Survey on Idle Resources Utilization

Gerd Dauenhauer
Department of Computer Sciences
University of Salzburg, Austria
Gerd.Dauenhauer@cs.uni-salzburg.at

Salzburg, July 11 2007

1 Introduction

As cited in papers on performance measurement and profiling [14], [15], computing power of desktop workstations is available, i.e. unused 50 – 70% of the time – not only on weekends and during the night but also during busiest working hours. Since computing capacity steadily increases, these numbers are likely to grow. Usual resource request patterns, however are bursty, i.e. phases of multiple consecutive requests and phases of idleness are alternating. Although the papers focused on the CPUs, the term resource is of course much wider. This survey therefore also presents papers specifically dealing with using disks and network interfaces, as well as a paper dealing with resources for background use in general.

2 Process and Data Migration Systems

Process and data migration systems are a classical concept, used to distribute resource intensive jobs between nodes in a pool of computing nodes. Processes or work packages are preferably transferred to idle nodes to maximize the overall performance.

2.1 V-System

The idea of the V-System is to treat idle workstations as a pool of processors – although in [16], there is no explanation about how to identify idleness. The designers concentrated on a transparent execution environment should be transparent, which means that processes should not notice whether they are executed on the local machine or remotely. Another aspect is the migration of a process, i.e. copying its state must be an atomic action. A process also must not depend on the computer it is currently executed on, i.e. it must not create temporary files locally.

The System is implemented on Sun hardware using a kernel server and manager processes. Tools for executing processes on specific hosts and migrating processes between hosts are provided. The V-System project seems to be inactive now.

2.2 Condor

The Condor system [6] provides a processor pool abstraction based on single workstations. The projects main research aspects were:

- The analysis of workstation usage patterns. Observations cited in [10] showed that only 30% of the machine's resources were utilized – even during the busiest hours – with long intervals of idleness.

- Management and allocation algorithms for resources. Jobs demanding much capacity should be granted a large time share, without slowing down other jobs to much. The Up-Down algorithm [13] is proposed to provide fair access.
- The implementation of remote execution facilities. In [10] on an early version, multiple implementations are mentioned, including the V-System [16] and the Remote UNIX (RU) facility [11].

Condor is still in use today and has evolved to support grid-style computing on most UNIX and Windows NT/2000 platforms.

2.3 Mether

Mether [12] is based on the idea of a distributed shared memory, based on SunOS 4.0 workstations connected by Ethernet. Processes access this shared memory by opening a special mether raw device and mapping this device into memory. Mether therefore allows a process to utilize unused memory of remote machines.

2.4 SETI@home, Genome@home and Folding@home

SETI@home [18], Genome@home and Folding@home [9] are examples of data migration systems. Instead of copying processes to remote machines, these systems feed applications already installed on remotemachines with data (work units) to process. The X@home projects all make use of idle computing power of tens of thousands of workstations – the accumulated CPU time of such systems can go up to $400,000$ years within three years [9]. Idleness of these workstations is usually detected in an ad-hoc way, like CPU-usage below some threshold, no user logged in or running screensaver.

2.5 BOINC

BOINC [5] is a generalization of the ideas behind SETI@home, Folging@home and Genome@home. It emphasizes on public-resource computing, which contrasts with Grid computing – Grid computing usually depends on organizationally-owned computers. The BOINC framework supports a wide range of languages (C, C++ and FORTRAN) and computer platforms (Mac OS X, Windows, Linux, UNIX).

3 Network Priorization Systems

Using idle network resources can help to increase the user-perceived service quality of an application: the application, e.g. a web browser, may speculatively pre-fetch and cache large amounts of data to answer future requests. Caching data relates to use of idle disk resources, described later.

3.1 Bandwidth Capacity Allocation

The framework for explicit allocation of network bandwidth described in [1] is based on an extension to the traditional internet protocols. Its aim is to provide different levels of service, increasing the overall utilization of network resources. It also provides means for charging for usage.

Where the internet protocols provide best effort service, the bandwidth allocation provides a predictable service for different types of data. Network congestion is handled based by a ranking of packets by price. Only those packages are served, that are willing to bid above the cutoff price.

3.2 TCP Nice, TCP-LP

As stated in [17], today's applications could benefit from making large backround transfers of data, no user is currently waiting for. This pre-fetched data would increase the perceived service quality. Hand tuning these background requests, however is a challenge.

The authors of TCP Nice therefore present another congestion algorithm for the TCP protocol – applications could choose between traditional congestion control and the new algorithm, specifically designed for background traffic. The reference implementation for Linux allows senders to select between algorithms on a per connection basis, without any modifications on the receiver side.

Another TCP extension very similar to TCP Nice is TCP-LP [8], which also provides support for low priority background transfer. The major difference between both algorithms is how sense for congestion: TCP-LP uses one-way delay, where TCP Nice uses round trip time.

