# A Framework for Dynamic CPU Allocation with Proportional Share Schedulers

V.L.Jyothi[1] S. K. Srivatsa[2]

**Abstract**

Applications such as interactive multimedia and web application require real-time computation from the operating system in order to be effective. The growing popularity of these applications has spurred research in the design of large multiprocessor servers that can run a variety of demanding applications. These applications are increasingly hosted on general purpose operating system. The underlying operating system should include a scheduler which satisfies certain requirements such as flexibility and fairness. These requirements can be addressed by means of proportional share resource allocation. Recently, the proportional share allocation problem has received a great deal of attention in the context of operating systems and communication networks. Allocating more CPU than is needed will be wasted, whereas allocating less than is needed will delay the job. The required proportion may change dynamically as the resource requirement of the application changes. Hence, a mechanism is needed to monitor the progress of an application and to dynamically allocate the CPU based on the progress.

**Keywords :** Multiprocessor, Operating Systems, Proportional Share Schedulers, Dynamic CPU Allocation.

## 1. Introduction

The basic idea of proportional share schedulers is that each process has an associated weight, and resources are allocated to the process in proportion to their respective weights. Proportional share schedulers were first developed decades ago with the introduction of weighted round-robin scheduling [Siberschatz et al,1998]. In the weighted round-robin case, each process is assigned a time quantum equal to its share. A process with a larger share, then effectively gets a larger quantum than a process with a small share.

Weighted round-robin (WRR) provides proportional sharing by running all process with the same frequency but adjusting the size of their time quanta. WRR is simple to implement, but provides weak proportional fairness. Fair-share schedulers evolved as a result of a need to provide proportional sharing among users. These algorithms are based on controlling priority values were developed and incorporated into some UNIX operating systems[Essick.R,1990, kay.J et al, 1988] . These earlier mechanisms were typically fast, requiring only constant time to select a process for execution. In UNIX time-sharing, scheduling is done based on multi-level feedback with a set of priority queues. Each process has a priority which is adjusted as it executes. The scheduler executes the process with the highest priority. The idea of fair-share is to provide proportional sharing among users by adjusting the priorities of a process. Fair-share provides proportional sharing by effectively running the process at different frequencies. Fair-share schedulers are compatible with UNIX scheduling and relatively easy to deploy in existing UNIX environments.

Proportional share resource allocation is particularly well suited to the problem of providing real-time services because its underlying scheduling mechanism is a

[1]Research Scholar, Sathyabama University.

[2]Senior Professor, St.Joseph's College Of Engg.

quantum-based round-robin like scheduler. Much of this work is rooted in an idealized scheduling abstraction called generalized processor sharing(GPS) [Parekh.A et al. 1993]. Under GPS, scheduling tasks are assigned weights, and each task is allocated a share of the 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 SFS[Chandra et al, 2000], SMART [J.Nieh et al, 2003], DFS[Micah Adler et al,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. A technique is needed to dynamically estimate the proportion required by an application. With these estimates, the system can assign the appropriate proportion to an application.

The rest of this paper is structured as follows. Section 2 deals with the related work. Section 3 presents the system architecture. Section 4 presents the results of our experimental evaluation and we present our conclusion in section 5.

## 2. Backround and related work

There are many methods proposed in the literature for dynamic CPU allocation. In Li and Nahrstedt (1998) a general framework is proposed for controlling the application requests for system resources using the amount of allocated resources. Nakajima (1998) shows how an application can adapt its requirement during transient overloads by scaling down its rate. Lu et al. (1999), considered a mechanism for adjusting the system workload when tasks' execution times are not known precisely. However, this approach does not permit to control each task's utilization individually. Giorgio et al. (2002), a mechanism is considered to compute the actual execution time of a task by monitoring the current load. David C.Steere et al. (1999) propose a mechanism to dynamically allocate the resource based on the progress of the application. However the progress of an application is measured by a symbiotic interface. This interface requires the kernel interpretation. This approach requires an application to be of producer-consumer type. Also, the CPU is dynamically allocated by means of a reservation based scheduler.

The objective is to design a framework to estimate the progress of an application at the user level space. The CPU cycles are dynamically allocated using a proportional scheduler based on the estimation.

