Machine Utilization Strategies for Desktop Grids

Wouter Leenards, wls400@cs.vu.nl
June 2008
Department of Computer Science
Vrije Universiteit, Amsterdam, The Netherlands

Abstract

Desktop Grids belong to the largest distributed systems in the world. In a desktop grid, normal desktop machines are used to run compute intensive applications, and contribute to the system as a whole in this way. Since these machines are also used by (local) users and processes, the distributed applications have to run in the background or when a machine is idle. Ideally, normal user activity is not or only barely disturbed while utilizing as many processor cycles as possible. In this paper we present a selection of simple machine utilization strategies for Desktop Grid systems, using the Zorilla P2P grid middleware software as a case study. For this work, we augmented Zorilla with Zolo, the Zorilla Loader, adding a number of machine utilization strategies for both the Microsoft Windows ® and GNU Linux platforms.

1 Introduction

Clusters are relatively cheap, but an even cheaper and easier alternative exists: Desktop Grids. Using already purchased, installed and connected machines in people's homes or corporations. SETI@home[1], FightAIDS@Home[2], climate*prediction*.net[3], and Folding@home[4] are a few examples of large scale desktop grid systems. SETI@home, one of the largest and well known projects, is currently in use by more than 800.000 users, running on almost two million hosts [3]. Users provide their idle CPU cycles to large research laboratories, universities and commercial organizations using specially designed client applications. Some organizations use their own available computer power to get better return of investment for the thousands of desktop machines purchased. Also, some universities and research facilities are using the same Enterprise Desktop Grids by utilizing idle student or employee workstations to do scientific research.

There are also a lot of companies and organizations that are using the idle CPU cycles of many volunteers all over the world. By giving these volunteers an incentive, a good feeling about what they do, the organizations hope many people will participate: “Just because you don't know much about biology or medicine won't stop you from helping to someday cure diseases like malaria, HIV, or cancer. In fact, all you need is a computer and an Internet connection and you can play a pivotal role in the search for treatments and cures for some of the world's biggest killers.” [15].

As computers get faster and cheaper, more computing power is available every day. To utilize the combined power of all the machines connected to a grid, an intelligent and robust grid middleware layer is needed. Most of the traditional grid systems are centralized and therefore not completely fault-tolerant and scalable.

2 Zorilla

Zorilla [6] is a peer-to-peer based supercomputing grid middleware system. Because there are no central components and every node in the system is dispensable, it is easy to set up and resilient against failures. Zorilla is built as part of Ibis [11], an efficient Java-based platform for grid computing[5]. Zorilla is built up around a Peer-to-Peer overlay network, consisting of a number of Zorilla nodes. Together, these nodes can handle any work or job, put in by one or more users. A job can be submitted at every node in the network. This node will then look for the resources required to complete the job by sending a request to other nodes in the network.

1 http://setiathome.berkeley.edu
2 http://fightaidsathome.scripps.edu/
3 http://www.climateprediction.net
4 http://folding.stanford.edu/

5 More information about Ibis, including downloads, can be found at http://projects.gforge.cs.vu.nl/ibis/

Zorilla can be started, paused and stopped at any time by a simple call.

3 Utilization strategies

In this section we will introduce ten simple machine utilization strategies for Zorilla. Some of these strategies are implemented in Zolo, the Zorilla Loader. Zolo is an extension for Zorilla, specially developed for this research, able to start, pause and stop Zorilla, based on the combined output of the different strategies. Zolo will be discussed more thoroughly in section 4.

Since we want to disturb the user as little as possible, the application can only run when the user does not need (all) the resources (memory, CPU). As we can see in Figure 1 and [12], desktop computers and laptops are active or at low power more than half the time. Given the fact that the average computer user uses a computer 2.48 hours per day during a work week [16], an application like Zorilla can utilize a lot of idle time.

Several resource availability strategies and models exist [9], [5], [7], [2], [13], [10] trying to predict, model and manage machine availability, for example the duration that a machine will run until it restarts. These machine availability strategies are beyond the scope of this paper but can be used for Zorilla as well.

