Real-Time OS Needs Multi-Resource Coordination

Kartik Gopalan Tzi-cker Chiueh
State University of New York at Stony Brook
{kartik,chiueh}@cs.sunysb.edu

Abstract

Most research on real-time operating systems have focused on scheduling and admission control of a single system resource, such as CPU, disk, and network link. However, real-world real-time applications require access to multiple system resources. Coordinating the allocation and scheduling of multiple resources raises new and interesting research problems, especially for latency-bound applications. The goal of this position paper is to identify these research issues, present sample solutions, and outline open problems for future research.

1 Motivation

In the past decade, applications of real-time technology have grown beyond the traditional mission critical systems to include more diverse applications such as multimedia, real-time transaction processing, interactive applications and web services. In addition to stringent performance requirements, this new class of real-time applications also need to access multiple system resources such as CPU, disks and network links. To satisfy the end-to-end performance requirement of these applications requires solutions that address new design issues beyond single-resource scheduling and admission control, which has been the focus of most past research in real-time systems.

When a real-time application uses multiple resources, these resources should be coordinated because they are not completely independent of one another. Multi-resource coordination exploits load information on each resource, the ordering of resource usage in each real-time application, and the tradeoff between utilization efficiency and performance guarantee, to achieve both higher overall system utilization efficiency and simpler application development. In this position paper we argue that multi-resource coordination should be an integral part of every real-time OS.

There are numerous real-world time-sensitive applications that would greatly benefit by using a coordinated allocation and scheduling support from the operating system. For instance, cluster-based web-hosting services are increasingly required to provide bounded response times and throughput guarantees for web access requests to their clients' websites [2, 9]. Servicing each web access request requires coordination across network, CPU and disk resources. Multimedia applications could be other prime beneficiaries. A network video server, for example, periodically reads a group of compressed video frames from disk, processes the frames (e.g. transcoding/frame skipping) and ships the processed frames over the network to the requesting clients. Interactive applications such as Voice over IP (VoIP) and video conferencing require coordination across usage of multiple resources such as voice/video capture, transcoding at CPU and data transmission over the network. Similarly, a real-time database server would receive a database query over the network, retrieve the required information from disk-resident database, process the information (e.g. SELECTs/JOINs) and transport the result over the network back to the client. Proposals for active networks [12] require programmable routers to support flow-specific computation in addition to data-path forwarding. Flows that need bounded delay require coordination between flow-specific computation at CPU and packet forwarding at the network interfaces.

Hence, a wide spectrum of real-time applications can potentially gain from inter-resource cooperation. This paper attempts to motivate a wider research into the multi-resource coordination problem and presents the related design issues.

2 Benefits of Multi-Resource Coordination

Coordination simplifies real-time application development. When operating system takes care of coordinating multiple resources, real-time applications are relieved of the burden of having to manage inter-resource interactions. Developers only need to declaratively specify the application-level performance requirements, while the operating system transparently maps these requirements to low-level resource reservations used in admission control and scheduling. This greatly simplifies real-time application development, permitting the application programmer to focus on functionality rather than the nitty-gritty details of inter-resource coordination, timing management and scheduling.

Coordination improves resource usage efficiency. A multi-resource *allocator* can help to achieve system's optimization objectives, such as to improve the long-term resource usage efficiency. Multi-resource allocation could balance the loads across heterogeneous resources to minimize the probability that one resource gets depleted well before others.

Higher resource usage efficiency in turn maximizes the number of real-time applications that can be admitted with timeliness guarantees. A coordinated resource *scheduler*, on the other hand, can facilitate cooperation between individual resource schedulers to improve the instantaneous run-time efficiency of resource usage. For instance, the CPU scheduler could preferentially schedule computation tasks that are followed by disk I/O operations with tight deadlines or those that lie along the SCAN order movement of the disk head.

There are two stages at which multi-resource coordination can be performed: static resource reservation while admitting a real-time application and dynamic scheduling of applications at run-time. In Section 3, we provide an overview of existing systems that attempt to perform multi-resource coordination in one or both of these phases. In Sections 4, and 5 we look at the fundamental issues that need to be addressed in the real-time multi-resource allocation and scheduling respectively. Finally, in Section 6 we identify directions for future research.

3 Existing Research

Continuous Media (CM) model [1] of the DASH system was the first to pioneer the idea of coordinated multi-resource allocation for continuous media applications in order to optimize general system-wide cost metrics. In the CM model, a *meta-scheduler* coordinates with the CPU, network and file subsystems and negotiates end-to-end delay guarantees and buffer requirements on behalf of applications that handle time-sensitive continuous media. However, run-time scheduling of individual resources is performed independently by rate-based schedulers. The CM model has been followed by relatively sparse research efforts in multi-resource allocation and scheduling.

