Temperature-Aware Operating System Scheduling

A Thesis
in STS 402

by

Eugene Otto

Mar. 31, 2006

Technical Advisor: Kevin Skadron
STS Advisor: Professor Helen Benet-Goodman

____

LIST OF FIGURES

ABSTRACT

The goal of my research was to implement and test an operating system scheduler modified to selectively throttle the processor clock frequency depending on the active task. Throttling refers to lowering the processor clock speed and is generally invoked to reduce processor temperature. This process, however, inherently reduces the performance of the system which can create dangerous conditions in some cases and is generally undesirable.

My work reduces the penalty caused by throttling by allowing the user to specify tasks for which throttling is not invoked. When throttling is invoked and a user specifies that a task should not be throttled, the system runs that task at full speed while throttling all of the other non-critical tasks. This arrangement allows the system to cool down while still providing high responsiveness for critical tasks.

I tested my system by playing DVD video files. This test was chosen because it relates to an everyday situation and would cause the system temperature to elevate significantly, resulting in unwanted fan noise. My goal was to demonstrate that my modifications to the scheduler allow DVD playback at lower temperatures with less fan noise.

My work was successful: I demonstrated that my modifications would allow a task to perform well while the rest of system was throttled. This paves the way for research into more sophisticated temperature-aware scheduling algorithms that can be applied to a wider swath of workloads.

TABLE OF CONTENTS

CHAPTER 1: THE CASE FOR SELECTIVE THERMAL THROTTLING

Modern computer processors are extremely powerful tools, but this functionality comes at a price: they operate at temperatures almost high enough to cause themselves irreparable damage. Of the various cooling techniques, clock throttling (lowering processor clock speed to reduce heat generation) has seen the most development in recent years. Unfortunately, when throttling is invoked, it affects every active task indiscriminately, impairing tasks that require high responsiveness such as an application that plays DVD video. I have modified the throttling cooling procedure in Linux to be selective so that high-priority tasks can run at their normal clock speeds while the processor's temperature is still safely regulated.

THERMAL THROTTLING

Processor cooling techniques are grouped into the two general categories: active and passive. Active cooling techniques, such as fans, use power and generally cause noise. Passive cooling techniques, such as clock throttling, do not use power and operate silently. The current trend in mobile computing is toward silent operation, which means that passive cooling techniques are becoming more important.

Recent Intel processors, like the Pentium M, automatically shut off when the temperature reaches a point beyond which it might be damaged. Although this saves the chip from burning out, it can result in data loss, corruption, and inconvenience to the user. Thermal throttling exists as a middle ground between this jarring shutoff and normal operation. Because processor activity is directly correlated with processor temperature, reducing the amount of processor activity will cause the temperature to fall. As shown in Figure 1, when the processor's temperature reaches the throttling trip-point, throttling is invoked to reduce the temperature. Once it drops, throttling is disengaged which allows the processor to operate again at full speed.

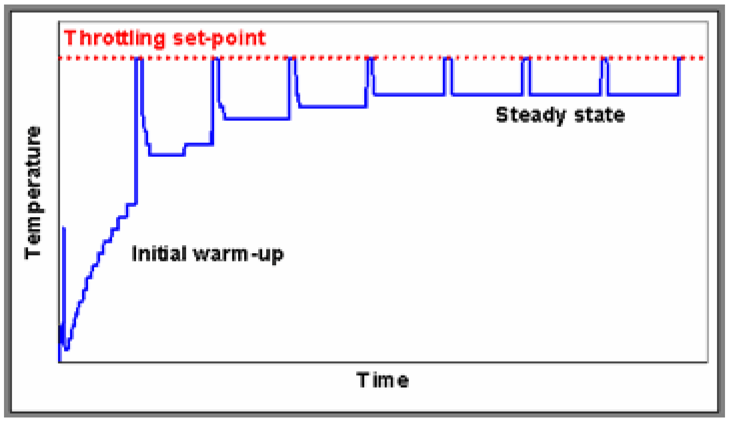

Figure 1 –The Effect of Processor Throttling – Processor throttling can be used to maintain a safe, steady processor temperature level. Image taken from [11].

Clock throttling, illustrated in Figure 2, works because it decreases the amount of activity occurring on the processor. Computer code is broken down from a human-readable language like C++ into the smaller instructions of a language called assembly language. A varying number of assembly instructions are executed each clock cycle. By reducing the number of clock cycles occurring during a given amount of time, the number of instructions that are executed are likewise reduced, which lowers the amount of activity on the processor and hence the temperature, as well. This, however, impacts performance – running fewer instructions means tasks take longer to execute.

