Constrained online resource control using convex programming based allocation

Lindberg, Mikael

2009

Constrained online resource control using convex programming based allocation

Mikael Lindberg
Department of Automatic Control
Lund University
lindberg@control.lth.se

ABSTRACT

A resource allocation algorithm aimed at embedded multimedia systems is presented. Particular emphasis is placed on computational efficiency, suitability for fixed point implementation and being able to solve the allocation at runtime when parameters or dynamics change. The algorithm is derived from classic convex optimization theory and the resulting real time properties are studied in simulations.

1. INTRODUCTION

Modern portable electronics is expected to perform multiple complex functions simultaneously, while remaining inexpensive, reliable and efficient. Cellular phones are an example of equipment that operates under these requirements. In a typical situation, the cellular phone will simultaneously have to perform audio and video processing, network signaling, user interaction and peripheral device communication. As the operating condition of these types of devices vary greatly from situation to situation, feedback techniques become more and more important for optimizing system performance. Very often, resources will be insufficient to guarantee maximum performance from every concurrently running subsystem and a solution must be able to make rational compromises.

From a resource consumption point of view, the phone software is dominated by video and audio processing. These are both what we consider *timing sensitive* applications, by which we mean that while they have real time timing requirements, they do not fail should these not be met. Rather, their performance can be expressed as a function of how well these requirements are met.

A common factor for media applications is that their computational requirements are heavily influenced by the data stream they process. While some parameters, such as desired playback rate and overall quality settings, are explicitly encoded in the stream, we are dependent on online parameter estimation for control decisions. The parameters can be expected to change over time and feedback techniques must be employed to keep the system performing optimally.

Priority based scheduling is still commonly used in cellular phone operating systems today but as priorities only specify relative importance, analysis is difficult unless the parameters of the entire system are known beforehand. Modern scheduling theory provides us with powerful tools for resource allocation in the form of reservation-based scheduling algorithms, such as the Constant Bandwidth Server [1] or pFair [4] scheduling. These differ perhaps most importantly from their predecessors in that task behavior depends only on the reservation it executes inside and not its surroundings. This property is called *temporal isolation* and greatly simplifies modeling task dynamics.

Techniques for optimal and constrained control are available today in the form of Model Predictive Control (MPC) based schemes. In their generalized form, they are unsuited for embedded systems as they themselves require substantial computational resources. Optimization algorithms for this problem domain should

- consume minimal resources
- be deterministic in time
- be numerically suitable for fix-point implementation

This article aims to demonstrate how such an algorithm can be designed for the cellular phone media use case.

2. RELATED RESEARCH

Resource management is traditionally part of operations research and there applied to a variety of problems such as optimizing storage utilization. A very known variety of this is the knapsack problem, which has been treated extensively in the literature, with books such as [9] covering many variants and solution techniques. It is, however, predominantly a combinatorial formulation which can be computationally expensive to handle. Another classic formulation of allocation is the water filling problem, which has become a popular model for resource allocation in wireless networks and problems of communications theory. Recent contributions include [15] and [10].

In computer systems, resource allocation problems have been treated as traditional allocation problems in articles by R. Rajkumar in e.g. [11] and [8]. Several general resource models

are considered here, including both constraints and multiple types of resources. This article instead focuses more on simple structures and the computational aspects of the allocation, in order to make that problem feasible to solve on modest hardware and finite word length.

Scheduling schemes for resource reservations with feedback has been treated using e.g. the Constant Bandwidth Server approach by Abeni and Buttazzo [2] [3]. The techniques have been extended to the resource constrained case with the elastic task model. Related is the Fair Queuing and Fair Allocation techniques that have been applied to similar problems, where [7] is among the earlier works and [13], [14] more recent contributions.

