Utilizing Idle Workstations*

Graham P. Phillips
Computer Science Department
University of Southern California
Los Angeles, CA 90098-0781
graham@isi.edu

May 1, 1997

Abstract

The decreasing cost of personal workstations over recent years has led to their proliferation in computing environments. Studies have shown that personal workstations are under-utilized for long periods of time during regular working hours. This paper identifies this under-utilization as the idle-workstation problem.

A taxonomy of computational models of systems that have attempted to solve the idle workstation problem is presented. The computational model includes issues such as, whether the user view of resources is location-transparent, whether heterogeneous processor instruction sets are supported, whether systems can interoperate and whether job migration is supported. The paper also presents those features that designers of future distributed systems should address to motivate users to pool their workstations.

1 Introduction

The decreasing cost of personal workstations over recent years has led to their proliferation in computing environments. Studies have shown that personal workstations are under utilized for long periods of time during regular working hours and especially during nights and weekends. One study of a group of workstations at the University of Wisconsin found that over a period of 5 months

*This paper was submitted in partial fulfillment of the Advanced Operating Systems class (CS555) held at USC during the Spring of 1997.

only 30% of their capacity was utilized [LLM88]. Similarly another study of workstations at Stanford University found that over one third of the workstations were idle during the busiest times of the day [TLC85]. These studies reveal that a tremendous amount of computational resources is largely untapped. This trend is likely to worsen as hardware prices fall and the proliferation of workstations continues.

We call the under-utilization of workstations the *idle workstation problem*. Our aim is to investigate existing systems that provide a solution to the idle workstation problem. In particular we are interested in answering the following questions:

- What is the model of computation used? Are parallel programs supported?
- Are the workstations closely coupled in that they share resources? For example, do workstations share file systems?
- Is the job scheduler a central server or a distributed one? Are jobs statically or dynamically assigned?
- Can jobs be migrated when a user returns to his workstation or are intermediate results lost?
- How well does the system interoperate with existing (heterogeneous) operating systems?

We use the following terminology from [Dou89]. The host on which a program is invoked is called its *home* while a host initiating process migration is called the *source*.

The limitations of this paper are that security considerations are not investigated. Clearly security must be addressed before the widespread deployment of a system will occur.

The remainder of this paper is organized as follows. Section 2 motivates the study of the idle workstation problem, section 3 provides a taxonomy of computational models of existing systems, section 4 presents issues that future distributed systems should address and finally, section 5 investigates ongoing research work.

2 Motivation

One of the goals of distributed operating systems is to present resources in a location-transparent fashion to the user. Modern operating systems provide transparency for some resources, for example, the Andrew File System in Andrew, distributed shared memory in Clouds, X Windows in Athena and interprocess communication ports in Mach [CDK94]. Few operating systems provide an integrated and transparent user view of processors, notable exceptions are the Amoeba, Clouds and Plan 9 [CDK94, page 44]. We believe that this problem is caused by the economics of computer manufacturing. In particular, most computing hardware is packaged as autonomous computing units in the form of workstations (or personal computers). Economies of scale reduce the price of these units which further promote their sale. The consequence is that users typically view their workstations as personal environments (because they paid for them) and interference from other users is not tolerated.

We encourage an alternate view, held by a growing number of researchers, that providing idle computational resources to other users is also beneficial to the provider. For example, a study of resources in the Condor system showed that for every minute of local capacity consumed to support local remote execution of jobs, nearly 22 hours of remote capacity was received [LLM88].

3 Taxonomy of models

This section presents a taxonomy of existing systems that provide a solution to the idle workstation problem. We have attempted to *abstract* and *identify* the computational models of these systems. Consequently, the taxonomy is structured not by existing systems but by the our abstracted models. We are unable to provide a completely disjoint taxonomy and therefore we cite their dependencies. All the systems investigated have the one feature in common that they support the dynamic assignment of jobs to workstations at job invocation time.

3.1 Processor pool

The processor pool model effectively avoids the idle workstation problem by logically decoupling the processor from the desktop platform and placing it in a pool. The first system based on the processor pool model was the Cambridge Distributed Computing System [NH82]. The most cited system that incorporates a processor pool is the Amoeba system [TvRvS^{+}90] while more recent systems include Plan 9 and Clouds [CDK94]. In this model processes are allocated to processors for their lifetimes and thus process migration is unnecessary.

