of the Simulated Peer-based Desktop Grid Systems

In our simulator, we implement the peer-based desktop grid systems using the following components: overlay network construction, host selection criteria, host discovery strategy, local scheduling policy, and scheduling scheme. The systems we evaluated use the same host discovery method and local scheduling policy, but differ in overlay network construction, host selection criteria and scheduling schemes. We present the scheduling schemes, in section 3.2.

Overlay network construction. Both the *no-migration system* and the *random-migration system* use the CAN overlay network [16]. WaveGrid uses the CAN-based timezone-aware structured overlay network described above.

Local Scheduling. The local scheduling policy on a host determines the type of service a host gives to a foreign job that it has accepted. We use a *strict screensaver model.* We assume that one host can only accept one foreign job, and foreign jobs can only run when it is admitted by local user policy such as there is no recent mouse/keyboard activity and the CPU utilization is low. When the host is available, the foreign job concurrently shares cycles with other local jobs.

Host selection criteria. A client uses its host selection criteria to select one host among multiple candidates. The primary selection criteria in a desktop grid system is host availability. As discussed in section 2, we use a strict host availability model. The following terms define the criteria regarding CPU availability which we use in the simulation. *Unclaimed* means that there is no foreign job on that host. *Available* means that there is no foreign job on that host and the host is idle. The host's local user policy described in local scheduling is used to decide whether the host is idle.

Different systems use different host availability criteria. The simple no-migration system relaxes this criteria to use any *unclaimed* hosts, while WaveGrid and random-migration try to schedule foreign jobs on *available* hosts for instant execution.

All three systems use CPU power as a criteria to choose the best host among all the candidates.

Host Discovery. The purpose of the host discovery scheme is to discover candidates hosts to accept the foreign job. WaveGrid does an expanding ring search in the targeted zone, while the other two systems do expanding ring search in the neighborhood of the client or a random area of the system. The benefit of the latter approach is to create a balanced load in case of a skewed client request distribution in the overlay.

3.2 Simulated Scheduling Strategies

The scheduling scheme has two distinct steps: *initial scheduling* and *later migration.* In initial scheduling, the initiator of the job uses host discovery to discover hosts satisfying the host selection criteria and schedules the job on the chosen host. The migration schemes also use host discovery to discover candidate hosts.

The *no-migration system* follows the SETI@home model. It uses the more relaxed host selection criteria: any *unclaimed* host can be a candidate.

No-migration: With no-migration, a client initially schedules the task on an unclaimed host, and the task never migrates during its lifetime. The task runs in screensaver mode when the user is not using the machine, and sleeps when the machine is unavailable.

WaveGrid and random-migrate all use migration schemes, which differs in where to migrate. WaveGrid migrates jobs to *available night-time* hosts, while random-migrate migrates jobs to *random available* hosts. In regarding to when to migrate, WaveGrid and random-migrate both adopt three different options: *immediate migration*, *linger migration* and *adaptive migration.*

Immediate migration. With immediate migration, the client initially schedules the task on an available host. When the host becomes unavailable, the foreign jobs are immediately migrated to another available host. In the best case, the task begins running immediately, migrates as soon as the current host is unavailable, and continues to run right away on a new available host.

Linger migration. With linger migration, a client initially schedules the task on an available host. When the host becomes unavailable, linger migration allows the task to linger on the host for a random amount of time. If the host is still unavailable after the lingering time is up, it then migrates. Linger migration avoids unnecessary migration as the host might be temporarily unavailable. Linger migration can also be used to avoid network congestion or contention for available hosts.

Adaptive migration. For initial scheduling, adaptive migration tries to find a host that is *available*. If it cannot, migration-adaptive schedules the task on an *unclaimed* host. When the host becomes unavailable, adaptive migration tries to migrate the task to a new host that is available. If it cannot find such a host, it allows the job to linger on the current host for a random amount of time and tries again later. A cycle of attempted migration and lingering is repeated until the job finishes.

Adaptive migration is designed to avoid rescheduling. However, it puts a bigger burden on the host since it may retry several times on behalf of the foreign task.