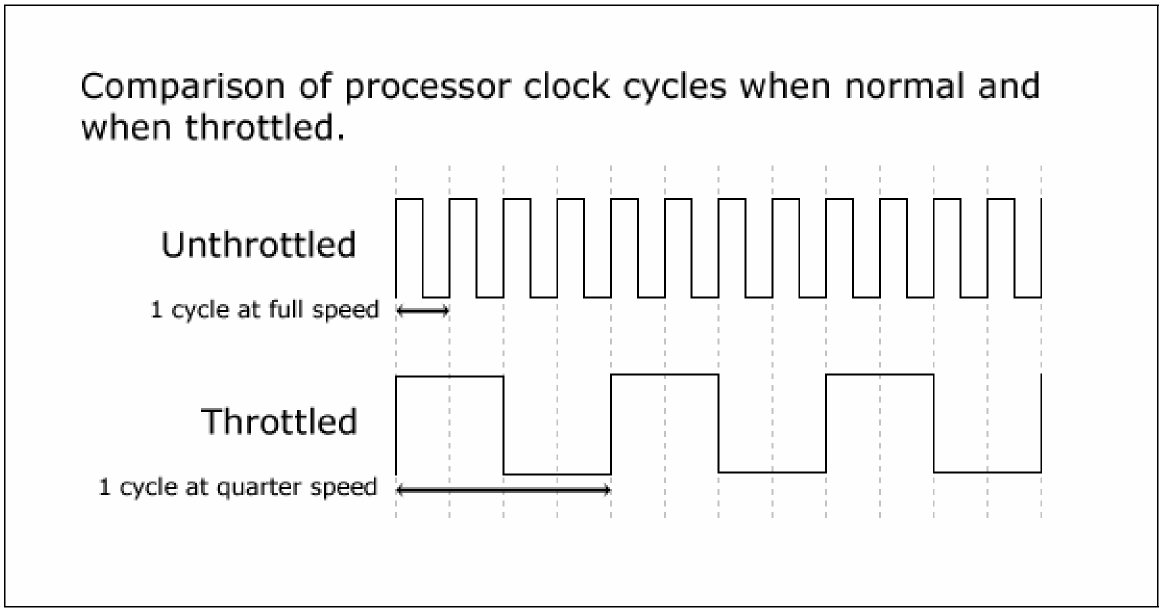
Comparison of processor clock cycles when normal and when throttled.
Unthrottled
1 cycle at full speed
Throttled
1 cycle at quarter speed

Figure 2 – Comparison of Processor Clock Cycles, Throttled and Unthrottled -- Computer code is broken down into small assembly instructions. Notice that in the bottom throttled timeline, only one fourth as many clock cycles occur, thus the number of instructions executed is cut down 75%. So, for example, when unthrottled, the processor might be operating at 1GHz, whereas when throttled as shown in the figure, it would be operating at 250MHz. Note that for ease of presentation, I am assuming that exactly one instruction executes per cycle. Image adapted from Intel spec sheet [10].

SCHEDULING AND MULTITASKING EXPLAINED

Modern operating systems, such as Windows and Linux, have a scheduling component that manages task execution. It is common, for instance, to be listening to a CD, writing a paper, and browsing the web all at the same time. Although it is physically impossible to run more than one task simultaneously on a single processor (except for the most advanced), the illusion of multitasking is created by maintaining a list of active tasks, executing the task at the top of the list for a very brief amount of time (30ms, for example), sending it to the back of the list, and then repeating the procedure for the next task. As new tasks start and active tasks finish, they are added and removed from the active task list. This continues for as long as the computer is running [12]. In Figure 3, I illustrate the process of multitasking. Notice that although the three applications, MPlayer, AbiWord, and Firefox are never really running at the same time, they execute at such small increments that to the user it feels as if they are running concurrently. High priority tasks that require a large amount of processing power, such as multimedia applications, are allotted longer timeslices and low-priority tasks, like a word processor, are allotted shorter timeslices [1].

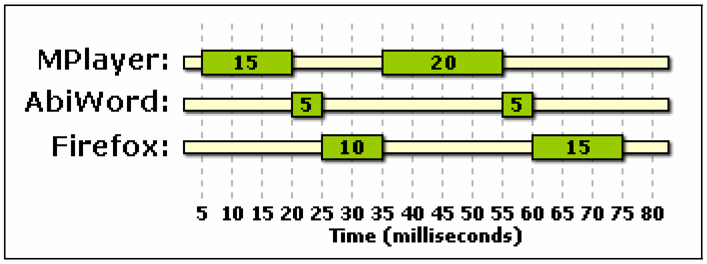

Figure 3 – Multitasking -- The processor runs three tasks (MPlayer, AbiWord, and Firefox) one at a time, but the amount of time allotted to each task (the task's timeslice) is so small that the illusion of simultaneous execution is created. Notice that timeslices can vary in length. (Source: Author)

The problem with the way that throttling is currently implemented is that the priority of the active tasks is never considered. For example, if the processor were throttled down to 50% of full speed from time 5ms to time 75ms in the scenario shown above in Figure 3, it would affect the three tasks equally, with no regard for MPlayer's high priority.