The strategies we are going to present only take account of CPU and memory utilization, system state (i.e. number of users logged in, workstation locked) and physical user activity (use of the mouse and keyboard).

First we will describe the strategies used in Zolo and how they are implemented. Second, we describe another set of strategies, not implemented in Zolo, but with an implementation proposal.

3.1 Zolo strategies

The following strategies are implemented in Zolo. Zolo can be configured to run Zorilla when at least one of the strategies satisfies the rules or when all strategies agree to start.

3.1.1 No Users Logged In

When there are no users logged on at a machine, a user can not be *directly* disturbed by other processes like Zorilla using the machine. The *No Users Logged In* strategy is probably a good solution for university computer rooms or companies were employees keep their computers on during the night. On most operating systems it is fairly easy to determine which users are currently logged on to the system. On Linux machines, the `users` command can be used to get a list of all users logged on. In Java, we can use the `psloggedon.exe`[6] program, provided by Microsoft.

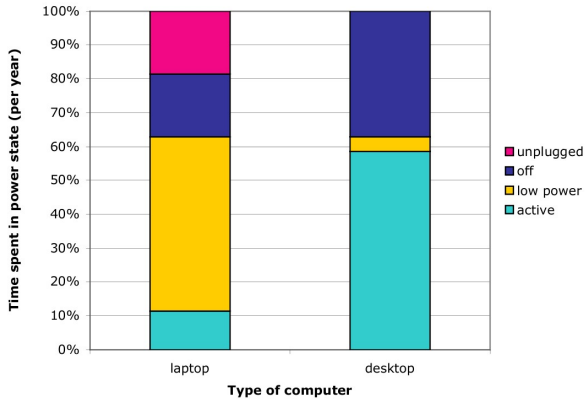

Figure 1: Typical usage pattern of laptop and desktop computers [4]

3.1.2 Physical User Activity

Normally, a desktop computer is equipped with a pointer device (we will refer to it as a *mouse*, of course a trackball, tablet pen or any other input device capable of moving the 'mouse' cursor will do) and a keyboard. By using the keyboard and mouse, the user can command the computer. When the keyboard and mouse are not used for a certain amount of time, we assume the user has become inactive and maybe has even left the computer. This assumption is the base for the *Physical User Activity strategy*. Using the JNA library (discussed later) we are able to gather the Windows idle time. This strategy is only implemented for the Windows platform. The amount of idle time before Zorilla will start can be specified in the configuration file.

3.1.3 Time Interval

A lot of home and office workstations are only used from early morning till late evening, and the latter often only during business hours [17]. While users used to shut down their workstations in the past, nowadays more and more users leave the computer on during the night. Reasons could be:

6 http://technet.microsoft.com/en-us/sysinternals/bb897545.aspx.

- not willing to boot every day
- better hardware reliability
- background processes (downloads/file sharing, automatic updates, television recording, etcetera)
- a running web-, mail-, compute- or fileserver

The TimeInterval strategy will only return a GO when the current time is between time *x* and time *y*. Checking the system time in Java is done with the Calendar library. Multiple time intervals can be provided using the configuration file.

3.2 Strategies not implemented in Zolo

The following strategies are not implemented in Zolo but could be added to Zolo very easily.

3.2.1 Load Threshold

If the machine is constantly in use by one or more users and we would still want the machine to participate in a distributed application like Zorilla, we define a load threshold strategy. If the machine load drops below the defined threshold for a certain amount of time, we can start (or un-pause) the Zorilla node. This will generally be the case when running only low CPU intensive programs like an e-mail client, a browser or a word processor.

Running Zorilla will cause the CPU load to increase. To prevent immediate shutdown of the node because of too high CPU load, we can:

- introduce a new threshold, adding the load generated by the grid node
- only measure the CPU load of all applications except the Zorilla node
- stop measuring the CPU load and instead let another external system, probably a strategy, take care of pausing/stopping Zorilla
- run for a certain amount of time, pause, check the CPU load and resume if the CPU load is still below the threshold
- combine this strategy with the resource limit strategy, discussed later

3.2.2 Screensaver

On various Operating Systems, a screensaver is installed and activated by default to prevent CRT displays from burn-ins. LCD computer monitors and laptop screens are not subject to burn-in because the image is not directly produced by phosphors. Nevertheless, screensavers are still largely in use.

