Platform Independent Resource Allocation for Multimedia Applications

Neil Stratford and Tim Granger*

March 1998

Several modern operating systems are providing multimedia applications with guaranteed access to resources such as network, disk and CPU to enable them to meet Quality of Service (QoS) requirements. The key to efficient resource usage is in determination of minimum resource guarantees subject to the constraints of given application quality of service requirements. We present a measurement based resource allocation mechanism that enables application requirements of timeliness to be specified in a machine independent manner.

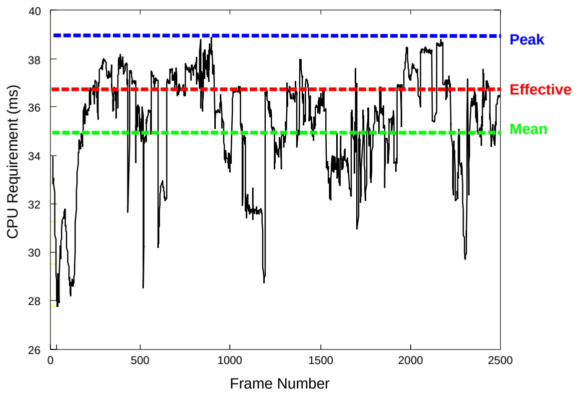

Figure 1: ESR Example

Introduction

Resource allocation is the task of determining the levels of guarantees that applications require to meet their quality of service bounds. The desire to meet a guaranteed level of service must however be traded off against the operating system goal of maximising the overall use of possibly scarce resources.

Traditional real-time operating systems approach this problem by forming models of application behaviour and deriving worst case time bounds [5]. This scheme works well for known workloads, but in a general purpose operating system where the application behaviour and mix will not be known in advance a more flexible scheme is required. The demands of multimedia applications inherently depend on the nature of any continuous media being processed. Figure 1 shows the CPU requirements of a motion JPEG video player application over time. If the application is given a service rate equal to its mean requirement it will quickly fall behind when any significant demand in excess of the mean is made and overflow any buffers. Alternatively, if it is allocated its peak requirement the operating system will be unable to guarantee the resulting extended periods of idle CPU time to any other application.

Multimedia applications can be broadly characterised by their timeliness requirements. To enable an application to meet user requirements it is important that a bound can be placed on the latency of any task that it performs.

*{Neil.Stratford,Tim.Granger}@cl.cam.ac.uk. University of Cambridge Computer Laboratory, Cambridge CB2 3QG, U.K.

In the Chorus based system developed at Lancaster [2] a *quantum* QoS parameter is used to represent the level of resource access required. This parameter is initially derived from a measurement taken for the time to perform a 'dummy' upcall of an *rthandler* and relies substantially on the skill of the programmer to craft a representative dummy execution path. Given that a resource requirement will depend largely on the unpredictable properties of any continuous media being processed this is clearly a difficult task. The Rialto operating system from Microsoft Research [4] provides feedback on CPU usage to applications in the form of how much CPU time they have used since creation. There is an obvious need for an automated mechanism to perform the translation from QoS bounds into resource access requirements.

We propose the concept of an *Effective Service Rate* (ESR) which is defined to be the level of service required to enable an application to meet a given stochastic QoS bound. It must be possible to specify the QoS bound in a machine independent manner, as the service rate required will depend greatly on the details of the host system. In [7] it is argued that processor cycle counts are a system independent measurement of service rate. In general however the *ESR* is highly dependent on other factors apart from clock speed, such as cache and branch prediction behaviour which are affected by application mix and scheduling parameters. Given the complexity of modeling such a system to obtain a portable unit of ESR, we propose a QoS parameter set based on latency that may be converted at run-time into a machine (and application) dependent ESR using on-line measurement.