3. SYSTEM MODEL

3.1 Rate-based processing

For media applications, the quality of the output is strongly connected to the processing rate. This connection holds true for all parts of the media processing chain, from encoding to decoding. It is therefore natural to consider how resource allocation decisions impact the processing rate of the system and thereby indirectly the quality of the service. It is possible to view media stream processing as a special case of data flow or stream processing. In these programming models, data is often contained in packets or tokens which are then processed by a network of computational elements. The rate at which data tokens are processed is a very tangible metric for the application performance. Data flow formulations exist for a large group of software relevant to embedded situations, ranging from automatic control to 3D-graphics. This supports making rate an important basis for resource allocation in heterogenous systems.

Given its central position in this work, the term *rate* deserves a clear definition.

Definition 1. **Rate** signifies the number of occurrences of a pre-specified event during a counting time-period.

The pertinent choice of event and counting period is strongly situational. Consider for instance the difference between digital audio and video. The ear is much more sensitive to audio jitter than the eye is to frame jitter. The audio stream is also sampled at a significantly higher rate than the typical video stream (16 kHz vs 25 Hz). While loosing a single or several movie frames during a second might not even be noticeable for the viewer, loosing the same percentage of audio samples will make the audio sound very distorted. In order to make resource allocations in time to preserve quality, the audio stream will need to be monitored using a much shorter counting period than necessary for the movie stream. As computations and changing scheduling parameters tend to introduce latency in the control loop, it might even be necessary to introduce predictive filters. In both the case of audio and video, the events are expected to be evenly distributed over the counting period. This in not required in general, but non-uniform distributions will make the rate estimator more complex to design. This article will not go deeper into the design of such estimators, but will instead assume that it is possible to design one for each type task of in the set.

3.2 Task model

For the purposes of this article, a rate-based processing task is modeled by the following parameters:

- r - desired execution rate
- y - actual execution rate
- ρ - assigned CPU share (bandwidth)

Depending on the application, the units of these parameter vary. For a video playback system running on a Constant Bandwidth Server (CBS) resource scheduler, r and y would be the desired and actual frame rate of the video stream respectively, and ρ would be a real number in the interval $[0, 1]$. If instead the resource scheduler would be the Completely Fair Scheduler (CFS) now part of the Linux kernel, ρ would be an unsigned integer value. This article will make the assumption that the processing system consists of a set $\tau_1, ..., \tau_N$ of independent CPU-bound tasks, which means that the stationary or mean execution rate y_i of the task τ_i can be approximated by a function $f_i(\rho_i)$. This corresponds to what is in resource management called the *utility function* and is a positive monotonically increasing function with parameter domain $\mathbb{R}^+$. For most rate-based applications, utility gains will decrease when the amount of afforded resource grows very large and it is reasonable to assume that f_i is a concave function. In the case where the task repeats the same calculation over and over again, a simple piecewise linear (PWL) model such as (1) is often sufficient.

$$f_i(\rho_i) = y_i = \begin{cases} k_i \rho_i & 0 \leq \rho_i \leq r_i/k_i, \\ r_i & \rho_i \geq r_i/k_i \end{cases} \tag{1}$$

For tasks with real-time requirements, this leads to the necessity of adaptation to insufficient resources. For a control task, it can mean down sampling (that is reducing r_i) or using less complicated calculations (increasing k_i). Conversely, for a media playback task it will mean either reducing post-processing (increasing k_i) or skipping frames (reducing r_i). It is important to note that the self adaptation depends greatly on the application at hand (dynamics of the controlled plant or the specific encoding scheme used). Experimental validation indicates however that the model holds even for choices of media streams and decoders which were not optimized for self adaptation. Figure 1 shows two cases which were produced using MPEG-4 video streams and the free MPlayer software. The video streams are encoded at a fixed rate, in this case 30 frames per second (fps). When throttling the CPU bandwidth available to the player below what is required for full rate playback, it starts to skip frames to keep up.

The parameter k_i must be assumed to be unknown at design time and is also likely to be time varying. Online estimation is therefore necessary and the resulting control structure is shown in Figure 2. A discussion of structural elements follow.