When the user becomes inactive, the screensaver starts after a user defined amount of time. In some Desktop Grid Systems, like SETI@home [1], [14], a special screensaver is manufactured to display information about the data analysis progress[7]. While the screensaver is active (the workstation is idle or the user started the SETI application by hand), the distributed application will start.

For Zorilla the screensaver strategy can be deployed in two ways: we could create a Zorilla specific screensaver, optionally showing some kind of progress information, graphs or other images like SETI@home uses. We could also *detect* when the screensaver is active and run Zorilla accordingly. Polling the operating system at a set interval for screensaver activity is probably the best implementation for this strategy.

3.2.3 Workstation Locked

A coffee break, a call of nature, a staff meeting, there are a lot of occasions in which the user leaves his workstation for a while or does something else. When at home, a user normally does not have to lock his workstation, but when at the office or at a public location, it is considered wise to lock your workstation. When a computer is locked, processes can continue, but since the user (for simplicity we only take into account a single user system) is locked out, he cannot experience any hinder from a system being less responsive ('slow').

This strategy could be implemented using the JDIC Incubator Project [8].

3.2.4 Manual Start

This is the only strategy in which the user is in full control of when he runs the Zorilla node. The user has to manually start Zorilla. This strategy is added for completeness.

3.2.5 Resource Limit

On heavily used or real-time systems, or when a users wants to run the Zorilla application *in the background*, the strategy of resource limitation can be used. When the node is started, it is dynamically assigned a resource limit. In Unix-like systems, the *nice* program can be used to run a program with modified scheduling priority. The nice

7 http://setiathome.berkeley.edu/sah_graphics.php

level can be set in the configuration of the strategy.

3.2.6 Power Suspend/Hibernate

A lot of computers support a hibernation or suspend mode. When a machine is in this mode, for example when then laptop screen is 'closed', no or only a few programs are running and the system is not in use. Unfortunately, most machines switch to power savings mode, downscaling the CPU speed and probably halt the hard drives when in hibernation mode. This will prevent Zorilla from running fast and maybe from running at all. It is not an ideal strategy, but can be used for non-CPU intensive programs, like gathering system information or monitoring software.

3.2.7 Always On

The 'always on' strategy does not take the other system user(s) into account. When Zorilla uses this strategy, it will simply run when the host boots and stop when the system goes down. In Linux systems, we can implement this strategy using a daemon process. When running Windows, a *service* can be used. This strategy is certainly the best for dedicated Zorilla processing and worst for the user, because the system is completely controlled by Zorilla.

4 Zolo

In this section, we introduce Zolo, the Zorilla Loader. Zolo is designed to start and stop Zorilla using a set of utilization strategies. Zolo is an extension for the Zorilla Starter[8] application. Zorilla Starter is a simple application written in Java to start, download and run Zorilla. Zolo allows the user to choose one or more machine utilization strategies from a predefined set of available strategies. Together with the Zolo configuration, these strategies are used to determine when Zorilla is allowed to run. Zolo takes care of starting Zorilla using the Zorilla Starter and the *Zet* program. Like Zorilla, Zolo is written in Java. By request of Zorilla's author Niels Drost we mainly focus on the Windows operating system, due to the fact that Zorilla will mainly be used on Windows machines. Therefore, we developed most utilization strategies with Windows in mind. Some Windows strategies are OS independent and work on Linux too. Zolo uses the following libraries:

- *JNA (Java Native Access)*[9]: provides easy access to native shared libraries (DLLs on Windows) without writing anything but Java code.
- *Apache Commons CLI*[10]: provides an API for processing command line interfaces.
- *Apache Commons Configuration*[11]: provides a generic configuration interface which enables a Java application to read configuration data from a variety of sources.

Because some strategies only work on a certain platform, Zolo has to check the OS it is running on, using `System.getProperty("os.name")`. Zolo is tested on Ubuntu Linux 8.04 and Microsoft Windows XP ®.

