Convex programming-based resource management for uncertain execution platforms

Lindberg, Mikael

2010

Convex Programming-Based Resource Management for Uncertain Execution Platforms

Mikael Lindberg
mikael.lindberg@control.lth.se
Department for Automatic Control, Lund University

***Abstract*—An approach to constrained resource allocation for real time software components executing on nondeterministic hardware is considered. A model for resource consumption based on execution rate is investigated together with an event-based measurement and parameter estimation scheme. Finally, an algorithm for real time constrained optimization of resources is presented together with results from a case study with synthetic software components.**

I. Introduction

The consolidation of telephony, media and general utility computing into modern smart phones has led to increased focus on managing limited resources to make devices flexible while also robust, powerful and yet efficient. A large number of components are integrated to make up a complex platform that must function in a variety of use cases and in different working environments. More and more often, functionality is implemented in software running on a general purpose CPU rather than as discrete hardware components. As a consequence, the method for allocating the CPU resource to competing subsystems becomes central to the performance of the product.

A CPU is normally seen as a resource where utilization must be multiplexed, but recently reservation based scheduling techniques have led to models more related to fluid resource sharing. This approach allows for more component oriented design and simplifies testing and integration. In most cases however, the underlying framework still utilizes classic realtime scheduling results, depending on conservative worst case properties of the concerned software. With the increasing gap between average and worst case performance, this is a growing concern when designing consumer devices. Unit cost is a driving factor and over-provisioning of resources is highly undesirable.

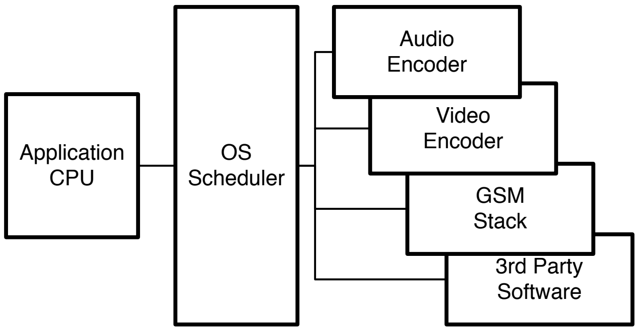

Fig. 1. Overview of the intended target system displaying some typical components.

Another reason why classic results are hard to employ is that they rely on knowing many of the software properties in advance. This becomes especially difficult to handle for media type application such as video decoders or games, as the software behavior is very much determined by the specific usage. For instance, a low resolution video can consume orders of magnitude less computational power to decode than one in high definition. Designs derived from a-priori knowledge are often static in nature and a sudden change in available resources or resource need, could render such solutions unusable.

The combination of uncertainty and time variability suggests a solution based on feedback and estimation techniques. Linear feedback controllers, which are most commonly used, are well suited for situations with unconstrained dynamics. However, the type of system discussed in this paper will be working close to or on the limit of resource availability, something that is difficult to handle using purely linear designs. Instead, optimization-based schemes are often employed for constrained dynamics. Particularly convex optimization lends itself well to online use, as the convexity properties make it easier to design efficient and robust solvers.

This paper presents a framework for constrained resource management for real-time computer systems, with focus on resource heavy applications that perform some repeated computation, such as media playback or Model Predictive style control. These types of applications can be called *timing sensitive*, meaning that they have timing constraints but rather than break down if a constraint is not met, performance is degraded.

The framework consists of three parts:

- a component-based system model,
- an event-based parameter estimation scheme and
- a convex programming-based allocation scheme.

The paper will demonstrate how feedback control and convex optimization theory can be used to pose the allocation problem for an uncertain computation platform running software components with time varying parameters. A model for uniform rate components, which can be considered the analogue to periodic tasks in traditional realtime theory, is presented together with a technique for online parameter estimation. Provided as proof of concept are also experimental results from running the framework on an unmodified Linux kernel.

The intended target system is a cellular phone with multimedia functionality or similiar embedded device.

II. Related Research

The ideas presented in this paper rely on the existence of a reservation based scheduling layer that can be used to partition the CPU-resources predictably. Theory for reservations can be derived from traditional EDF scheduling, resulting in for instance the Constant Bandwidth Server (CBS) formulation proposed by Buttazzo in [2]. The concept has been extended to include constrained resource situations through Elastic Reservations in [4] and [5]. The work in this paper differs primarily in that it allows a more general formulation of the resource tradeoff and puts more focus on online estimation of unknown parameters.

Resource allocation is often posed as a optimization problem, with a prominent example in R. Rajkumar's work on Q-RAM, which was originally described in [14]. This has since been extended to include multi-resource cases in [9]. This paper takes a similar approach, but special care is taken so that the optimization is solvable in realtime and also focuses more on parameter estimation.