3.3 Task Set block

From a control perspective, the resource consuming tasks together with the resource scheduler makes up the plant. It is assumed that the number of tasks will change over time

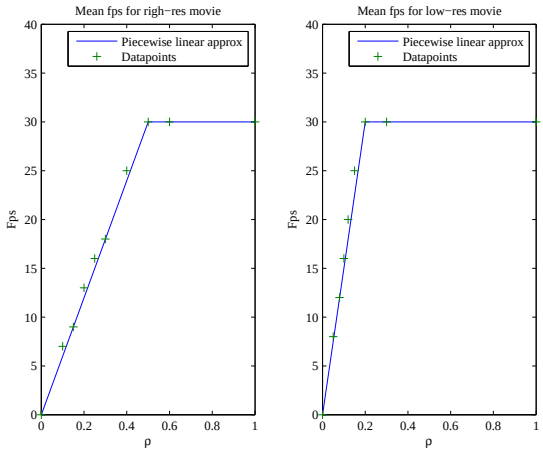

Figure 1: Experimental results of throttling CPU-share for the MPlayer decoder using Linux 2.6.27 and Control Groups. The diagrams show how the frame rate per second (fps) depends on the amount of CPU share allocated to the decoder. The rate increases linearly with share until the movie can be played back at encoded rate.

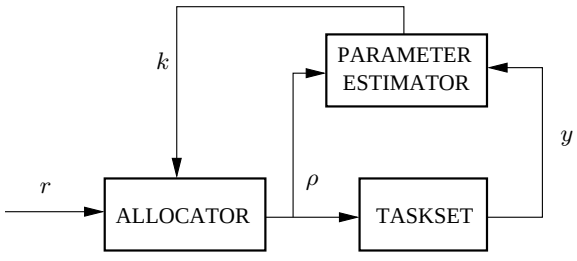

Figure 2: Proposed control structure.

and that the dynamics of each task is both time varying and mostly unknown. Neither is any assumption about the actual resource allocation algorithm made. It is possible to keep the model this abstract through the use of a suitable parameter estimator. As discussed in previous sections, it is assumed that it is possible to formulate an estimator for each task and that this gives a current and correct estimate of execution rate and resource consumption.

It is entirely possible that the task set represents just that subset of tasks in the system which can be described as rate-based. However, it is assumed that they can be isolated from the remaining system through RBS techniques and that the amount of resources allocated to the subsystem is known at all times (though possibly time varying).

3.4 Parameter Estimator block

This block represent some algorithm (or algorithms) that can from real-time measurements estimate the processing rate of the task set and the corresponding resource use. From this it calculates the task CPU gain k_i. The estimator can be tailored for different forms of applications, which can be seen as an added degree of flexibility, as there is no assumption on the execution pattern (such as periodicity). It is also possible to handle different counting periods. For normal periodic tasks, counting events during a sliding time window is a straight forward and intuitive approach. It has been experimentally verified to work well if the rate of allocation decisions is sufficiently lower than the event arrival rate.

3.5 Allocator block

The allocator is responsible for deciding how the resources should be allocated to the tasks in the task set so to maximize the global performance of the system. It will make decisions based on the estimates supplied by the parameter estimator and the desired execution rates as set by user or supervising applications. Physically, it can be assumed to be a task in itself and therefore needs to consume resources to perform its function. A simple approach is to make a static reservation for the allocator and have it execute outside the task set over which it presides. It can also be included in the task set, but the interaction between its own resource allocation and the overall system behavior then becomes much more complex.

4. CONSTRAINED ONLINE ALLOCATION

Allocating resources under constrained conditions requires a compromise in performance for the task set. To evaluate such a compromise, a global performance metric is needed. For the set of independent tasks, a natural choice would be an aggregate of the individual utility functions. Finding such an aggregate which represents the system performance that the user subjectively perceives is a non-trivial task and beyond the scope of this paper. It is however possible to simplify utility tradeoff comparisons by observing that performance is typically only tolerable when y_i is close to r_i. For this purpose a lower rate bound y_i' is introduced, which represents the lowest execution rate at which the performance can be considered acceptable. It would then be desirable to allocate resources so that all tasks (if possible) are within their respective good regions of performance. To reformulate

