ISSN: 0974-2115
www.jchps.com Journal of Chemical and Pharmaceutical Sciences

Integration of feedback control loop with proportional share schedulers to fine tune the QOS of real-time application

V.L.Jyothi[1], S.Chockalingam[2]
1Department of Computer Science & Engineering, Jeppiaar Engineering College, Chennai, Tamilnadu, India
2Software Engineer, CISCO Systems Pvt. Ltd, Chennai.
***Corresponding author: E-Mail: jyothivl15@yahoo.com**

ABSTRACT

Proportional share scheduling works based on predefined shares of applications. It faces the challenge of setting reasonable share for a set of tasks with dynamically changing resource requirements. Hence, it is not widely accepted for general purpose systems. We propose a mechanism to set the CPU share (weight) based on the progress metrics of an application. The weight allocates the required processor cycles. As the workload characteristics change, the proportion needs to be recomputed and reallocated to the tasks. Hence, we include a feedback mechanism to meet current resource needs of any application. The PS scheduler is restructured to integrate with the feedback mechanism. The environment for the feedback scheduler is identified to achieve high throughput and to minimize service time error. The performance of feedback scheduler is evaluated and its related metrics are analyzed. We tested the performance with a mix of workload which includes real time and non real time tasks.

KEY WORDS: Proportional Share Schedulers, Real-Time Scheduling, Feedback Control Loop, Feedback Schedulers, Dynamic Resource Allocation.

1. INTRODUCTION

Proportional share resource allocation is particularly well suited to the problem of providing real time services because its underlying scheduling mechanism is a quantum based round robin like scheduler. Much of this work is rooted in an idealized scheduling abstraction called generalized processor sharing(GPS) (Parekh A. and R. Gallager 1993). Under GPS, scheduling tasks are assigned weights, and each task is allocated a share of th resource in proportion to its weight. Thus, each task's designated share is guaranteed (fairness) and any misbehaving task is prevented from consuming more than its share. Recently, many algorithms such as SFQ (Bennett and Zhang, 1996), SFS (Abhishek Chandra 2000), SMART (Nieh, 2003), DFS (Micah Adler 2004) are proposed based on the concept of generalized processor sharing (GPS).

GPS based algorithms are extensively used for real time systems. It is not widely accepted for general purpose systems. The reason is difficulty in estimating the correct weight assignment. In this paper, we have described a technique to dynamically estimate the proportion of processor capacity required for an application.
We employ a feedback mechanism to a proportional scheduler to estimate the required proportion by an application. However, many of today's applications have time-varying resource demands. This means that the resource must be dynamically allocated. The idea of dynamically allocating the resource based on the progress is proposed by (Steere, 1999).

Recently, there has been a lot of interest in feedback control theory and it has been successfully applied to several computer system projects. At the network layer, Hollot, (Hollot, 2001) applied control theory to analyze the RED congestion control algorithm on IP routers. The idea of using feedback information to adjust the schedule has been used in general-purpose operating systems in the form of multi-level feedback queue scheduling (Blevins, 1976). The system monitors each task to see if it consumes a time slice or does I/O and adjusts its priority accordingly. This type of control seems to work via ad hoc methods. No systematic study has been done so far. In the area of CPU scheduling, Steere, (Steere, 1999) developed a feedback based CPU scheduler. It synchronizes the progress of consumers and producer process of buffers. Recently real-time application has become a focus area of feedback control because the unpredictability of the workload.

Related Work: The related work (real time systems with feedback control) falls into three types: Integrated control and real time system design, flexible and adaptive real time system algorithms and architectures and QoS approaches in real time systems.

Cervin (2002), have considered sampling period selection for a set of control tasks. The performance of a task is given as a function of its sampling frequency, and an optimization problem is solved to find a set optimal task periods. Co-design of real time control systems is considered by Liu, J, (Liu, 2003), wherein the multiple tasks are expressed as functions of time for real-time embedded software. Shin and Meissner (Shin, Meissner 1999) deal with on-line rescaling and relocation of control tasks in a multi-processor system. The elastic model of Buttazzo, (Buttazzo (2000) allows run time task timing adjustment in order to improve schedulability and thereby enhance the control performance. However, its task timing constraints do not incorporate information in terms of control performance. Another approach proposed by Pau Marti, (2002) forwards the idea of flexible timing constraints for a set of control tasks.