RESEARCH GOAL

I have modified the Linux operating system's scheduler so that high-priority tasks are executed at full speed and low-priority tasks are executed at lower-than-normal speeds to compensate. Figure 4 illustrates this achievement. Notice that although throttling has been invoked, MPlayer still operates at full speed while Firefox operates at about half speed and AbiWord is barely given any clock speed at all.

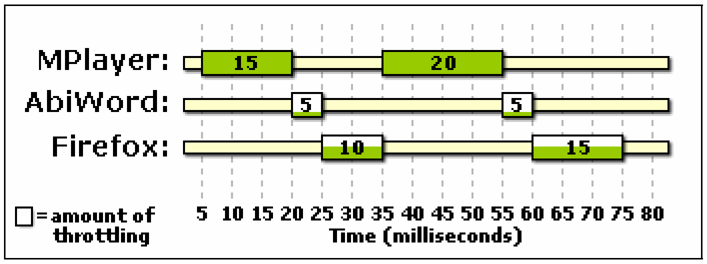

Figure 4 – Selective Throttling -- My changes to the Linux operating system make it possible for tasks to be throttled based on their responsiveness requirements: MPlayer, a video decoder, requires a large amount of processor power, so it is not throttled at all; however, AbiWord needs very little processor power, so it is throttled almost the entire way. Firefox ends up in-between. (Source: Author)

CHAPTER 2: FOUNDATIONS OF TEMPERATURE-AWARE SCHEDULING

This section describes previous research in reducing thermal/power density and in temperature-aware system scheduling.

THERMAL RESEARCH

As previously mentioned, modern processors can run at temperatures high enough to cause system instability, and even catastrophic failure, if left unchecked. Finding ways to deal with this problem at the software and hardware level has been an intense research subject. In *Reducing Power through Activity Migration* [9], Heo, Barr, and Asanović describe a thermal model to help exploit spatial granularity in chips by shifting computations to other areas of the die when components overheat. Powell, Gomaa, and Vijaykumar describe a similar technique in their paper *Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System* [13]. These papers describe methods of software-based processor heat management that parallel my research in temperature-aware operating system scheduling.

In their paper *Potential Thermal Security Risks* [7], Dadvar and Skadron describe the possibility of a malicious program causing processor damage or at the least, user inconvenience, by making the processor overheat and thus throttle down its speed for protection. This subtly underlines the fact that because processor temperature is highly correlated with the programs being run, it should be possible to reduce processor temperature by altering the manners in which these programs are run -- this fact is the underlying principle behind my work.

TEMPERATURE-AWARE SYSTEM SCHEDULING

In *OS-Directed Throttling of Processor Activity for Dynamic Power Management* [4], Bellosa describes a technique for improved temperature management by modifying the operating system scheduler to change the processor frequency based on certain characteristics of a program. This paper reflects my goals closely but it seems to be the extent of his research on this specific topic. I have made the necessary modifications to the Linux kernel to allow selective throttling and have implemented a simple algorithm to throttle and dethrottle processor frequency depending on the active task. In *Power Aware Operating Systems: Task-Specific CPU Throttling* [8], an unpublished project, Sunny Gleason, a University of Pittsburgh graduate student, describes a method of modifying the Linux scheduler to throttle tasks to different levels. Although my investigations began long before I stumbled across this paper, my research will in effect build directly on his work. The similarities between his work and my own give me confidence that I made the correct design decisions in my project. The difference between my research and his is that although I implement a similar means of incorporating throttling into the Linux scheduler, I have benchmarked my changes and shown this technique to be effective, a necessary step before additional research can take place to build on this idea. Gleason's paper describes the first step in attacking the problem, a step I have moved beyond with my own work.

In *Dynamic Thermal Management for Distributed Systems* [15], Weissel and Bellosa describe a way for datacenters (facilities that operate large numbers of servers), to balance the heat generated by their servers. Datacenters are the focus of a lot of thermal research because of the large amount of money spent on cooling solutions in these environments. This topic is further described in *Schedulability Analysis and Utilization Bounds for Highly Scalable Real-Time Services* [2], *Web Search for a Planet: the Google Cluster Architecture* [3], and *Managing Server Energy and Operational Costs in Hosting Centers* [6]. Although my research is not specific to datacenter thermal management, it will help to guarantee set service levels. These papers helped me to frame my research against the work being done to optimize a particular market.