this as a classic optimization problem, we introduce the rate error $e = r - y$. We can now pose the convex optimization problem

$$\begin{aligned} &\min J = ||e||_w^\infty \\ &\sum_{i=1}^{N} \rho_i \leq \rho_{tot} \\ &\rho_i \geq f_i^{-1}(y_i'), \forall i \end{aligned} \tag{2}$$

or in other words, find the allocation which minimizes the worst weighted rate error under the constraint that all tasks should perform with some minimum performance. The weights w can be used to balance a problem where rates are very different in size. Compare for instance audio vs video where they differ by a factor 10^3. Inversely, it can also be used to model how certain tasks are more sensitive to rate errors.

If the posed optimization problem is feasible, all tasks can have acceptable performance. Testing for feasibility is merely checking that

$$\sum_{i=1}^{N} f_i^{-1}(y_i') \leq \rho_{tot}. \tag{3}$$

Should the test fail, the problem must either be relaxed (e.g. through increasing ρ_{tot}) or some task be deactivated.

Another possible choice of utility aggregation is

$$J = ||e||_w^2 \tag{4}$$

How (4) compares to (2) as a measure of overall system performance is not possible to say in the general case, but solving them is strongly related as will be shown, and the choice can be left to the system designer.

Though parameters in this system might not change often, events such as the reconfiguration of an application, the arrival or deactivation of a new task or the change in CPU resource availability in response to risk of overheating might require that we quickly redistribute resources. Therefore, an algorithm suitable to solving this online with limited computational resources and numerical precision is needed. More specifically, it is desirable that it

1. takes minimal system resources,
2. accounts for changing parameters as quickly as possible,
3. produces results in deterministic time and memory,
4. can improve upon a previous allocation even if aborted before optimum was computed and
5. is suitable for implementation in fixed point arithmetics

5. INCREMENTAL OPTIMIZATION

This section proposes an algorithm which can solve (2) or (4) efficiently and with desirable time and memory characteristics. The central idea of the algorithm is to see the solution as a sequence of resource transfers between two tasks, in effect solving the problem as a series of one-dimensional problems. The benefit of this approach is that each step is computationally inexpensive, predictable in execution time and has numerical properties well suited for fixed point implementations.

Assume that two tasks τ_i, τ_j are picked from the set during th k:th step of the algorithm. Let $J(k)$ be the cost at the beginning of the step and $J_{i,j}(k)$ denote the contribution by τ_i, τ_j to $J(k)$. Specifically, if

$$v = (e_i, e_j) = (r_i - f_i(\rho_i + \delta), r_j - f_j(\rho_j - \delta)) \tag{5}$$

then we have

$$J_{i,j}(k+1) = ||v||_w \tag{6}$$

By solving the subproblem

$$\min_{\delta} J_{i,j}(k+1) \tag{7}$$

$$s.t. -\rho_i \leq \delta \leq \rho_j \tag{8}$$

the relation

$$J(k+1) \leq J(k) \tag{9}$$

is ensured. In other words, by in each step solving a subproblem to the original allocation problem, performance will improve incrementally. Solving this minimization problem for general concave utility functions can be done by simple modifications to unconstrained methods such as Newton-Rhapson. In the case of tasks modeled by (1), near closed form expressions can be obtained for the costs (4) and (2). In the unconstrained case, $||v||^\infty$ is minimized when the elements are equal, that is

$$r_i - k_i(\rho_i + \delta) = r_j - k_j(\rho_j - \delta) \tag{10}$$

while $||v||^2$ is minimized when $\frac{d||e||^2}{d\delta} = 0$, which corresponds to

$$k_i(r_i - k_i(\rho_i + \delta)) = k_j(r_j - k_j(\rho_j - \delta)) \tag{11}$$

A unified expression

$$c_i(r_i - k_i(\rho_i + \delta)) = c_j(r_j - k_j(\rho_j - \delta)) \tag{12}$$

could then represent both optimum points. The terms on the left- and right hand side will be referred to as the *potentials* for τ_i, and τ_j respectively. The optimal transfer can then be written as

$$\delta = \frac{c_i r_i - c_i k_i \rho_i + c_j k_j \rho_j - c_j r_j}{c_i k_i + c_j k_j} \tag{13}$$

to which we apply the limitation in (7). From a numerical point of view, this expression is structurally simple and allows for easy fixed point implementation.

