Balancing processor shares of scheduling classes through controlled allocation of memory

by K. V. SASTRY
Sperry Univac
Roseville, Minnesota

INTRODUCTION

In a paged, virtual memory computer system used by several different classes of users simultaneously, it is reasonable to expect that each class will demand certain service rates. Service rates for a particular class can be affected by having each class use its share of (a specified percentage) Instruction Processor (IP) time at regular intervals. There are also two other important resources in the system—main memory and the IO system. The service rate of a class is very much dependent upon the availability of (or lack of) these two resources even if a processor is dedicated to a class. On the other hand, a specified IP share can be achieved by using an "unfairly large" portion of main storage for an "unfairly large" population of processes in the READY state. Although it may be favorable to do so for a particular class, it may, in fact, prevent the other classes from achieving their processor shares because of lack of enough storage.

Problem statement

The problem then is threefold—

1. How do we ensure that a proper set of processes in each class is in the READY state so that the IP share for each class is satisfied?
2. How do we ensure that main storage is available to contain the proper set of processes mentioned in (1)? In other words, how do we map the requirement for residence of the proper set into availability of main storage? This question is especially pertinent in a class-oriented scheduling scheme.
3. What measures do we take to ensure that the direction of resource distribution is from those classes that are meeting or exceeding their IP shares through "unfairly large" allocation of resources or "unfairly large" population of processes, to those classes that are not satisfying their IP shares?

DETERMINATION OF THE MULTIPROGRAMMING SET

The multiprogramming set is that set of processes waiting for IP or IO Service and having main storage allocated. In the algorithms described below, the multiprogramming set will be determined based upon the expected storage requirements of a process before it is rescheduled. We will here use a memory management policy based upon the concept of locality of page referencing [1]. In a paged virtual memory system the main storage requirements of a process can be represented by its working set [1,2]. It is defined to be the set of distinct pages referenced by a process in the last time interval $T$.

We will identify [3] three states for a process to be in. These are as follows;

ACTIVE—processes whose working sets are in main storage and are ready to be dispatched, except when they are waiting for page fault to be resolved or a data resource to be released. A process in this state is dispatched until its quantum runs out, or a force-deactivate occurs.

STANDBY—a set of processes eligible to be dispatched.

BLOCKED—a set of processes waiting on an external condition; e.g., a tape mount or a terminal input.

We will also assume that the system is a paging machine with virtual memory. Movement of processes into the ACTIVE set is caused and controlled by the memory balance (or imbalance) condition defined below.

Let $C$ be the total number of scheduling classes in the system. Also, let the scheduling characteristic of class $i$ be denoted by $(m,p,T_i)$, where $m_i$ is the memory requirement of processes in class $i$, $p_i$ and $T_i$ are the IP and IO system share which class $i$ requires in order to satisfy the required rate of progress and response time averaged over all processes in the class. The memory balance condition is denoted by:

$$
\sum_i m_i \leq M
$$
where $M$ is the total main storage size usable by all classes and $m_i$ is the total working set requirement of all processes active in class $i$.

Classical working set theory [1] says that a process can be placed on the ACTIVE state if there is enough free storage available to contain its estimated working set.

Let $w_{e,j}$ be the estimated working set size of a process at the time it is moved into the ACTIVE state. Let the current (actual) working set size of a process be denoted by $w_{a,j}$.

At the time a process is placed in the ACTIVE set, then

$$w_{a,j} = w_{e,j}$$

Let us assume that $f$ number of processes are placed in the ACTIVE state in a particular class to cause processor balance and satisfy processor share of that class. Let us also assume that the memory balance condition permits us to do this, i.e.,

$$\sum_j w_{e,j} \leq (FL)$$

where $(FL)$ is the size of the free list of page frames.

The underlying assumption here is that these $f$ processes that are newly placed in the ACTIVE set will execute soon enough to cause:

$$w_{e,j} = w_{e,j}$$

During all the time that it takes these $f$ processes to build (or rebuild) their working sets there are

$$\sum_j w_{e,j} - \sum_j w_{a,j}$$

number of free page frames that are taken. Nearly all the current virtual memory systems use this approach. Perhaps that is why the force-deactivate function is used very infrequently in those systems.

This is not an efficient approach for a class-oriented scheduling system. All the actually-free-but-unavailable page frames are unavailable for other classes to be used in order to satisfy their IP shares. To this end, a more pragmatic approach is proposed which is based on the notion of "let there be some virtual overcommitment of memory," that is, we activate more processes than whose estimated working sets will fit in memory. A consequential notion is "let there be some inter-class thrashing."

How do we know how many more processes to activate than the memory balance condition permits?

