Achieving Predictable Timing and Fairness through Cooperative Polling

Anirban Sinha (student), Charles Krasic
Department of Computer Science
University of British Columbia, Canada
{anirbans, krasic}@cs.ubc.ca

Ashvin Goel
Department of Electrical and Computer Engineering
University of Toronto, Canada
ashvin@eecg.toronto.edu

Time-sensitive applications that are also CPU intensive are increasingly being used in commodity environments. Examples of such applications include video games, graphical simulations, video playback, and some recent visually-heavy graphical desktops. These applications run on commodity operating systems that are targeted at diverse hardware, and hence they cannot assume that sufficient CPU is always available. Increasingly, these applications are designed to be *adaptive*. For example, they may adapt visual fidelity according to the diverse capabilities and usage patterns of users' existing and future computers. When executing multiple such applications, the operating system must not only provide good timeliness but also allow co-ordinating their adaptations so that applications do not interfere with each other (e.g., their fidelity is stable).

In this work, we aim to meet three important requirements for supporting adaptive time-sensitive applications in a general-purpose OS: a) good timeliness: tasks must receive predictable and low latency execution, b) fairness: long term throughput of all tasks (or fidelity of the time-sensitive tasks) should be assured, avoiding starvation, and c) full utilization: unnecessary idle periods should be avoided (work conservation). We meet these requirements by combining an event-driven application model called *cooperative polling* with a fair-share scheduler. Cooperative polling allows sharing timing or priority information across applications via the kernel thus providing good timeliness, and the fair-share scheduler provides fairness and full utilization. We describe these components below.

Cooperative polling uses a new system call called `coop_poll` that time-sensitive applications use to share event information such as deadlines and priorities with the kernel. Across applications, intra-application event dispatchers use this shared information to determine appropriate times to yield, and optionally to achieve co-ordinated quality adaptations. By yielding in an informed fashion, applications minimize involuntary preemption thus achieving more predictable timing. Our kernel scheduler uses the information available from `coop_poll` to provide better responsiveness to applications that use `coop_poll`, hence providing an incentive for such cooperation. Besides rewarding cooperation among time-sensitive tasks, our model which combines fair-share scheduling, enables two other significant contributions: a) we use preemptive scheduling to prevent the possibility of `coop_poll` being abused (either intentionally or otherwise) to gain unfair advantage, and b) unlike existing approaches that have attempted to integrate conventional real-time scheduling algorithms into general-purpose operating systems with limited success, our approach allows time-sensitive and best-effort tasks to co-exist in a tightly unified framework.

While there is a large body of work in this area, our work is closest to the borrowed virtual time (BVT) scheduling algorithm [1] and the SMART scheduler [2]. Unlike SMART, deadlines in our scheduler represent times at which an application needs to run, as opposed to times by which units of work need to be completed. Also like BVT and unlike SMART, fair sharing in our algorithm is based simply on earliest virtual time, as opposed to adapting a priority-based real-time scheduler. On the other hand, unlike BVT, our algorithm uses application specific event deadlines for low latency dispatch and for scheduling realtime process as opposed to the *warp* value.

We have implemented cooperative polling in the Linux kernel. Our scheduler employs and benefits from several relatively recent infrastructural components in the Linux kernel, such as fine-grained kernel preemption, high-resolution process accounting, and high resolution kernel timers. We have performed experiments to evaluate our prototype. In our evaluation, we use a quality-adaptive video playback application to demonstrate how complex adaptation policies may be realized. We have also performed comparisons that show the performance and timing benefits of cooperative polling. Our experiments show that cooperative polling leverages the inherent efficiency advantages of voluntary context switching versus involuntary preemption. In CPU saturated conditions, we show that the scheduling responsiveness of cooperative polling is five times better than a well-tuned fair-share scheduler, and orders of magnitude better than the best-effort scheduler used in the mainstream Linux kernel. We plan to release all the code of our system as Open Source and demonstrate it at SOSP if our poster is accepted.

REFERENCES

[1] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 261–276, 1999.

[2] J. Nieh and M. S. Lam. The design, implementation and evaluation of SMART: a scheduler for multimedia applications. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 184–197, 1997.