

Plimsoll: a DVS Algorithm Hierarchy

Hrishikesh R Amur

National Institute of Technology Karnataka, Suratkal.

hrishikesh.amur@gmail.com

Gautham R Shenoy

Dipankar Sarma

Srivatsa Vaddagiri

Linux Technology Center, IBM India Systems and Technology Lab

{ego, dipankar, vatsa}@linux.vnet.ibm.com

Abstract

Finding a compromise between high performance and low power consumption has recently become important even in non-battery-operated systems. An increasing number of researchers have taken advantage of recent processors' capabilities of scaling down frequency and the operating voltage of the CPU to obtain significant power savings. But previous efforts have aimed at policies tailor-made for specific kinds of workloads and most policies run the processor in a "power management mode" where the performance of all the running processes is affected. In this paper, we present Plimsoll, a set of algorithms implemented as an extension to Linux[®]. Plimsoll allows the performance degradation values to be set per-process and guarantees that degradation stays within the specified limit for that process, giving best-effort energy savings.

1 Introduction

Dynamic Voltage Scaling (DVS) techniques have been researched extensively in the past [1][2]. These techniques are based on the fact that the peak frequency of a processor is directly proportional to the supply voltage and energy consumed for a workload is proportional to roughly the square of the supply voltage. Therefore running the processor at a lower frequency means that the supply voltage can also be lowered, resulting in a quadratic decrease in energy consumption [3]. But past work has tended to focus on specific domains when fixing performance guarantees. This is simply because given the diverse range of workloads it is extremely difficult to devise an algorithm which would perform admirably for all the workloads. Plimsoll sidesteps this problem by letting the user decide acceptable performance degradation, which is specified in terms of excess time taken beyond normal execution time, on a per-process basis. It then guarantees that the degradation due to DVS stays below this value for that process. For example, if the performance degradation for a particular process has been set to 10%, then Plimsoll guarantees that the process will complete within 10% excess of the time it would have taken executing at full frequency. The power saved is on a best-effort basis. We compare Plimsoll to an efficient interval-based algorithm such as the Ondemand governor[4], and evaluate some optimizations to Plimsoll.

2 Related Work

While the case for power management in battery-operated and hand-held systems has always been strong[5], there has been enough evidence in recent literature that significant power savings can be achieved with tolerable performance hits in web servers[6][7]. Previous research into DVS algorithms can be dichotomised into approaches that obtain task deadlines from real-time kernels and interval-based approaches. We concentrate on interval-based approaches which often use processor utilization history to derive useful information that can be used in prediction. Attempts to predict short-term processor requirements based on past history have generally proved unsatisfactory and the complexity of the prediction methods do not increase prediction accuracy significantly[2]. A recent approach combined a prediction-based algorithm, which made conservative guesses, with a performance-checking algorithm[3] based on the idea that a hierarchy of performance-setting algorithms each specialized for different workload characteristics can be used for controlling the processor's performance.

But most methods still do not differentiate between processes and all executing processes take performance hits. We recognize that for certain critical processes, this is clearly unacceptable and being able to set tolerable degradation levels in a per-process manner, these critical processes can run without any

performance hit while power can be saved by slowing down other, concurrently running processes.

3 Implementation

Plimsoll follows the multiple-algorithm model used by Flautner and Mudge[3] where a decision hierarchy is introduced where the decisions taken by the algorithms of the lower layers tend to maximise energy savings and the upper layer which can override the decisions from below seek to introduce checks to ensure that performance does not deteriorate beyond acceptable limits that have been specified per-process by the user. In this paper we introduce new algorithms for the higher layers which estimate per-process performance degradation and take decisions to keep deterioration within the limits specified for that particular process and to minimise the overhead associated with the power state changes. The lowest algorithm makes conservative decisions in stretching execution bursts into predicted idle time to maximize the power saved.

3.1 Counting time

Our measurements require high resolution timers to timestamp events such as entry into runqueue, scheduling for execution etc. Since the timestamp counter in the hardware used varies with the frequency of the processor, we maintain a timestamp counter, *dpm.tsc* normalized to maximum frequency. At each synchronization point, *dpm.tsc* is calculated as:

$$dpm.tsc_i = dpm.tsc_{i-1} + \frac{cycles_i - cycles_{i-1}}{f_{i-1,i}}. \quad (1)$$

Where *dpm.tsc_i* is the value of the timestamp counter at synchronization point *i*, *cycles_i* is the number of cycles executed by the processor from boot-up to point *i* and *f_{i-1,i}* is the frequency of the processor between points *i* - 1 and *i*. The synchronization points are events that change the execution state of a process such as being scheduled to execute, preemption, timeslice expiry, sleeping on I/O and waking up.