Historically, allocation is often seen as knapsack- or bin-packing problems (see e.g. [10]), but the difficulty of solving these types of problems makes the formulation ill suited for use in embedded systems.

A promising approach, using convex optimization in real-time, has been discussed in recent publications by S. Boyd [12]. While these algorithms can be used to solve much more general problems, they rely on code generation to produce specialized solvers. This imposes restrictions on how the structure of the problem can vary over time. The proposed framework in this paper allows the problem to be modified over time by adding or removing components and makes very few assumptions on the individual utility functions, allowing heterogeneous problems to be solved.

Using control theory for computer systems is a strong emerging trend. This has successfully been done by e.g. Tarek Abdelzaher, who treats it in several publications, including [1]. This work is very relevant for the problem studied in this paper, but more focused on computer server farms than the embedded space.

III. System Model

Consider a system of software components C_i that are part of a computational system and executing on the same CPU, as exemplified in Figure 1. A component can be monolithic and correspond to a single operating system thread or process, or it can be an aggregate set of other components. From the outside, the component is opaque and is assigned resources as a single entity. The component produces some relevant results of work and it is assumed that each component uses all assigned resources for that. In order to make decisions regarding how to allocate resources, a model is needed of how resources are converted into results. As it is assumed that the components have unknown characteristics a-priori, it must be possible to estimate the parameters online.

A. Rate-based processing

For media applications, the quality of the output is strongly connected to the processing rate. Higher frame-rates means more fluid video playback, higher bit-rates means more information in each frame. Similarly, in a control system, control performance is related to sample-rate. Control systems and media can be seen as a special cases of data flow or stream processing. Data is often contained in packets or tokens that are processed by a network of computational elements. The rate at which data tokens are processed is a tangible metric for the application performance. This supports making rate an basis for resource allocation decisions in heterogenous systems.

In this paper, rate is defined as the number of occurrences of some event per time period. The pertinent choice of event and counting period is highly 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 sampled at a significantly higher rate than the typical video stream (16 kHz vs 25 Hz). Loosing a few movie frames during a second may not be noticeable for the viewer, while 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 for the movie stream.

As computations and changing scheduling parameters introduces latency in the control loop, it can 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 is not required in general, but non-uniform distributions will make the rate estimator more complex to design. Section III-C will discuss in more detail how to pose the estimation problem.

B. Uniform rate component model

The typical model of media processing or control application is a periodic real-time task. The constant job release interval T and completion deadline d correspond to a uniform rate of completed calculations. In the scheduling formulation of the problem, knowledge about job worst case execution time w is assumed in order to test if all jobs meet their deadlines under some assumptions on the scheduler and task dependancies. The price one pays for this very strong guarantee is that w needs to be a true upper bound, which on uncertain hardware and highly data dependent software can be very pessimistic. It has been demonstrated that feedback control is robust to jitter in sampling and setting of control signal (see e.g. [6]). Similarly, a video playback can suffer both jitter and the occasional frame loss without significant degradation in quality [7]. In this paper, these types of applications will be considered timing sensitive rather having than hard real-time timing properties. Using these properties and relaxing the requirement that all deadlines must be met, it is instead possible to formulate a desired average completion interval, which would in turn correspond to a rate of execution.

A rate-based processing task is then 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 $[0, 1]$, denoting the quota between budget and period, Q/T. 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 use as a weight in the proportional share-based scheme. This paper makes the assumption that the processing system consists of a set $C_1, ..., C_N$ of independent CPU-bound components, which means that the mean execution rate y_i of the component depends primarily on the allocated CPU resource ρ_i and can be approximated by a function $f_i(\rho_i)$. In resource management this is 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) can be sufficient.

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

Figure 2 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.

C. Event based estimation

Estimating the parameters k_i in (1) would be straight forward if the rate y was a continuous signal that could be sampled. As it is, there is only new information about the execution rate when a calculation cycle completes or when an expected event is missing. There are two main alternatives to estimate the execution rate from this, sliding time window event counting and event based filtering. It is worth noting that a benefit from measuring the rate through the completion events is that this poses a very mild requirement on the software. Since in cellular phone design, it is common to use 3rd party components, this is highly desirable as it reduces the cost and complexity of the components. The methods of event based estimation is discussed further in the chapter IV.