## 3. Architectural Components

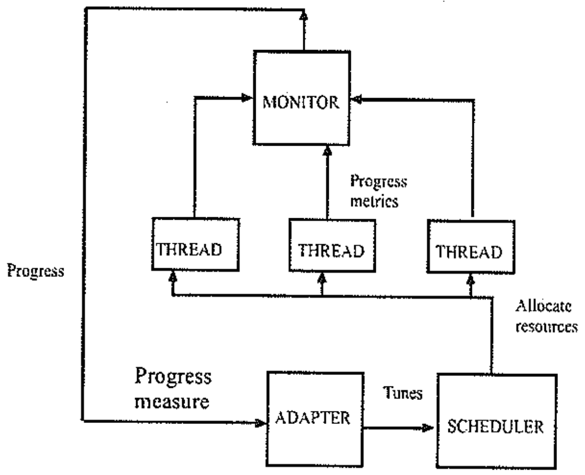

**Figure 1: System Architecture**

**Figure 1: Shows the Main Architectural Components.** Our proposed work monitors the progress of application and increases or decreases the allocation of CPU to those applications as needed. The high-level architecture is shown in fig. 1. The scheduler dispatches threads in order to ensure that they receive their assigned proportion of the CPU during their period. An adapter adjusts each

application's proportion automatically. A feedback is included for dynamic adaptation. The monitor samples progress metrics from the application and it is fed as a progress measure to the adapter.

**A . Monitor**

The main purpose of the monitor program is to estimate the correct proportion required by an application. Progress estimation can be done in several ways. The BeNice program monitors an application's progress via Windows NT performance counters. An alternate method of obtaining progress information of an application is to tap into the progress bar[MSDN library, 1998]. The progress bar is a very poor metric of the actual progress that a program makes [Seltzer.M et al, 1997]. The proposed work estimate the progress based on the importance of an application. Priority may be used to differentiate a low importance process or a high importance process. This may result in starvation or priority inversion. The progress of an application is measured by a symbiotic interface [David C.Steere et al, 1999]. This interface requires the kernel interpretation. The proposed method measures the progress of an application by means of a progress metrics. The progress metrics depends on the nature of an application.

Periodically, a process provides an indication of its progress, through either a library call or a standard reporting interface. A rate calculator combines this progress indication with temporal information from a system clock to determine the process's progress rate. This progress rate is used for two purposes: First, it is fed into a target calibrator, which analyzes many progress rate measurements to determine a target rate for the process. Second, the progress rate is fed into a rate comparator, which compares it against the target rate from the target calibrator. The rate comparator judges whether the current progress rate is less than the target progress rate.

The proposed mechanism [10] allows the metrics to be converted into a suitable progress measure and it is fed into an adapter.

$$\text{Progress measure(Pm)} = \frac{\text{Target rate}(T_r)-\text{Progress rate}(P_r)}{\text{Target rate}(T_r)+\text{Progress rate}(P_r)}$$

This method avoids starvation by ensuring that every job is assigned a non-zero percentage of the CPU. It also avoids priority inversion by allocating CPU based on need as measured by progress. It provides fine grain control since process can request specific portions of the CPU. Resource allocation should ensure that every application maintains a sufficient rate of progress towards its completion.

**B. Adapter**

The main function of the adapter is to achieve higher CPU utilization and better control performance. An adapter tunes the scheduler according to the progress measure from the monitor program.

An adapter may increase or reduce the CPU allocation to the process based on the following scenario :

**CASE 1 :** If progress rate is greater than target rate then,

New desired allocation becomes :

$$\mathbf{w_i = (w_i - P_m)k}\ ;$$

k – constant scaling factor.

**CASE 2 :** If progress rate is lesser than target rate then,

New desired allocation becomes :

$$\mathbf{w_i = (w_i + P_m)k}\ ;$$

**CASE 3 :** If progress rate equals to target rate then,

New desired allocation becomes the ideal allocation.

Case 1 implies to reduce CPU allocation. Case 2 implies to increase CPU allocation.
Case 3 indicates ideal allocation.
The new relative allocation (i.e. with respect to other active process)

$$W_i = \frac{W_i}{\Sigma W_j}$$

The adapter, which typically executes more frequently than the scheduler. It is responsible for signaling the scheduler as the requirement of the tasks changes

**C .Scheduler**

The scheduler used is basically a proportional share system. In PS system, each shared resource is allocated in discrete quanta. At the beginning of each time quantum a process is selected to use the resource. A weight is associated with each process which determines the relative share of the resource that the process should receive.
The algorithm used is closely related to weighted fair queueing (WFQ) algorithms. WFQ [J.C.R.Bennett et al,1996]was proposed for fair allocation of network bandwidth. It can be modified to apply in the domain of processor scheduling. WFQ can be designed to emulate a hypothetical weighted round robin server. The service received by each thread in a round is proportional to the weight of the thread.
The algorithm is formulated in terms of virtual time. The concept of virtual time was introduced by [Zhang]. WFQ introduced the idea of virtual finishing time V(F) to do proportional share scheduling.

