EXPANDING THE OVERALL EFFICIENCY OF TOMORROW'S DATA CENTERS THROUGH PROCESSOR OPTIMIZATION

William Putnam
Computer Science/Computer Engineering

ABSTRACT: In order to solve the impending issues of data center inefficiencies, micro-level hardware-based and software-based approaches were studied. A power model based on various processors was observed and calculated in order to determine which type of processor works best given the type of process and the number of instances it must be carried out. In addition, an application that can implement these models over an intranet of data servers was developed. The application allows for toggling of various processors and their cores.

INTRODUCTION

As consumers start to utilize cloud-based services, the data centers that house them can be very inefficient. On average, data centers worldwide consume an average of 30 nuclear power plants' worth of power, as well as a supplement of round-the-clock diesel generators, in the hopes of preventing any outage for any period of time. At an average of 10% processor utilization per server, the wasted power conflicts with the desire to shift to environmentally-friendly trends.

While it's easy to suggest turning off servers to save power, the need to keep the servers running at maximum capacity is greater than the amount of power used to run them. This is because of the possibility of a distributed-denial-of-service, or DDoS. When too many clients try to access the server network at once, the servers must scramble to accommodate all requests. Due to the large traffic, these servers may not be able to respond to all the requests before a request time-out. When too many of these time-outs occur, the server network experiences DDoS. The recovery time can take anywhere from minutes to days, and any important data that has been transferred during this time may be lost.

Two of the major players in the recent rise of data centers are the financial industry and the social media industry. In the financial industry, financial transactions can occur at the microsecond, and an estimated 86% of all transactions today are computerized. There are financial networks all over the world, as well as communications between cities and even continents. In the social media industry, the importance of networking sites such as Facebook, Twitter, and YouTube have exerted influence in current world events. When a certain event occurs, such as a revolution or a mass public event, the amount of traffic about that event may spike in a short time. To these two industries, a DDoS is not an option, as it can mean money lost or business jeopardized. Therefore, these industries and others will gladly pay a hefty premium to make sure that such an event never happens.

And even these safeguards are not enough. In 2010, Amazon's data centers in Virginia were drawing so much power off the grid that they caused a massive power outage, both at the center and in the regional area. Not even the mindset of keeping everything at maximum capacity is working out for the industry.

And turning off some of the servers results in downtime, as they need to be powered down and powered up once more. It takes, on average, one to two minutes to get a data server up and running properly, even with automated methods. In times of heavy traffic, this gap in downtime is all a potential DDoS needs to occur.

ASSESSING A POSSIBLE APPROACH

There are already a few limited implementations to reduce the possibility of a DDoS and save power. An especially relevant example is here at the University of New Haven, where the servers in Echlin Hall are used for virtualization on the network for the thousands of students and faculty accessing the network at university-owned computers. If there is a server with a heavy load and a server with little to no load, some of the load is distributed from the former to the latter. This is a fantastic macro-based implementation that focuses on the network as a whole, both saving power and preventing DDoS.

However, there are no real micro-based implementations in practice that focus on each individual server. While macro-based implementations are useful, they tend to see all the systems on a network as ideal and uniform in spec. This is usually not the case in some networks, where there may be different types of hardware or software. Therefore, this project will focus on the micro-based implementation over the macro-based.

Think of the computer as a mammal. The two major components of the mammal are the brain and the body. The brain controls the body, and the body gives its services to the brain. Without one, there is no point in having the other. The same can be said for computers, only instead of the body, there is the hardware, and instead of the brain, there is the software. When designing an implementation to a computer, both of these components must be taken into consideration. In terms of this project, the processor is the primary focus for the hardware, and an application of some sort is the primary focus for the software.

CALCULATION OF A POWER MODEL

The processor's name essentially describes its role; it processes various values and carries out operations on them. These values are then stored on the computer and/or displayed to the user. Up until the late 2000s, a processor had only one core, which meant that only one value could be processed at a time, albeit within a small fraction of a second. Hyper-threading technology, mainly utilized by Intel, provided a duplicate of key hardware components to simulate two processes at once, but was not widely

implemented. In recent years, however, processors with multiple cores can be acquired at reasonable cost. These processors allow multiple processes to be carried out at once on each of its cores. A processor can have two, four, eight, or even up to 64 cores, which allows for greater processing abilities without necessarily having the same number of processors with only one core.