The second area relates to the wealth of flexible scheduling algorithms available. Buttazzo, (Buttazzo.G 2000) proposes an elastic task model for periodic tasks. Stankovic, (Stankovic, 1999) present a scheduling algorithm, the FC-EDF, that explicitly uses feedback in combination with EDF scheduling. The same approach is extended using an additional proportional integral derivative (PID) by Lu, (Lu, Stankovic 2002). The optimization of a control system's performance subject to schedulability has been treated by Rehbinder, (Rehbinder, 2000), Seto, (Seto, 1996).

The third area of related work is on QoS aware real-time software. In order to maximize the performance, the resource allocation is adjusted online. Feedback control theory has been applied to solve performance or QoS problems in computing systems Hellerstein, (Hollot, 2001), Abdelzaher, (Abdelzaher, 2000), Miguel D (Miguel de, 2002) analyzes QoS-Aware Component Frameworks. Proportional share schedulers offer weaker guarantees for applications with time constraints than the traditional real time based schedulers. However, they tend to be more flexible and ensure a graceful degradation in overload situations (Nieh, 2003). A high level architecture with feedback control loop to allocate the appropriate cpu share as required by the application is proposed (Jyothi, 2007)

Thus, the literature survey differentiates the work pertaining to real time systems with feedback control into three types. Our approach is to apply the feedback control to fine tune the QoS or performance of a real time application.

2. METHODS AND MATERIALS

Feedback Scheduling Framework: The goal of the feedback scheduler is to maximize CPU utilization, throughput, minimize deadline miss ratio and service time error. The goal is achieved by maintaining the execution rate at a desired level by manipulating the CPU share. The basic scheduler used is proportional share scheduler. The scheduler with the feedback control loop is termed as feedback scheduler. The feedback scheduler estimates the correct proportion of the weight from the execution rate of a task.

To apply feedback control techniques in scheduling, schedulers must be restructured based on the feedback control framework. Feedback scheduling uses feedback control loop to schedule the task. The CPU share required by the task is estimated and manipulated by the feedback control loop. The CPU share is manipulated in order to maintain the execution rate at the desired level.

To apply feedback control to a scheduling system, the components corresponding to Figure 1 have to be decided. The controlled variable and set point of the system must be selected. The requirements of real-time scheduling algorithm are to achieve i) high CPU utilization ii) high system throughput iii) low miss ratio and iv) low service time error. To satisfy these requirements, our feedback scheduler chooses execution rate of a task as the set point. We control the progress of execution rate to the set point so as to maximize the CPU utilization and minimize the deadline miss ratio.

Feedback Control Scheduling Architecture: Feedback control CPU scheduling consists of: a task model, a set of control related variables, a feedback control loop that maps a feedback control system structure to real-time CPU scheduling, and a basic scheduler. Figure 1 shows the architecture of a feedback scheduler.

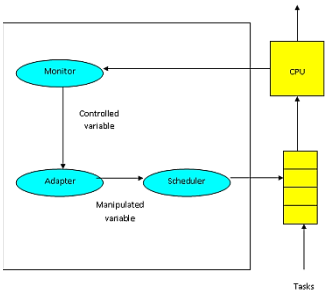

Figure.1.Architecture of a feedback scheduler

Task Model: Our workload model consists of real time and non-real time applications. These applications have very diverse characteristics. Real time applications have some well-defined computation that must be completed before an associate deadline. The goal of real time applications is to complete their computations before their respective deadlines. If it is not possible to me,l deadlines, it is generally better to complete as many computations as possible by their respective deadlines. In contrast, non-real-time applications have no explicit deadlines.

- Each task is assigned a unique id
- The period of a task is the deadline of a task for real time task.
- The weight or the CPU share is estimated by the feedback controller. As the process executes, the weight is dynamically changed.
- The target rate is expressed in terms of execution rate. For a real time task, it is the number of frames processed per second. For a non real time task, it is the number of lines processed per second.
- The sampling interval is defined for invoking the feedback control loop. The feedback control loop is tested for various sampling intervals of different durations.