The papers above illustrate the amount and diversity of thermal research being conducted. Because the scheduler is an essential component to any operating system, my work will likely have some impact on all of the research described above. The next section will extend the context of my work from the research community to society in describing ways it is relevant to the ordinary user.

CHAPTER 3: SOCIETY'S NEED FOR COOLER COMPUTERS

This chapter will explain how my work benefits society. The different application domains will give a sense for the complexity of the problem and the different metrics by which computer performance is measured.

CONSUMER ELECTRONICS

Most laptop users have undoubtedly experienced the discomfort caused by the heat generated by a laptop placed on one's lap. A recent study found that laptop use causes significantly elevated scrotal temperature [14]. While it is unclear how damaging this raised temperature actually is, the researchers recommend that men of reproductive age avoid scrotal heat exposure. By implementing selective thermal throttling, laptop temperatures can be reduced, lowering the impact of laptop temperature on sperm health while allowing the user to remain productive. The mere fact that this study exists shows that the consumer base is aware of rising device temperatures which places the results of my thermal optimizations in a genuine social context.

COOLING IN DATA CENTERS

The current trend in web-services like those offered by Google, is to use a large number of clustered low-cost commodity-class computers instead of a few expensive servers [3]. These computers run processors like the Pentium 4 which are capable (or at risk, one might say) of throttling. If companies can run these clustered computers at close to maximum temperatures while being confident that throttling will be applied intelligently across running tasks, then they can reduce the amount of money invested in air cooling solutions or elect to purchase fewer servers knowing that they can push a smaller number of servers to the same limit without risk of lowered performance. This technique would be useful for a stock-trading company that guarantees that trades will be processed within a second. The particular program responsible for transactions would only be throttled if it had no unprocessed transactions; other less-prioritized tasks would be throttled at higher levels to compensate.

AMBIENT TEMPERATURE

Another area my work might impact is in military applications. Imagine, for example, an officer's personnel tracking application used in the Iraqi Desert where it regularly reaches temperatures in excess of 100^{o}F. The application uses 3D graphics to display terrain, buildings, and the locations of the members of the officer's unit. The officer needs to know exactly where his unit is so that he can coordinate their movements with other officers and direct them around obstacles and potential attacks. This application is run on a laptop powered by battery. Thermal safety was a big design consideration when it was built, but now, just as importantly so is high performance. The officer needs the tracking application to run at full performance and with full detail and high responsiveness when his troops are on the ground. He does not want his tracking application to be throttled when it starts to overheat when, for instance, his computer's search indexer starts up. My work will allow the user to specify the thermal priority of different tasks in order to prevent situations like this from occurring. The tracking application would be given the highest thermal priority, and all other applications would be given very low thermal priorities.

To test my system, I wanted to use a benchmark (a program that measures performance) that fit in the context of normal everyday use. The scenario I decided to replicate was DVD video playback. DVD video playback requires a large amount of processing power which inevitably leads to elevated CPU temperatures. Additionally, when a user plays a DVD he does not want outside distractions, such as the noise that would be caused by a fan cooling system but, of course, with increased temperatures come increased fan speeds. I will discuss my implementation and benchmarks in the next chapter.

CHAPTER 4: DEVELOPMENT TOOLCHAIN AND FRAMEWORK

This chapter will describe my tools and methodology with the goal of allowing a reader to replicate and verify my work. I will describe in-detail my changes to the Linux kernel, my development process, and the benchmarks I created to test my system.

TOOLS

My work was software-based so I did not need any physical tools besides a laptop with throttling capability. The software I used is all open source and freely available via the internet. I will briefly list the hardware I used and then describe the main software tools I used.

Hardware

I did all of my research on a Dell Inspiron 4150 laptop with a 1.60 GHz Pentium 4-M processor. The only thing special about this computer is that its processor has throttling capabilities, a property now commonplace in most laptop processors.

Software

I made my changes to the scheduler component of a Linux 2.6.13 kernel running under the Ubuntu 4.0.1 Linux distribution. I used the C compiler, gcc version 4.0.2 to recompile my changes after every modification I made to the kernel.

I wrote my benchmarks in Perl because its powerful text processing capabilities made it ideal for extracting system performance statistics. I used the MPEG decoder mpeg2dec in one benchmark to stress my system as much as possible and the media player mplayer in another benchmark to simulate a realistic workload. My kernel modifications and testing framework are discussed in detail below.