3.2 Definitions

3.2.1 CPU burst length

The CPU burst length for a process is measured as the time spent executing on the CPU from when it was scheduled, to the point where it is taken off the runqueue when it sleeps. This means that the burst

length is tallied over preemptions and timeslice expiries which makes the nature of the process independent of the working of the process scheduler.

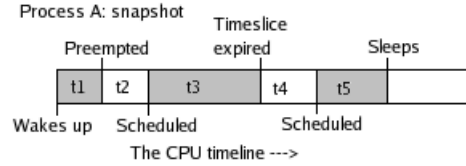


Figure 1: Calculation of burst length

For example in Figure 1, the CPU burst length for Process A would be calculated as $t_1 + t_3 + t_5$ when the process goes to sleep and is taken off the runqueue.

3.2.2 Performance Degradation

Performance degradation for a process is measured in terms of the excess time taken by the process to complete execution. The execution time of a process consists mainly of time spent executing on the CPU, waiting while runnable on the runqueues and blocked on I/O.

3.3 The Stretch Algorithm

Stretch seeks to maximise energy savings by running the processor in lower power states. The aim is to bring system idle time close to zero and increase CPU utilization, while minimizing chances of overstretching and causing congestion in the runqueue. When a process is scheduled to execute, this algorithm predicts the expected execution burst length based on execution history and stretches this burst into the available idle time by moving to a lower power state. Figure 2 shows the process profiles for the main make thread in kernel compilation. With Plimsoll enabled, the CPU bursts are stretched.

This means that frequency and voltage scaling is done only when the system has some idle time. On a busy CPU, all the processes would run with highest performance.

The prediction can be made using a number of well-researched methods[2]. Since our requirement is to minimize the number of underestimations, we found that using a system based simple average of previous bursts works well in predicting the next CPU burst length. The frequency is calculated as:

$$f_{stretch} = \min(f_i) \quad (2)$$

such that

$$f_i \geq \frac{f_{max} * remaining_burst_length}{stretch_space} \quad (3)$$

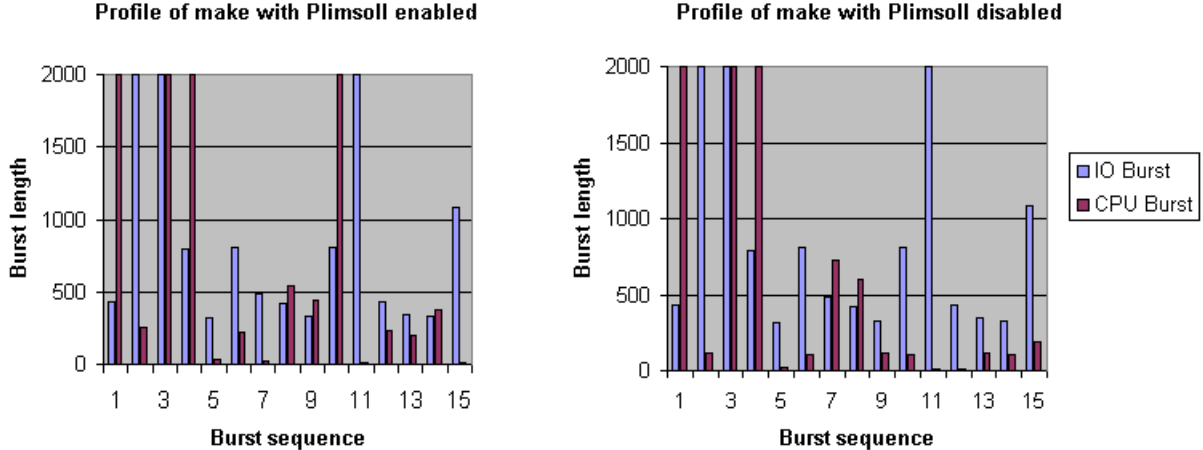


Figure 2: Stretching of process - make

Where the *stretch_space* is calculated from system idle time and *remaining_burst_length* is initially set to the full predicted length and updated each time (if) the process is scheduled out because of a timeslice expiry or a preempt. In the case of underestimation of the next burst length, the excess is run at maximum frequency as a corrective mechanism.