Control Related Variables: To apply feedback control techniques in scheduling, schedulers need to be restructured based on the feedback control framework. The function of the feedback control loop depends on the controlled variable and manipulated variable. The choice of the controlled variable depends on the system goal. The goal of our system (feedback scheduler) is to maintain a high CPU utilization. The execution rate of an application can be used to determine CPU utilization. By controlling the execution rate of an application the goals of the system can be met. Hence, we define the execution rate as the controlled variable. The target rate in terms of execution rate is defined as the set point.

The execution rate of a task highly depends on the CPU share or its weight. By controlling the execution rate of a task to the desired execution rate, the task receives its appropriate CPU share. In other words, CPU shares are manipulated so as to control the progress execution rate to the set point. Hence, CPU share is used as the manipulated variable.

To facilitate the design of feedback scheduler, variables related to our objectives are determined. The list of variables is depicted in Table 1.

Table.1.Control related variables

Set point	Target execution rate
Controlled variable	Progress execution rate
Manipulated variable	CPU share

Feedback Control Loop: The main objective of our feedback control loop is to minimize the deviation of progress execution rate from the target execution rate. The minimum deviation improves the performance of a task. Feedback control loop comprises of a monitor and an adapter. It is activated at every sampling interval. To apply feedback control techniques in scheduling, schedulers need to be restructured based on the feedback control framework.

Function of Feedback Control Loop: Each feedback control loop is composed of a monitor, and a controller/adapter. The monitor and adapter are a part of feedback control loop and they are executed whenever the feedback control loop is called.

- The monitor measures progress metrics from the application and feeds the samples back to the adapter.
- The adapter adjusts each application's proportion automatically based on the progress measure. It tunes the scheduler to allocate the CPU cycles required for an application.

The feedback control loop is responsible for the dynamic CPU allocation for the real-time and non-real-time threads. The dynamic CPU allocation is expressed in terms of weight. The scheduler allocates the desired CPU cycles to a thread in proportion to its weight. The weight for a particular thread is calculated by te monitor and it is adjusted by the adapter.

Monitor: The monitor in the feedback control loop is responsible for estimating the proportion of CPU required by an application. Our feedback control loop assigns the fraction of CPU to an application based on the rate of execution. We characterize the execution rate as target execution rate and progress execution rate.

The processor percentage consumed by an application over a period of time is defined as progress execution rate. The target execution rate is fixed for every application. As the application executes with mix of workload, its progress rate of execution may vary. The workload corresponds to real time and non-real time applications.

Figure 2 shows the block diagram of a monitor in the feedback control loop. The monitor includes a progress rate calculator which measures the rate of progress execution of an application. The progress execution rate is then fed into an adapter which compares it with the target rate. The target execution rate is directly applied to the adapter in the feedback control loop.

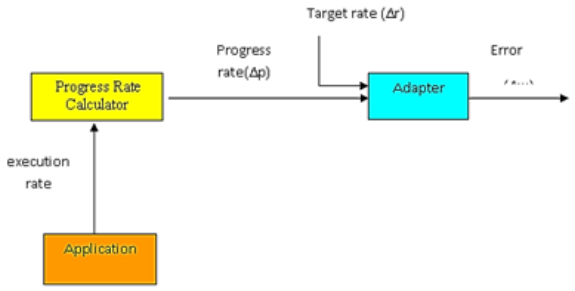

Figure.2. Block Diagram of monitor

The progress rate of an application is sampled every 100 ms. It is measured as the amount of progress (Δp) made in a particular interval. The progress rate calculator, during each sampling interval calculates the amount of progress made in a particular interval and computes the average progress rate. The average progress rate is measured as the sum of the progress rate made at various intervals.

Progress rate calculator employs simple moving average to calculate the progress in the rate of execution made by an application.