KERNEL MODIFICATIONS

The kernel is the core of an operating system, responsible for managing resources, facilitating communication between software and hardware, and scheduling tasks, among many other things [5]. I made a number of infrastructural changes to the kernel in order to facilitate my main goal of modifying the kernel's scheduler component to take into account throttling priority and throttling level.

Infrastructural Changes

The Linux scheduler maintains a list of running tasks, each one represented by a structure called task_struct (a structure is a collection of variables grouped together into one entity). The task_struct structure contains about 150 variables describing various properties of a task such as task ID and task owner. I added a variable to the task_struct structure called throttle_level to describe the amount of throttling a task is willing to endure. throttle_level can take any integer value, 0 to 7; a throttle_level of 0 means that the task is willing to be throttled any amount up to the maximum while a throttle_level of 7 means that the task is not willing to be throttled at all. The default throttle_level value is 0.

To make the throttle_level value useful to a user, I needed to provide a way to set it outside the kernel. I did this by writing a utility named "ice" that would allow a user to start a program with a specific throttle_level value. To start mplayer with a throttle_level value of 7, for example, one would execute the following command:

```
$ ice -n 7 mplayer
```

A user mode program such as ice, however, cannot modify kernel level parameters like throttle_level without some extra help – this help comes in the form of system calls.

System Calls

The Linux operating system has two modes of operation: user mode and kernel mode. In general, different tasks will be active under each mode. The key difference between the two modes is the amount of privilege: user mode is given very little privilege while kernel mode is given the highest privileges. For example, the system scheduler operates in kernel mode because it needs knowledge of minute system details whereas a media player runs in user mode because all it needs is a video file [5].

Sometimes, user mode tasks need to execute kernel mode operations. An example of this is when a task wants to read a file. Reading a file is a privileged operation that can only happen in kernel mode. But, reading files is a common operation that most user mode tasks need to be able to do. The solution to this problem is for the kernel to provide system calls.

A system call allows a user mode program to execute a privileged kernel mode operation; the kernel currently provides about 200 system calls, each of which providing a different capability. I added two system calls, set_throt_level and get_throt_level, to allow user mode programs to set and retrieve a task's throttle_level value.

Scheduler Changes

The changes I made to the scheduler were relatively minor, a fact I attribute to the infrastructural changes I made to the rest of the kernel and the supporting utilities I wrote to facilitate user mode access. During a context switch (when the scheduler sets a new task to be active and puts the current task to sleep), the scheduler checks the new task's throttle_level value. If throttle_level is not equal to 0, that is, if the task is set to be resistant to throttling, and if throttling is currently invoked, then the scheduler reduces the throttling level for the duration of the task's timeslice. If throttle_level is equal to 0, then the scheduler makes sure that the throttling level is set to the level last set outside the scheduler.

BENCHMARKS AND RESULTS

I made two benchmark programs to measure how well my scheduler modifications worked. My first benchmark involved decoding 190 DVD video files (a total of about 50,000 video frames), approximately 1.3 MB in size each. After decoding each file, the benchmark would log the amount of time it took and take a snapshot of system statistics. The idea was to simulate a component of DVD video playback (decoding) and to put continuous stress on my system. In addition to the video decoding, I ran a continuous compile in the background for the duration of the benchmark to put additional stress on the system. I will discuss my second benchmark after I explain the results of this first benchmark.

I ran the two benchmarks on each of the following three system configurations:

1. Original kernel, no throttling
2. Original kernel, maximum throttling
3. Modified kernel, maximum throttling

Benchmarking the original kernel without any throttling would give a sense of the system's maximum performance. It would also most likely push the system to the highest temperatures and fan speeds. Benchmarking the original kernel with maximum throttling would give a sense for the type of performance to expect on a normal system with throttling invoked. Performance here was expected to be poor but temperatures and fan speeds would also be at a minimum. Benchmarking the modified kernel with maximum throttling would give a sense of how well my changes worked. I would expect the benchmarked program to have good performance and for the system to have low temperature and fan speed.

Listed in Table 1 are the results of the first benchmark. As expected, when the benchmark was run on the original kernel with throttling disabled, temperatures and fan speeds were much higher than for the other two. Also, the time to decode all 190 video files was only 359s, a fraction of the time it took for the two throttled benchmarks to complete. Notice, however, that it took close to one forth the amount of time for the throttled modified kernel to decode all 190 video files that it took for the original throttled kernel to do so. And notice also, that the average temperatures and fan speeds for the modified and original kernels, when throttled, are very comparable.

