Absolute Bandwidth Scheduling via Group-based Partitioned Proportional Share Scheduling and Dynamic Weight Management on Varying-Speed Multiprocessors

Sangwon Shin, Shakaiba Majeed, and Minsoo Ryu*
Department of Computer Science and Engineering, Hanyang University, Seoul, Korea
{swshin, shakaiba}@rtcc.hanyang.ac.kr, msryu@hanyang.ac.kr

Abstract - *For the past few years there has been an increase in the use of compute intensive applications running on high-performance embedded systems based on multi-core platforms. These applications demand an absolute share of CPU bandwidth to guarantee a certain level of QoS (Quality of Service) and fulfil their timing constraints. Unfortunately, traditional proportional share or priority scheduling algorithms employed in general purpose operating systems are not able to provide an absolute share of processor resources for such time sensitive tasks. In this paper, we present an absolute bandwidth scheduling scheme which aims at providing an absolute share of CPU bandwidth to groups of soft real-time tasks regardless of the work load conditions and varying speeds of CPU. The proposed scheme provides a mechanism of CPU bandwidth allocation for groups of soft-real time tasks by dynamically changing the overall weight of the group while maintaining the proportion of share of each task in the group. Our proposed approach works on top of a traditional proportional share scheduler and does not require any modifications to the kernel layer. To demonstrate the effectiveness and the correctness of our scheme, we have implemented a prototype using Linux cgroups and the existing completely fair scheduler (CFS). A series of experiments are conducted to prove that each soft real-time task in the group maintains its required absolute bandwidth.*

Keywords: Absolute bandwidth scheduling, group-based proportional share scheduling, dynamic weight management, QoS, soft real-time.

1 Introduction

For the past few years there has been an increase in the use of compute intensive applications such as multimedia processing, online gaming and data encryption/decryption running on high-performance embedded systems based on multi-core platforms. These applications demand an absolute share of CPU bandwidth to guarantee a certain level of QoS (Quality of Service) and fulfil their timing constraints. For instance, a video application may require 30% of processor bandwidth at 1GHz to decode 30 frames per second for smooth playback.

Unfortunately, traditional proportional share or priority scheduling algorithms employed in general purpose operating systems are not able to provide an absolute share of CPU resources for time sensitive tasks. The proportional share algorithm aims at providing relative fairness to the tasks proportional to their weights, for distributing CPU bandwidth. As a result, the relative share of each task decreases as the system load increases.

The second hindrance in allocating a guaranteed share of CPU bandwidth is the assumption of fixed speed or performance of each CPU core. In a real world system, however, the speed of CPU may vary with new tasks being dynamically added to the CPU and the underlying DVFS (dynamic voltage and frequency scaling) policy. As a result, each task may generate a different utilization value depending upon the current speed of the CPU. Consider for example that a soft real-time task such as an HVEC (High Efficiency Video Coding) decoder demands 60% of CPU utilization on ARM Cortex A9 processor running at maximum frequency of 1.5 GHz. If the task is scheduled on a CPU which is running at 1.0 GHz, HVEC may not meet its performance metrics because CPU utilization in that case should be 90%. This may adversely affect the resource allocation problem especially in multi-core environments where CPUs can run at varying speeds and task migration occurs frequently. Hence, to provide efficient CPU bandwidth allocation to tasks, the individual as well as overall computation performance of all the CPU resources in a system under a certain condition must be take into account.

In this paper we propose an absolute bandwidth scheduling (ABS) scheme to accomplish absolute bandwidth guarantees on top of an existing partitioned proportional share scheduler. The proposed scheme performs two important functions. First, it partitions tasks running on each processor into two groups, absolute bandwidth tasks that require absolute bandwidth guarantees and proportional share tasks that require traditional proportional share services. Second, it dynamically adapts the weights of absolute bandwidth task groups to satisfy their absolute bandwidth requirements taking into account the dynamic change in workload characteristics and varying processor speeds. We show through experimental evaluation that the proposed scheme can efficiently achieve absolute bandwidth guarantees in conjunction with an existing proportional share scheduler.