The simple moving average is an equally weighted sum of the previous k data points. We represent the amount of progress made in a particular interval as Δp. The weight at each data point equals 1/k.

$$\Delta p(t)=1/k *(\Delta p(t) + \Delta p(t-1) + \ldots + \Delta p(t-k-1)) \quad (2)$$

$$\Delta p(t) = 1/k \ *\left(\Delta p(t) \ + \ \Delta p(t-1) \ + \ \ldots \ + \ \Delta p(t-k-1)\right)$$

Where, Δp(t)=amount of progress at t^{th} interval and K=number of data points or observations.

The amount of progress made by a thread is measured as the average sum of the amount of progress made at the previous intervals. The ideal target execution rate is determined for all the applications. Ideally, this target rate represents the expected progress rate.

The ideal target rates for various multimedia applications are tabulated as follows. The test sequences are taken for motion pictures experts group (MPEG) files are shown in Table 3.

Test Sequence - 1, ideal target rate →25 frames/sec.

Table.3.Test Sequences

Number of frames	150
horizontal_size	352
vertical_size	288
Bit rate	1152000.0(bits/s)

Monitor employs a progress rate calculator to measure the average progress execution rate. The target execution rate represents ideal progress rate. The ideal target execution rate for an application can be obtained from a test sequences. The target execution rate can be set by the user depending on the preferred execution rate for an application. Thus, monitor in the feedback control loop computes the progress measure of an application based on its execution rate.

Adapter: The adapter adjusts each application's proportion of processor capacity based on the progress measure. The progress measure includes the progress execution rate and target execution rate. The main function of the adapter in the feedback control loop is to compute the difference between the progress execution rate and target execution rate. The difference is termed as error. The error determines the rate of execution of an application. That is, the error will decide that whether the application is progressing towards the ideal target rate. The larger the deviation of progress rate from the target rate results in slower performance of an application. Lesser deviation improves the performance of an application.

The error which is computed based on the progress measure of an application is then mapped into CPU share. Hence, CPU share is assigned based on the progress measure of an application. It then tunes the scheduler to allocate the CPU cycles required for an application. It maps the progress measure of the execution rate to the change in the CPU share so as to drive the execution rate back to the set point.

The error is determined from the target and the progress of the execution rate of an application.

error(e) = Target rate – Progress rate

$$error(e_i) = \Delta r - \Delta P_i(t) \quad (3)$$

Parameters

- The target rate of a thread.
- The time period is denoted as t.
- The amount of progress made by an individual thread is represented by Δp_i.

Adapter manipulates the CPU share according to the measured error. The adapter employs a Proportional Integral (PI) controller to measure the error. The error e(n) measured in a particular interval (n) and the sum of the recent errors are added. The error is calculated for all threads at different intervals. The rate at which error change can be measured using a PI controller. PI controller tracks the error at recent interval and also at the previous intervals.

The PI controller algorithm corrects the proportional term and the integral term. The proportional value determines the reaction to the current error and the integral value determines the reaction based on the sum of recent errors. The weighted sum of these two actions is used to adjust the scheduler. The sum of these two terms constitutes the manipulated variable. The manipulated variable used by our control loop is the CPU share.

Hence,

$$MV(t) = P_{out} + I_{out} \quad (4)$$

where, $P_{out}$and I_{out}, are the contributions to the output from the PID controller. MV(t) is the manipulated variable. The proportional term is given by:

$$P_{out} = K_P \, . \, e(n) \quad (5)$$

Where, *P*out is Proportional term of output, K_P is Proportional constant, e is Error = target rate – progress rate and n is n^{th} sampling interval (the present)

The integral term is given by :

$$I_{out} = K_I . \int e_i \quad (6)$$

where I_{out} is Integral term of output, K_I is Integral constant.

Measurement of Error by PI Controller: The feedback control loop is invoked at every sampling interval. Adapter which is in the feedback control loop computes the error at all intervals. The error e(n) measured at a particular interval (n) and the sum of the recent errors are added to get the error of a thread at a particular interval.

So, $$\Delta w_i(t) = K_p e_i(t) + K_I \sum_{j=1}^{t} e_i(j) \quad (7)$$

where **$\Delta w_i(t)$ is e**rror of a task i at t^{th} interval, $K_p e_i(t)$ is proportional error and $K_I \sum_{j=1}^{t} e_i(j)$ **is** integral error. The value for K_p is set according Ziegler-Nichols method and proportional constant K_p is taken as 0.9. The value for K_I is set according Ziegler-Nichols method and integral constant K_I is taken as 0.1.