Table 1 – Benchmark 1 Results -- Statistics for decoding 50,000 frames of DVD video

	Time	Temp (°C)			Fan Speed (RPMs)		
	(s)	Max	Min	Avg.	Max	Min	Avg.
Original, Unthrottled	359	80	51	72.78	5940	3780	4958.947
Original, Throttled	6749	56	53	55.11	3840	3780	3808.684
Modified, Throttled	1885	56	48	54.41	4080	4020	4079.842

Figure 5 gives a visual representation of the temperature differences as the number of decoded frames progresses. Notice that for the throttled systems, although the modified kernel runs at about the same temperature as the original kernel, the modified kernel finishes decoded the video files in about one forth the time. The same goes for Figure 6 – the modified kernel decodes the video files in a fraction of the time that the original kernel takes, but its fan speed is barely greater than the original kernel's.

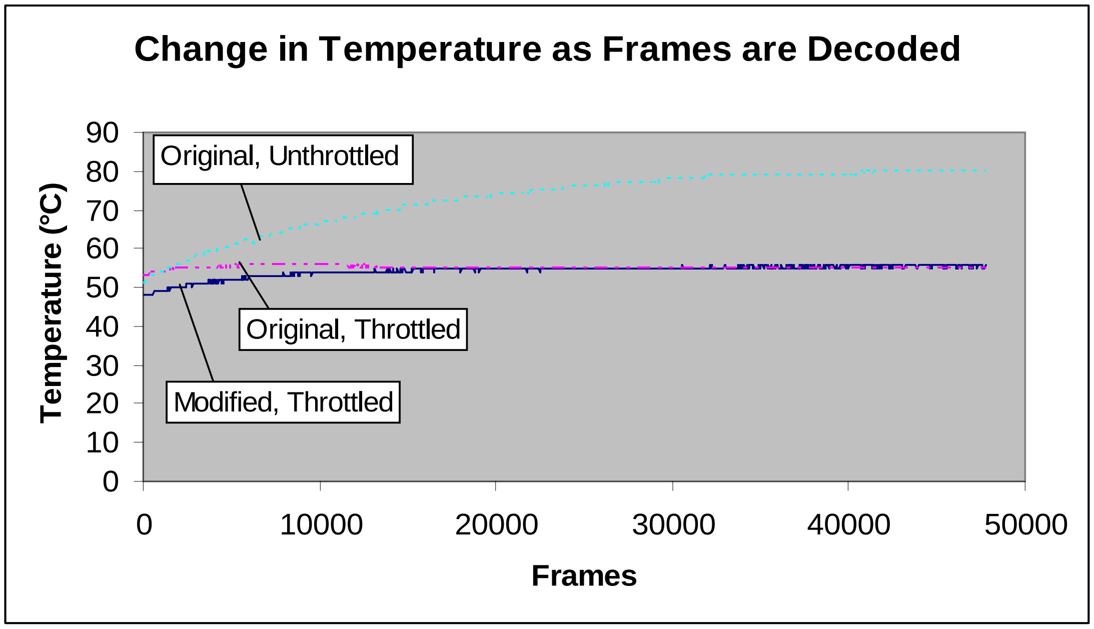

Figure 5 – Change in Temperature as Frames are Decoded – This shows the difference in temperatures for the three different system configurations. (Source: Author)