4 Heterogeneous Host CPU Power Profile

We use a heterogeneous host CPU power profile derived from statistical data from the BOINC project in this study [3]. BOINC is client-server-based Internet-wide cycle sharing system, which attracts millions of users. BOINC supports several scientific applications and uses a credit-rewarding scheme to motivate hosts to donate cycles. Each time when a host returns a result, once the result is verified, the host is awarded some credits. Therefore the number of credits a host earns is directly proportional to the number of results it generates. For each application supported by BOINC, there is statistical data (http://www.boincstats.com) providing information such as total number of credits and average number of credits grouped by type of CPU, or type of operating system, or geographic regions. Table 1 shows some sample entries from the BOINC statistics website, which is grouped by types of CPUs.

We processed the statistical data about different CPUs in the system to generate the heterogeneous host CPU power profile via the following method. Hosts with the same type of CPU are regarded as in the same group. We exclude those groups with a very small population, e.g. rare kind of CPUs such as high-end multiprocessors or vanishing types of CPUs. As the credits assigned to one host is directly related to the number of results returned by it, we use the average number of credits per CPU to predict the power of that type of CPU. For each CPU group, we compute the credit ratio as maximal average number of credits earned among all different groups divided by average number of credits earned by hosts in that group, and the population of that group as a percentage of the total number of hosts. The smaller the credit ratio is, the better the CPU power of that group is.

Figure 3 shows the cumulative distribution graph, in which each data point represents one group. The graph shows that for different types of applications, the distribution of CPU power is different, probably related to the structure of the particular program and the type of users attracted to that particular project. Except for Climateprediction.net, the other three applications exhibit a distribution similar to a normal distribution: a large number of groups have a similar ratio.

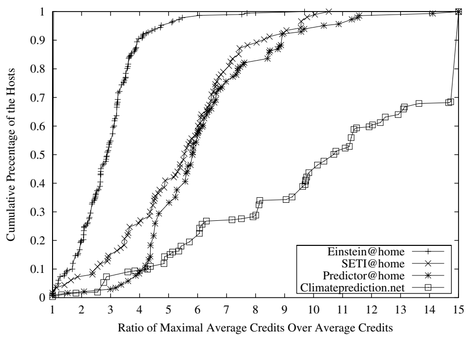

Figure 3. CPU powers of different CPU groups using empirical data from BOINC (collected in Aug. 2005). The credits are averaged over all the CPUs in the same group.

We further convert the credit ratio into power rank. If the credit ratio of a group falls in the range of $[k, k+1)$, the rank of that group equals k. The results in Figure 4 confirm that the distribution of the CPU power is far from a uniform distribution. The majority of the hosts are clustered on one or a few ranks. We use the information in Figure 4 as the heterogeneous host CPU power profile in our simulation.

The empirical data from BOINC also shows that the types of operating systems used by different hosts are quite uniform. About 89.1% of the hosts use Windows operating system, and 7.6% of the hosts use Linux. In our simulation study, we assume that all the hosts use the same type of operating system or invoke the same type of virtual machines to run the foreign job. The BOINC data provides some information about the geographic distribution of the hosts, in which it shows that the majority of the hosts reside in North America. We believe that this is biased information which derives

Pos.	CPU	#CPU	Total credit	Average credit	Credit per CPU	Average credit per CPU
1	Intel(R) Pentium(R) 4 CPU 3.00GHz	27,910	249,454,384.00	2,100,610.25	8,937.81	75.26
4	AMD Athlon(tm) Processor	13,934	47,707,040.00	356,019.28	3,423.79	25.55

Table 1. Sample statistical data for SETI@home organized by types of CPUs. (Note: "Average credit" is the average credit granted over the last few days to the hosts with this type of CPU.)

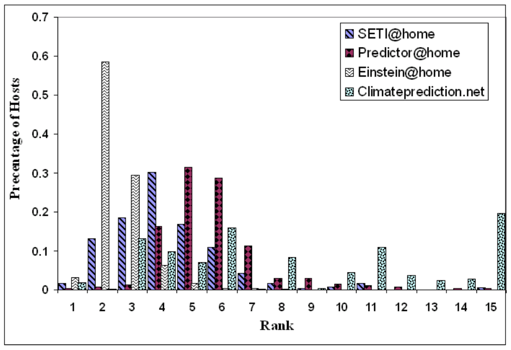

Figure 4. Percentage of CPU groups by rank

from the fact that North America is the birthplace of the BOINC project, and thus it attracts many users in North America. According to statistics about the population of Internet users [11], 34.2% Internet users are in Asia, 28.5% Internet users are in Europe, and 23.4% Internet users are in North America. Therefore considering the fast growth of the Internet and the end host connection speed, the population of hosts in different parts of the world on a symmetric global desktop grid system will be more equalized geographically than what is indicated by BOINC data.