The error computed by PI controller at $(t+1)_{th}$ interval is :

$$\Delta w_i(t+1) = K_p e_i(t+1) + K_I \sum_{j=1}^{t} e_i(j) \quad (8)$$

The computation of error may vary at each interval. Hence, the change in error is given for $task_i$ at t^{th} interval is computed as the difference between the error at t^{th} interval and $(t-1)^{th}$ interval. It is given by:

$$\Delta w_i(t) = (K_p+K_i) * e_i(t) - K_p * e_i(t-1) \quad (9)$$

Where Δw_i is error of task i at t^{th} interval, K_p is proportional constant, K_i is integral constant, $e_i(t)$ is error of task i at t^{th} interval and $e_i(t-1)$ is error of task i at $(t-1)^{th}$ interval.

During any interval, there may be a set of tasks for which error needs to be computed. The error for an individual task is computed using the above mentioned equation. That is, the error computed for each task is as follows:

$$\Delta w_1(t) = (K_p+K_i) * e_1(t) - K_p * e_1(t-1)$$
$$\Delta w_2(t) = (K_p+K_i) * e_2(t) - K_p * e_2(t-1)$$
……
……
……
$$\Delta w_m(t) = (K_p+K_i) * e_m(t) - K_p * e_m(t-1)$$

Hence, the computation of error for a set of threads taken at a particular time is given by:

$$\Delta w(t) = K_1 E(n) - K_2 E(n+1) \quad (10)$$

Where, E is the error for a set of threads in a particular time.

$E = e_1 + e_2 + e_3 + \ldots + e_m$; $K_1 = (K_p+K_i)$ and $K_2 = K_p$; $e_1 + e_2 + e_3 + \ldots + e_m$ = error for a set of threads in a particular interval.

As the workload characteristics change, the error computed for each task changes. The change in the error is reflected in the CPU share. The error measured by a PI controller is directly mapped into CPU share which is represented by w. This direct mapping is possible because error is computed based on the progress measure of an application. The CPU share represents the amount of CPU cycles that a task should receive. Our feedback mechanism assigns the CPU cycles for a task based on its progress measure. Hence, the error computed by the adapter can be directly mapped into CPU share.

Manipulation of CPU Share Using an Adapter: Adapter manipulates the CPU share according to the measured error. The error e(n) measured at a particular interval (n) and the sum of the recent errors are added to get the error of a thread at a particular interval. The error is measured for all the tasks at various intervals.

The error measured at t^{th} interval is different from the error that is measured in $(t+1)^{th}$ interval. The error difference between the two intervals is defined by the equation 8. The CPU share is manipulated according to the change in error.

The CPU share (weight) of a task at t^{th} interval includes the weight at the previous interval and the error. The error is measured by the PI controller. Thus, the CPU share (weight) calculated at t^{th} interval includes the CPU share (weight) at $(t-1)^{th}$ interval and the controller output. Every time, the CPU share or weight of a task is adjusted using the manipulated variable. Here, 'w' is the manipulated variable.

The new weight assignment at t^{th} interval for any task i becomes

$$w_i(t) = w_i(t-1) + \Delta w_i(t) \quad (11)$$

Where, $\Delta w_i(t)$ is PI controller output, $w_i(t-1)$ is CPU share or weight of task i at previous interval, $w_i(t)$ is CPU share or weight of task i at current interval.

The measured error represents the rate of execution of an application. The error can either be positive or negative. The positive error indicates that the progress of an application is below the target rate. That is, the progress

execution rate of an application is less than the target rate. The negative error indicates that the progress execution rate of an application is higher than the target rate.

The term CPU share or weight is used to define a portion of the CPU resources that is allocated to a task. Our feedback mechanism allocates the CPU resources based on the execution rate of a task. The execution rate is measured and compared with the target execution rate. The difference is computed as error. The CPU share is manipulated according to the measured error.

- If the measured error is positive, then the manipulated CPU share gets incremented.
- If the measured error is negative, then the manipulated CPU share is decremented.
- If the error is zero, the progress rate has reached the ideal target rate and there is no change in the CPU share.