**Scheduling Mechanism**

Each process is associated with a weight $w_i$. The virtual time of a process is a measure of the degree to which a process has received its proportional allocation relative to other clients. When the process executes, its virtual time increases at a rate inversely proportional to the sum of the weights of all active process.
Given a process' virtual time, virtual finish time V(F) can be computed. Virtual finish time V(F) is defined as the virtual time the process would have after executing for one time quantum.
At a high-level, the algorithm can be briefly described as :

- Arrange the process in increasing order of weight.
- Compute the virtual time V(t),virtual finish time V(F) for each thread.
- A process which appears earlier than any other process, (i.e) a process with the smallest virtual finish time V(F) is selected for scheduling.

The dynamic CPU allocation is achieved by the above framework using proportional share schedulers. The adapter tunes the CPU allocation based on the progress of an application which is monitored by the monitor. Based on the tuned weight from the adapter, the scheduler computes the virtual time, virtual finish time and using the above mechanism, schedules the job.

## 4. Performance Evaluation

Our test machine is a Pentium IV 800-MHz personal computer with 640KB of Base memory, 256KB of Cache Memory and 20 GB of Hard Disk. The operating system used is Windows XP/Windows 2000 Server.
We tested this mechanisn using two class of processes : i) Numerical Solver and ii) File Archive Utility.
Progress based regulation for the numerical solver is estimated as the count of iterated solution steps. The progress estimated for file archive utility is the number if files it scans. For all experiments except the calibration test, a target progress rate is established by running the application on an idle system until the initial calibration phase completed.

Fig. 1 illustrates the progress of a numerical solver using this mechanism. The x-axis is run time. The y-axis indicates the progress rate, expressed in the normalized target duration between test points. Values greater than one indicate progress above the target rate; values less than one indicate progress below the target rate.

## FLOW DIAGRAM

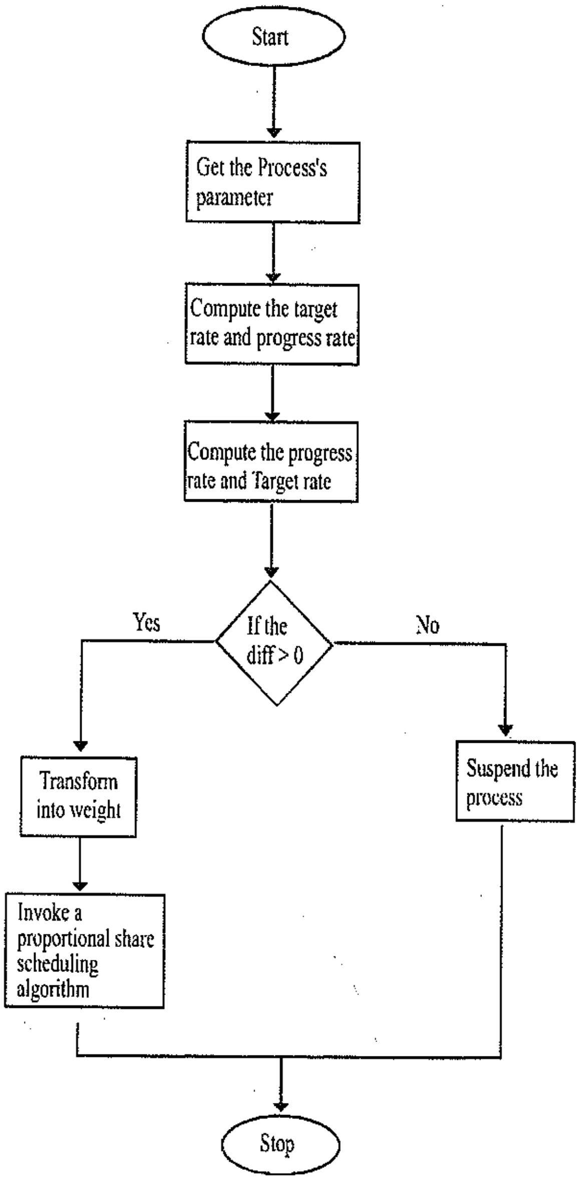
Start
Get the Process's parameter
Compute the target rate and progress rate
Compute the progress rate and Target rate
If the diff > 0
Yes
No
Transform into weight
Invoke a proportional share scheduling algorithm
Suspend the process
Stop