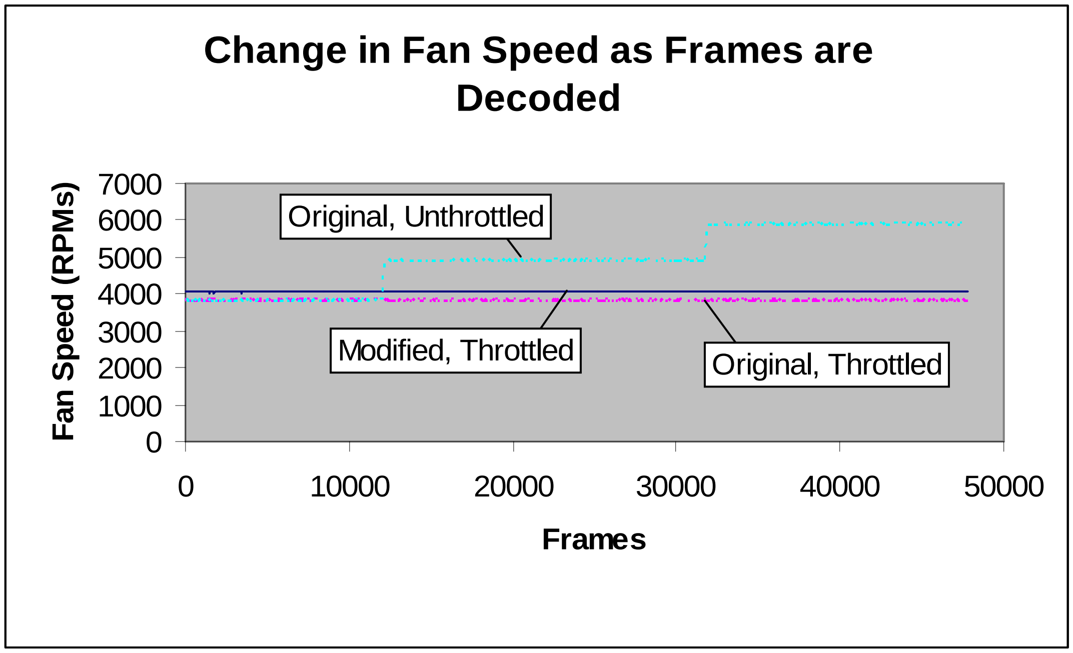

Figure 6 – Change in Fan Speed as Frames are Decoded—This figure shows that difference in fan speeds for each of the three system configurations. (Source: Author)

Having seen that the modified kernel does provide better performance in throttling mode but that it does not provide the same level of performance as the original kernel when unthrottled, I wanted to know if the system was efficiently enough to play DVD video. So, I created a second benchmark.

Second Benchmark

My second benchmark was meant to simulate a more realistic workload, so I decided to actually play a DVD video file rather than simply decode it. I wanted to see how much I could throttle my computer running the original kernel before DVD video playback became unfeasible. At the point that DVD playback would not work, I wanted to see if running the modified kernel would improve performance and whether it would allow me to throttle the system to an even greater extent.

Table 2 – Benchmark 2 Results – DVD video playback at different throttling levels.

Throttling Level	Original Kernel	Modified Kernel
0.0%	Yes	-
12.5%	Yes	-
25.0%	Yes	-
37.5%	No	Yes
50.0%	-	Yes
62.5%	-	No
75%	-	-
87.5%	-	-

As shown in Table 2 above, the original kernel could not play DVD video beyond throttling level 25% whereas the modified kernel could play DVD video when the system was throttled as high as 50%.

CHAPTER 5: CONCLUSION

The purpose of my project was to implement and test a scheduler that selectively varies the amount of processor throttling, depending on the active task. My results, described in the previous chapter, were very encouraging, and suggest that more research into this technique would be valuable. I will summarize my work and suggest future avenues for research here.

SUMMARY

My data show that selective throttling can reduce system temperature and thus fan speed while still allowing high performance of certain tasks. Further, it shows that processor-intensive applications can be run at lower throttling levels on a modified kernel through selective throttling.

INTERPRETATION

My work has shown that system performance can be improved in the following ways: IO-Bound tasks, such as games and DVD playback, can be executed in near realtime even when the system is throttled and that fan noise can be limited for multimedia applications that generally create a large amount of heat.

One limitation to my work is the limited applicability it might have to real workloads. My tests were limited to a narrow application field and they do not show how well selective throttling would perform in critical situations. Also, I have not tested throttling levels between maximum and minimum throttling which would most likely provide many opportunities for optimization. Another major limitation is the limited likeliness that my modifications will be adopted into the mainstream Linux kernel. Even though this is the case, however, it is possible that independent vendors, such as portable electronics manufacturers, would see value in my work and adopt it.

RECOMMENDATIONS

There is much room for future research. One thing that could be done is to automate the process of throttling and dethrottling certain programs. Currently, to change the throttling level of a task, a user must explicitly set the throttling level of that task. I believe that this process is too reliant on the end-user and can be improved so that user input is minimized. A solution might be to use the existing priority levels to infer what type of throttling to use for a task. An input-bound task such as a game or movie would need to be responsive and would require to be throttled lower than a CPU-bound task such as a compile or download. Another improvement would be the use of feedback to change continuously tasks' throttling levels to optimize system performance with respect to operating temperature.

I believe that my work has paved the way for more sophisticated temperature-aware scheduling algorithms which can be generalized to apply to a wider variety of applications such as the military and e-commerce sectors.

SOURCES CITED

[1] Aas, J. (Feb. 17, 2005). Understanding the Linux 2.6.8.1 CPU scheduler. Retrieved Oct. 16, 2005 from http://josh.trancesoftware.com/linux/

[2] Abdelzaher, T.F., Lu, C. Schedulability analysis and utilization bounds for highly scalable real-time services. Presented at Proceedings of the Seventh Real-Time Technology and Applications Symposium (2001).