5 Simulation Experiments

We compare WaveGrid with two other peer-based desktop grid systems, *no-migration* and *random-migration*, using varied migration schemes described in section 3 under an empirical heterogeneous host CPU power model. We used four metrics to evaluate the performance of the system: slowdown, makespan, number of rescheduling attempts, and migration overhead.

5.1 Simulation configuration

In all simulation runs, we use a 5000 node structured overlay network built with the CAN protocol.

Workload. A random group of peers (10% to 90%) are chosen as clients. Each client sends out one workpile application at a random time during the day. The *runtime* of a task is defined as the time needed for the task to finish on a dedicated machine with median CPU power rank over all the hosts in the system. This means that it will take a slow dedicated host more than the runtime to finish the job, and it will take a fast dedicated host less than the runtime to finish the job. The median CPU power is computed based on our empirical host CPU power profile, and the runtime is randomly distributed from 12 hours to 24 hours. Tasks belonging to the same application have the same runtime.

Profile of available cycles on hosts. A coarse-grain hourly synthetic profile is generated for each machine as follows: during the night-time (from 12am to 6 am), the host is available with a very low CPU utilization level, from 0% to 10%. During the daytime, for each one hour slot it is randomly decided whether the machine is available or not. The local CPU load in a free daytime slot is generated from a uniform distribution ranging from 0% to 30%. We assume that when a host is running a foreign job, it can still initiate resource discovery for migration and relay messages for other hosts. The percentage of available time during the day varies from 10% to 90%.

The computation power of the hosts follows the heterogeneous model discussed in section 4. The amount of available time on each host is weighted by its CPU power. The normalized available time equals the unweighted available time multiplied by the CPU power divided by the median CPU power in the whole system.

Rescheduling. When a client fails to find an available host during initial scheduling or a host fails to find an available host to migrate the job to, it will try to reschedule the task after a random amount of time. The waiting time in the simulation is chosen between 1 to 2 hours, which includes the time to restart the task. The amount of time the rescheduling takes is included in the total execution time of the task.

Host discovery parameters. The search scope for expanding ring search is 3 hops for all three systems, based on earlier studies on expanding ring search [22].

Migration parameters. The migration delay is added into the total execution time. The migration delay includes time to discover available hosts, record the current status of the program, ship the code and data to the new host and restart the program on new hosts. For most of the simulations, the migration delay is 5 minutes, which is a fairly conservative figure for the type of applications suitable to run in WaveGrid.

We explored a range of lingering times for linger migration and adaptive migration. The results presented are representative, and these experiments randomly choose the lingering time in the range 1 to 2 hours. For eager migration, we tested different wakeup intervals and chose to do background host discovery hourly.

Wave scheduler. For the Wave scheduler, a 1x 24 CAN space is divided into 6 wavezones, each containing 4 time zones based on its second dimension.

5.2 Simulation Metrics

Average slowdown factor: The slowdown of a task is its turnaround time (time to complete execution in the peer-to-peer desktop grid system) divided by the task runtime. Turnaround time includes both migration time and waiting time due to rescheduling. We average the slowdown over all tasks. The slowdown of one task can be less than 1 in this scenario as the runtime of the task is defined using the median CPU power of the system while the actual execution time on one host is weighted by its CPU power.

Average makespan: The makespan of an application is the time the first task of that application is submitted until the last task finishes, divided by runtime of the task (Tasks in one application have equal runtime.). We averaged the makespan over all applications.

Average number of migrations per task: the number of times a task migrates during its lifetime in the system, averaged over all tasks.

Average number of retries per task: the number of times a task is rescheduled during its lifetime in the system, averaged over all tasks.

5.3 Simulation Results

Our simulation study investigates the performance gains achievable through timezone-aware organization of the hosts, efficient migration, and scheduling strategies that consider the heterogeneity of the system.

In this section, the legends in each graph are ordered from top to bottom to match the relative position of the corresponding curves. Each data point is the average over 15 simulation runs.

1) Overall Performance of WaveGrid

The overall performance of WaveGrid is stable and is better then the other systems with limited available cycles, as WaveGrid efficiently organizes hosts according to timezone and migrates using this timezone information.

