CMPSCI 754 Multimedia Systems **Spring 2001**

Lecture 17: 10 April

Lecturer: Prashant Shenoy *Scribe: Rahul Gupta*

In last lecture, we have looked at HSFQ. With proportional sharing, a real-time scheduling policy and a conventional scheduling policy are each given a proportional share of the machine to manage by the underlying proportional share mechanism, which then timeslices between them. Real-time apllications will not take over the machine, but they also cannot meet their time constraints effectively as aresult of the underlying proportional share mechanism taking the resource away from the real time scheduler at an inopportune and unexpected time in the name of fairness.

In this lecture, we will discuss about SMART which takes care of above thing.

SMART: A CPU scheduler for Multimedia Applications

Application requirements

The workloads on computers is changing rapidly. In the past, most of the applications were textual based such as word and accounts processing in the offices. These applications require only a best-effort service and they are easy to schedule. But nowadays, images, video, audio and interactive graphics have become common place. A growing number of multimedia applications are available. These applications have strict latency requirements. It is difficult to schedule an application mix which consists of both textual and real time applications.

Scheduler demands

The salient features of multimedia applications and their special demands that distinguish them from the conventional applications current operating systems are designed for:

1. *Soft real-time constraints* : Real-time applications have application-specific requirements that need to be met. The data is useful only when received on time. Time constraints may be periodic or aperiodic in nature. Missing a deadline diminishes the quality of the results and does not lead to catastrophic failures.

2. *Insatiable resource demands and frequent overload* : These applications present practically an insatiable demand for resources. The applications such as real time video are highly resource intensive and can consume the resources of an entire machine, resources are commonly overloaded, with resource demand exceeding its availability.

3. *Dynamically adaptive applications* : When resources are overloaded and not all time constraints can be met, multimedia applications are often able to adapt and degrade gracefully by offering a different quality of service.

4. *Co-existence with conventional computations* : Real-time applications must share the desktop with already existing conventional applications.

SMART : Principles

1. *Priority* : The system should not degrade the performance of a high priority application in the presence of a low priority application.
2. *Proportional sharing among real-time and conventional applications iin the same priority class*.
3. *Graceful transitions between fluctuations in load* : The system load varies dynamically, new applications come and go, and the resource demand may also fluctuate. The system should be able to adapt to the changes gracefully.
4. *Provide QoS* : It should be able to meet deadlines and should provide low latencies.
5. It should be able to trade off instantaneous fairness for better real-time and interactive response time.
6. Notification of resource availability : It should allow the applications to specify if and when they wish to be notified if it is unlikely that their computation will be able to complete before their given deadlines.

SMART Operation

The process that is to be scheduled has two dimensions: importance and urgency. An urgent task is one which has an immediate real-time constraint. An important task is one with a high priority, or one that has been least serviced proportionally among applications with the same priority. An urgent task may not be the one to execute it it requests more resources than its fair share. Same way, impoirtant task can not be run immediately. So, urgency is different from importance. So, process schedule = f(importance, urgency). For importance dimension, we can use priorities and weighted fair queuing. For urgency, we can use EDF scheduling.

SMART Scheduling Principles

1. Each task is assigned a tuple (P,S).
   - P: priority assigned to a process.
   - S: share of CPU bandwidth within a priority level.
2. Active tasks progress at the rate of 1/S.
3. It is kind of similar to WFQ (weighted fair queuing.
4. Virtual finishing time (VFT): virtual time + Q/S
5. We can use bias to compute VFT. Bias is a measure of delay tolerance of tasks. Real-time tasks have zero-bias. Bias is reset to zero on user interaction. It is used to specially handling for tasks using less than their fair share.
6. Biased VFT (BVFT) : VFT + b/S

SMART Scheduling Algorithm

The SMART scheduling algorithm used to determine the next task to run is as follows:

- If the task with the highest value-tuple is a conventional task (a task without a deadline), schedule that task.

- Otherwise, create a candidate set consisting of all real-time tasks with higher value-tuple than that of the highest value-tuple conventional task. (If no conventional tasks are present, all the real-time tasks are placed in the candiadate set.).
- Apply the best-effort real time scheduling algorithm on the candidate set, using the value-tuple as the priority in the original algorithm. By using the given deadlines and service time estimates, find the task with the earliest deadline whose execution does not cause any tasks with higher value-tuples to miss deadlines. This is achieved by considering each candidate in turn, starting with the one with the higher value-tuple. The algorithm attempts to schedule the candidate into a working schedule which is initially empty. The candidate is inserted in deadline order in this schedule provided its execution does not cause any of the tasks in the schedule to miss its deadline. The scheduler simply picks the task with the earliest deadline in the working schedule.
- If a task cannot complete its computation before its deadline, send a notification to inform the respective application that its deadline cannot be met.

COMPLEXITY : The cost of scheduling with SMART consists of managing the value-tuple list and the cost of managing the working schedule.

Managing the tuple list takes O(N) time. But it can be reduced to O(log N) by using a tree data structure. N is the number of active tasks. It is same as in WFQ.

Creating the work schedule takes O(Nr**2) time. But it can be reduced to O(Nr) using optimizations. Nr is the number of active real-time tasks of higher value than the highes value conventional task.

SMART Interface

1. Application developer interface.
   - Time constraints: application provides time bounds.
   - Notifications: scheduler provides notification of constraint failures.
2. User interface
   - Priorities: strict priority schdeduling.
   - Weights : CPU bandwidth is distributed in proportion to weights within a priority level.

Performance Results

Application mix consists of

1. Batch: Dhrystone (CPU intensive)
2. Interactive: Typing (I/O intensive)
3. Soft real-time : media player.

SMART performs better than UNIX TS, SVR4-RT and WFQ. More detailed results are given in paper ”The design, implementation and evaluation of SMART: A Schfulere for multimedia applications” by Jason Nieh adn Monica S. Lam.

The experiments show that SMART:

- reduces the burden of writing adaptive real-time applications.
- has the ability to cooperate with applications in managing resources to meet their dynamic time constraints.
- provides resource sharing across both real-time and conventional applications.
- delivers improved real-time and interactive performance over other schedulers without any need for users to reserve resources, adjust scheduling parameters, or know anything about application requirements.
- provides flexible, predictable controls to allow users to bias the allocation of resources according to their preferences.