To determine the amount of power that a processor uses, we must consider the base, or idle, level of power required to make the chip itself operational, plus the amount of power needed to carry out the process multiplied by the time required to carry it out, multiplied by the number of instances that a processor needs to carry out the task. The equation for the power needed can be represented as in Equation 1.

$$P_T = (P_{idle} + t_{test} * P_{test} * n) + t_{test} * n$$

Equation 1: The equation used for the power models.

Overall, there were four different systems with individual processors tested to find suitable power models. They are as follows:

- BeagleBoard XM, 1 GHz ARM® Cortex™-A8 processor
- Dell™ Latitude E6500, 2.80 GHz Intel® Core 2 Duo™ T9600 processor
- Dell™ Optiplex 780, 2.66 GHz Intel® Core 2 Quad™ Q9400 processor
- Asus® Maximum IV Extreme Z, 3.40 GHz Intel® Core™ I7-2600 Quad-Core processor

The processors were tested in eight different benchmarking categories. All tests were conducted using a National Instruments™ myDAQ and Labview data acquisition software for the Beagleboard, and a Watts Up? Pro plug load meter with Microsoft® Joulemeter software for the other models. The BeagleBoard ran on a Linux-based distribution known as Ångström, and the other machines ran on the Windows® 7 operating system. Most of the source code used was included in the Phoronix Test Suite, which is available for both Windows® and Linux machines. The tests were as follows:

- Idle
  - No additional loads are placed on the processor.
- Batch processing
  - The resources needed for processing are loaded first, and then the actual processing takes place.
  - A custom C program doing various calculations is used for testing. It includes all four forms of arithmetic, plus modulus operations, if statements, for loops, and functions separately from the main function.
- Interactive processing
  - The processor waits for user input/output, such as from a keyboard to a text editor.
- Realtime processing
  - A multimedia file (audio, video, etc.) is played.
- Server benchmark
  - The server carries out standard testing protocols for web-based hosting via the Apache client.
- System benchmark
  - All machines spend a few minutes running Nexuiz, a fullscreen video game, which utilizes many different components of the computer.
- Processor benchmark
  - The Beagleboard compresses files using Phoronix's 7-zip utility multiple times.
  - For all other machines, the Primesieve application's algorithm is computed with a large set of numbers as input.
- Blowfish encryption
  - Files are encrypted and decrypted a certain large number of times.
  - The source code was obtained by MiBench, which is hosted by the Electrical Engineering and Computer Science department at the University of Michigan at Ann Arbor.

Power data was collected similarly for all processors except for the BeagleBoard. The Joulemeter software would calibrate with the Watts Up? hardware to ensure accurate readings of power consumption. It would then calculate the power consumed over various instances of time for all computer components and display them in a .csv file. Only the processor's power consumption was considered for data collection.

The BeagleBoard could not be tested in this manner because Joulemeter is only available for Windows® operating systems. Therefore, the power needed to be collected using the myDAQ. Two wires were connected to a pair of prongs on the board that measured the current of the processor and the voltage of the board. Since the resistance of the prongs was given, we were able to determine the power consumed over a short period of time in Labview, as shown in Figure 1, using the equation $P(t) = I(t)^2 R$.

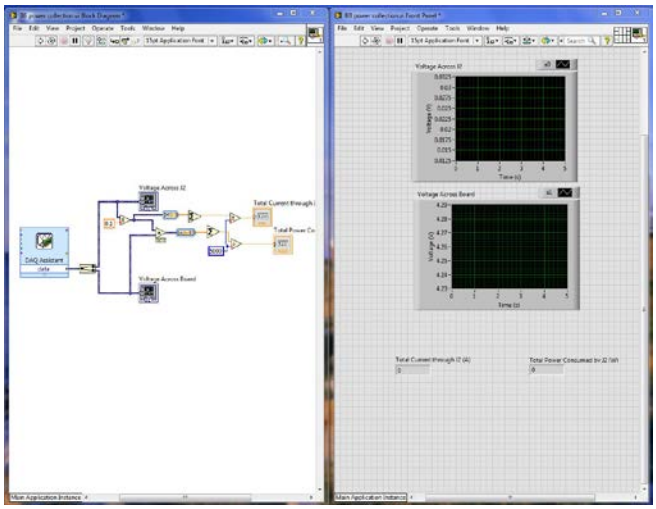


Figure 1: The Labview application used for current to power conversion for the Beagleboard.