3.4 The Performance Algorithm

This algorithm is responsible for ensuring that performance degradation for each process does not exceed the respective specified value. It accomplishes this by associating a net frequency f_{avg} which is calculated from the acceptable performance degradation, d_0 as:

$$f_{avg} = \frac{f_{max} * 100}{100 + d_0} \quad (4)$$

f_{avg} is an estimate for the average frequency which would guarantee that performance degradation remains below d_0 . A system of credits is used for each process where, running at a frequency higher than its f_{avg} accumulates credits while a process running at a frequency lower than its f_{avg} loses credits. By keeping the credits value around zero, or setting frequencies that tend to do so, we can ensure that the degradation remains in check.

The rationale behind this claim lies in the fact that when quantifying the slowdown experienced by a process due to changing to a lower power state, it is sufficient to consider the degradation due to running at a lower power state only. The effects of scheduling latencies which are difficult to measure and control need not be considered. The latencies that may occur during scheduling are:

1. Time spent waiting on the active runqueue - T_r .
2. Time spent in changing power state.

Slowing down the processor may cause some of these latencies to increase when compared with values at peak performance, but they do not need to be included explicitly in the calculation of performance degradation for a process. In the case of T_r , an increase in latency due to running at a lower power state is expected but the cumulative effect of increases in T_r for all the processes will cause system idle time to reduce because of increased queueing on the active runqueue. This will cause less stretching for the processes that are henceforth scheduled to execute causing them to run at higher power states.

Power state change latencies in currently available hardware however are not negligible and since in the worst case, Plimsoll would cause a power state change for every context switch where the overhead becomes sizeable, we introduce a third algorithm in the hierarchy which tries to keep the processor running at the same power state for longer periods of time. This effectively reduces the number of power state changes.

3.5 The Sandbag Algorithm

This algorithm seeks to smoothen out the power state changes for the processor. Whenever a context switch occurs, the algorithm tries to run the process being scheduled in, at the current power state of the processor. A buffer `SANDBAG_LIMIT` is maintained for each process which accumulates the difference in the power state recommended for the process by the lower algorithms and the power state that it actually runs at. When this buffer becomes full (positive or neg-

Table 1: Workload with two processes with varying degradation

Process A			Process B		
Degradation (%)	Performance (sec)		Degradation (%)	Performance (sec)	
	Total time	CPU time		Total time	CPU time
0	7.17	0.08	0	23.38	3.14
20	7.43	0.16	0	23.82	3.11
20	7.50	0.15	20	26.18	6.74

ative), the power state of the processor is stepped up (or down). The threshold values are configurable and the granularity of power state changes can be adjusted depending on the overhead associated with a power state change in hardware.

4 Evaluation

Our measurements were performed on an IBM® T42 Thinkpad using the Intel® Pentium™ M processor with Enhanced Speedstep® [8] running at six levels of frequency 600, 800, 1000, 1200, 1400 and 1500 MHz. The power management extensions of Plimsoll were added to the Linux 2.6.18 kernel [9].

We compared Plimsoll with another popular interval-based algorithm used by the Ondemand cpufreq governor which monitors CPU utilization at frequent intervals and scales frequency up or down using threshold values. We used a variety of workloads and found that in the case of Plimsoll, the processor executes for equal or slightly larger fractions of the total execution time in lower performance levels as compared to Ondemand, translating into greater power saved. In addition is the added advantage of being able to set performance degradations per-process and in effect choose which processes to save power with. To demonstrate this, we ran a workload consisting of two processes with differing values for acceptable performance degradation. Plimsoll meets requirements for both processes. Since performance levels are being assigned per-process, in the worst case the overhead of a power state change might be incurred during every task switch. To alleviate this, a buffering algorithm was used in Plimsoll, in which using a SANDBAG_LIMIT of 10 credits, we reduced the number of power state changes by an average of 80%.