4.1 Native access

Due to Java security measurements, the existence of a Virtual Machine and portability, Java is not capable of interacting directly with the underlying operating system. Instead, we use JNA to call native OS functions. In the *No User Activity* strategy, we use JNA to get the Windows idle time. We prefer using JNA instead of JNI (Java Native Interface)[12], because it is much easier to implement calls to the native operating system using JNA. We use JNI to check if the Windows user session is locked.

4.2 Zolo design

In this section we will give an overview of the Zolo design choices made. The main class, Zolo.java, handles command line options, creates an instance of the Strategy Manager and determines the strategies that have to be loaded based on the configuration file. When verbose mode is enabled (through the `--verbose` command line option), Zolo will give extensive output on what it is currently doing. By supplying a configuration file with the command line option `-v`, a user-defined configuration file is loaded instead of the default `zolo-config.xml` file. The Output object handles the (exception) output.

8 Zorilla (including Zorilla Starter can be downloaded from http://projects.gforge.cs.vu.nl/ibis/zorilla.html)

9 Java Native Access. https://jna.dev.java.net/

10 Apache Commons Command Line Interface library. http://commons.apache.org/cli/

11 Apache Commons Configuration library. http://commons.apache.org/configuration/

12 Java Native Access interface. http://java.sun.com/j2se/1.4.2/docs/guide/jni/

4.2.1 Strategy Manager

The Strategy Manager is the heart of Zolo. It loads and starts strategies and keeps track of the strategy states. Every running strategy can change his state by calling the `reportState()` method of the Strategy Manager. When a strategy reports a state, the Strategy Manager determines if Zorilla should start, resume or pause, based on the `strategy-check-operator` configuration file option, the current states and the reported state. The manager is capable of starting, pausing and stopping Zorilla by using the Zorilla Starter and *Zet* program. Zorilla will be stopped when Zolo stops.

4.2.2 Strategies

Zolo provides a StrategyInterface and a Strategy class to make it very easy to add new strategies to Zolo. A strategy can have three states, defined in the StrategyStates enumeration: `GO` (conditions are met), `NO_GO` (conditions are not met) and `ERROR` (something went wrong, i.e. the strategy is not capable of checking the conditions). When a strategy is started, the state will be set to `NO_GO` and optionally, a scheduler can be started to invoke timed checks on strategy conditions. Strategies are responsible of reporting to the Strategy Manager.