Figures 2 (batch processing) and 3 (Blowfish encryption) show examples of the data collected and plotted to determine the power usage per processor. The data is graphed with the number of instances to run the program on the x-axis and the wattage consumed by the processor on the y-axis. (The respective colors are blue, red, green, and purple.)

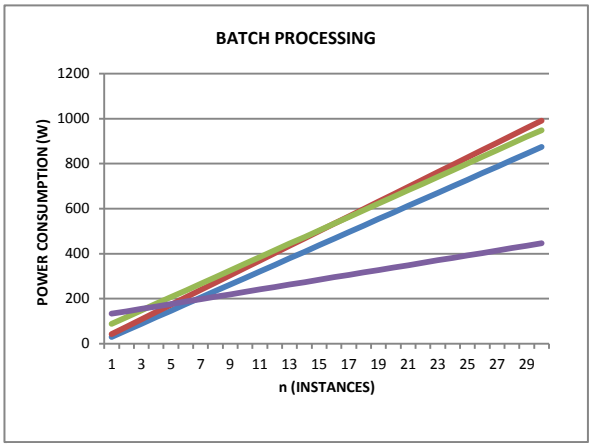


Figure 2: The data collected for batch processing.

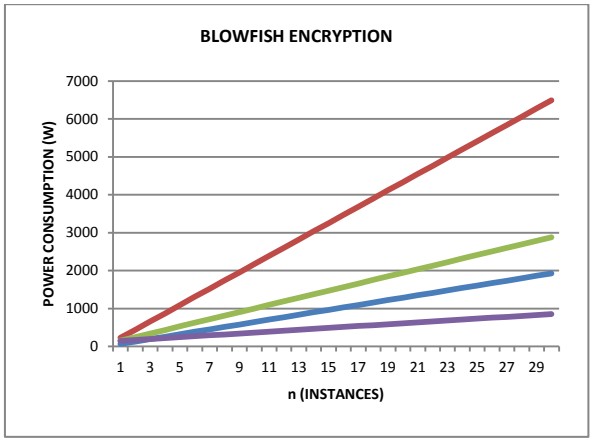


Figure 3: The data collected for Blowfish encryption.

As observed in Figure 2, there are some points in which there is a tradeoff in power consumption. For three instances of the batch processing, the ARM® processor would be the most power-efficient processor, but for 30, the I7 processor would be the most power efficient. In some cases such as Figure 3, however, there were no noticeable tradeoffs. It should be noted that these figures are not inclusive of:

a) all possible processors on the market
b) the base power required by the system housing the processor to make the system operational

DEVELOPMENT OF A CONTROLLER APPLICATION

In data servers today, it is expected to have multiple processors in one machine to allow for extra processing power. An average server today can have four processors, with about eight cores on each chip. In some cases, it's not necessary to have extra cores or processors running. Therefore, a method of controlling these processors and cores on each system is required.

Pandect, meaning comprehensive overview, is the application that controls the processors on the network. It was developed in the C++ programming language paired with Win32 API headers. The application was developed in the Windows® operating system for versions XP and newer, both due to its availability in the workplace and the fact that each type of operating system communicates with its processor differently. In that regard, the time constraints for this project prevented development on multiple operating systems, whose methods of accessing the processor vary by the operating system used.

Pandect monitors the status of all processor and cores of each computer on an intranet, or local network, and toggles which cores and/or processors should be used for a program. There are three levels to the breakdown of the network: a machine class for each server on the network, a processor class for each processor on that system, and a core class for each core on the processor. The processor class derives the core class through a concept in computer science known as polymorphism.

In the program window, an internal window is dynamically created for each different machine on the network, and a window within that window is created for each different processor on the system. Each instance of a core is stored in a vector-type variable in the processor class, and each instance of a processor is stored in a vector-type variable in the machine class.

Pandect's controls are quite straightforward. Toggling a core "off" moves all processes off it to other cores, while toggling it "on" allows for assignment to them. Toggling a processor off and on works in the same manner. However, a machine cannot turn off all the processors of a machine, as at least one is needed for its operation. This is the same manner with the number of cores on the system.