4 Disk Scheduling

Usage of idle disk resources has two aspects: use of the free capacity for caching and scheduling disk access between low and high priority processes. Such low priority processes that should not interfere with foreground processes are e.g. maintenance tasks such as virus scanning or disk de-fragmentation.

4.1 Anticipatory Disk Scheduling

Traditional disk schedulers are work conserving, which means that as soon a request is finished, the next waiting request is served. Application that frequently issue synchronous disk requests, followed by short pauses like intervals of computation, could therefore be delayed by concurrently running applications also issuing disk requests – the disk scheduler makes its decision to handle this concurrent disk requests to early, causing delays by positioning the disk head.

The anticipatory disk scheduler [7] tries to solve this problem by introducing short delays before a new disk request is handled. This way, the overall system performance is increased and low priority background processes do not interfere with foreground processes that make heavy use of disks.

The scheduler was first implementation as a FreeBSD kernel extension. A more general solution, supporting arbitrary resources is presented in [3].

5 Preemption Intervals

The idletime scheduler with preemption intervals is a general concept, applicable to any class of resources. The implementation presented in [3] supports disks and network interfaces.

The idea is to relax the work conserving principle for resources used by low priority processes: Requests from foreground processes with higher priority are executed by the scheduler as soon as they arrive, while the scheduler imposes a short delay after each request from a low priority background process. If no new foreground request arrives, one enqueued background request is served. If, on the other hand, a new foreground request arrives, the background request is delayed again.

The current algorithm, however, requires manual tuning and therefore does not seem to be useful in this form.

The scheduler was implemented as an extension to the FreeBSD kernel. Further details and performance evaluations are presented in [4].

6 Application Level Mechanisms

6.1 MS Manners

MS Manners [2] is a library for regulating progress of Windows applications based on monitoring the applications progress. Little progress is an indicator for resource contention, where low priority background processes would be throttled.

Usually, programs would use a single function for measuring the progress, but the authors also described a progress regulator implemented as an external program which measures progress of an application by evaluating its Windows performance counters and suspending the threads accordingly.

The framework does not require any kernel modifications and no manual tuning. Example applications include a disk defragmenter, which does not interfere with foreground work.

References

[1] David D. Clark and Wenjia Fang. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Trans. Netw.*, 6(4):362–373, 1998.

[2] John R. Douceur and William J. Bolosky. Progress-based regulation of low-importance processes. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 247–260, New York, NY, USA, 1999. ACM Press.

[3] Lars Eggert and Joseph D. Touch. Idletime scheduling with preemption intervals. *SIGOPS Oper. Syst. Rev.*, 39(5):249–262, 2005.

[4] Lars Rene Eggert. *Background use of idle resource capacity.* PhD thesis, 2003. Adviser-Joseph D. Touch.

[5] BOINC Project Homepage. http://boinc.berkeley.edu/.

[6] Condor Project Homepage. http://www.cs.wisc.edu/condor/.

[7] Sitaram Iyer and Peter Druschel. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous i/o. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 117–130, New York, NY, USA, 2001. ACM Press.

[8] Aleksandar Kuzmanovic and Edward W. Knightly. Tcp-lp: low-priority service via end-point congestion control. *IEEE/ACM Trans. Netw.*, 14(4):739–752, 2006.

[9] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande. Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology.

[10] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[11] Michael J. Litzkow. Remote unix: Turning idle workstations into cycle servers. In *Proceedings of the Summer 1987 Usenix Conference*, pages 381–384, June 1987.

[12] Ronald G. Minnich and David J. Farber. The Mether system: Distributed shared memory for SunOS 4.0. pages 51–60, Summer 1989.

[13] Matt Mutka and Miron Livny. Scheduling remote processing capacity in a workstation-processing bank computing system. In *7th International Conference on Distributed Computing Systems*, pages 2–9, Berlin, Germany, September 1987.

[14] Matt W. Mutka and Miron Livny. Profiling workstations' available capacity for remote execution. In *Performance '87: Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*, pages 529–544, Amsterdam, The Netherlands, The Netherlands, 1988. North-Holland Publishing Co.

[15] Matt W. Mutka and Miron Livny. The available capacity of a privately owned workstation environment. *Perform. Eval.*, 12(4):269–284, 1991.

[16] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the v-system. In *SOSP '85: Proceedings of the tenth ACM symposium on Operating systems principles*, pages 2–12, New York, NY, USA, 1985. ACM Press.

[17] Arun Venkataramani, Ravi Kokku, and Mike Dahlin. Tcp nice: a mechanism for background transfers. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 329–343, New York, NY, USA, 2002. ACM Press.

[18] Dan Werthimer, Jeff Cobb, Matt Lebofsky, David Anderson, and Eric Korpela. Seti@homemassively distributed computing for seti. *Comput. Sci. Eng.*, 3(1):78–83, 2001.