Figure 5 shows the makespan of applications when the percentage of available time on hosts varies. Figure 6 shows the slowdown of applications in the same scenario. All systems perform better as the amount of available time increases. No-migration performs the worst, as a lot time is wasted waiting for the hosts to be available again.

The performance of WaveGrid is quite stable over different host availabilities. When the available time on hosts during the day changes from 90% to 10%, the performance of WaveGrid degrades less than 10%. In contrast,the performance of random-migrate degrades more than 85%.

WaveGrid performs better than all the other systems as it makes efficient use of large chunks of night-time cycles, when the available cycles are limited. Random-migration improves with increasing host availability. When the hosts are mostly available during the day, it performs slightly better than WaveGrid. The reason for that is random-migration selects candidate hosts from a larger pool than WaveGrid, i.e. it is not restricted to the hosts in the next wavezone. Thus it has higher chance to identify some hosts with better CPU power in this heterogeneous system.

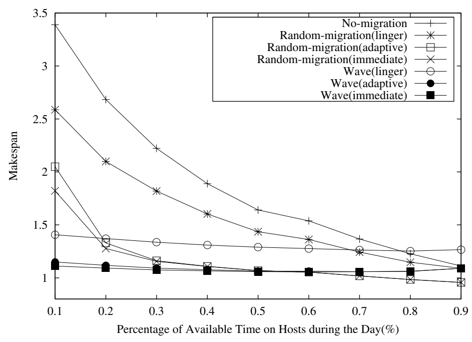

Figure 5. Average makespan vs host availability (The percentage of clients is 20%)

Figure 7 confirms that systems utilizing migration perform better than the no-migration option. The makespan of applications in the no-migration system has a long-tailed distribution, and in the extreme case the makespan is as high as 8. In contrast, the makespan of applications in migration-based systems, except for

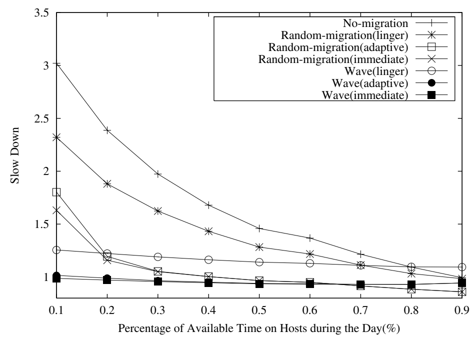

Figure 6. Average slowdown vs host availability (Percentage of clients is 20%)

those using linger migration strategies, is less than 2. WaveGrid using immediate migration strategy performs the best with the highest percentage of applications having a makespan which is less than 1. Recall that makespan can be less than 1, since it is defined in terms of a host with median CPU power.

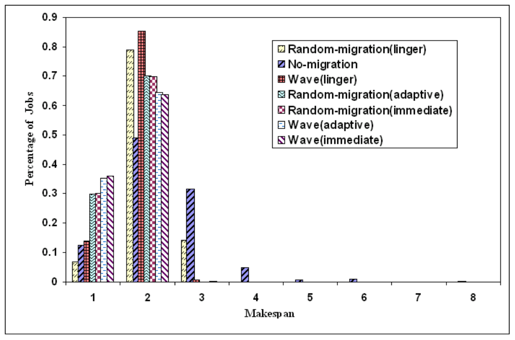

Figure 7. Histogram of distribution of makespan (Percentage of clients is 20%. Percentage of available time on hosts during the day is 40%.)

2) Effective migration in WaveGrid

The migration schemes used by WaveGrid effectively reduce the number of migrations, migration delay and number of rescheduling attempts.

Figure 8 shows the average number of migrations. As expected, jobs scheduled with WaveGrid finished with fewer migrations, because it exploits the long available intervals at night while the others may end up using short, dispersed time slots during the day. WaveGrid minimizes disturbance to hosts as it uses fewer hosts in migration and therefore contacts fewer hosts for resource discovery, which is important to cycle donors.

The graph also shows that when the percentage of host availability increases, the number of migrations in random-migration increases first then decreases. The reason is that there are two factors that influence the number of migrations: the number of currently available hosts and the length of free time slots on the hosts. With more available hosts, there is higher chance of migration success and therefore a larger number of migrations. With longer free time slots, the need for the jobs to migrate is reduced. With higher percentage of free time, the amount of currently available hosts increases and the length of free time slots also increases.