Q-RAM model [8, 13] is an analytical framework for allocating multiple resources along single or multiple QoS dimensions such that general system utility metrics can be optimized. Unlike the CM model, Q-RAM model accounts for multiple QoS dimensions, such as timeliness, cryptography and loss rate, in addition to handling multiple resources. Q-RAM formulates multi-resource allocation as a mixed integer programming problem.

Integrated Real-Time Resource Scheduling (IRS) [5] is similar in spirit to the CM model. In IRS, a multi-resource allocator transparently coordinates with multiple system resources to provide end-to-end delay guarantees to real-time applications. Instead of optimizing general cost metrics, IRS specifically maximizes the number of real-time applications that can be admitted with guarantees. Additionally, unlike the CM and Q-RAM models, IRS performs coordinated multi-resource scheduling at run-time. Multi-resource run-time scheduling takes care of dispatching precedence constrained tasks to individual resource schedulers in a timely manner.

Cooperative Scheduling Server (CSS) [14] is a dedicated server that performs admission control for one specific controlled resource, such as disk, while using CPU as the controlling resource. Timing constraints for disk I/O requests are partitioned into multiple stages each of which is guaranteed to complete before its deadline on a particular resource. The Srishti [12] framework supports network flow specific computations in a programmable router. Srishti performs integrated scheduling of CPU and network resources in order to provide timeliness guarantees for network flows. Management System for Heterogeneous Networks (MSHN) [4] addresses multi-resource allocation for non-real-time applications with the goal of minimizing the total schedule length of a set of precedence constrained tasks.

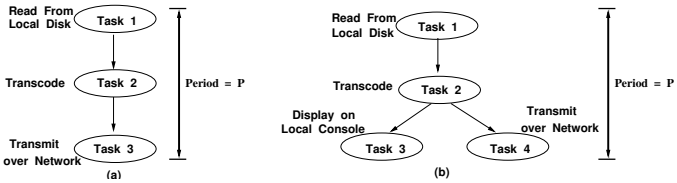

Figure 1: *Task precedence graphs (TPG) for video playback applications. (a) Linear TPG (b) General TPG.*

Multi-resource allocation in real-time OS has much similarity with the problem of resource allocation in multi-hop networks. Specifically, the problem explored by networking research community is how to partition the end-to-end delay requirement of a real-time flow over a multi-hop path to achieve an optimization criteria [6, 10]. Here resources in an operating system are analogous to network links in a multi-hop path.

These analytical models and prototype systems notwithstanding, most state-of-the-art operating systems such as [3, 7, 11] are yet to embrace the concept of multi-resource coordination mechanism. The principal obstacles to widespread acceptance of this concept includes lack of clear cut optimization objectives and absence of a integrated multi-resource interfaces for real-time applications.

4 Coordinated Resource Allocation

Resource allocation deals with the problem of what share of different resources to reserve for an application such that its end-to-end performance requirement can be satisfied. Resource allocation in real-time systems has two fundamental objectives: 1) To provide timeliness guarantees to real-time applications, and 2) To maximize a system wide utility metric.

Different utility metrics can result in different resource allocation strategies. Let's take an example of a specific utility metric: *the number of real-time applications that can be admitted with guarantees*. We will show how careful multi-resource allocation is necessary for maximizing this utility metric.

A typical real-time application repeatedly executes a set of tasks in a periodic or aperiodic fashion, and has a time-bound for execution of the entire set of tasks. There is a specific order in which tasks need to be executed; we call this ordering as the *task precedence graph* (TPG). For instance, Figure 1(a)

shows the linear TPG of a network video server that executes a set of three tasks every 33msec. Task 1 is a *disk read* operation which reads a video frame from local disk. This video frame is then transcoded (Task 2) and transmitted over the network to a remote client (Task 3). In general, the TPG could be a directed acyclic graph, as shown in Figure 1(b). However, for simplicity of exposition, we will restrict our example to Figure 1(a).

In order that the three tasks in the TPG can complete within 33ms, the total delay budget must be partitioned among Tasks 1, 2, and 3. But how does one assign delay budgets to each task in a TPG so that the system can satisfy both the ordering and timing constraints among the tasks, and at the same time maximize the number of real-time applications that can be admitted into the system? This is called the *Task Deadline Assignment* problem.