In the Amoeba system, the pools of processors is managed by the run server. Processors can be part of a multiprocessor or multicomputer, be a collection of single board computers or a group of workstations. Multiple processor architectures are supported in a transparent fashion. Furthermore, parallel programs can utilize multiple processors simultaneously. Memory is not shared but is private to each processor.

A limitation of Amoeba is that it does not provide binary-level compatibility with widely used operating systems such as Unix, instead it provides only a partial library-level emulation of Unix.

The Clouds system is based on an object-thread paradigm in which each object has an address space that encapsulates code and data. Threads are activities that execute methods within objects which in turn may call methods within other objects. When a method in an object is invoked, a compute server is chosen to execute the method. The compute server fetches the object from storage and then executes the particular method.

3.2 Worm programs

The worm programs, developed at Xerox, were one of the first systems to hunt for idle workstations [SH82]. The worm programs consist of a number of program segments where each segment runs on a separate computer. The worm segments remain in communication with all the other segments and collectively they are responsible for constructing another segment, should a segment be terminated (by a workstation becoming unavailable). Idle workstations are sought by probing the hostname address space, starting with the address of the local host and then working through the address space. Application programs are written using the services provided by the worm programs.

Because of the particular method used to search for idle workstations, the hunting capability will not work well in wide-area networks. In particular, the Internet hostname address space is substantial (2^{64} addresses). Vast regions of the Internet address space are without an associated host and consequently it would take an inordinate amount of time and network traffic to locate an idle host.

One advantage of the above search method is that collecting information about workstation availability is automatic and there is no need to manually configure workstation pools. However, these worm programs operated in a *homogeneous* environment consisting of identical hardware and worm migration was limited to the local area network. The most limiting factor is that all worms are required to communicate with each other. Therefore these programs will not behave correctly across network partitions nor will these programs scale to a large numbers of worms.

3.3 Supervisor-worker model

A task is divided into a supervisor process and several worker processes where typically the workers are identical. Although the supervisor is responsible for completing the particular task, it plays a passive rôle because the worker processes initiate communication by requesting a subproblem from the supervisor. That is, the supervisor relies on workers making themselves available.

Tasks that can be mapped into the supervisor-worker model are an important subset of distributed computation for two reasons. Firstly, massive parallelism of work is possible and secondly, the complex mechanism of process migration is unnecessary. Process migration is unnecessary because any intermediate results that are lost (from a worker becoming unavailable) are ignored and the supervisor simply hands the subtask to another worker.

This model is particularly suited for tasks that can be divided into subtasks that have few interactions, thereby allowing implementations with low communication requirements. Examples include the generation of primes numbers [Mul94, page 83] and the solution of encryption codes [Hug97].

Recently, a RCS code with a 48-bit key was solved on the Internet, utilizing various combinations of hardware and operating systems [Hug97]. The encryption code was solved in 13 days with an average test rate of 140 million keys per second. At one point during the computation over 5000 machines were testing 440 million keys per second! This demonstration is proof that significant computational resources are available on the Internet.

The disadvantage of this model is that not all problems can be decomposed into subproblems that have few interactions.

3.4 Location-transparent execution

Programs belonging to this category have the property that their behavior is invariant whether executed locally or remotely. We ignore the differences in the behavior of a program running on faster or slower hardware. System calls from a remote process are typically forwarded to the process' home.

The Sprite operating system provides a location-transparent execution environment by making processes appear in all ways to execute on a single host throughout its lifetime [Dou89]. Location-transparent execution is supported by a shared file system. Interprocess communication is implemented through the file system using pipes and pseudo devices, thereby providing access to the X Window System and the Internet. Traffic to the home is reduced by making *most* system calls location-independent.

The degradation of performance due to remote execution depends on the ratio of location-dependent system calls to the amount of computation. Tasks such as compilations and text formatting show little effect from remote execution. Conversely, performance is improved for computation-bound parallel programs. For example, Sprite provides a parallel make facility called pmake with good speedup properties.

The advantage with this model is that remote execution is invisible to the user and therefore the user is not burdened with separately managing different classes of jobs. Alternatively, if execution was not location-transparent and if certain programs were *required* to run locally then these programs would have to be managed separately from other programs. The disadvantage with the Sprite system is that remote execution can occur only between machines of the same architecture.