The rest of this paper is organized as follows: section 2 gives a brief review of the related work, section 3 elaborates our proposed scheme. In section 4 we give an implementation of the scheme, section 5 explains our experimental set up and finally we conclude with section 6.

2 Related Work

Many researchers proposed resource reservation schemes to provide guaranteed allocation of resources to a group of tasks present in an application. For example, the processor capacity reserve scheme proposed by Mercer et al. [2] suggests using priority based scheduling to grant resource reservation to each application. Later a kernel module is used to keep track of the CPU usage of each application to implement the granted reservation. A similar approach, called ACTORS [3], reserves resources for each application and readjust the assignment by using feedback from the application. This solution gives user a comprehensive control over resource allocation but the technique requires non-trivial modification to the kernel.

To provide a constant share of CPU bandwidth regardless of the workload conditions a fixed share scheduling (FSS) policy has been proposed in [4]. FSS enables a traditional proportional share scheduler to provide constant share of CPU bandwidth by dynamically modifying the weights of soft real-time tasks under changing workload conditions. However, FSS assumes that the speed or performance of the CPUs remains fixed over time. Note that the dynamic voltage frequency scaling (DVFS) mechanism implemented in modern computing systems scales up or down the speed of CPUs as new tasks arrive and/or depart. Hence, to provide absolute CPU bandwidth allocation to tasks, dynamic CPU speeds must be taken into account.

3 Absolute Bandwidth Scheduling

There are two general approaches to multiprocessor scheduling, global scheduling and partitioned scheduling. The global scheduling approach maintains single task queue and schedules tasks selecting eligible tasks guided by a global scheduling policy and assigning them to appropriate processors. On the other hand, the partitioned scheduling approach maintains a separate task queue for each processor and schedules tasks in a way similar to distributed scheduling. In a partitioned scheduling system, tasks are first assigned to processors and each processor runs a separate scheduler instance to schedule them independently of other processors.

In this work, we consider partitioned scheduling systems with proportional share scheduling support. In order to satisfy absolute processor bandwidth requirements in such partitioned proportional share scheduling systems, we present a group-based proportional scheduling scheme with dynamic weight management as described below

3.1 Group-based Proportional Share Scheduling

The goal of group-based proportional scheduling schemes is to meet the performance requirements of a group of tasks within applications. The key idea of group-based proportional share scheduling is to allocate resources to task groups relative to their weights such that the share of each group is defined by a proportional share with respect to the other groups present in the system. The share of each task within a group is defined by a proportional share of its parent group [4]. For example, let W_{g_i} be the weight of group g_i and let G be the set of all active groups at time t, then the share $s_{g_i}(t)$ of a group g_i at time t is defined as below.

$$s_{g_i}(t) = \frac{W_{g_i}}{\sum_{j \in G} W_{g_j}}$$

Let w_{τ_i} be the weight of task τ_i and T be the set of all active tasks included in group g_i at time t. The share $s_{\tau_i}(t)$ of a task τ_i with weight w_{τ_i} at time t is defined as below.

$$s_{\tau_i}(t) = s_{g_i}(t) \times \frac{w_{\tau_i}}{\sum_{j \in T} w_{\tau_j}} \qquad (1)$$