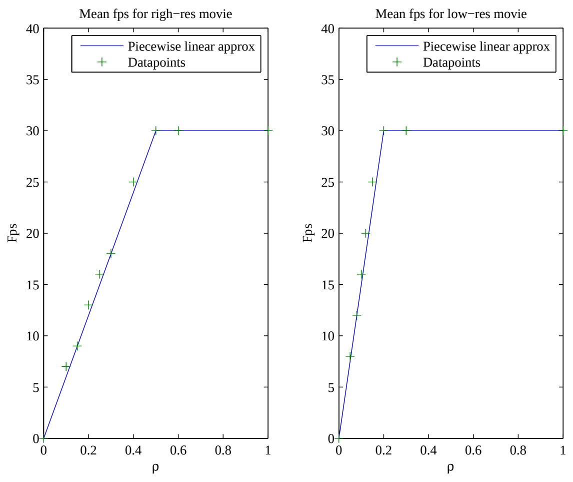

Fig. 2. Experimental results of throttling CPU-share for the MPlayer decoder using Linux 2.6.27 and Control Groups. The diagrams show how the frame rate per second (fps) depends on the amount of CPU share allocated to the decoder. The rate increases linearly with share until the movie can be played back at encoded rate.

IV. Event based estimation

This section will discuss two approaches to form an estimate of the rate y and the parameter k from (1). The following properties are considered important for the resulting algorithm

- Time complexity
- Space requirements
- Sensitivity to noise
- How fast it can detect a change in rate

The following assumptions will be made on the component

- The rate is constant over a small interval of time t_s
- k can in t_s be considered to be the mean of a weak stationary stochastic process.

A. Sliding window event counting

Using the definition of rate (events/time period) it is natural to consider an approach where the number of events occurring over a predetermined time period is mechanically counted. Given a suitable window length, the method is straight forward in implementation, but suffers from needing an unknown amount of memory to keep the events and that the time complexity is proportional to the rate, i.e. unknown beforehand.

B. Event based filtering

An alternative approach is to see the problem as a prediction problem, where the objective is to at any given time estimate the time between the last and the yet not arrived event. If $\Delta(k)$ denotes the time between event k and $k-1$, using the assumed stationarity stochastic properties of k, a predictor can be written on the discrete time shift operator form

$$\hat{\Delta}(k+1) = \frac{B(q^{-1})}{A(q^{-1})}\Delta(k) \tag{2}$$

$$\hat{y}(k) = \frac{1}{\Delta(k+1)} \tag{3}$$

The selection of the polynomials B and A makes it possible to filter out specific noise components of the sequence and as long as the filter has unit stationary gain $(B(1)/A(1) = 1)$ the proper mean will be obtained. There is one caveat however when dealing with a decreasing rate. If the prediction states that an event should occur but there is none, the estimate must be updated to reflect this. In this work the update is done through noting that if t time has passed since the last event occurred and $t > \hat{\Delta}(k+1)$, then the highest possible current rate would be sustained if an event would arrive at the time $t + \epsilon$. A way to check for this is to tentatively update the prediction as if an event had occurred at the time t and check if the estimated rate would be lower. If t_k denotes the arrival time of event k and $\Delta_e(k)$ denotes the extended sequence $\{..., \Delta(k-1), \Delta(k), (t - t_k)\}$, the resulting estimator for $y(t)$ would then be

$$\begin{aligned} \hat{\Delta}(k+1) =& \frac{B(q^{-1})}{A(q^{-1})}\Delta(k) \\ \hat{\Delta}_e(k+1) =& \frac{B(q^{-1})}{A(q^{-1})}\Delta_e(k) \\ \hat{y}(t) =& \frac{1}{\max(\hat{\Delta}(k+1), \Delta_e(k\hat{+}1))} \end{aligned} \tag{4}$$

Advantages with this approach is that the filter is fixed in time and space complexity. There is also the added degree of freedom in selecting the filter polynomials, but the downside is that badly chosen polynomials can yield a very noisy estimate.

C. k-parameter estimation

Given an estimate of the current execution rate $\hat{y}(t)$, falling back on the model (1) results in the following estimate:

$$\hat{k}(t) = \frac{\hat{y}(t)}{\rho(t)} \tag{5}$$

Unfortunately, this estimate assumes that the amount of resource fed to the component is constant. Due to scheduler dynamics and control actions, this is rarely the case and the estimate would have bad convergence properties. Instead, as accumulated resource use for any specific component can be measured directly through system calls, a better estimator is

$$\hat{k}(t) = \frac{\hat{y}(t)}{u_{acc}(t_1) - u_{acc}(t_0)} \tag{6}$$

if $u_{acc}(t)$ is the accumulated amount of resource at time t and t_0 and t_1 are such that the events used to form $\hat{y}(t)$ occur in the interval (t_0, t_1).

V. Constrained Allocation

Allocating resources under constrained conditions requires a compromise in performance for the component set. To evaluate such a compromise, a global performance metric is needed. For a set of independent components, a natural choice would be an aggregate of the individual utility functions. Finding an aggregate that well represents the user perceived system performance will be situation dependent and the task of the system designer. In this paper, some restrictions are posed on the selection of aggregate in order to fit the target platform.