A simple-minded approach would be to assign equal delay budget of 11ms to each of the three tasks. However, this would be exactly the wrong thing to do for the following reason. A smaller delay budget for a task leads to higher reservation requirement and hence higher load on the corresponding resource. An equal allocation strategy ignores the loads on individual resources while performing task deadline assignment.

The key to making the system resources last longer is to balance the loads on individual resources. Load balancing ensures that more critical resources in the system do not get depleted at a faster rate than less critical resources. For instance, if the disk resource in the above example was more critical than CPU and network resources, then an equal allocation strategy would lead to a situation where we will be left with ample spare CPU and network bandwidth but no spare disk bandwidth.

A judicious resource allocation strategy would, on the other hand, conserve the disk bandwidth by assigning tighter deadlines (say 5ms each) for CPU and network tasks and a looser deadline (such as 23ms) for the disk I/O component. Thus careful task deadline assignment can significantly improve the overall system resource usage efficiency by taking into account the current loads and predicted demands on different resources.

As an example, in IRS [5], the multi-resource allocation algorithm partitions the end-to-end delay specification D of the TPG into task-specific delay budgets. Each task i in the TPG comes with a workload specification W_i at resource j. The allocator calculates the minimum possible delay budget $d_i^{min} = W_i/R_j$ for each task i by assigning all the unreserved capacity R_j at resource j. If $\sum_i d_i^{min}$ is smaller than the end-to-end delay budget D then the new real-time application can be admitted. The allocator next proceeds to divide the *slack* in delay budget $D - \sum_i d_i^{min}$ among the tasks of the TPG in such a manner that the loads across multiple resources remain balanced and no single resource in the system gets depleted earlier than other resources. This strategy is called *load-based slack allocation* (LSA).

Figure 2 compares the LSA algorithm against a simpler equal slack allocation (ESA) algorithm. ESA divides the slack

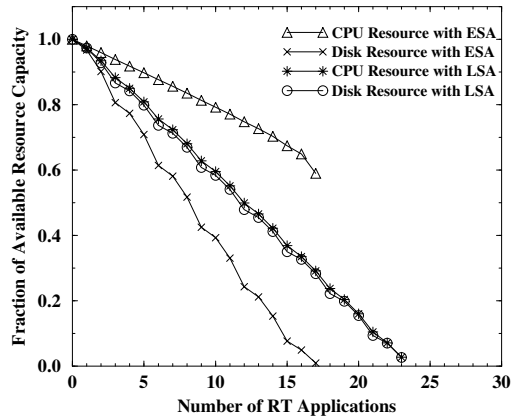

Figure 2: *Variation in available capacity of CPU and disk resources with number of processes in ESA and LSA algorithms.*

equally among all the tasks of the TPG leading to a skewed resource utilization and runs out of disk I/O bandwidth sooner than CPU bandwidth. On the other hand, LSA is able to keep the resources more balanced by allocating more slack to the disk resource which is more loaded than the CPU resource. Thus in the long run, LSA is able to admit more real-time applications than ESA.

As another example, the Q-RAM model [13] provides an analytical framework for multi-resource allocation along multiple QoS dimensions, which may not necessarily be related to real-time guarantees. The optimization criteria in Q-RAM is to maximize a system utility function that is composed of individual application utilities. Application utilities can be controlled by the level of QoS assigned along each QoS dimension.

Like the IRS framework, Q-RAM model also allows tradeoffs between QoS levels across different resources. QoS level at a more critical resource can be lowered for a higher QoS at another less critical resource. For instance, extra CPU cycles could be reserved to perform higher quality compression of video data so that less network bandwidth would be consumed in transmitting the video to a remote client.

These examples convincingly demonstrate that coordinated multi-resource allocation for real-time applications provides ample opportunities to advance global optimization objectives, such as resource usage efficiency of the system or the net utility accrued by real-time applications.

5 Coordinated Scheduling

Resource allocation is a largely static operation in which resources are reserved for each real-time application in the system. On the other hand, it is the run-time resource scheduler which guarantees that the promised share of resources is indeed delivered to a real-time application.