Selecting the pair τ_i, τ_j for each step is the last element of the algorithm. From (7) and (12) it follows that a transfer will occur if potentials are not already balanced and and $\rho \geq 0$ for the task with highest potential. As long as τ_i, τ_j are chosen so, (9) will hold. If there is no such pair, the Karush-Kuhn-Tucker (KKT) conditions show that the current allocation is optimal (see e.g. [5]). It is thus proven that the algorithm will converge to the optimum. The convergence speed will obviously depend on the specific transfer sequence. As the intended domain is real-time allocations, an efficient strategy is needed. It is desirable that each step reduces $J(k)$ as much as possible and from (7) and (12) it is evident that the size of the gain depends on

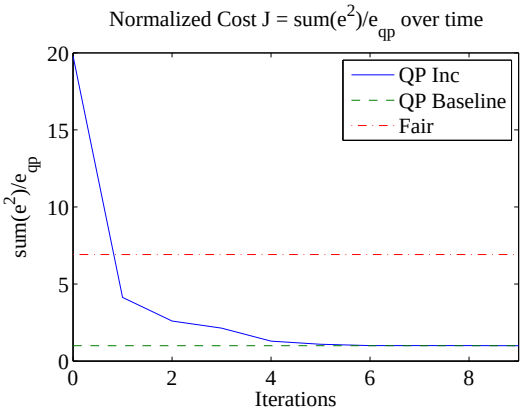

Figure 3: Incremental QP Optimization compared with QP and the default fair allocation. The cost has been normalized with the optimal allocation error e_{qp} which is given by solving the QP problem with eg. `quadprog` in MATLAB.

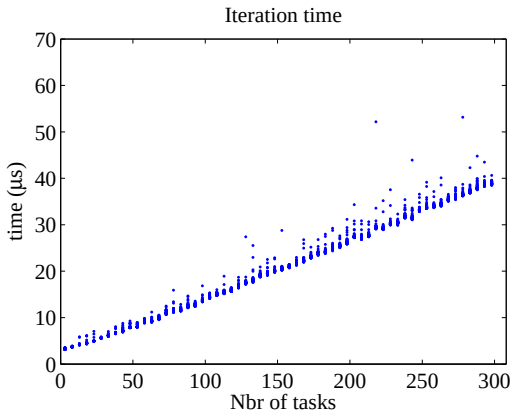

Figure 4: Measurements of iteration times during simulations. Each random taskset was run 10 times. The variance comes from a combination of sorting artifacts and cache dynamics.

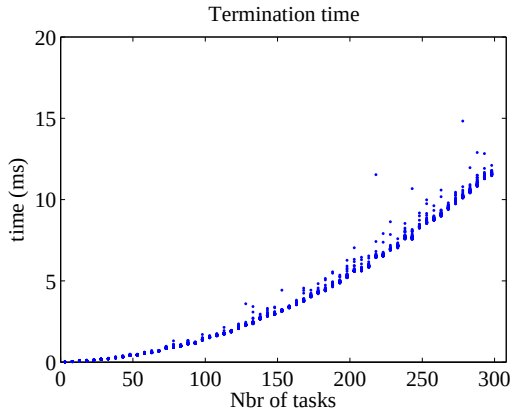

Figure 5: Measurements of optimization termination time. Each taskset was run 10 times. Variance is due to sorting artifacts and cache dynamics.

- the difference in potentials between the two tasks and
- the amount of resource available to redistribute.