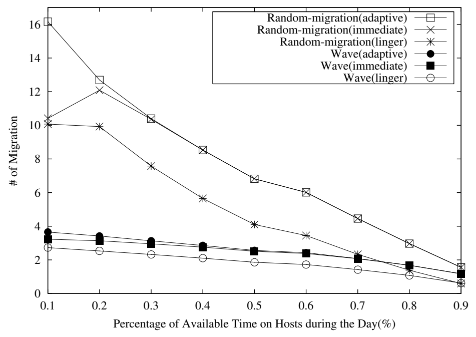

Figure 8. Average number of migrations (Percentage of clients is 20%)

Figure 9 shows how makespan varies as a function of migration delay. It shows that makespan increases with the increasing migration delay. The migration delay has a much larger impact on the performance of random-migration than WaveGrid. When the migration delay increases from 2 minutes to 22 minutes, the makespan in random-migration(immediate) increases about 19.4%, while the makespan in WaveGrid(immediate) only increases about 6%.

Figure 10 shows that WaveGrid has much fewer scheduling retries than random-migration. The reason for that is in random-migration the scheduler has no knowledge about where to find potential hosts with available cycles.

3) Heterogeneous Environment

Performance of all systems improves when the scheduling strategy chooses the most powerful host from multiple candidates.

Figure 11 shows the slowdown of applications when the scheduler selects one random host instead of the most powerful host among multiple candidates. (Only selected strategies are shown due to limited space.) Figure 12 shows the percentage of performance im-

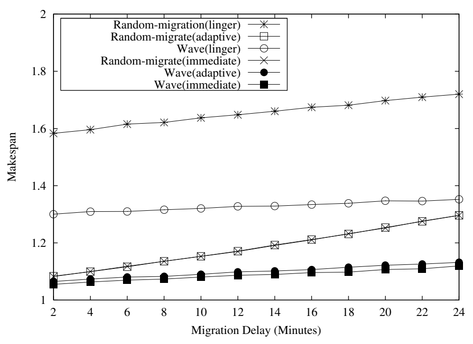

Figure 9. Average makespan when the migration delay varies (Percentage of clients in the system is 20%. Percentage of free time on hosts during the day is 50%.)

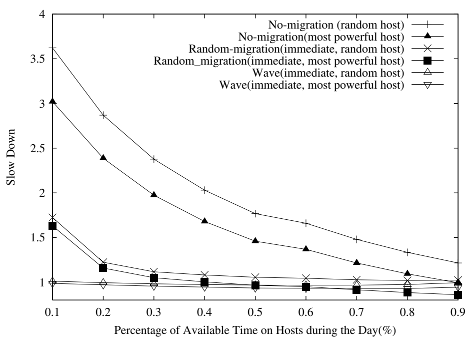

Figure 11. Average slowdown vs host availability when using random host selection (Percentage of clients is 20%)

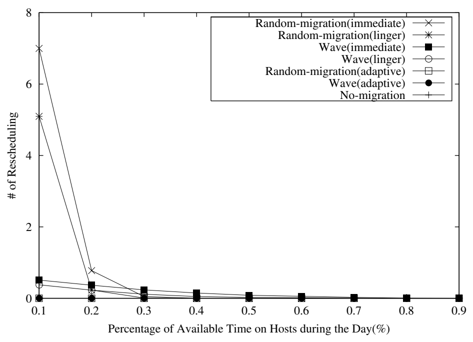

Figure 10. Average number of retries during the job execution (Percentage of clients is 20%)

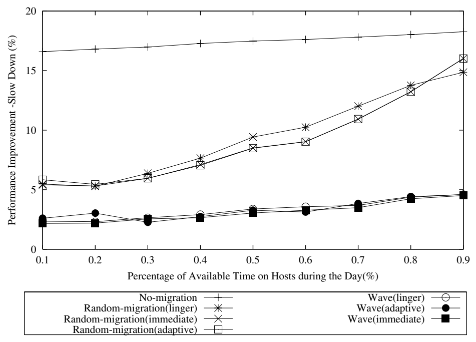

Figure 12. Percentage of performance improvement when using the most powerful host selection instead of a random host selection (Percentage of clients is 20%)

provement for all the strategies. Our results show that WaveGrid's superior performance is not greatly impacted by using the host selection strategy based on CPU power, although it does improve performance. We also see that this host selection strategy yields significant improvements for no-migration and random-migration.