Most operating systems support at best independent scheduling of real-time applications' tasks across multiple re-

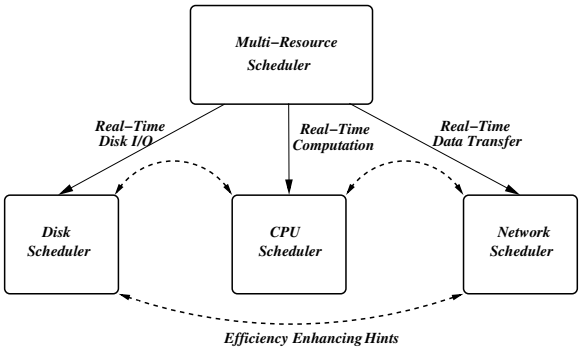

Figure 3: *Multi-resource scheduler coordinates dispatching of tasks to to local resources and enables exchange of efficiency enhancing hints.*

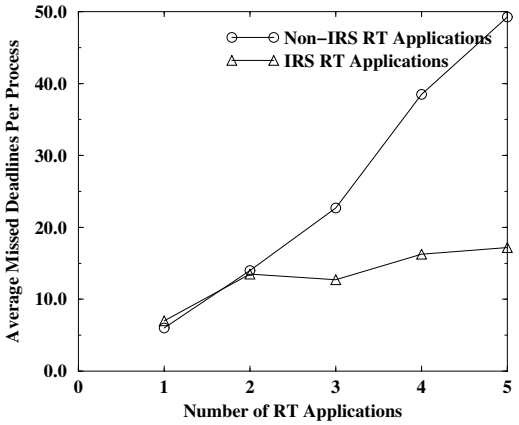

Figure 4: *Average number of missed deadlines per process for non-IRS and IRS real-time applications with increasing number of processes.*

sources and leave it up to the application to manage any form of run-time coordination across resources. Figure 3 shows an entity, called the *multi-resource scheduler*, that acts as a unifying agent across disparate local resource schedulers. Multi-resource scheduler primarily provides two forms of inter-resource coordination.

Transparently scheduling tasks at multiple resources: Traditional real-time applications that require access to multiple resources need to manage intricacies of multi-resource interaction by themselves. This includes managing nitty-gritty details about inter-task dependencies and timings. This results in additional complexity for the application developer.

On the other hand, a multi-resource scheduler can transparently take care of the timing details and dependencies of tasks in the TPG of an application. If the TPG simply consists of tasks within a single process, then the multi-resource scheduler can dispatch the tasks with minimal overhead since it exercises direct control over underlying resources. If the TPG extends to tasks spread across multiple processes, the multi-resource scheduler eliminates the need for complex and expensive inter-process communication to signal task eligibility across processes. Furthermore, a multi-resource scheduler, that has complete knowledge of an application's entire TPG, could dispatch all eligible tasks simultaneously, avoiding the problem of process blocking on I/O directed tasks.

Figure 4 demonstrates the benefits of multi-resource scheduling in the IRS [5] framework. The plot compares the average number of missed deadlines per process when five instances of a real-time video filtering applications were executed with and without the multi-resource scheduling of IRS. With multi-resource scheduling, the average number of missed deadlines is small, the deadline being missed mainly during a learning phase when the scheduler attempts to estimate the computation workload of the application. On the other hand, the number of missed deadlines steadily increases with increasing workloads in the case of independent resource scheduling.

Enabling inter-scheduler cooperation: A multi-resource scheduler can provide efficiency enhancement hints to local resource schedulers by exploiting task dependency information.

For instance, consider a future disk I/O task D that depends upon completion of a ready computation task C. Before dispatching C to the CPU scheduler's queue, the multi-resource scheduler can query the disk scheduler about *desirability* of D in terms of disk usage efficiency. The disk task D would be considered more desirable if it can be serviced along the current SCAN order movement of disk head without compromising other disk I/O deadlines. Similarly, if the task D has a very tight deadline then the disk scheduler might prefer to have D in its ready queue relatively early so that it has enough flexibility to schedule D in a timely manner. If task D is highly desirable from disk I/O point of view then the corresponding preference for scheduling task C at the CPU scheduler should also be increased. This is an example of how scheduling a task at one resource indirectly leads to better run-time scheduling at another resource.

Another example of a resource with preference for one task over another is a multi-CPU resource, where being able to schedule computation tasks of the same process on the same CPU will help the utilization efficiency, because of better cache hit ratio and thus less CPU stall. Thus the multi-resource scheduler can examine the TPG of an application to determine all the computation tasks and schedule these tasks on the same CPU.