The workloads used in the evaluation were: tar compression utility [10], the Mplayer library [11], kernel compilation and a more CPU-intensive prime number generator. The workloads were run a number of times and the average value used, taking care to clear cache contents between two successive runs. Table 1 shows the results of a workload consisting of two processes. The workload was first run at

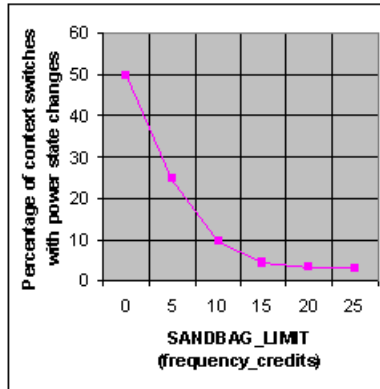


Figure 3: Number of power state changes

peak performance and then with Process B deemed as critical and not allowed any degradation in its performance. Plimsoll continues to run Process A at lower power states and thus reduce power consumption while guaranteeing peak performance for Process B. This is the major advantage of Plimsoll over other interval-based algorithms. Table 2 shows the percentage of total execution time that each of the workloads spent at different power states for Plimsoll and Ondemand. The average performance level of the processor for each of the workloads shows that power savings with Plimsoll are roughly the same as that of Ondemand. Most previous DVS algorithms fail to account for the often considerable overhead of making the power state changes.

Especially in Plimsoll, this issue has to be addressed since making power state changes at every context switch is wasteful. Figure 3 shows that using a buffer dramatically reduces the number of power state changes. All tests were run using a SANDBAG_LIMIT of 10.

5 Conclusion

We have shown that Plimsoll allows varying levels of performance degradation to be set for different processes, thus enabling saving of power even when

Table 2: Performance of Plimsoll and Ondemand with various workloads

Workload		Percentage of execution time - (MHz)						Average Performance (%)
		600	800	1000	1200	1400	1500	
tar	Plimsoll	61.03	35.82	2.99	0.08	0.06	0	45.76
	Ondemand	89.25	0.75	0.23	0.09	0.26	8.8	45.26
mplayer	Plimsoll	89.65	8.94	0.07	1.32	0.01	0	41.69
	Ondemand	91.56	0.12	3.41	0.02	0.02	4.86	43.87
prime	Plimsoll	65.32	30.13	3.05	1.48	0.36	0	45.75
	Ondemand	93.87	1.65	0	0	0	4.30	42.73

critical processes are being run, unlike previous algorithms which switched into a power-saving mode where the performance of all running processes was affected. With degradation set to 25%, the power saved is equal to or slightly better than the Ondemand governor for conventional workloads such as compression, kernel compilation and CPU-intensive computation. To decrease the overhead involved with the power state changes, a buffering algorithm was used which drastically reduces the number of state changes.

References

- [1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy", *Proceedings of the First Symposium of Operating Systems Design and Implementation*, November 1994.
- [2] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU", *Proceedings of the First International Conference on Mobile Computing and Networking*, November 1995.
- [3] Krisztian Flautner and Trevor Mudge, "Vertigo: Automatic Performance-Setting for Linux", *OSDI '02: Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.
- [4] Venkatesh Pallipadi and Alexiy Starikovskiy, "The Ondemand Governor - Past, Present and Future", *Proceedings of the Linux Symposium Volume Two*, July 2006.
- [5] O.S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems", in *Proc. IEEE, Special Issue on Real-Time Systems*, vol. 91, no. 7, July 2003.
- [6] R. Bianchini and R. Rajamony, "Power and energy management for server systems", in *IEEE Computer, Special issue on Internet data centers*, vol. 37, no. 11, Nov. 2004.
- [7] Tibor Horvath, Tarek Abdelzaher, Kevin Skadron, Xue Liu, "Dynamic Voltage Scaling in Multi-tier Web Servers with End-to-end Delay Control" in *IEEE Transactions on Computers*, to appear.
- [8] Enhanced Speedstep, www.intel.com/support/processors/mobile/pentium4/sb/CS-007499.htm.
- [9] The Linux Kernel Archives, <http://kernel.org>.
- [10] The gzip Homepage, <http://www.gzip.org>.
- [11] <http://mplayer.org>.

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM.

IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds.

Intel, Pentium and Enhanced Speedstep are registered trademarks of Intel Corporation in the United States, other countries or both.

Other company, product, and service names may be trademarks or service marks of others.