3.2 Dynamic Weight Management for Groups

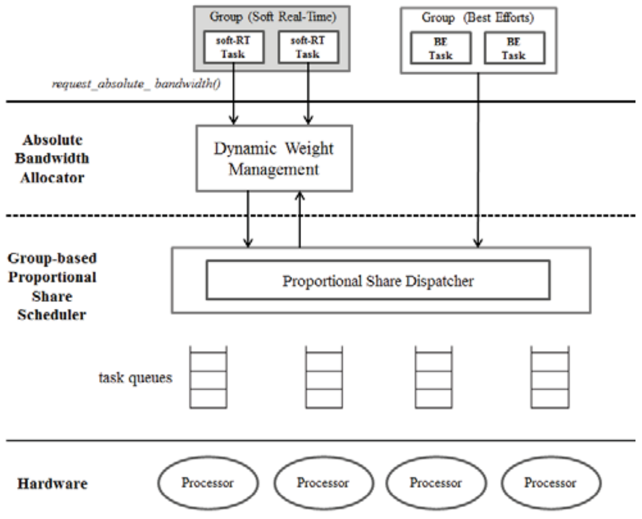

Group (Soft Real-Time)
soft-RT Task
soft-RT Task
Group (Best Efforts)
BE Task
BE Task
request_absolute_bandwidth()
Absolute Bandwidth Allocator
Dynamic Weight Management
Group-based Proportional Share Scheduler
Proportional Share Dispatcher
task queues
Hardware
Processor
Processor
Processor
Processor

Figure 1. The absolute bandwidth scheduling model.

In order to obtain absolute bandwidth allocation guarantees from the existing proportional share scheduler, we propose to add an absolute bandwidth allocator on top of it with a minimal impact on the existing system architecture. The primary goal of the absolute bandwidth allocator is to receive absolute bandwidth requirements from the soft real-time tasks and changing the weights of groups of tasks by examining the run queue of each processor and its current speed.

Figure 1 shows the proposed absolute bandwidth scheduling model with an absolute bandwidth allocator running on top of an existing group-based proportional share scheduler on a multiprocessor platform. The platform consists of m processors $P = \{P_1, P_2, \dots, P_m\}$. Each processor has identical maximum processing speed F_{max} but can be operating on varying speeds depending on the workload conditions and DVFS (dynamic voltage and frequency scaling (DVFS) mechanism. Let us denote the current frequency of a processor by F_{p_i}.

For the proposed model, we define a set of active tasks $T = \{\tau_1, \tau_2, \dots, \tau_m\}$ running on the multiprocessor platform and divide these tasks into two groups such that

$$T = T_{ab} \cup T_{ps}$$

where T_{ab} is a group of soft real-time tasks that require absolute bandwidth guarantees and T_{ps} is a group of best effort tasks that require proportional share bandwidth guarantees.

Each processor P_i has a separate task run queue T_{p_i} such that

$$T_{p_i} = T_{abp_i} \cup T_{psp_i}$$

where T_{abp_i} is a group of soft real-time tasks allocated to processor P_i and T_{psp_i} is a group of best effort tasks allocated to processor P_i. The best effort tasks are allocated bandwidth shares in proportion to their weights in accordance with the existing group-based proportional share scheduling scheme. We denote the weight of a best effort task $\tau_i \in T_{psp_i}$ by w_{τ_i} so that the overall weight of the best effort group T_{psp_i} scheduled at P_i is given by

$$W_{psp_i} = \sum_{\tau_i \in T_{psp_i}} w_{\tau_i}.$$

The goal of absolute bandwidth scheduling is to provide an absolute bandwidth to soft real-time tasks. This can be achieved by receiving a absolute bandwidth requirement from each task and then dynamically changing the weights of each soft real-time group to maintain the requested bandwidth requirements. We define the absolute bandwidth request by any software task $\tau_i \in T_{abp_i}$ as CPU utilization U_i required from a CPU when running at F_{max}. In the proposed model each task with in T_{ab} may put such request through an application programming interface (API) request_absolute_bandwidth() as shown in Figure 2.

For a group of soft real-time tasks assigned to a processor P_i, the absolute CPU utilization U_{abp_i} is defined by

$$U_{abp_i} = \frac{F_{max}}{F_{p_i}} \times \sum_{\tau_i \in T_{abp_i}} U_i. \quad (2)$$

It is worth noting that when dispatching the tasks to a specific processor it is necessary that the sum of bandwidth requests $\sum_{\tau_i \in T_{abp_i}} U_i$ cannot be greater than 1. However, the resulting U_{abp_i} can be greater than 1 in case when the current operating frequency F_{p_i} is less than the maximum speed F_{max} of CPU. This will enable DVFS to increase the operating frequency of CPU as required.