The concept of expected working set accumulation

The rate of progress of a class in real time depends upon the processor share of the class. The rate of progress of a process in a class depends upon the dynamic execution characteristics of the process when it was last placed in the STANDBY set. Nevertheless, such a process is going to page-fault until it builds its working set. This is particularly true of a newly created process. The initial storage allocation to such a process or the criteria used for memory balance to activate such a process should depend upon the fraction of the estimated working set that the process can actually acquire before one of the following happens:

1. The process is quantum-timed-out.
2. The class is quantum-timed-out.
3. Free page frames are made available by the paging algorithm working on some process.

The probability of one of the above happening—in particular (3)—is indeed high. The reasoning is as follows. Let $\ell$ be the number of processes in the ACTIVE state in class $i$ at the time a process is selected to be moved into the ACTIVE state;

- $e_j$ be the mean execution interval of process $j$ in the class;
- $t$ be the average service time of a page fault;
- $q_i$ be the remaining quantum time of class $i$;
- $a_{i,j}$ be the remaining quantum time of process $j$ in class $i$;
- $f$ be the process we are trying to select to activate;
- $p_i$ be the IP share of class $i$ as a fraction of unity;
- $I$ be the current class;
- $Q_i$ be the total quantum for class $i$.

Assume for the moment that $\ell=0$; then the minimum real time, $T$, required for the process to acquire its estimated working set $w_{e,j}$ depends upon the following different conditions. Let $w = w_{e,j} - w_{a,j}$.

1. If $w.t < q_i$, then $T = w.t$.
2. If $w.t \geq q_i$, let $w.t = kQ_i$ and $n$ be the largest integer such that $n < k$.

(i) For $k \leq 1$

The minimum real time that must elapse before $w$ pages are acquired is increased by the dispatching time allocated to classes other than $I$. Depending on the monitoring granularity, this time can be computed as a function of the IP share and the class quantum of class $I$.

For every unit of time allocated to class $I$, all other classes are allocated $(1-p_i)/p_i$ units of time, so that a uniform rate of progress is achieved for all the classes. Hence

$$T = \frac{(1-p_i)}{p_i}Q_i + w.t$$

(ii) For $k > 1$

The elapsed real time consists again of two components. The first is the dispatching time allocated to all other classes for each full quantum of class $I$, which is $(1-p_i)/p_i Q_i$. The total elapsed time for each $Q_i$ of class $I$ is then

$$\frac{(1-p_i)}{p_i}Q_i + Q_i = \frac{Q_i}{p_i}$$

By definition class $I$ receives $n$ such $Q_i$'s. The second component is the remaining fraction of $Q_i$'s that must elapse to satisfy a total class virtual time of $w.t$. Therefore

$$T = \frac{nQ_i}{p_i} + (k-n)Q_i$$
However, the above formulas should be modified if $\ell > 0$. On each page fault, the process is blocked until the fault is resolved. Some other process in the class is dispatched until it faults and so on. For the purpose of analysis let us assume a simple round robin dispatching algorithm among the processes in class $i$ that are READY for IP service. For each execution interval $e_j$ that elapses, the class virtual time that elapses is the sum of all $e_j$'s. The modified time, $T'$, is therefore related to $T$ as follows:

$$T' = \sum_{i} e_j \frac{T}{e_j}$$

The contention now is that the actual storage allocation for a process being activated should be related to the ratio of the minimum time to acquire the estimated working set if there were no other classes to the real time it takes to acquire the same working set. Taking also into account the remaining quantum time for the process in question then, the actual storage allocation should be:

$$w_{a,j} = \frac{\min(w_i T_j) T_j}{T'} \frac{T}{\ell}$$

This scheme will not only insure an equitable distribution of available storage to the processes in the class, it will also allocate only enough storage that the process can actually use.

AN INTER-CLASS ALGORITHM FOR CORRECTING MEMORY OVERCOMMITMENT

Memory overcommitment occurs when a process in the multiprogramming set page-faults and there are no free page frames available. In keeping up with the working set concept of memory management free page frames must be made available by deactivating one of the processes in the multiprogramming set. While there may be several processes qualified to be the deactivation candidates, we must ensure that the deactivation of a process occurs in a proper class.

It was previously pointed out that we do not want a class to achieve its IP share by merely having an "unfairly large" population of processes in the ACTIVE state. On the same token, we do not want a class to overcommit memory to itself thus preventing other classes from achieving their processor shares due to lack of storage.

To this end, let us introduce a parameter called the concurrency factor. The concurrency factor $f_j$ for a class $i$ is defined as the average number of processes that the site administrator or the subsystem designer deems necessary to be in the ACTIVE state such that a specified processor share and response time for that class can be achieved. For the same reason that it is possible to specify processor shares, it should be possible to specify concurrency factors.

Let $p_i$, $T_i$, $f_i'$ be the processor share, page traffic rate and concurrency factor respectively specified for class $i$.

Also let $p_i'$, $T_i'$, $f_i'$ be the current processor share, page traffic rate and concurrency factor respectively achieved by class $i$.

Algorithm

On memory overcommitment due to a page fault in class $I$, the following steps are used to correct the deviations.