The deviation of progress execution rate from the target rate determines the performance of an application. If the performance is slow, our feedback mechanism increases its CPU share so as to meet the target rate with minimal variability. If the performance is fast, our feedback mechanism decreases its CPU share.

The manipulated weight has to be adjusted to the total weight of the tasks that exists during a particular interval. The total weight corresponds to the total number of CPU shares of all tasks in a particular time interval. The weight adjustment of a task i with respect to the sum of the weight of other tasks is termed as relative weight and it is given by:

$$w_i^1(t) = \frac{w_i(t)}{\Sigma wj(t)} \qquad (12)$$

Implementation of PS Scheduler with Feedback Mechanism: We integrate the feedback mechanism with the proportional share schedulers and its characteristics are studied. We implement the feedback mechanism to assign weight to the tasks and also to adjust the weight dynamically. The weight assignment is based on the execution rate of an application.

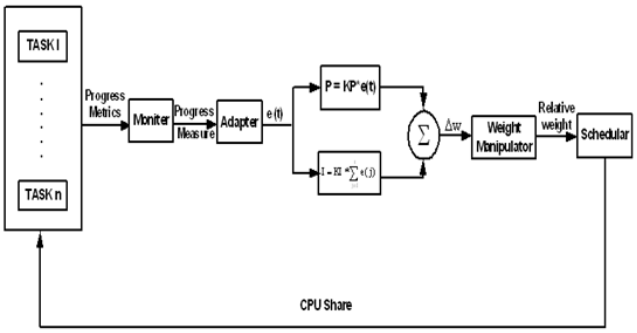

Figure.5. Implementation of feedback scheduler

Figure 5 shows the block diagram of a feedback scheduler. Monitor and adapter constitute the feedback control loop. Monitor measures the execution rate of a task. As the task executes, the execution rate is measured periodically. Each time, the measurement is filtered by means of a filter in the feedback control loop. The progress measure measured by the monitor is then fed into an adapter. The adapter computes the error difference between the target rate and the progress execution rate. A PI controller is employed in order to compute the error over a time interval. The measured error (Δw) is fed into a weight manipulator. The weight manipulator computes the current weight and the relative weight (Δw^1). The current weight of a task depends on the previous weight and on the measured error. The weight is always manipulated with respect to the weight of other tasks which exists at that particular time.

The scheduler employed is basically a proportional share scheduler. We have employed the scheduling mechanism of WFQ, SFQ, and SFS. The three PS schedulers use virtual time domain to make scheduling decisions.

$$v(t) = \frac{1}{\sum w_j} \qquad (13)$$

where Σw_j is the sum of the shares of all active tasks.

Based on the tuned weight from the adapter, the scheduler computes the virtual time. It increases at a rate inversely proportional to the sum of the weights of all active processes. The scheduler is responsible for allocating the processor cycles based on the tuned weight (CPU share). Each task is allocated CPU time based on its CPU-shares value divided by the sum of the CPU-shares values for all active tasks. The characteristics of these schedulers are studied and they are compared with the characteristics of open loop schedulers.

3. RESULTS AND DISCUSSION

Service Time Error:

$$f_i(t) = \frac{w_i}{\sum w_j} \quad (14)$$

The task's share of a given CPU will change over time as the load changes. As the total weight of processes in the system increases, each task's share of the resources decreases. As the total weight of processes in the system decreases, each task's share of the resources increases. When a task's share varies over time, the service time that a task should receive during any interval is given by

$$S_i(t_1,t_2) = \int f_i(t) \quad (15)$$

When the CPU is allocated in time quanta, it is not possible for a task to always receive exactly the service time it is entitled to in all time intervals. The difference between the service time that a task should receive at a time t, and the service time it actually receives is called the service time error.

Service time error (SE) is determined as:

$$SE_i(t_1,t_2) = S_i(t_1,t_2) - w_i(t_1,t_2) \quad (16)$$

Where, i represents a task, $SE_i(t_1,t_2)$ is service time error measured during the interval t_1,t_2, $w_i(t_1,t_2)$ is CPU share (weight) assigned during t_1,t_2 and $S_i(t_1,t_2)$ is ideal service time that a task should receive during t_1, t_2.