[3] Barroso, L.A., Dean, J., Hölzle, Urs. Web search for a planet: the Google cluster architecture. IEEE Micro, March/April, 2003.

[4] Bellosa, F. OS-directed throttling of processor activity for dynamic power management. (1999). Technical Report TR-I4-99-03, Department of Computer Science, University of Erlangen.

[5] Bovet, B.P., Cesati, M. Understanding the linux kernel, third edition. O'Reilly Media Inc. Sebastopol, CA (2005).

[6] Chen., Yiyu, Das, A., Qin, W., et al. Managing server energy and operational costs in hosting centers. Presented at Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2005.

[7] Dadvar, P., Skadron, K. Potential thermal security risks. 21st IEEE SEMI-THERM Symposium, 2005.

[8] Gleason, S. Power aware operating systems: task-specific CPU throttling. Retrieved Nov. 23, 2005 from http://oldwww.cs.pitt.edu/PARTS/implementation/sunny-paos.pdf

[9] Heo, S., Barr, K., Asanović, K. Reducing power density through activity migration. Presented at The International Symposium on Low Power Electronics and Design (ISLPED), 2003.

[10] Intel Corporation. IA-32 Intel® architecture software developer's manual, volume 3: System programming guide. Denver, CO (2005).

[11] Intel Corporation. Intel centrino mobile technology. Intel Technology Journal, Volume 7. May 2003. Denver, CO (2005).

[12] Love, R. *Linux kernel development, second edition.* Novell Press. Indianapolis, IN (2005).

[13] Powell, M.., Gomaa, M., Vijaykumar, T.N. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. Presented at The Tenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2004.

[14] Sheynkin, Y., Jung, M., Yoo, P., Schulsinger, D., Komaroff, E. Increase in scrotal temperature in laptop computer users. Human Reproduction (Advance Access), Dec. 2004.

[15] Weissel, A., Bellosa, F. Dynamic thermal management for distributed systems. Presented at The Proceedings of the First Workshop on Temperature-Aware Computer Systems, 2004.

BIBLIOGRAPHY

Aas, J. (Feb. 17, 2005). Understanding the Linux 2.6.8.1 CPU scheduler. Retrieved Oct. 16, 2005 from http://josh.trancesoftware.com/linux/

Abdelzaher, T.F., Lu, C. Schedulability analysis and utilization bounds for highly scalable real-time services. Presented at Proceedings of the Seventh Real-Time Technology and Applications Symposium (2001).

Barroso, L.A., Dean, J., Hölzle, Urs. Web search for a planet: the Google cluster architecture. IEEE Micro, March/April, 2003.

Bellosa, F. OS-directed throttling of processor activity for dynamic power management. (1999). Technical Report TR-I4-99-03, Department of Computer Science, University of Erlangen.

Bovet, B.P., Cesati, M. (2005). *Understanding the linux kernel, third edition*. Sebastopol, CA: O'Reilly Media Inc.

Chen., Yiyu, Das, A., Qin, W., et al. Managing server energy and operational costs in hosting centers. Presented at Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2005.

Corbet, J., Rubini, A., Kroah-Hartman, G. (2005). *Linux device drivers, third edition.* Sebastopol, CA: O'Reilly Media, Inc.

Dadvar, P., Skadron, K. Potential thermal security risks. 21st IEEE SEMI-THERM Symposium, 2005.

Gleason, S. Power aware operating systems: task-specific CPU throttling. Retrieved Nov. 23, 2005 from http://oldwww.cs.pitt.edu/PARTS/implementation/sunny-paos.pdf

Heo, S., Barr, K., Asanović, K. Reducing power density through activity migration. Presented at The International Symposium on Low Power Electronics and Design (ISLPED), 2003.

Intel Corporation. IA-32 Intel® architecture software developer's manual, volume 3: System programming guide. Denver, CO (2005).

Intel Corporation. Intel centrino mobile technology. Intel Technology Journal, Volume 7. May 2003. Denver, CO (2005).

Love, R. (2005). *Linux kernel development, second edition.*. Indianapolis, IN: Novell Press.

Powell, M.., Gomaa, M., Vijaykumar, T.N. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. Presented at The Tenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2004.

Sheynkin, Y., Jung, M., Yoo, P., Schulsinger, D., Komaroff, E. Increase in scrotal temperature in laptop computer users. Human Reproduction (Advance Access), Dec. 2004.

Weissel, A., Bellosa, F. Dynamic thermal management for distributed systems. Presented at The Proceedings of the First Workshop on Temperature-Aware Computer Systems, 2004.