1) If $p_i' < p_i$.
   then (i) If $f_i' < f_i$
       then select class $j$ such that
       (a) $[p_i' - p_j]$ is maximum
       (a.1) and $f_j' > f_i$
       (a.2) If however $f_j' < f_i$ for all $j$ then select class with
            $\max[p_i' - p_j]$
       (b) If $p_j' < p_i$ all $j$
           then (b.1) select class $j$ with $\max[f_j' - f_i]$
           (b.2) If, however, $T_j' < T_j$
                for all $j$, then select class with $\min[f_j' - f_i]$
   (ii) if $f_i' > f_j$
        then select class $j$ such that
        (a) $[p_i' - p_j]$ is maximum
        (a.1) and $f_j' > f_i$
        (a.2) If however $f_j' < f_i$ for all $j$ then select class with
             $\max[p_i' - p_j]$
        (b) If $p_j' < p_i$ all $j$
            then (b.1) select class $j$ with $\max[f_j' - f_i]$
            (b.2) If $f_j' < f_i$, all $j$
                 then select class $I$.

2) If $p_i' > p_i$
   then (i) If $f_i' < f_i$
       then select class $j$ such that
       (a) $[p_i' - p_j]$ is maximum
       (a.1) and $f_j' > f_i$
       (a.2) If however $f_j' < f_i$ for all $j$ then select class with
            $\max[p_i' - p_j]$
       (b) If $p_j' < p_i$ all $j + I$
           then select class $I$
   (ii) If $f_i' > f_j$
       then select class $j$ such that
       (a) $[p_i' - p_j]$ is maximum
       (a.1) and $f_j' > f_i$
       (a.2) If $f_j' < f_i$ all $j + I$
            then select class $I$
       (b) If $p_j' < p_i$ all $j + I$
            then select class $I$

Note that in the above algorithm, we are in general trusting the $p$'s more than the $f$'s. It is only when $p' < p$ that we are choosing a class with a maximum $f$-deviation. While a class may maliciously specify large $f$, all classes together cannot specify more than 100 percent of the available IP time. Thus, the $p$'s are inherently more trustworthy than the $f$'s. Since the deviations in the IP shares are corrected through adjustment of the $f$'s, we have a situation that is in-

From the collection of the Computer History Museum (www.computerhistory.org)
herently self correcting. Thus, while the scheduler will select the class to force—deactivate a process from, it is up to the class or subsystem scheduler to select the process to force—deactivate.

CONCLUSIONS

In a multi-application environment consideration must be given to user processing requirements in addition to optimization of resource utilization. These processing requirements can be specified to the system in terms of percentages of IP, IO and memory resources, for each class of users. The system in turn will translate these requirements into scheduling parameters and monitor the progress made by each class in achieving its stated goal. We have developed algorithms to achieve these goals by first activating a process to satisfy a memory balance condition based on only the expected use of memory before this process is scheduled again. Secondly, when an imbalance occurs in the use of resources by the classes, a class and a process within the class is selected to be deactivated from the system based upon the degree and type of imbalance in resource use. Thus the algorithm is inherently self correcting and provides a dynamically adjusted path to the solution. Although the algorithms are developed for a paged, virtual memory system, they can be applied to non-paged systems as well with somewhat longer degree of imbalance in resource use. The equations to compute the expected use of main storage can be made more accurate by using distributions of quantum times for classes and more complex IP service disciplines.

REFERENCES

Management in Data Processing

This group of sessions is directed toward management in data processing and consists of three presentation discussion sessions and two panel sessions. The sessions have been selected for their appropriateness to today's problems in data processing management.

The first part of “Change Management” deals with the problems associated with the preparation and planning required to implement a data processing system (new or renew) and how to cope with the change. The second part will discuss the advances women have made in data processing management and their roles in change management.

Potpourri, for want of a better title, will be a panel discussion concerning computing careers and education: deciding exactly what you want to do in life and where you want to do it, and related topics.

“Computer Aided Systems Integration” will be a panel discussing the reality, dilemma, or myth of this topic. The panel will include such notables as Samuel C. Phillips of TRW, Dan Roman from George Washington University, and Albert Rubenstein from Northwestern University. This panel hopes to be catalytic and focus industry-wide attention to the task of identifying, processing, and displaying “System Integration.”

“Management” will focus on two presentations. The first will be concerned with new applications for data entry and will discuss the various aspects and techniques of on-line data entry and how these affect management. This portion will range far afield from the collection of inventory data in a supermarket to the latest techniques in voice input. The second portion of this session will be directed toward the future management concerns regarding office automation. Attention will be further focused on methods to make top management aware of the new technologies in word processing and how to sell the idea.

“Management Performance” is a session in conjunction with the half-day workshop and is a must for the workshop participants. It will deal with why managers succeed, among other topics. Attendance should not be limited to workshop participants, however.