6 Related Work

Our work belongs to the group of peer-based desktop Grid systems [2, 4, 15, 8], which harness idle cycles on desktop machines using peer-to-peer techniques. Each node in these systems can be either a single machine or an institution joining the peer-to-peer overlay network. Each peer can be both a cycle donor and a cycle consumer. OurGrid [2] proposed an accounting scheme to aid equitable resource sharing in order to attract nodes to join the system. Flock of Condors [4] organized the nodes in a Pastry [17] overlay network. SHARP [8] is aimed at secure resource sharing. Our CCOF project [15] is a generic scalable modular peer-to-peer cycle sharing architecture which supports automatic scheduling for arbitrary client applications.

Migration was originally designed for load sharing in distributed systems to move processes from a heavily loaded machine to a lightly loaded machine. Theoretical and experimental studies have shown that migration can be used to improve turnaround time [5, 18]. We are the first to investigate migration strategies in a peer-based desktop grid systems [21].

To our best knowledge, none of the previous work has addressed the fast turnaround scheduling problem in a scalable peer-based cycle sharing system. A recent paper [12] describes scheduling for rapid application turnaround on enterprise desktop grids. A central server chooses the best host based on criteria such as clock rate and number of cycles delivered in the past. Their work did not considering migration schemes, and it is limited to scheduling within one institution.

7 Conclusion

We propose a novel heterogeneous scalable fast-turnaround desktop grid system, WaveGrid. WaveGrid allows hosts to register themselves in a structured overlay network according to their long idle time slots at night. A client can quickly target an area to search for available hosts without sending a large amount of messages. Application schedulers migrate jobs from busy hosts to idle hosts with a potentially large chunk of available time, maximizing utilization of available cycles. The simulation results show that WaveGrid outperforms other systems with respect to turnaround, stability and minimal impact on hosts.

Heterogeneity is inherent to the nature of peer-based desktop grid systems. To accommodate heterogeneity of the system, we used a host selection criteria based on CPU power. In our simulation, we used a heterogeneous host CPU power model based on empirical data from BOINC. This model is helpful for designing and evaluating other global desktop grid systems.

Our future work includes a number of extension to WaveGrid, extending the night-time concept to any long interval of available time, a peer-to-peer checkpointing scheme using the underlying DHT to store program state, and merging or splitting wavezone according to population of the hosts in the geographic areas.

References

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45, 2002.

[2] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of JSSPP'03*.

[3] BOINC: Berkeley open infrastructure for network computing, http://boinc.berkeley.edu/.

[4] A. Butt, R. Zhang, and Y. Hu. A self-organizing flock of condors. In *Proceedings of SC2003*, 2003.

[5] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. on Software Engineering*, 12(5), 1986.

[6] Folding@Home http://folding.stanford.edu/.

[7] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11, 1997.

[8] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: An architecture for secure resource peering. In *proceedings of SOSP 2003*, 2003.

[9] R. Gupta and A. Somani. Compup2p: An architecture for sharing of computing resources in peer-to-peer networks with selfish nodes. In *Proceedings of P2P Econ 2004*.

[10] A. Iamnitchi and I. Foster. A peer-to-peer approach to resource location in grid environments. In J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, editors, *Grid Resource Management*. 2003.

[11] Internet world stats (usage and population statistic) http://www.internetworldstats.com/stats.htm.

[12] D. Kondo, A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of SC2004*, 2004.

[13] M. Litzkow, M. Livny, and M. Mutka. Condor -a hunter of idle workstations. In *Proc ICDCS'88*, 1988.

[14] V. Lo and J. Mache. Job scheduling for prime time vs. non-prime time. In *Proc 4th IEEE International Conference on Cluster Computing (CLUSTER 2002)*, 2002.

[15] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster Computing on the Fly: P2P scheduling of idle cycles in the Internet. In *IPTPS*, 2004.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. ACM SIGCOMM*, 2001.

[17] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, 2001.

[18] N. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *Computer*, 25(12), 1992.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, 2001.

[20] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computng: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2002.

[21] D. Zhou and V. Lo. Wave scheduler: Scheduling for faster turnaround time in peer-based desktop grid systems. In *Proc JSSPP'05*.

[22] D. Zhou and V. Lo. Cluster Computing on the Fly: resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of the 4th International Workshop on Global and P2P Computing (GP2PC'04)*, 2004.

[23] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23(12), 1993.