Queue Server Paradigm

A modern trend in operating system design [3, 6] is to adopt a vertical structure where applications are responsible for performing many of the tasks usually associated with a monolithic kernel, or a shared server. This structure enables us to treat applications as independent *virtual single server queues*, that are served at the current rate r. QoS bounds can now be expressed in terms of maximum *latency* of work in a queue, l, and the probability of exceeding this latency, p. This model can be applied to many applications and is general enough to express detailed QoS constraints as shown in figure 2. It should be emphasised that applications do not need to be structured internally as a queue for this model to be applied. Consider as an example two common multimedia applications: compressed M-JPEG video playback from disk and MPEG audio layer 3 decoding. Work arrives into the *virtual queue* as data becomes ready for processing (eg. a video/audio frame presentation time passes). In keeping with human perceptual abilities we can set the QoS bounds of the video applications to have a high maximum latency, say 200ms, while the audio application can have a much stricter bound of 20ms. Work arrives into the *virtual queue* as data becomes ready for processing. The probability of exceeding this latency can then be used to set a variable QoS level which directly affects the perceived presentation quality.

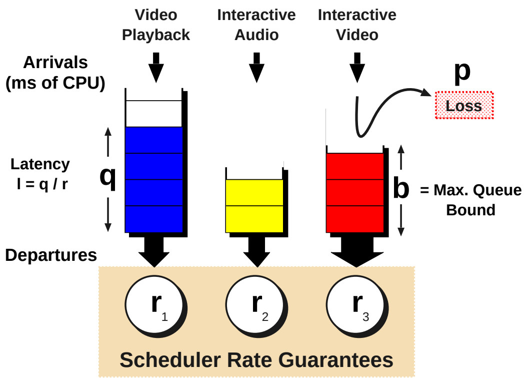

Figure 2: Queue Server Paradigm

Applications can react in various ad-hoc ways in response to these parameters. Some may choose to discard work that exceeds the maximum latency, while others may be written to adapt in output quality level and degrade gracefully. An example of such an application is a wavelet based video decoder that we have developed, which may discard higher subbands in response to a lower resource allocation.

Our algorithm for converting the QoS specification (l, p) into an ESR is based on observations of the entropy of an applications resource usage and applies a result of Large Deviation Theory.

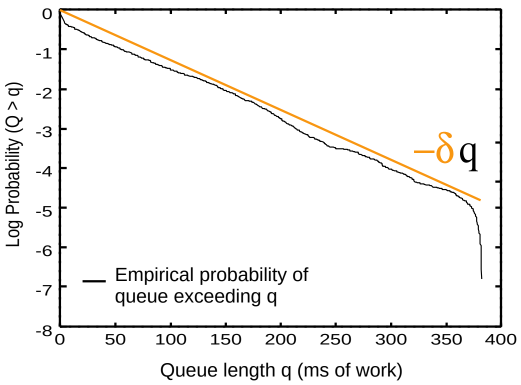

Figure 3: Empirical Queue Lengths

Relating QoS to ESR

The cumulative queue distribution of an instrumented M-JPEG decoder is shown in figure 3. The distribution is shown to have a linear approximation with gradient $-\delta$. The gradient $-\delta$ is a measure of performance of the application, and can be directly related to its QoS bounds by equation 1.

$$\delta = \frac{-\log p}{lr} \tag{1}$$

To satisfy a given QoS specification, (l, p), the application will require a service rate r that results in a cumulative distribution with slope equal to $-\delta$.

From observations of application behaviour, it is possible to construct a compact approximation to the scaled cumulant generating function (SCGF, λ) that is analogous to the entropy of the application. This function directly relates the target value of δ to the required *ESR* (r).

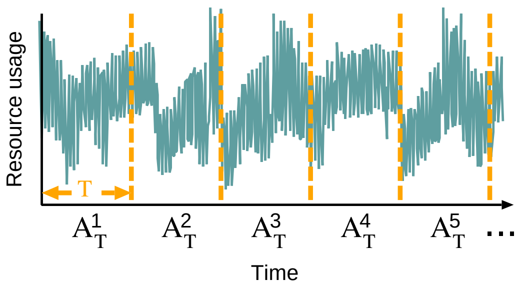

Figure 4: Workload Observations

More specifically we construct an estimate of the SCGF from observations of resource usage A_T^k over N successive periods of length T (figure 4) using equation 2.

$$\lambda_T(\theta) := \frac{1}{T} \log \frac{1}{N} \sum_{k=1}^{N} e^{\theta A_T^k} \qquad (2)$$

The required value of r is then the gradient of a line connecting the origin and the point on the SCGF, $\lambda = -(\log p)/l$. [1]

This algorithm provides a mechanism for determining a value for the *ESR* that relies only on knowledge of past application behaviour, and the desired latency requirements. Full details and justification can be found in [1].