The primary restriction is on the component utility function in that is should fit the convex framework presented in section V-A. This allows for simplified solver algorithm design without putting too severe limits on the choices available to the designer. A secondary restriction is numerical simplicity. Computing the value of the function and its derivative must be relatively inexpensive on a limited precision platform. For evaluation purposes, one such choice will be suggested in the next section.

A. A Convex Formulation

The proposed problem structure in this paper is

$$\begin{aligned} \min J =& \sum_{i=1}^{N} w_i J_i(\rho_i) \\ \rho \geq& 0 \\ \sum \rho_i \leq& \rho_{tot} \end{aligned} \tag{7}$$

under the restriction that $J_i(\rho_i)$ is a convex differentiable function. Assuming that a component C_i have a known desired execution rate r_i, let $e_i = r_i - y_i$ denote the rate error. It is then assumed that it is desirable to minimize the aggregate rate error, resulting in the cost function

$$J = \sum_{i=1}^{N} w_i e_i^2 = \sum_{i=1}^{N} w_i(r_i - f_i(\rho_i))^2 \tag{8}$$

where $f_i(\rho_i)$ is taken as (1). While the rate error e_i is a convex function of ρ_i and thus fits in the framework, it is perfectly possible to use a mix of utility functions when models for more complex components have been derived. This formulation is much like the water filling problem (see [3, pp 245]) used for power allocation in communications theory, with the main difference in that the set of utility functions can be heterogenous.

As previously stated, an important property of the problem is that the parameters are expected to change over time. It is therefore not possible to solve for the optimal allocation once and leave it at that. Changes to the setup can come in many different ways, including

- a new component becomes active
- a component changes its internal structure thereby changing its utility function
- the total amount of resources decreases due to e.g. CPU becoming too hot and needs to be throttled
- properties of the data processed lead to changes in utility function parameters

The solver thus needs to run continuously, making it 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 and
4) can improve upon a previous allocation even if aborted before optimum was computed

VI. Incremental Optimization

In response to the properties 1 - 4, it seems that an incremental approach is suitable, meaning that the algorithm computes the answer as a sequence of relatively simple operations, where each operation improves the solution a bit. As parameters can change at any time, it makes sense to try to use small increments so that as little work as possible is wasted if parameters change in mid increment. A guiding principle behind the proposed solution is that computers are generally good at doing simple things over and over again. This has implications on cache usage, compiler optimizations and stack memory requirements.

Assume that two components C_i, C_j are picked from the set during the 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 C_i, C_j to $J(k)$. Consider now what happens if an amount of resource δ is transferred from C_i to C_j so that their combined contribution to $J(k+1)$ is minimized, i.e. by solving

$$\begin{aligned}\min_{\delta} J_{i,j}(k+1) =& w_i J_i(\rho_i(k)+\delta) + w_j J_j(\rho_j(k)-\delta) \\ & s.t. -\rho_i(k) \le \delta \le \rho_j(k)\end{aligned} \tag{9}$$

This ensures that

$$J(k+1) \le J(k) \tag{10}$$

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 convex functions $J_i(\rho_i)$ can be done by modifications to unconstrained methods such as Newton-Rhapson or even bisection. In the case of components modeled by (1), near closed form expressions can be obtained for some common cost function, see [11] for some.

Selecting the pair C_i, C_j for each step is the last element of the algorithm. The proposed strategy is derived from the Karush-Kuhn-Tucker (KKT) conditions (see e.g. [3]). Posing (7) on standard form, the Lagrangian becomes

$$L(\rho,\lambda,\nu) = \sum_{i=1}^{N} w_i J_i(\rho_i) + \sum_{i=1}^{N} -\lambda_i \rho_i + \nu \sum_{i=1}^{N} \rho_i \tag{11}$$

The KKT-conditions state that $\nabla L(\rho,\lambda,\nu)$ is 0 in an optimal point. By studying the expression

$$\frac{\partial L(\rho,\lambda,\nu)}{\partial \rho_i} = w_i \frac{\partial J_i(\rho_i)}{\partial \rho_i} - \lambda_i + \nu = 0 \tag{12}$$

it can be seen that in an optimal point, either $\rho_i = 0$ or $-w_i \partial J_i(\rho_i)/\partial \rho_i = \nu$. Let $\psi_i(\rho_i) = -w_i \partial J_i(\rho_i)/\partial \rho_i$. If $\rho_i = 0$ and therefore $\lambda_i > 0$, then $\psi_i(\rho_i)$ must be less than ν. In other words, a point where $\psi_i(\rho_i) > \psi_j(\rho_j)$ and $\rho_j > 0$ does not minimize (9).