Having obtained the absolute CPU utilization U_{abp_i} of a soft real-time group, the weight of the group can be obtained as

$$W_{abp_i} = \frac{U_{abp_i} \times \sum_{\tau_i \in T_{psp_i}} w_{\tau_i}}{1 - U_{abp_i}}. \quad (3)$$

The group weight obtained from Equation (3) guarantees that each soft real-time task gets an absolute bandwidth allocation and the other best effort tasks present on the run queue are assigned a proportional share of the CPU bandwidth.

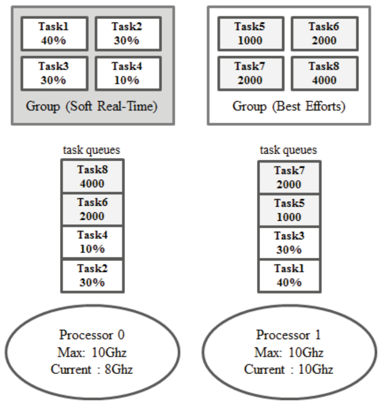


Figure 2. Examples of absolute bandwidth scheduling

Figure 2 shows an example of the proposed absolute bandwidth scheduling model. Consider two sets of tasks $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ and $T_{ps} = \{\tau_5, \tau_6, \tau_7, \tau_8\}$ running on a dual-processor platform. Maximum frequency of each processor, F_{max} is 10 GHz while current frequency of P_0 is $F_0 = 8$ GHz and current frequency of P_1 is $F_1 = 10$ GHz .

Soft real task τ_2 and τ_4 are allocated to processor P_0 and have absolute bandwidth requirements of $U_2 = 0.3$ and $U_4 = 0.1$ at 10 GHz, and best effort task τ_6 and τ_8 are allocated to processor P_0 and have weights $w_{\tau_6} = 2000$ and $w_{\tau_8} = 4000$. By applying Equation (2) to group of soft real-time tasks, we have $U_{abp_0} = \left(\frac{10\times(0.3+0.1)}{8}\right) = 0.5$ and by applying Eq. (3), we get $W_{abp_0} = \left(\frac{8\times0.5\times(2000+4000)}{8-8\times0.5}\right) = 6000$. By using this weight value for the group we obtain the share of each soft real-time task present in the group as desired. We can verify the result by applying Equation (1): $s_{\tau_2} = \frac{6000}{6000+6000}\times\frac{0.3}{0.3+0.1} = 0.375$ and $s_{\tau_4} = \frac{6000}{6000+6000}\times\frac{0.1}{0.3+0.1} = 0.125$ and $s_{\tau_6} = \frac{6000}{6000+6000}\times\frac{2000}{2000+4000} = 0.167$ and $s_{\tau_8} = \frac{6000}{6000+6000}\times\frac{4000}{2000+4000} = 0.333$.

Soft real task τ_1 and τ_3 are allocated to processor P_1 and have absolute bandwidth requirements of $U_1 = 0.4$ and $U_4 = 0.3$ at 10 GHz, and best efforts task τ_5 and τ_7 are allocated to processor P_1 and have weights $w_{\tau_5} = 1000$ and $w_{\tau_7} = 2000$. By applying Equation (2) to group of soft real-time tasks, we have $U_{abp_1} = \left(\frac{10\times(0.4+0.3)}{10}\right) = 0.7$ and by applying Eq. (3), we get $W_{abp_1} = \left(\frac{10\times0.7\times(1000+2000)}{10-10\times0.7}\right) = 7000$. Using this weight value we can verify the share of each soft real-time task by applying Equation (1): $s_{\tau_1} = \frac{7000}{7000+3000}\times\frac{0.4}{0.4+0.3} = 0.4$ and $s_{\tau_3} = \frac{7000}{7000+3000}\times\frac{0.3}{0.4+0.3} = 0.3$ and $s_{\tau_5} = \frac{3000}{7000+3000}\times\frac{1000}{1000+2000} = 0.1$ and$s_{\tau_7} = \frac{3000}{7000+3000}\times\frac{2000}{1000+2000} = 0.2$.