Results and Further Work

To measure the performance of our mechanism for resource allocation we have instrumented several applications to keep detailed information on the status of their work queue. This enables us to compare the latency experienced by applications when given a calculated ESR to their desired QoS specification.

Figure 5 shows the results for an instrumented M-JPEG application. The application decodes a sequence of JPEG frames from local disk at a fixed rate, with a maximum bound on latency for any single frame. In the normal operating region, for latency violations of 0.01 or less, the predictions are close and slightly conservative. Given the theory predicts worst case behaviour and the differences between the actual and virtual queues, due to overheads, these results are very promising.

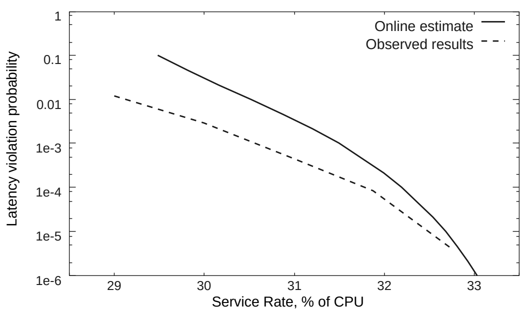

Figure 5: ESR Calculation Results

For several applications the mean of their resource requirements is close to the peak. This means that there will be little variation in any ESR, but setting the service rate correctly is still important to avoid break-up and loss. The algorithm presented above has been shown to perform well in these cases and hence can be used as a general mechanism for operating system resource allocation.

Our current implementation uses the Nemesis operating system and considers only the CPU resource. We are currently working on applying similar techniques to other resources, such as memory allocation. Here we define an *Effective Memory Requirement (EMR)* which is the required size of a physical memory allocation to give a probability of paging, p_{page}. Equation 1 is then replaced with $\delta = -log(p_{page})$ which is combined with the SCGF obtained from observations of memory usage to give the EMR.

Through tackling multiple resource allocation problems in a uniform and consistent manner we hope to gain insight into the more general and harder problem of resource co-scheduling.

[1] This corresponds to the point at which $\theta = \delta = \frac{-\log p}{lr}$, but does not require previous knowledge of r.

References

[1] Paul Barham, Simon Crosby, Tim Granger, Neil Stratford, Meriel Huggard, and Fergal Toomey. Measurement based resource allocation for multimedia applications. In *Proceedings of the Multimedia Computing and Networking Conference, SPIE Volume 3310*, 1998.

[2] Geoff Coulson, Andrew Campbell, Philippe Robin, Gordon Blair, Michael Papathomas, and David Hutchison. The design of a QoS controlled ATM based communications system in Chorus. *IEEE Journal on Selected Areas In Communications*, 1994.

[3] Dawson Engler, Frans Kaashoek, and James O'Toole Jr. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the 15th ACM SIGOPS Symposium on Operating Systems Principles, Operating Systems Review*, 1995.

[4] Michael B. Jones, Daniela Rosu, and Marcel-Catalin Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proceedings of the 17th ACM SIGOPS Symposium on Operating Systems Principles, Operating Systems Review*, October 1997.

[5] M. Joseph, editor. *Real-time Systems; Specification, Verificiation and Analysis*. Prentice Hall, 1996.

[6] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas In Communications*, 14(7):1280–1297, September 1996. Article describes state in May 1995.

[7] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Proceedings of the Multimedia Computing and Networking Conference, SPIE Volume 3310*, 1998.