3.5 Process migration

Process migration is a more general abstraction than remote execution. Remote execution can be thought of as process migration just before the process starts to execute. There are a number of operating systems that provide process migration facilities, including V [TLC85], Locus [WPE+83], Sprite [DO87] and Accent [Zay87]. We introduce the term *full process migration* to denote a process, that once migrated, is no longer dependent on the source machine (with the exception of being dependent on a terminal I/O process on the home, in the case that the home and the source are the same machine). Locus, Accent and Sprite do not implement *full process migration* while the V system does.

- Locus is upwardly compatible with Unix and allows a parent and child process to run on different machines. For Unix compatibility, the parent and child must share the same process environment. The implication is that if one of these processes is migrated individually then the shared environment introduces a dependency with the process' source machine.

- Accent uses a lazy virtual memory transfer mechanism to improve migration performance. The implementation requires that the source dedicate memory to the process after migration has completed.

- Sprite implements location-transparent execution by forwarding those system calls that are location dependent to the process' home. Therefore a dependency always exists between the process and its home. Therefore when the source and the home are the same machine (ie. the process is migrated from the home rather than some other machine), Sprite does not comply to our definition of full process migration.

- The V system implements full process migration by providing a logical host abstraction in which processes are executed. Process migration is implemented by simply migrating the logical host containing the process. The limitation of the V system is that it does not provide complete execution transparency. In particular, the migration facility is restricted to applications that execute 'only those operations whose output is independent of the host'. This limitation does not preclude the remote execution of *interactive* programs. For example, the designers claim that they perceive no difference when remotely executing interactive editors such as emacs [Che88, page 328].

The disadvantage of process migration in general is twofold. First, it introduces complex interactions with the kernel which makes kernel implementation cumbersome. Second, the large amount of state accumulated by processes increase their migration time. Process state typically includes virtual memory, accessed files, message channels, execution state and kernel state [DO89]. Note that it is not only *open* files that are included in the file state. Temporary files that have been created and *closed* must also be migrated because the program may access them at a later stage.

The advantages of process migration is that processes can be evicted from workstations when users return. Clearly, eviction can occur without process migration, by simply terminating the remote process, but the check-pointing function of process migration ensures that intermediate computations are not lost. For jobs with long execution times, process migration is indispensable.

3.6 Central resource registry

This model maintains information about distributed workstations in a central registry or database. The registry typically contains information on a per workstation basis such as availability, processor architecture, the amount of memory and available local disk space. The registry provides an interface for entries to be removed, added or updated.

Systems that fall outside this model are either those systems that maintain remote resource information at each workstation, for example Locus [WPE+83] and Nest [AE87] or those systems that maintain resource registries in a distributed way, for example the Prospero Resource Manager [NR94] or the resource data base in Grapevine [BLnS82]. This model is identified because it is common to several distributed systems:

- Sprite implements the resource registry as a shared file [DO87]. This approach suffers from the problem that locking and synchronization of the shared writable file must be enforced.

- The Butler system implements the registry as a set of files in a special directory in the shared file system [Nic87]. A process called the butler runs on each workstation and is responsible for updating a single file in the directory. To speed up the search process, this information is cached by the mreg process which runs on a subset of the workstation. The list of available machines is only a hint, the particular workstation must first be interrogated before a program can be remotely executed on it.

- The V System implements the registry as a server process and nodes proclaim their availability by periodically broadcasting to the server [TLC85].

- In Amoeba, the run server implements the resource registry [TvRvS+90].

- In Condor, a central coordinator polls each workstation every two minutes to determine its availability [LLM88].

The advantage of using a registry is that resources are available to all users in the workstation pool. Furthermore, the advantage of structuring the registry as a central server is simplicity of implementation. On the other hand, the central server presents a single point of failure. However, this disadvantage is mitigated if the registry is considered only a *hint* of available resources (as in the Butler system) because of the existence of a fall-back interrogation mechanism initiated from the home machine.

3.7 Fair access to resources

This model maintains information about the use of workstation clusters so that resources are allocated fairly amongst users. The model prevents heavy users from acquiring all the available resources at the expense of those users who acquire resources occasionally. Fair allocation requires up-to-date information about the availability all workstations in the cluster.

The Up-Down algorithm used by the Condor system provides fair access to remote workstations [LLM88]. The algorithm accounts for user workload by maintaining an index for each workstation. The index is increased when remote resources are allocated and decreased when remote resources are denied. Every two minutes a central coordinator checks each workstation for new jobs. If a new job is present on a workstation and there are no idle workstations left then a job is preempted from a workstation with a higher index so that the new job takes its place.