- If the algorithm tries to select C_i, C_j so that $\psi_i(\rho_i) > \psi_j(\rho_j)$ and $\rho_j > 0$, solving (9) results in $J(k+1) < J(k)$.
- If there is no such pair to select, then that point satisfies the KKT-conditions of (7) and the allocation is optimal.

It follows that such a strategy will make the algorithm 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 (9) it is evident that the size of the gain depends on

- the difference in $\psi(\rho)$ between the two tasks and
- the amount of resource available to redistribute.

The two criteria can be in conflict, particularly if there is a strong correlation between low $\psi_i(\rho_i)$ and low but non-zero ρ_i. It will in this paper be assumed that the components require resources of the same magnitude. A conflicting situation should then only exist initially before the allocation evens out.

An intuitive strategy would be to sort the components according to $\psi_i(\rho_i)$ and select the two furthest apart, skipping the ones with zero resources on the lower end. The proposed implementation uses a red-black tree that makes finding the pair an $O(1)$ operation and inserting them back after the transfer an $O(\log n)$ operation (see e.g. [8] for complexity analysis of red-black trees). As the algorithm uses an iterative loop and the persistent data allocated scales linearly with the problem size, memory need for a system with a known max size can easily be calculated.

To illustrate the workings of the algorithm, consider a case with three components. Let component C_i be represented by the tuple $(r_i, k_i, \rho_i, \partial J_i/\partial \rho_i)$, unit weights are assumed for all components. In the example, the components $C_0 = (25, 30, 1, 300)$, $C_1 = (25, 40, 0, -2000)$, $C_2 = (15, 20, 0, -600)$ will be used.

Step 1, $J = 875.0$. The algorithm finds that the highest ψ component is C_0 (with $\psi_0 = 300$) and the lowest ψ component is C_1 (with $\psi_1 = -2000$). The subproblem to solve then becomes

$$\min J_{0,1} = (25 - 40(1-\delta))^2 + (15 - 20\delta)^2 \tag{13}$$

$$1 \ge \delta \ge 0 \tag{14}$$

which gives the new allocation $C_0 = (25, 30, 0.540, -528)$, $C_1 = (25, 30, 0.460, -528)$, $C_2 = (15, 20, 0, -600)$.

Step 2, $J = 346.0$. C_2 is now the worst of component while $\psi_0 = \psi_1$. The implementation used for this paper uses the component index as secondary sorting criteria, so C_0 is selected. After solving the new subproblem, the allocation becomes $C_0 = (25, 30, 0.512, -578)$, $C_1 = (25, 30, 0.460, -528)$, $C_2 = (15, 20, 0.0277, -578)$.

Subsequent steps are done in the same way, resulting in the sequence

J	(ρ_0, ψ_0)	(ρ_1, ψ_1)	(ρ_2, ψ_2)
345.0	$(0.512, -578)$	$(0.448, -568)$	$(0.0401, -568)$
344.7	$(0.514, -574)$	$(0.445, -574)$	$(0.0401, -568)$
344.7	$(0.514, -574)$	$(0.447, -569)$	$(0.0390, -569)$

Note that while J seems to have converged, the real criteria for termination must be the difference in ψ:s, as derived from the KKT-conditions.

VII. Implementation Aspects

For experimental purposes, an implementation of the framework has been done for Linux 2.6 using the CFS scheduler

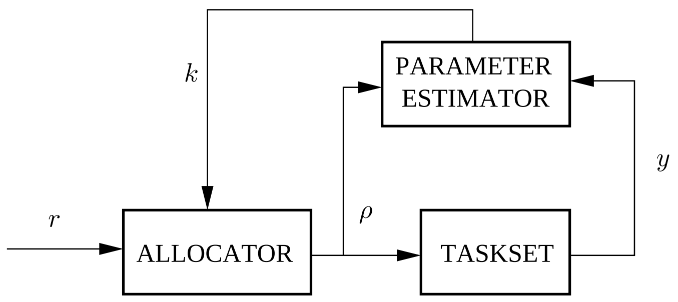

Fig. 3. Proposed control structure.

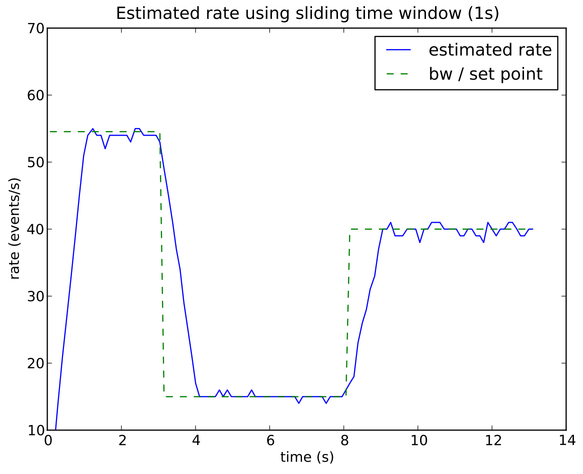

Fig. 4. Event based estimation using a sliding time window. The estimated parameter k is used in feedforward control to show that the model can be used to accurately control the process.