4 Implementation

We have implemented a prototype of the proposed scheme on Linux kernel 3.18.3 using control groups (cgroups). Cgroups provide a mechanism to aggregate or partition tasks into hierarchical groups categorized by their peculiar behavior primarily for the purpose of efficient resource management among tasks.

To exploit the benefits of cgroups and to efficiently manage CPU resources we implemented the absolute bandwidth allocation scheduling using CPU subsystem of cgroups. However, we add two new parameters to the existing CPU subsystem. The first parameter cpu.softRT is used to classify each task on the system as a soft real-time or best effort task. The other parameter cpu.absoluteBW is implemented as a structure and indicates absolute bandwidth requirement of each task in a group.

In order to provide absolute bandwidth allocation to soft real-time tasks we implemented dynamic weight management for groups on top of the existing Linux's completely fair scheduler (CFS) [8]. Our addition of parameters in CPU subsystem of cgroups and the process of dynamic weight management does not modify the existing kernel implementation and has no impact on the existing system if newly defined parameters are not used.

5 Experimental Evaluation

We evaluated the effectiveness and correctness of the implementation of absolute bandwidth scheduling scheme by conducting a series of experiments. These experiments were performed on an Intel Core i5-4690 CPU which has four cores with 3.50 GHz maximum speed, running on Linux 3.18.3. Each set of experiments was conducted 10 times to ensure that the experiments and their results are repeatable.

In the first set of experiments, we observed the CPU utilization of soft real-time tasks under varying workload conditions. We used four target tasks $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with absolute bandwidth requirement $U_{ab} = \{60\%, 60\%, 50\%, 40\%\}$ at 3.5 GHz. Each task used in this experiment is a busy-waiting task which consumes 100% of CPU utilization when executed alone on a processor. Each processor was running on its maximum speed. We started adding new best efforts tasks at 1000 millisecond and noticed that the actual CPU utilization of target tasks is maintained to their demanded absolute bandwidth even though the number of running best efforts tasks on each CPU was dynamically changing. Notice that since the processors are running on their maximum frequency hence the actual CPU utilization and the absolute bandwidth utilization demanded by tasks is same. Figure 3 shows that actual utilization of target soft real-time task is maintained around absolute bandwidth even though the number of tasks varies dynamically.

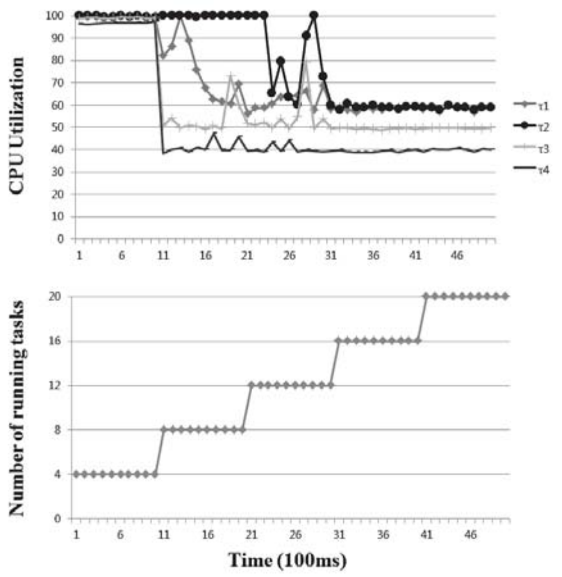
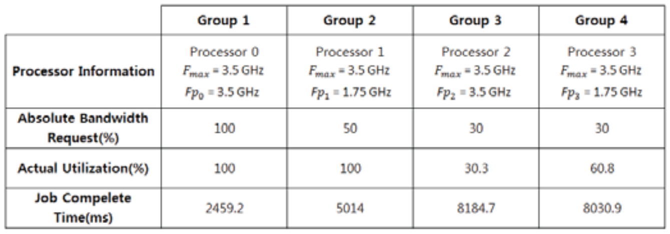