The aim of the Prospero Resource Manager (PRM) is to provide scalability to a large number of workstations by dividing management tasks between system managers and job managers [NR93, NR94]. Each system manager is responsible for a disjoint subset of the full set of resources. The job manager on the other hand, acts as an agent for a single job, requesting resources from a particular system manager on behalf of the applications in the job. Fairness is achieved because the PRM model allows system managers to communicate. The advantage of both Condor and PRM is that they provide binary-level compatibility for Unix programs.

The advantage of this model is that greedy users are prevented from dominating occasional users. The disadvantage is that accurate state of all workstations must be maintained. This problem is common in distributed systems, particularly when scaling these systems to a large number of nodes.

3.8 Non-interactive remote execution

This model specifically limits the interactive support provided to remotely executed jobs. Because the submitted jobs are non-interactive they can be decoupled from the home workstation and executed on widely distributed workstations.

The Distributed Batch Controller (DBC) implements a federation of workstation pools [CSL96]. Each pool is autonomous and managed by the Condor Resource Manager [LLM88]. A DBC agent interacts with Condor to request resources while a DBC coordinator assigns work to the agent. The coordinator does not control the workstations but rather uses resources allocated to it by the Condor Resource Manager. Note that the passive rôle played by the coordinator is similar to the supervisor in the supervisor-worker model.

The computational model of DBC is simple, programs read a single input file and produce a single output file. The input and output files are transferred to the remote pool using the FTP protocol, thereby avoiding the need for a shared file system. The simple model precludes the use of certain operations such as execution pipelines (where the output of one job forms the input of the next job). Another limitation is that DBC interacts only with Condor pools. On the other hand, DBC can harness computational resources over a wide geographical area and therefore has access to a larger number of processors than other systems.

4 Discussion

Systems that provide a solution to the idle workstation problem have not been widely deployed. This statement is substantiated by observing that most workstations do not belong to workstation pools. Condor is one system that has enjoyed some success, for example, in 1994 a federation of Condor pools was established at universities in Madison, Delft, Amsterdam, Geneva, Berlin and Warsaw [ELvD+95]. Reasons for its success include binary-level compatibility with Unix and a mechanism for controlling access to resources.

We believe that distributed systems should provide an integrated solution to the idle workstation problem. In particular, designers should address the following issues:

- Program execution should be location-transparent so that program behavior is invariant whether executed locally or remotely. This invariant allows users to manage programs in a uniform way.

- The job scheduling mechanism should be as simple as possible to ensure widespread deployment in a heterogeneous environment. Furthermore, the simple scheduling mechanism of placing no more that one job on an idle machine is sufficient to prevent low processor utilization.

- The system should interoperate with other heterogeneous systems thereby enabling jobs to be transferred between systems. Interoperability will ensure that a large number of resources will be available to all users. Designers should also note that those systems that are compatible with Unix have enjoyed the most widespread deployment.

- The system should support multiple processor architectures in a transparent fashion. New processor continue to be developed and users are likely to upgrade their machines are different times.

- A shared file system within pools greatly simplifies remote execution and process migration mechanisms. However, if the system is to support the transfer of jobs between widely distributed pools then the system should manage file systems that are not shared.

- Process migration is recommended. The advantages of process migration is that the checkpointing function prevents long running jobs from losing intermediate results when workstations are reclaimed.

5 Future research

Three systems—Condor, the Prospero Resource Manager and the Distributed Batch Controller—are adopting the Parallel Virtual Machine (PVM) framework [Sun90] to enable the execution of parallel programs on a collection of machines [PL95].

PRM researchers plan to include mechanisms for program check-pointing (provided by Condor), relinquishing workstations and dynamic reconfiguration of system managers [NR94].

The Condor system was recently extended to provide resources beyond the boundaries of Condor pools by forming a federation of Condors [ELvD+95]. New mechanisms must be developed to provide adequate controlled access to resources in autonomously administered domains. Also, various scheduling issues must be resolved. One scheduling issue is that available local capacity should be selected over remote capacity, which implies that simultaneous bidirectional transfers of jobs between pools should be avoided. Another scheduling issue is that one might want to match the job with the longest expected run time with the fastest machine.