Fairness can be measured in terms of service time error. When the shares are predefined, the service time that a task receives during one interval will not be the same during another interval. When the system is overloaded, there will be a large variation in the service time of a task between various intervals. Hence, the service time error deviates considerably during later intervals. The predefined shares become critical when the system is overloaded.

Our proposed feedback mechanism integrated with PS schedulers assigns weight based on the execution rate of an application. This type of weight assignment can work well even in overload conditions. When the workload characteristics change, there will be a change in CPU share of a task. The service time that a task receives is highly dependent on the CPU share. As CPU shares are decided by the progress execution rate, there will not be much difference in the service time of a task between any two intervals. Hence, the service time error deviates less during later intervals. We compute the CPU share during an interval and let it be stored as x_i*. The actual allocation which is measured as the service time at a particular interval be assigned as x_i. Then, for 'n' number of tasks, service time error is calculated as:

$$SE = \frac{SQRT\left((x_1 *- x_1)^2 + (x_2 *- x_2)^2 + (x_3 *- x_3)^2 + \ldots + (x_n *- x_n)^2\right)}{n} \quad (17)$$

Where, n is the number of tasks, x_i*is the ideal allocation, and x_i is the actual allocation.

We measure the service time error for all three PS schedulers with and without feedback mechanism. Figure 6 through 8 illustrate that PS scheduler with feedback mechanism results in minimum service time error which improves CPU allocation. As the CPU shares are manipulated based on the progress execution rate, the variation in the CPU allocation is minimum at all intervals. In case of pre-defined CPU share allocation, the CPU allocation varies linearly during all intervals.

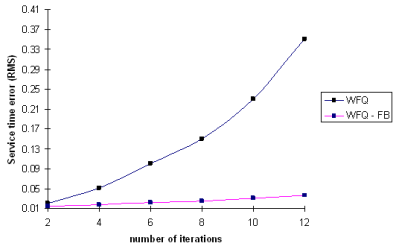

Figure.6. Service time error with WFQ feedback scheduler

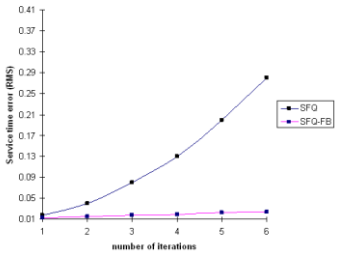

Figure.7. Service time error with SFQ feedback scheduler

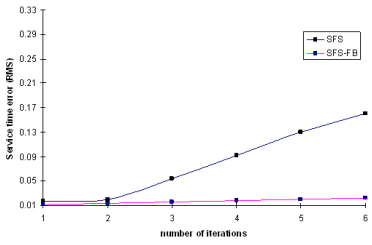

Figure.8.Service time error with SFS feedback scheduler

Throughput: Throughput is defined as the number of processes completed per unit time. We measure the number of processes completed by repeating the experiment ten times. All iterations are made to run for different durations. We measure throughput for two different job categories: i) real time applications ii) mix of real time and non real time applications.

Figure 9 illustrates throughput for different job categories when scheduled using WFQ with feedback mechanism. We observe that the policy can result in high throughput when the workload is purely real time. When there is a mix of workload, the throughput is not high. The experiment is carried out for different number of iterations.

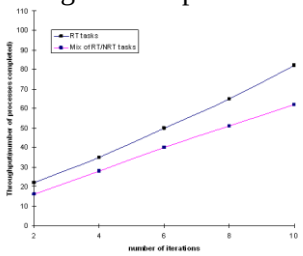

Figure.9. Throughput for different categories of tasks

4. CONCLUSIONS

A feedback mechanism is integrated with the proportional share schedulers and its characteristics are studied. We implement the feedback mechanism to assign weight to the tasks and also to adjust the weight dynamically. The weight assignment is based on the execution rate of an application.

The feedback scheduler is structured and the related environmental variables is identified to achieve high throughput and to minimize service time error. The performance of feedback scheduler is evaluated and its related metrics are analyzed.

