Idle Injection Mechanism Implemented in the BFS scheduler

Andrea Mambretti, *student id: 783286*
Luca Muccignato, *student id: 783274*

***Abstract*—In these days the problems of power consumpion and temperature management have become two of the most studied fields in computer architecture. Lots of applications are built with different methods and different levels of abstraction to try to manage them. Various techniques are implemented at the hardware level, others are at software level either in the kernel or in userspace.**

In this paper we want to discuss about the concepts, implementation and results of a system based on the BFS scheduler of Linux that uses the idle injection method to reduce the workload and therefore the temperature of a stressed system. This kind of solution, as you will undestand, is already used and implemented but there's nothing for that kind of scheduler and with this flexibility of work.

***Index Terms*—Operative System, Linux Kernel, Power Consumption, Temperature, Scheduling, Idle Injection**

I. Introduction

In modern computer architecture, developers are obliged to consider the problems caused by high temperature and power consumpion. This can really have a bad impact on the performance of a system, in terms of throughput and overheating of components.

To avoid this situation, we can work at hardware level using more parallelized cores, processor with threading support, do nothing well technique, lower power state for DRAM or disks and dynamic voltage frequency scaling.

However, we can also work at software level using idle injection or frequency scaling techniques. Today, more or less all modern CPUs have the Frequency Scaling Support, which is used by the operative system to control the power consumpion (they have policy like powersave, ondemand etc), whereas there are only few implementations of idle injection technique.

In this paper we will present our own solution for the linux kernel 3.2.6 patched with the BFS scheduler by Con Kolivas. The first chapter is an overview on what is the situation of the modern schedulers in the linux kernel.

II. Actual Situation in the Linux Kernel

In today linux boxes, we can find two main versions of the scheduler. One is in the mainline of linux kernel, created by Ingo Molnar and called CFS; the other is an unofficial version, developed autonomously by Con Kolivas and called BFS.

CFS, which stands for Completely Fair Scheduler, is in the mainline since 2007. The idea behind is based on the Fair-Share Scheduling policy, which divides the CPU time among entities, such as users, groups, or tasks. CFS can support all this levels, but was originally created for scheduling at task level. Each one of these should have a fair share of the CPU, but this is an ideal case. CFS keeps track of how much time is given to a task respect the others, so it can schedule the one with the highest level of unfairness. To easly know which task it should chose, CFS organizes them in a red-black tree, descending ordered by their total unfairness. The choosen task is the leftmost and the total time required to schedule it is $O\left(\log n\right)$, where n is the number of tasks present in the system. When the current task has finished running on the CPU, just before selecting the new one, the scheduler increments the task's *vruntime* field, which saves the amount of time that the CPU has dedicated to it. Then the scheduler choses the most unfairly treated task by selecting the one with the lowest vruntime. At its creation, a new task is given the minumum value of current vruntime, which is stored and mantained in a variable in order to speed up the whole process. So, there is no concept of timeslice, but a task run until it's no longer the most unfairly treated. In order to reduce the context switching overhead (the time used for sending a content switch signal), CFS introduces a minimum granulrity of time (a minimum time that a chosen task must at least run), used to control the tradeoff between latency and content switching's overhead. In case of multiprocessor architectures, CFS keeps separate data structures for each CPU, reducing lock contention but requiring some sort of load balancing to spread tasks across processors.

BFS, which stands for Brain Fuck Scheduler, was written by Kolivas as an alternative to CFS. It doesn't adopt the modular framework of CFS; instead it uses one single system-wide runqueue, containing all non-running tasks. In this way, it's able to determine the next scheduled task without using some sort of special and complicated heuristic. BFS implements an earliest effective virtual deadline policy. A virtual deadline is the ideal time that any two tasks with the same niceness will have to wait before running on the CPU; each task has its own assigned when it asks for CPU usage. Even if it's virtual, so called because there is no assurance that a task will be scheduled by that time, however tasks will definitly be scheduled from earlier to later virtual deadlines. Because of this approach, earlier virtual deadlines are given to tasks with higher priority. If a task blocks, it keeps the remainder of its timeslice and virtual deadline, so it gains higher priority when is rescheduled. Because of there is no order of tasks in the runqueue, they cannot be placed in a tree like in CFS' solution and the lookup for the next scheduled task takes an $O\left(n\right)$

scan over the whole queue. As a consequence, BFS scales poorly with large amounts of tasks. Furthermore, a share data structure among all CPUs increases lock contention, but tasks can be quick scheduled on different CPUs without problems as soon as they became available. This results in a lower latency.

On the net is possible to find several stress tests and, some of them, really well made, using benchmarking programs, like latt.c a tool specific for schedulers's benchmarking, or just using gcc instruction, make for exemple, or playing a video. Those tests can be done on netbook, laptop and desktop too. Results generally demonstrate that BFS scheduler has a lower latency, which is not a surprise indeed as it was written with this aim. However CFS scheduler result to have a better performance in terms of turnaround time (which is the total time taken between the submission of a task for execution and the return of the output) under different load conditions, expecially with multicore machines. Finally, another result is that BFS scheduler, playing a video, drops much less frames than CFS, under large amounts of load.

As a conclusion, we can say that, generaly, CFS is better in terms of turnaround time, and so for batch processing, while BFS has a less latency, more indicated for interactive tasks.

III. THE HEAT PROBLEM

More than in past years, heat is a main problem that must be taken into account while building systems. Components are susceptible to malfunction or even break if overheated, expecially integrated circuits like CPUs. Because of these, components are often designed to generate as little heat as possible and operating systems try to reduce power consumption which generates heat. In modern computers, integrated circuits are the prime source of heat, due to the fact that they work at high frequency and voltage. Increasing CPUs workload in order to exploit better performance in term of thrughput and speed up, with ILP i.e., results, as a drowback, in an increasing amount of warmth produced by components.