The two criteria can be in conflict, but it will be assumed that the allocation is not too skewed and that therefore the tasks have resource allocations of roughly the same size. A reasonable strategy would then be to select a pair with an as large potential difference as possible, where the high potential task has non-zero resources. The proposed implementation uses a red-black tree to keep the tasks sorted according to potential, which makes finding the pair an $O(1)$ operation and inserting them back after the transfer an $O(\log n)$ operation (see e.g. [6] for complexity analysis of red-black trees). As the algorithm uses an iterative loop and the persistent data allocated scales linearly with the task set size, memory need for a system with a known max size can easily be calculated.

6. SIMULATION RESULTS

A series of simulations were run where the algorithm was used to find an allocation for a random task set under overload conditions (i.e. $\sum_{i=0}^{N} r_i/k_i \geq \rho_{tot}$). The aim with the simulations were to show the computational efficiency and get a feel for the convergence rate. The simulations were run on an 2.40 Ghz Intel Pentium(R) 4 based computer with 512Mb memory which was running Linux 2.6.27. The compiler used was gcc 4.3.2 using the -O3 compiler flag. For the experiments, the $||e||^2$ case was chosen as this uses more calculations.

In Figure 3 we can see the algorithm work on a random task set with 6 tasks of which one is the idle task. The task set is initialized with all the resources being allocated to the idle task. The plot compares the aggregate utility function with the fair allocation, as defined by [7] and a baseline allocation calculated using the Quadratic Programming [5] (QP) solver `quadprog` in MATLAB. The values have been normalized using the optimal cost as calculated by the QP solver, so that an optimal cost i 1. The fair allocation can be seen as nominal as it is effected automatically by the current Linux scheduler. The algorithm converges rapidly in the beginning, which is a result of selecting the task with highest potential for the transfer. This is a desirable property as parameters might change over time and the optimization might not have time to finish.

Figure 4 shows the iteration time as function of the number of tasks and in Figure 5 we see the termination time of the optimization. The variance come primarily from sorting artifacts and cache dynamics. Task sets were generated randomly and run 10 times in succession.

As a comparison number, a 2 variable QP problem with the structure of (4) took 500 ms to solve with a general QP solver written in C++ using the same computer as the above simulations. The generic solver also has a large overhead for initializing the algorithm, something which will make it resource expensive to use for a problem where parameters

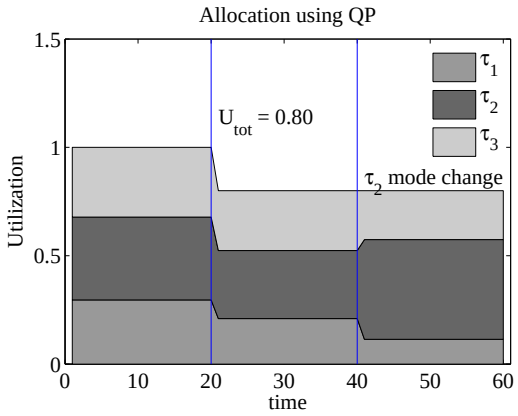

Figure 6: Allocations for a task set of 3 tasks with parameters that change over time.

change over time.

In Figure 6 the algorithm allocates resources to 3 tasks (4 if you include the idle task) with $r = (30, 40, 40)$, $k = (80, 90, 100)$. In this scenario, the available resources change from 1 to 0.8 at time 20 and task 2 has a mode change where r_2 changes from 40 to 60 at time 40. Reduction of resource availability could result from CPU-voltage scaling to deal with heat dissipation problems. This sudden change requires a fast reallocation and for this particular case we can expect the optimization to terminate within a handful of iterations. For the experiment setup this would be about 20 μs. If we assume that task 2 is a video encoder task, the mode change could be a result of a user switching to a higher frame rate. The transients in the figure are exaggerated to visualize the changing allocation better.

7. CONCLUSIONS

As a normal operating system will have tasks numbering in the range of 100 to 300, solving the allocation problem will be fairly efficient. In fact, even when taking into account that a typical cellular phone CPU will run perhaps 10-50 times slower, a 100 task system will take a about 50 ms to solve. This makes the solver feasible to run periodically at runtime which could then respond to dynamically changing parameters of the tasks or resource availability.