**Figure 1: Progress of Numerical Solver**

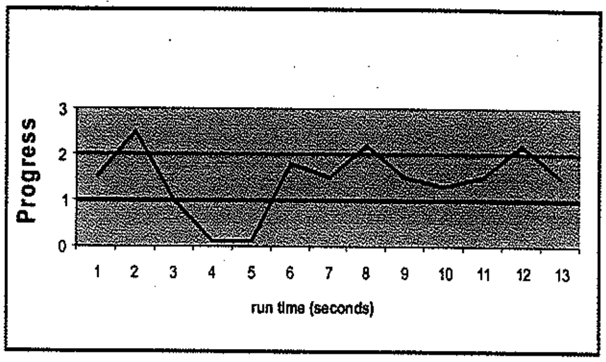

**Figure 1 : Progress of Numerical Solver**

Dynamic CPU allocation is evaluated with certain performance metrics. The proposed method is compared against a dynamic CPU allocation mechanism which use the progress of an application [David C.Steere et al.].

**• Overhead**

Overhead is calculated as the ratio of the CPU time consumed by the algorithm (PS scheduler with feedback mechanism) to the wall time duration of the experiment. Fig. 2 shows the overhead of the algorithm when process are scheduled at quantum lengths of 10,20,40 milliseconds. As quantum increases, overhead decreases. In general, overhead is very low for any number of process using the proposed algorithm.

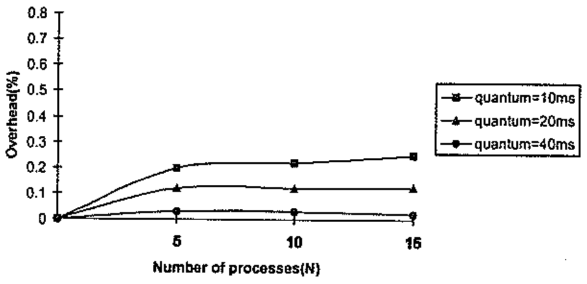

**Figure 2: Overhead of the Proposed Algorithm**

Figure 3. compares overhead for proposed method and existing method (David C.Steere et al. ). It is found that the overhead is below 30% for the proposed method when scheduled for 10ms.

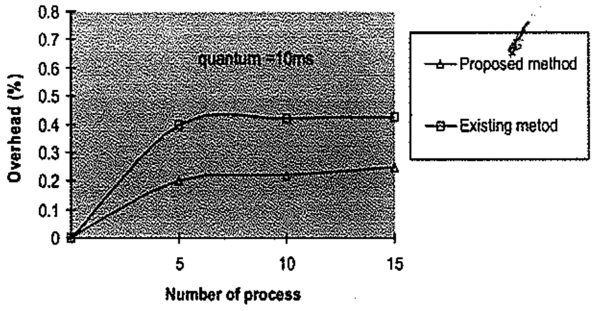

**Figure 3 : Overhead Comparison**

- **Fairness**

To evaluate the fairness of a proportional share algorithm, the service time error for a process has to be defined. The service time error is the difference between the ideal allocation and the actual allocation. Actual allocation can be defined as the amount of time allocated to a process during interval (t1,t2) under the given algorithm. Ideal allocation is the amount of time that would have been allocated under an ideal scheme. The goal of a proportional share scheduler should be to minimize the allocation error between the clients.

Fig. 4 shows the service time error between the processes is minimized. Thus, the proposed algorithm(WFQ) with feedback mechanism achieves fairness.