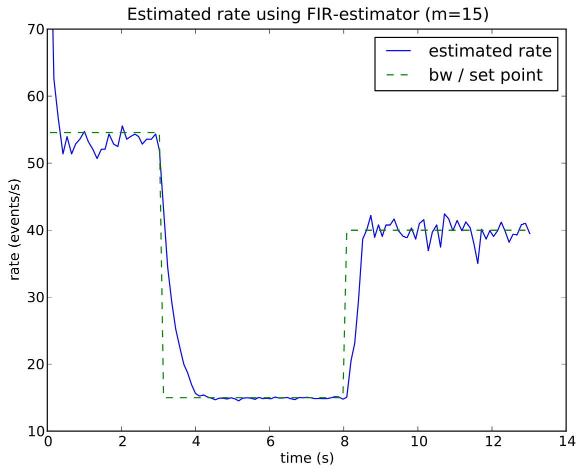

Fig. 5. Event based estimation using an event filter with FIR structure. The estimated parameter k is used in feedforward control to show that the model can be used to accurately control the process.

and control groups [13] for resource allocation. The resulting experiment platform can be said to consist of three major parts.

A. *Component Set*

The components are in this case Linux processes running code to emulate the behavior of periodic tasks that can adapt to varying resource availability by reducing execution rate. They communicate the completion of a calculation cycle by sending a unix datagram packets with the current time stamp and accumulated resource usage. The reason the sampling of resource usage is done by the process and not some other part of the system is to better synchronize the two measurements. Each packet is annotated with the sending process id, so that the estimator can distinguish the data. By means of the /proc file system, other process parameters are discovered, such as control group membership. The processes are multi-threaded in order to support command signals for changing parameters, but the cost of receiving these commands is negligible.

B. *Parameter Estimator*

The estimator is an application with a data collection socket that receive the incoming completion events. Upon collecting an event, the event based estimation algorithm updates the relevant parameter estimate.

C. *Allocator*

The allocation algorithm is in this application execute as a thread in the same process as the estimator application. Periodically, it uses the current estimates to calculate an updated allocation by means of the algorithm described in VI.

VIII. Simulation and Execution Results

A. *Event based estimation*

In order to validate the parameter estimation scheme an experiment was run where the objective was to control the rate of a single component. A comparison between a sliding time window approach and an FIR-structure event filter approach can be seen in Figure 4 and Figure 5. The sliding time window is less noisy for high rates, which is to be expected as it is using a larger number of events to form the estimate. However, the quantization noise can be troublesome when running on low rates. The FIR-estimator on the other hand is more noisy on high rates, but at a rate where the two use the same amount of events to form the estimate (the middle section where the rate is around 15 events/second), it seems a lot more stable. (4) suggests that the filter could have other structures, but that would require a model of the process noise and sensor dynamics to exploit.

B. *Optimization solver performance*

The optimization problem solver was implemented in ANSI C using an of the shelf implementation of a red-black binary tree. The correctness of the solver has been verified against the QP-solver available in MATLAB (quadprog). A simulated simple case with 3 components can be seen in Figure 6 The algorithm has been benchmarked using large sets of random components. The algorithm was run 10 times for each set. The

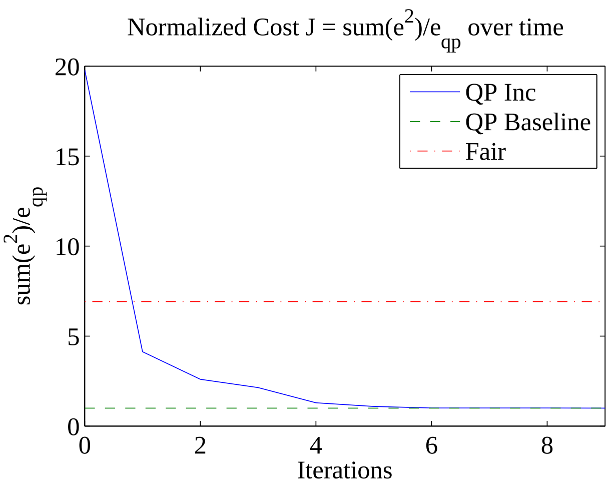

Fig. 6. Solver running over a set of 3 random components with the qp baseline solution computed by MATLAB as a baseline and the fair allocation provided for comparison.