6 Research Directions

Two factors complicate the design of multi-resource allocation algorithms. First, typically a system resource's capacity is measured in terms of throughput, e.g., bits/sec and instructions/sec. However, the application-level performance requirement does not have to be based solely on throughput, e.g., delay and delay jitter. The conversion between non-throughput performance requirements and system resource capacity requirements is non-trivial, and depends on the request sched-

uler used in each resource. The conversion formula, even if it exists, is mostly non-linear. Second, the major task of a multi-resource allocation algorithm is to map a separate performance requirement with each resource usage, based on the given application-level performance requirement. For example, if an application's performance requirement is delay bound, a multi-resource allocation algorithm needs to partition the delay budget among the resources involved in the application. The desirability of a particular partition is gauged based on the resulting degree of load balance among the resources in a system. The load of each resource in turn is determined based on the conversion formula discussed above. Therefore, a multi-resource allocation algorithm is essentially a search algorithm that looks for the best partitioning choice which leads to the most balanced load for each resource. IRS and CM model are but two example of partitioning algorithms specifically designed for delay bound. An entirely new class of multi-resource allocation algorithms that are targeted at application-level performance requirements other than delay bound or at different resource schedulers are waiting to be developed.

An important design issue of real-time operating systems is transparency to applications; how much information must the application programmers specify explicitly so that a real-time operating system can effectively render its services? While it is highly desirable that legacy applications can also benefit from the services of a real-time operating system without any code modification, in practice this may not be always possible. So the next best thing is a clean and expressive programming interface through which programmers can specify performance requirements and expected resource usage. When an application needs to access multiple resources, additional information regarding dependencies among tasks, each corresponding to a resource usage, is needed. IRS [5] requires the application programmers to specify a task precedence graph through a special system call interface. In contrast, the Spring system [15] is able to derive this information automatically through a compiler, but requires that all applications be written using a special Spring-C programming language. As real-time operating systems incorporate more sophisticated multi-resource coordination mechanisms, more application-level information may be needed. Keeping the application development as transparent as possible is an interesting challenge.

One of the benefits of multi-resource coordination is that it relieves application programmers of the burden of specifying the deadlines for individual tasks. This is an instance of the *performance-to-resource mapping* problem: Given an application-level performance requirement, how to translate it into low-level system resource reservations. For example, suppose the performance requirement of a real-time database management system is that each simple SELECT query should take less than 50msec. Given such a requirement, what are the minimal CPU and disk resource reservations that the processes assigned to handle these queries should make? This mapping is non-trivial because factors such as CPU cache misses, page faults, locking waiting overhead, and disk seek delay, could result in unexpected resource idle times that should be accommodated in the reservations. With multi-resource coordination, performance-to-resource mapping problem becomes highly complicated and deserves a comprehensive study.

References

[1] D.P. Anderson. Metascheduling for continuous media. *ACM Trans. on Computer Systems*, 11(3):226–252, Aug. 1993.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Measurement and Modeling of Computer Systems*, pages 90–101, 2000.

[3] M. Barabanov and V. Yodaiken. Real-time Linux. *Linux Journal*, Feb. 1997.

[4] D.A. Hensgen et. al. An overview of MSHN: The management system for heterogeneous networks. In *Heterogeneous Computing Workshop*, pages 184–198, April 1999.

[5] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *ACM/SPIE Multimedia Computing and Networking*, Jan. 2002.

[6] K. Gopalan and T. Chiueh. *Delay Budget Allocation in Delay Bounded Network Paths*. Technical Report TR-113, Experimental Computer Systems Labs, Dept. of Computer Science, State University of New York, Stony Brook, NY, June 2002.

[7] M.B. Jones, D. Rosu, and M. Rosu. CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Proc. of SOSP*, pages 198–211, Oct. 1997.

[8] C. Lee, J.P. Lehoczky, D.P. Siewiorek, R. Rajkumar, and J.P. Hansen. A scalable solution to the multi-resource QoS problem. In *IEEE RTSS'99*, pages 315–326, Dec. 1999.

[9] C. Li, G. Peng, K. Gopalan, and T. Chiueh. Performance guarantee for cluster-based internet services. In *Proc. of ICDCS*, May 2003.

[10] D.H. Lorenz and A. Orda. Optimal partition of QoS requirements on unicast paths and multicast trees. In *Proc. of INFOCOM'99*, pages 246–253, March 1999.

[11] LynxOS. *Hard Real-time OS Features and Capabilities*. http://www.lynx.com/.

[12] P. Pradhan, K. Gopalan, and T. Chiueh. Design issues in system support for programmable routers. In *Proc. of HotOS*, 2001.

[13] R. Rajkumar, C. Lee, J.P. Lehoczky, and D. P. Siewiorek. Practical solutions for QoS-based resource allocation. In *IEEE RTSS'98*, pages 296–306, Dec. 1998.

[14] S. Saewong and R. Rajkumar. Cooperative scheduling of multiple resources. In *IEEE RTSS'99*, pages 90–101, Dec. 1999.

[15] J. Stankovic and K. Ramamritham. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software*, 8(3), May 1991.