The expressions used in the algorithm are simple and therefore suited to implementation in fixed point arithmetic. This is seen as a necessary property of an embedded resource allocator as floating point support is rare and the use of library functions can make the size and memory usage of binaries undesirably large.

8. FURTHER WORK

A fixed point implementation of the algorithm targeted at ARM-type processors is underway. These are frequently seen in cellular phones and other types of consumer media devices. While smaller CPUs normally will not run complex task sets, allocation problems often arise for resources external to the CPU. It would there for be interesting to study how the these types of algorithms could be applied to such things as control allocation problems [12]. This class of problems deal with distributing control action between a number of actuators where the total amount of control authority is constrained, such as emergency breaking systems. In this particular case, control allocation can require fast adjustments should the vehicle suddenly drive over a spot of ice. This would mean extending the algorithm to deal with more complex dynamics and constraints. Multiple resource problems are also a natural extension as embedded CPUs are likely to be multi-core in the future. This could open up for parallel implementations. As the information structure becomes more and more complex and distributed, the cost of communication and information aggregation grows. This will likely lead the need for distributed algorithms were local decisions can be made.

9. ACKNOWLEDGEMENTS

This work has been supported by VINNOVA and Ericsson Mobile Platform as part of the project "Feedback Based Resource Management and Code Generation for Real-time System".

10. REFERENCES

[1] Luca Abeni, Giorgio Buttazzo, Scuola Superiore, and S. Anna. Integrating multimedia applications in hard real-time systems. In *In Proceedings of the 19th IEEE Real-time Systems Symposium*, pages 4–13, 1998.

[2] Luca Abeni, Scuola Superiore, S. Anna, and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *In Proceedings of the IEEE Real Time Computing Systems and Applications, Hong Kong*, pages 70–77, 1999.

[3] Luca Abeni, Scuola Superiore, S. Anna, and Giorgio Buttazzo. Hierarchical qos management for time sensitive applications. In *in Proceedings of the Seventh IEEE Real-Time Technology and Applications Symposium (RTAS âĂŹ01*, pages 63–72. IEEE Computer Society, 2001.

[4] J.H. Anderson and A. Srinivasan. Pfair scheduling: beyond periodic task systems. *Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on*, pages 297–306, 2000.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*, pages 1–12, New York, NY, USA, 1989. ACM.

[8] Sourav Ghosh, Jeffery Hansen, Ragunathan (Raj) Rajkumar, and John Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 12–22, Washington, DC, USA, 2004. IEEE Computer Society.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[10] M. Kobayashi and G. Caire. An iterative water-filling algorithm for maximum weighted sum-rate of gaussian mimo-bc. *Selected Areas in Communications, IEEE Journal on*, 24(8):1640–1646, Aug. 2006.

[11] Ragunathan Rajkumar, Chen Lee, John Lehoczky Y, and Dan Siewiorek. A resource allocation model for qos management. In *In Proceedings of the IEEE Real-Time Systems Symposium*, pages 298–307, 1997.

[12] Brad Schofield. *Model-Based Vehicle Dynamics Control for Active Safety*. PhD thesis, Department of Automatic Control, Lund University, Sweden, September 2008.

[13] Pablo Soldati, Björn Johansson, and Mikael Johansson. Proportionally fair allocation of end-to-end bandwidth in stdma wireless networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 286–297, New York, NY, USA, 2006. ACM.

[14] Y. Wang, L. Fan, D. He, and R. Tafazolli. Solution to weight-adaptive fair queuing. *Electronics Letters*, 44(5):385–386, 28 2008.

[15] Wei Yu, Wonjong Rhee, S. Boyd, and J.M. Cioffi. Iterative water-filling for gaussian vector multiple-access channels. *Information Theory, IEEE Transactions on*, 50(1):145–152, Jan. 2004.