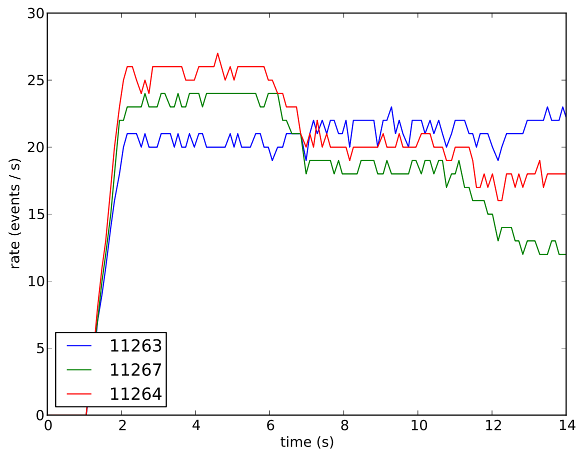

Fig. 9. 3 random components running in a constrained resource environment. Each component changes its dynamics every 3 seconds after which it takes about 1 second for the estimator to converge and a new allocation is calculated.

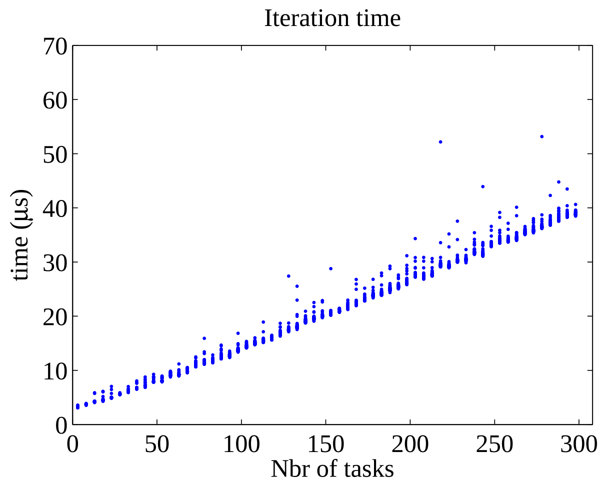

Fig. 7. Iteration time as a function of components in problem.

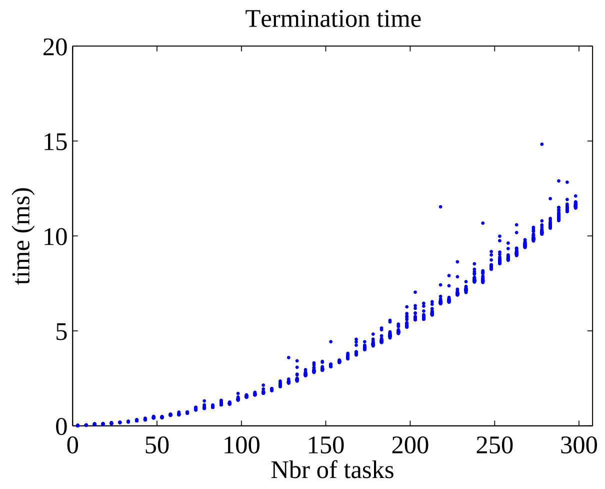

Fig. 8. Optimization time as a function of components in problem.

fluctuations in completion times is most likely due to sorting artifacts and cache misses. Figure 7 shows how the iteration time increases as the number of components in the problem grows, while Figure 8 displays how long it takes to complete the optimization. Even for a fairly large number of tasks, the time is reasonable and running it as part of a periodic controller is deemed reasonable.

C. *Online allocation*

Figure 9 displays the results of an experiment running three components with the same reference rate but with time varying k. Every 3 seconds, the components randomly changes their resource demands, resulting in a new allocation to maximize total system utility. The new k parameters are drawn from a uniform random interval, where $k_{max}/k_{min} = 2$. The setup is not unlike that from Figure 1. The quadratic cost function displays good robustness properties in that a small change in the parameter set only changes the optimum by a small amount. Figure 10 displays the cost function over time for the same experiment. It compares the cost using the dynamic convex programming based allocation (DCA) compared with a theoretical static worst case allocation (SWA) baseline. The DCA setup can provide a substantial improvement over SWA as long as the actual execution time is less than the worst case. As can be expected, the advantage decreases in the last portion of the experiment where the actual execution time is closer to the worst case.

As a final comparison between the DCA and SWA, Figure 11 shows the average cost for a number of setups corresponding to different ratios between k_{max} and k_{min}. For deterministic cases ($k_{max} = k_{min}$) the DCA actually underperforms the SWA. This is because the DCA algorithm relies on $\hat{k}$, which is initially unknown and will vary over time due to noise. The resulting allocation will therefore likely be suboptimal, even if there was enough resources to satisfy all components.