6 Conclusions

We have argued that users should cluster their workstations to benefit from the largely wasted computational resources found in autonomous workstations. We identified a taxonomy of computational models that provide a solution to the idle workstation problem. Our motivation for creating the taxonomy was to identify those features best suited for solving this problem in a distributed heterogeneous environment. These features were presented in the form of design guidelines.

Current work is focussed on increasing the number of pooled processors by connecting workstation pools together using high-level scheduling mechanisms. To date these systems have not enjoyed wide-spread deployment. However, we expect that users will cluster their workstations when these systems mature.

7 Acknowledgments

John Heideman and Ted Faber brought the research work of the Prospero Resource Manager to our attention.

This work was supported in full by a Graduate Research Assistantship held at Information Sciences Institute (USC), Los Angeles.

References

[AE87] R. Agrawal and A. K. Ezzat. Location independent remote execution in NEST. *IEEE Transactions on Software Engineering*, SE-13(8):905–912, August 1987.

[BLnS82] Andrew D. Birrell, Roy Levin, Roger M. needham, and Michael D. Schroeder. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[CDK94] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, second edition, 1994.

[Che88] David D. Cheriton. The V Distributed Sytem. *Communications of the ACM*, 31(3):314–333, March 1988.

[CSL96] Chungmin Chen, Kenneth Salem, and Miron Livny. The DBC: Processing scientific data over the Internet. *16th International Conference on Distributed Computing Systems*, May 1996.

[DO87] F. Douglis and J. Ousterhout. Process migration in the Sprite operating system. In *Proceedings of the 7th International Conference of Distributed Computing Systems, Berlin, West Germany*, pages 18–25. IEEE, September 1987.

[DO89] Fred Douglis and John Ousterhout. Transparent process migration: Design alternatives and Sprite implementation. Technical report, Electrical Engineering and Computer Sciences, Univeristy of California, Berkeley, CA 94720, 1989.

[Dou89] Fred Douglis. Experience with process migration in Sprite. Technical report, Electrical Engineering and Computer Sciences, Univeristy of California, Berkeley, CA 94720, 1989.

[ELvD+95] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pryne. A worldwide flock of condors: Load sharing among workstation clusters. `http://www.cs.wisc.edu/condor/doc/flock.ps`, 1995.

[Hug97] Brent Hugh. Distributed Internet crack. `http://www.ee.ethz.ch/~rsa_clng/en/dicinfo.html`, 1997.

[LLM88] M. Litzkow, M. Livney, and M. Mutka. Condor—a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111. IEEE, June 1988.

[Mul94] P. J. Muller. Environment for distributed programming. Master's thesis, University of Stellenbosch, South Africa, 1994.

[NH82] R.M. Needham and A.J. Herbert. *The Cambridge Distributed Computing System.* Addison-Wesley, Wokingham, 1982.

[Nic87] David A. Nicols. Using idle workstations in a shared computing environment. *ACM Operating Systems Review*, 21(5):5–12, 1987.

[NR93] B. Clifford Neuman and Santosh Rao. Resource managament for distributed parallel systems. In *Proceedings of the 2nd International Symposium on High Performance Distributed Computing, Spokane*, pages 316–323, July 1993.

[NR94] B. Clifford Neuman and Santosh Rao. The Prospero Resource Manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice and Experience*, 6(4):339–355, June 1994.

[PL95] Jim Pryne and Miron Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. Technical report, Department of Computer Sciences, University of Wisconsin-Madison, 1995.

[SH82] John F. Shoch and Jon A. Hupp. The "worm" programs—early experience with a distributed computation. *Computing Practices*, 25(3):172–180, 1982.

[Sun90] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.

[TLC85] M. M. Theimer, K.A. Lantz, and D. R. Cheriton. Preemptable remote execution facilities for the V-System. In *Proceedings of the 10th Symp. on Operating System Principles*, pages 2–12, December 1985.

[TvRvS+90] Andrew S. Tanenbaum, Robbert van Renesse, Hans van Stavern, Gregory J. Sharp, Sape J. Mullender, Jack Jansen, and Guido van Rossum. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33(12):46–63, December 1990.

[WPE+83] Bruce Walker, Gerald Popek, Robert English, Charles Kline, and Greg Thiel. The LOCUS distributed operating system. In *Proceedings of the Ninth Symposium on Operating Systems Principles*, pages 49–70. ACM, October 1983.

[Zay87] Edward R. Zayas. Attacking the process migration bottleneck. *ACM Operating Systems Review*, 21(5):13–24, 1987.