It should be noted at this point that it is not possible to fully "turn off" a processor or core, as there is always a

small amount of current keeping the processor ready for future possible loads. In that regard, it is possible to redirect the processes of an application to other cores and/or processors. Thanks to modern processor technology known as power gating, a processor's core is considered "disabled" if there are no processes on it for a certain amount of time. The additional power needed to otherwise run the core is therefore redirected to other cores or conserved altogether.

The application is also designed to include ceilings for core and processor usage. There is also an automatic mode for off-peak periods of time, and a manual mode for an administrator to control the limits. Finally, as a safety precaution to the network, there is an option for an emergency override that automatically brings the units to full power in case of a sudden spike in traffic to prevent a denial of service error. The utilization of these features is to be determined by the systems administrator in charge of the network.

ACTIVITIES PENDING

As of publication of this paper, the development of Pandect is still in progress. The application is currently experiencing initialization problems. The code structure, while being maintained as best as possible, requires significant reformatting for distribution and error-checking purposes. In addition to the application's incompleteness, the data for the power model has yet to be implemented into an appropriate manner to be read into the program. Further development of this application will continue in the school year, and its expansion may be considered as a separate project in the years to come.

In addition to the aforementioned features, plans are being drafted to expand the application to the OS X and Linux-based environments. This expansion will lead to better compatibility with data centers of many different interfaces. Also, terminal-based command issues are a possibility for future versions of Pandect. In environments where a GUI is unsuitable, Pandect can still be used provided that the administrator enters a command that specifies what mode the application should be in, what computers should be affected, what the thresholds are, etc.

Due to the incomplete nature of this project, it is not possible to draw a conclusion on whether or not the venture is a success. Only time will tell whether or not Pandect can truly live up to its name as an overview manager and controller of data center processing.

ACKNOWLEDGEMENT

The author wishes to thank the following persons and groups:

- Prof. Christopher Martinez, Ph.D., for his assistance with all hardware-related support
- Prof. Alice E. Fischer, Ph.D., for her assistance with all software-related support
- Prof. David W. Eggert, Ph.D., for his assistance with initial debugging of testing software
- Mrs. Janice Sanderson, Mrs. Carol Withers, and the personnel contributing to the SURF program, for their kindness and generosity for providing and administering a fantastic work environment

REFERENCES

1. Apache HTTP Server Project. Computer software. The Apache HTTP Server Project. Vers. 2.0.xx. The Apache Software Foundation, June 2013. Web. June-July 2013. <http://httpd.apache.org/>.

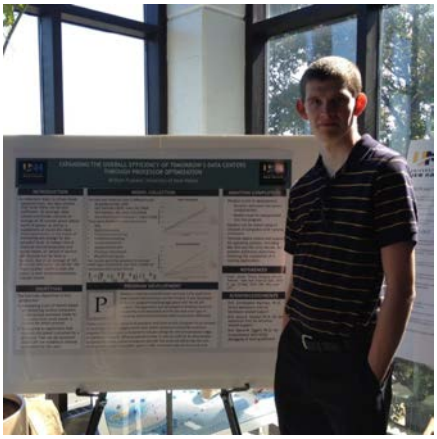
2. Glanz, James. "Power, Pollution and the Internet." New York Times 22 Sept. 2012: n. page 22 Sept. 2012. Web. May-June 2013.

3. Guthaus, Matthew, Jeff Ringenberg, Todd Austin, Trevor Mudge, and Richard Brown. Software.tar.gz. MiBench Version 1. Vers. 1.0. University of Michigan at Ann Arbor, 2002. Web. June-July 2013. <http://www.eecs.umich.edu/mibench/>.

4. Microsoft Research. Joulemeter. Computer software. Joulemeter - Microsoft Research. Vers. 1.2. Microsoft Research, 29 Sept. 2011. Web. June-July 2013. <http://research.microsoft.com/en-us/downloads/fe9e10c5-5c5b-450c-a674-daf55565f794/>.

5. Phoronix Test Suite. Computer software. Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing Framework, Open-Source Benchmarking. Vers. 4.6. Phoronix Media, 2008. Web. June-July 2013. <http:// http://www.phoronix-test-suite.com/>.

BIOGRAPHY



William Putnam, of Old Saybrook, CT, is a junior at the University of New Haven. He is pursuing a bachelor's degree in both computer science and computer engineering. His hobbies include gaming and playing the saxophone. He wishes to go for his master's degree in Japan, and hopes that one day he may start his own video game company.