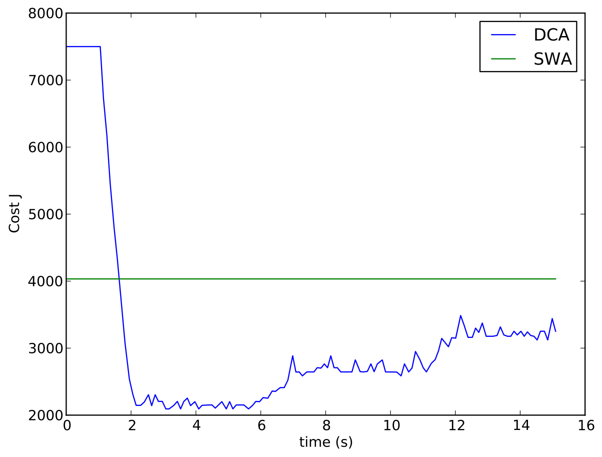

Fig. 10. The performance cost function over time for Dynamic Convex Allocation (DCA) compared with the Static Worst-case Allocation (SWA) baseline.

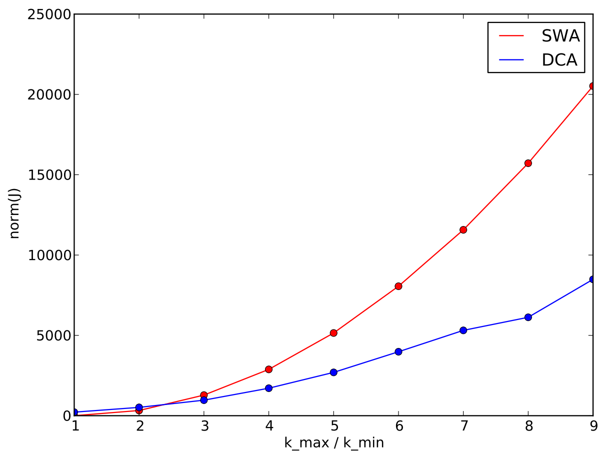

Fig. 11. A comparison between DCA and SWA for different variability of execution time.

50 experiments were run for each ratio and the plot shows the average of the results. This demonstrates that it is quite possible to do the allocation based on no a-priori knowledge about the execution time and with a substantial performance gain compared to the SWA solution.

IX. Future Work

This paper treats systems of software components, but in order to do systemwide resource management, hardware aspects need to be brought into the model. An important direction therefore must be to see how to model a system with mixed software and hardware (typically power) and generate the computational resources needed for the software components. It then directly follows that the model must also be extended to include cases where the components depend on resources from other components.

Mixing software and hardware components will make it hard to maintain global state knowledge and it is therefore reasonable to pursue distributed formulations.

For estimation performance, in the general case little can be said about event to event dynamics but under some assumptions on the resource consumption on the components and process noise, better performance in parameter estimates should be possible.

Finally, admitting new components onto a running system must be investigated as newly arriving components will have uncertain parameters in the utility function. One possibility is to apply reinforcement learning methods to promote active probing.

X. Acknowledgements

This research has partially been funded by the VINNOVA/Ericsson project ”Feedback Based Resource Management and Code Generation for Real-time System” and partly by the EU ICT project CHAT (ICT-224428)

References

[1] Tarek Abdelzaher, Yixin Diao, Joseph L. Hellerstein, Chenyang Lu, and Xiaoyun Zhu. Chapter 7 introduction to control theory and its application to computing systems.

[2] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 4, Washington, DC, USA, 1998. IEEE Computer Society.

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

[4] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 286, Washington, DC, USA, 1998. IEEE Computer Society.

[5] Giorgio Buttazzo and Luca Abeni. Adaptive workload management through elastic scheduling. *Real-Time Syst.*, 23(1/2):7–24, 2002.

[6] Anton Cervin, Bo Lincoln, Johan Eker, Karl-Erik Årzén, and Giorgio Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, Göteborg, Sweden, August 2004. Best paper award.

[7] Mark Claypool and Jonathan Tanner. The effects of jitter on the peceptual quality of video. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, pages 115–118, New York, NY, USA, 1999. ACM.

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

[9] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 12–22, Dec. 2004.

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

[11] M. Lindberg. Febid 2009: Constrained online resource control using convex programming based allocation. In *Proceedings to Fourth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2009.

[12] J. Mattingley and S. Boyd. Automatic code generation for real-time convex optimization, Dec 2009. http://www.stanford.edu/~boyd/papers/rt_cvx_opt.html.

[13] P. Menage. Linux kernel documentation :: cgroups, Dec 2009. http://www.mjmwired.net/kernel/Documentation/cgroups.

[14] 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.