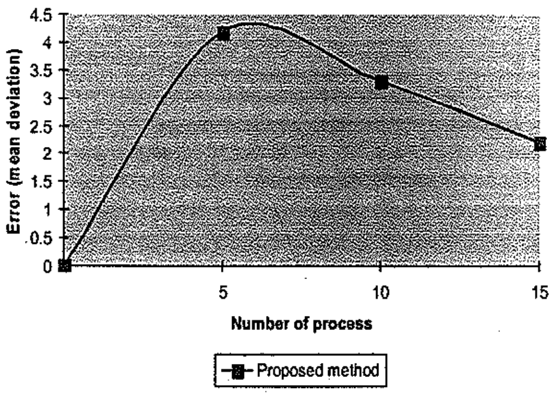

**Figure 4 : Service Time Error**

- **CPU Allocation Error**

A traditional metric used in scheduler's analysis and comparison is the error of CPU allocation. The decision of a controller is based on a difference between assigned and consumed CPU allocation. A scheduler with significant error in CPU allocation may cause unstable controller behavior, lead to a poor application performance.

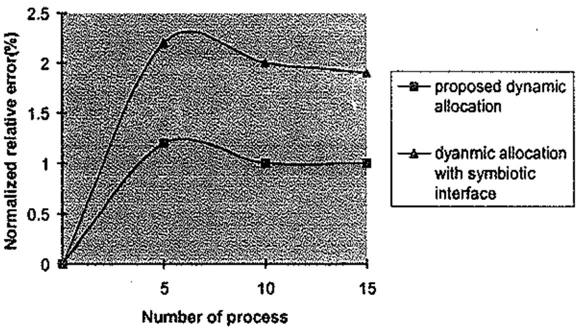

**Figure 5 : Allocation Error**

Fig. 5 plots the allocation error for the proposed dynamic allocation and the dynamic allocation with symbiotic interface. It is found that the proposed method results in minimum allocation error.

## 5. Conclusion

A framework is proposed to assign the required proportion based on the progress of an application and to schedule the process using proportional share schedulers. The amount of CPU time given to an application depends on its progress measure. The system uses a monitor, a controller and a proportional share scheduler. A feedback based controller dynamically adjusts the CPU allocation to meet current resource needs of applications. The advantage of dynamic scheduling based on the progress automatically scales as the application's requirement changes. All processes can dynamically achieve stable configurations of CPU sharing so that starvation is avoided.

## REFERENCES

1. A.Silberschatz and P.Galvin, *"Operating System Concepts"*,Reading, MA, USA: Addison-Wesley, 5th ed, 1998.
2. R.Essick, *"An Event-Based Fair Share Scheduler"*, in Proceedings of the Winter 1990 USENIX Conference,USENIX,Berkeley,CA,USA, PP.147-162, Jan.1990.
3. A.Parekh and R.Gallager, *"A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case"*, IEEE/ ACM Transactions on Networking, 1(3), PP.344-357, June 1993.
4. Abhishek Chandra, Micah Adler, Pawan Goyal & Prashant Shenoy, *"A Proportional Share CPU Scheduling Algorithm for Multiprocessors"*, In Proceedings of the Fourth USENIX Symposium on Operating System Design & Implementation (OSDI 2000), San Diego, CA, Oct'2000.
5. A.Chandra, M.Adler & P.Shenoy, *"Deadline fair scheduling : Bridging the theory & Practice of proportionate fair scheduling in multiprocessor servers"*,In Proceedings of IEEE Real-time Technology & Applications Symposium, June 2001.
6. J.Nieh, S.Lam, *"A Smart Scheduler for Multimedia Applications"* , In proceedings of ACM Transactions on Computer Systems, Vol. 21, No. 2, May 2003.
7. Li. B and Nahrsedt. K, *"A control theoretic model for quality of service adaptions"*, In Proceedings of Sixth International Workshop on Quality of Service, PP. 145-153.
8. Nakajima.T, *"Resource reservation for adaptive QoS mapping in real-time mach"*, In Proceedings of In Proceedings of Sixth International Workshop on Parallel and Distributed Real-Time Systems, PP. 1047-1056, 1998.
9. Lu,C.Stankovic, J.A.Tao, *"Design and Evaluation of a feedback control EDF scheduling algorithm"*, In Proceedings of the 20th IEEE Real-Time Systems Symposium, 1999.
10. D.C.Stteere, A.Goel, J.Walpole, *"A Feedback-driven Proportion Allocator for Real-Rate Scheduling"*, In proceedings of The ACM Symposium on Operating Systems Design & Implementation, 2000.
11. J.C.R.Bennett and H.Zhang, *"WFQ : Worst-case Fair Queueing"*, INFOCOM '96, San-Francisco, March 1996.