Our feedback mechanism integrated with PS schedulers assigns weights based on the execution rate of an application. We evaluated the performance for a mix of workload which includes real-time and non-real time tasks. The dynamic change in the workload characteristics is also considered.

REFERENCES

Abdelzaher T.F and C Lu, Modeling and Performance Control of Internet Servers', 39th IEEE Conference on Decision and Control, Sydney, Australia, 2000, 2234-2239.

Abhishek Chandra, Micah Adler, Pawan Goyal, and Prashant Shenoy, Surplus fair scheduling, A Proportional-Share CPU scheduling algorithm for symmetric multiprocessors, OSDI, 2000, 45–58.

Bennett J and H.Zhang, WF²Q: Worst-case Fair Weighted Fair Queueing', Proceedings of INFOCOM '96, San Francisco, CA, 1996, 120-128.

Buttazzo G and Abeni L, Adaptive rate control through elastic scheduling, Proceedings of 39th IEEE conference on Decision and Control, 2000, 4883-4888.

Cervin A, Eker J, Bernhardsson B and Arzen K.E "Feedback feedforward scheduling of control tasks." Real-Time Systems, 2002.

Goyal P, Guo X and Vin H.M, A Hierarchical CPU Scheduler for Multimedia Operating Systems, Proceedings of the second symposium on Operating Systems Design and Implementation, Seattle, WA, 1996,107-122.

Hellerstein J, Diao Y, Parekh S, Tilbury D, Feedback Control of Computing Systems, Wiley-Interscience, 2004.

Hollot C.V, Misra V, Towsley D, Wei-bo Gong, A Control Theoretic Analysis of RED',Proceedings of IEEE infocom, 2001, 1510—1519.

Jyothi V.L and Srivatsa S.K, A Mechanism for Dynamic Weight Assignment by Inferring Processing Requirement of an application', International Journal on Information Sciences and Computing, 1, 2007, 67–70.

Liu J and Lee E, Timed multitasking for real-time embedded software." IEEE Control Systems Magazine, 23(1), 2003.

Lu C, Stankovic J, Son S and Tao G, Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms,' Real-Time Systems, pp 23(1), 2002, 85–126.

Micah Adler and Petra Berenbrink, A Proportionate Fair Scheduling Rule with Good Worst-Case Performance', proceedings of the 15th annual ACM Portal on Parallel Algorithm & Architectures, 2004, 101-108.

Miguel de M.A, QoS-Aware Component Frameworks, in The 10th International Workshop on Quality of Service (IWQoS 2002), (Miami Beach, Florida), 2002.

Nieh J and Lam S, 'A Smart Scheduler for Multimedia Applications', Proceedings of ACM Transactions on Computer Systems, 21(2), 2003, 117-163.

Parekh A and Gallager R, A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks, The Single-Node Case', IEEE/ACM Transactions on Networking, 1(3), 1993, 344-357.

Pau Marti, Josep M, Gerhard F, Krithi R**,** Improving Quality-of-Control Using Flexible Timing Constraints: Metric and Scheduling Issues, Proceedings of the 23rd IEEE Real-Time Systems Symposium, **2002,** 91.

Rehbinder H and Sanfridson M, Integration of Off-Line Scheduling and Optimal Control', 12th Euromicro Conference on Real-Time Systems, 2000, 137-143.

Seto D, Lehoczky J.P, Sha L and Shin D.G, On Task Schedulability in Real-Time Control Systems', Proceedings 17th IEEE Real-Time Systems Symposium, 1996, 13.

Shin K, Meissner C, Adaptation of control system performance by task reallocation and period modification', Proceedings of the 11th Euromicro Conference on Real-Time Systems, UK, 1999, 29-36.

Stankovic C, Lu J, Abdelzaher T, Tao G, Son and Marley M, The case for feedback control real-time scheduling, Proceedings of the 11th Euromicro Conference on Real-Time Systems, UK, 1999, 11-20.

Steere D, Goel A, Gruenberg J, McNamee D, Pu C and Walpole J, A Feedback-Driven Proportional Allocator for Real-Rate Scheduling, Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, 1999, 145-158.