Nowadays all CPUs ang GPUs must use dedicated Heat Sinks or they will reach high temperatures in few time of work. But also via software there's the possibilty to reduce it, by writing programs properly or using specific software as well. Even solutions that involve idle injection.

Because it's the mainline kernel, proposals in this last way are already provided for CFS scheduler. One, for exemple, is Kidled by Google. The idea behind the project is to realize a "Power Capping through Idle Cycle Injection", as the project's title says. It consist in a module for the CFS scheduler, patched by Google developer Salman Qazi, which allows the system administrator to set the percentage of time that a specific CPU should be idle and an interval over which that percentage is calculated. To avoid important processes to be stalled, there is the special notation "interactive" with which a process can be marked and, when running, idle cycles will be forced only when necessary. Kidled code also allows system administrator to select a process that should be idled, so he can chose if he wants to affect all the system or just a single process. The result is a reduction of power consumption and, as a consequence, a reduction of heating.

IV. SYSTEM GOAL AND APPROACH

The goal of our system is to provide an injection control mechanism of idle processes for the linux BFS kernel which is able to reduce the power consumpion and the average temperature. The aim of doing that is to try to maintain the best condition of work, in terms of heating, also when a system is overloaded. To garantee the flexibility and the usability, our system has no policy hardcoded inside, whereas there are two interfaces where external monitors can plug in and control it specifing their own policies.

Our approch is based on the behaviour of the scheduler. We mainly worked on the *schedule function* and on the function that chooses the earliest deadline task. The idea behind our system is to inject idle every X calls of the *schedule function*, or Y times that a specific process is scheduled. We faked the condition inside the BFS to force it to schedule an idle cycle, instead of elected process. The way is to change, if needed, the <u>next</u> process with the idle process before that the *context_switch function* is called. Actually, our system doesn't consider realtime processes because it has no much sense to work on some process that needs all the resources given to it.

Our implementation provides two main functions, one which works globally on the machine and the other one on a specific process or thread (via its identifier, pid or tid).

The global function can be used to keep under control the temperature of the machine when it's rising.

The second function works a specific identifier can be use to control a single process or a thread (even more than one at a time), maybe because it's requiring more CPU time respect to the amount that what we want to give it (ex. updates manager).

Our system has two interfaces, working through procfs, which either the user directly or a monitor program as well can use to control the injection (see the Implementation Chapter below).

V. IMPLEMENTATION

In this section we want to present some more detailed aspects of our own implementation:

A. IDLE PROCESS

Our mechanism is an injector of idle, but what is an idle process? Actually the idle process can be different things. In some cases it is a process with pid=0 that executes a sequence of NOP instructions and it is executed if and only if there aren't other processes to execute (it's the process with the lower priority). In other cases it is represented as a turned off cpu to save power when the cpu hasn't nothing to execute. Inside the linux kernel the idle process is like a normal process that does nothing, it's the only process where the *task_struct* it the only one that is not dynamically allocated. It is staticaly defined at kernel build time and is called init_task. At the end we chose to use this special process because it's a natural process that is already inside in the system and do what we want to do that is nothing inside the processors.

B. *OPERATIVE SYSTEM*

The linux version choice is totally arbitrary to use our system. In our own implementation we chose Gentoo with a standard *kernel 3.2.6*[1]. That kernel was patched with a kovalis' patch porting did by us. We performed also another patch to our kernel with hrm system. We needed hrm to get details about the impact on the thoughput of our application. We had to also install lm_sensors module to get information from the temperature sensors.

C. *GLOBAL INJECTION*

The global injection is realized to control the global scheduling machanism. The maximum bound is variable and can be modified by procfs. The file in procfs can be found in */proc/schedidle/sched_global*. To insert a specific value we can simply use an echo into our file specifing the X value[2] Using this kind of injection it's possible to apply different type of policy, starting from a simple policy that checks the temperature ending with a more complex policy that takes also in account other metrics like the cpu load. We tested it with a simple metric that takes in account only the average temperature among all the temperatures of the testing machine (see Results Chapter). The point where we modified the schedule to achive this kind of injection is direct inside in schedule function.

D. *PID/TID CONTROL*

The second one way of usage is a more fine mechanism due to control processes or threads. It has also its file in procfs at path */proc/schedidle/sched_pid* where either a user or a monitor program can manage a list inside kernel space where therare stored tuples with the following information: 1) Identifier: number that tell to the kernel which process/thread control. 2) Y: this value means "put an idle process every Y times that the process with pid/tid = identifier". 3) Type: this field say if identifier refer to either process or thread identifier can be equal to p/t. Also there the insertion can be done using an echo on the file[3].

VI. FUTURE WORK

At the end our work wants to be a starting point to other interesting projects. Other works that can be done are either a monitor to take undercontrol the temperature with smart policy that consider also the number of processes or consider the namespace number of the processes to have a fine grane because in some version of kernel (such as openvz) can be that two processes have the same pid/tid because they belong to two different namespaces. Other jobs can be integrating a monitor inside hrm to inject idle using the througput parameter read by hrm itself and doing the porting to other scheduler of our system with its functionality.

[1]Actually the system work for sure only on this version but maybe it can work with previous version of the kernel

[2]Ex. "echo 200 >/proc/schedidle/sched_global". This means that one times every 200 calls of schedule() put an idle.

[3]Ex. "echo 1000,6,p >/proc/schedidle/sched_pid. The previous command means put an idle process instead the process with pid equal to 1000 every 6 times that the process is scheduled