Figure 3. CPU utilization of soft real-time tasks, number of running tasks in the proposed scheme

In the second set of experiments, we observed the CPU utilization and job completion time of target soft real-time tasks under the assumption of varying CPU speeds. We created four soft real-time tasks $T_{ab} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with absolute bandwidth requirement $U_{ab} = \{100\%, 50\%, 30\%, 30\%\}$ at 3.5 GHz. Each task was added to a separate group destined to schedule on a particular CPU using processor affinity. To simulate that two of the CPUs, CPU1 and CPU3 are running at slower speeds, we used a value of loop counter inversely proportional to the desired frequency in the busy-waiting loop of soft real-time tasks, such that the current frequency of CPU1 and CPU4 is 3.5 GHz and 1.75 GHz for CPU1 and CPU3.

Table 1 shows that for CPU1 and CPU3, the actual utilization of soft real-time tasks at simulated current frequency is increased as a result of applying Equation (2). We then used this utilization value to recalculate the weights of the soft real-time task groups scheduled at CPU1 and CPU3 to maintain their initial absolute bandwidth request. The last row in Table 1 shows the relationship between the absolute bandwidth requirement and job-completion time of each task; for a given operating frequency, the smaller the absolute bandwidth request the longer it takes to complete the task.

Table 1. CPU utilization and job completion time of each soft real-time groups



	Group 1	Group 2	Group 3	Group 4
Processor Information	Processor 0 F_{max} = 3.5 GHz Fp_0 = 3.5 GHz	Processor 1 F_{max} = 3.5 GHz Fp_1 = 1.75 GHz	Processor 2 F_{max} = 3.5 GHz Fp_2 = 3.5 GHz	Processor 3 F_{max} = 3.5 GHz Fp_3 = 1.75 GHz
Absolute Bandwidth Request(%)	100	50	30	30
Actual Utilization(%)	100	100	30.3	60.8
Job Compelete Time(ms)	2459.2	5014	8184.7	8030.9

6 Conclusions

In this paper, we have presented an absolute bandwidth scheduling scheme which guarantees absolute bandwidth for a soft real-time tasks and proportional bandwidth allocation to best effort tasks. We implemented absolute bandwidth scheduling using the notion of group-based scheduling and by dynamically changing the weights of groups of soft real-time tasks on top of the existing Linux CFS scheduler. We demonstrated with our experiments that the soft real-time tasks maintain their required absolute bandwidth when more tasks were added on the system and even when the current speed of the underlying CPUs was changed.

Acknowledgment

This work was supported partly by Seoul Creative Human Development Program (HM120006), partly by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (NRF-2011-0015997), and partly the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) (IITP-2015-H8601-15-1005) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600-625, 1996.

[2] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," in *Multimedia Computing and Systems, 1994., Proceedings of the International Conference on*, 1994, pp. 90-99.

[3] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, *et al.*, "Resource management on multicore systems: The ACTORS approach," *IEEE Micro*, vol. 31, pp. 72-81, 2011.

[4] H. Kim, H. Yoon, P. Wu, and M. Ryu, "Fixed Share Scheduling via Dynamic Weight Adjustment in Proportional Share Scheduling Systems."

[5] P. Holman and J. H. Anderson, "Group-based pfair scheduling," *Real-Time Systems*, vol. 32, pp. 125-168, 2006.

[6] J. Kay and P. Lauder, "A fair share scheduler," *Commun. ACM*, vol. 31, pp. 44-55, 1988.

[7] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, pp. 440-459, 2014.

[8] C. S. Pabla, "Completely fair scheduler," *Linux Journal*, vol. 2009, p. 4, 2009.