4.2.3 Configuration

The default configuration file, `zolo-config.xml` is an XML file containing the global program settings as well as the strategies and their options. The `strategy-check-operator` can be `AND` or `OR`. When it is set to `AND`, all the strategies have to be in `GO` mode for Zorilla to start. If it is set to `OR`, at least one of the strategies must be in `GO` mode. In order to add a strategy to Zolo, create a file *`StrategyName`*`Strategy.java` and add an `strategy` tag to the strategies node in the configuration XML. The `name`, `class` and `active` tags are obligatory in this tag. An example configuration file is shown in listing 1, containing two Windows strategies.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<zolo-config>
  <settings>
    <strategy-check-operator>OR</strategy-check-
operator>
  </settings>
  <strategies>
    <windows>
      <strategy>
        <name>NoUsersLoggedIn</name>
        <class>NoUsersLoggedInStrategy</class>
        <timer>10</timer>
        <active>true</active>
      </strategy>
      <strategy>
        <name>NoUserActivity</name>
        <class>NoUserActivityStrategy</class>
        <active>true</active>
      </strategy>
    </windows>
  </strategies>
</zolo-config>
```

Listing 1: Sample Zolo configuration

4.2.4 Daemon

Ideally, we want the Zolo program to run when a machine is active. With a daemon (called *service* in Windows) this can be done. The Java Service Wrapper[13] allows any Java program to be installed and controlled like a native Windows service or Linux daemon. The service wrapper comes with an extensive set of options, allowing full control over the daemon process. The service wrapper is implemented in Zolo and can easily be installed and uninstalled with a batch file. There is one drawback: for both Windows and Linux separate wrappers are available, so we have to ship them both with Zolo or provide two separate versions.

5 Summary and Future Work

In this paper we presented a number of machine utilization strategies that can be put to use for Zorilla, a P2P middleware grid system. Some of those strategies are implemented in Zolo, the Zorilla Loader application specially built for this research. We have implemented a set of strategies for the Microsoft Windows platform and some for the GNU Linux platform.

The strategies presented are only a subset of all possible strategies. Adding strategies to Zolo is fairly easy. The only thing a strategy has to do is report to the Strategy Manager with a simple method call. An interface and a base class for the new strategy are available within Zolo. Strategies can be easily switched on and off using the XML configuration file. All implemented strategies are

13 Java Service Wrapper by Tanuki Software. http://wrapper.tanukisoftware.org/

kept very simplistic and take only the current state of the host machine into account. The complexity of strategies can be increased by adding availability prediction, advanced CPU load statistics, and better pause strategies. Zolo itself could also be expanded with better strategy condition selection, for example more options than only AND and OR.

Acknowledgments

I wish to thank Niels Drost for his helpful comments and suggestions and for the modifications to Zorilla, allowing us to pause and resume the node.

References

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public- resource computing. In *Communications of the ACM*, voume. 45, number 11, pages 56-61, November 2002 .

[2] A. Andrzejak, P. Domingues, and L. Silva. Classifier-Based Capacity Prediction for Desktop Grids. Workshop of Integrated Research in Grid Computing, Pisa, Italy, November 2005.

[3] Berkeley Open Infrastructure for Network Computing (BOINC) project statistics. http://boincstats.com/.

[4] M. Bray. Review of Computer Energy Consumption and Potential Savings (white paper). Dragon Systems Software Limited, Hereford, United Kingdom, 2006.

[5] J. Brevik, D. Nurmi, R. Wolski. Quantifying Machine Availability in Networked and Desktop Grid Systems. University of California, Santa Barbara, Computer Science, UCSB Computer Science Technical Report Number CS-2003-37, Nov. 2003.

[6] N. Drost, R. van Nieuwpoort and H. Bal. Simple Locality-Aware Co-Allocation in Peer-to-Peer Supercomputing. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 14, 2006.

[7] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is Not Sloth. In *Proceedings of the USENIX Conference*, New Orleans, Los Angeles, USA, pages 201-212, January 1995.

[8] JDIC Incubator Project, Systeminfo package. https://jdic.dev.java.net/incubator/systeminfo.

[9] D. Kondo, M. Taufer, C. Brooks, H. Casanova and A. Chien. Characterizing and Evaluating Desktop Grids: An Emperical Study. In *proceedings of the 18th International Parallel and Distributed Processing Symposium*, page 26, April 2004.

[10] D. Kondo, G. Fedak, F. Cappello, A.A. Chien, H. Casanova. Resource Availability in Enterprise Desktop Grids. In *Journal of Future Generation Computer Systems,* issue 23(7), pages 888-903, 2007.

[11] R. Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. Bal. Ibis: an efficient Java-based grid programming environment. In *Joint ACM Java Grande - ISCOPE 2002 Conference*, Seattle, Washington, USA, pages 18-27, November 2002.

[12] K. Roth, F. Goldstein and J. Kleinman. Energy Consumption by Office and Telecommunications Equipment in Commercial Buildings: Volume I. Arthur D. Little, Inc., reference 72895-00, Report for Office of Building Equipment, Massachusetts, 2002.

[13] K. Ryu. Exploiting Idle Cycles in Networks of Workstations. PhD thesis, 2001.

[14] The SETI@home project. http://setiathome.berkeley.edu/.

[15] ScienceDaily. Donate Your Unused Computing Power To Aid Medical Research. http://www.sciencedaily.com/releases/2006/04/060417131952.htm, April 17, 2006

[16] K. Taylor. An Analysis of Computer Use Across 95 Organisations in Europe, North America and Australasia. White Paper, Reference: 2.03-8/2007, Wellnomics Limited, August 2007.

[17] C. Webber, J. Roberson, M. McWhinney, R. Brown, M. Pinckard and J. Busch . After-hours Power Status of Office Equipment in the USA. In Energy (the International Journal), volume 31, issue 14, November 2006.