A Convex Optimization-Based Approach to Control of Uncertain Execution Platforms

Lindberg, Mikael

2010

A Convex Optimization-Based Approach to Control of Uncertain Execution Platforms

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

***Abstract*— The problem of resource management in a system of a-priori unknown software components executing on nondeterministic hardware is considered. The approach uses on-line parameter estimation to address uncertainties and combines this with a convex optimization-based control scheme able to handle overload situations. An algorithm to solve the optimization in real-time is presented together with performance analysis through simulations. An implementation of the approach is experimentally compared with a static analysis scheme using worst case a-priori estimates. It is demonstrated that the presented approach outperforms the static scheme in situations with uncertainty and that the advantage increases as uncertainty grows.**

I. Introduction

The consolidation of telephony, media and general utility computing into modern smart phones has led to an increased focus on managing limited resources to make devices flexible while also robust, powerful and yet efficient. As a consequence, the method for allocating the CPU resource to competing subsystems becomes central to the performance of the product. This is complicated by growing performance uncertainties in emerging CPU technology.

The type of scenario that drives system design is illustrated in Figure 1. In a video telephony situation, a set of components execute concurrently, thereby competing for the CPU resource. Few assumptions can be made regarding the amount of resource each component needs in order to perform at an acceptable level. Two important reasons for this is the data driven behavior of algorithms such as compressed media encoding and limited knowledge about 3rd party components. Similarly, resource availability is hard to predict, as power management and other hardware monitoring mechanisms will enforce temperature constraints and maximize battery life by throttling CPU performance. The system is therefore expected to be working on the resource constraint all all time.

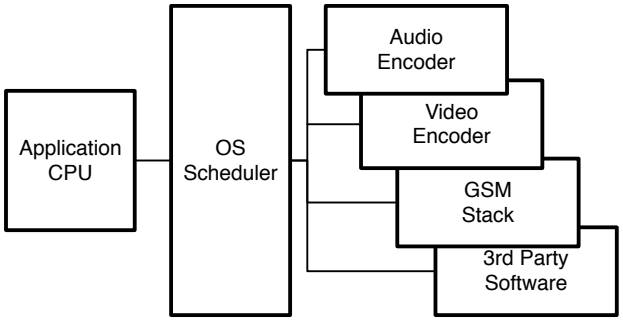

Fig. 1. Overview of a video telephony scenario with a typical set of components. Some of these are well known by the phone designer, while others can be downloaded to the phone by the user.

Growing uncertainties makes the problem difficult to address with static analysis methods, e.g. Rate Monotonic scheduling [18]. Worst case approaches yield designs that underutilize the system, which raise unit cost. Several works propose introducing feedback control in the scheduling mechanism ([2], [9]) to deal with uncertainty. This paper continues along these lines, using control and estimation techniques in order to maximize performance in environments with limited resources using minimum a-priori knowledge.

II. Contributions and outline

The three challenges to resource management for embedded systems this papers aims to solve are

- allocation for systems using CPUs with uncertain execution speed and software with uncertain execution time,
- allocation strategies in overload situations and
- implementation on limited precision hardware.

Section III presents a survey of related work. A component model intended for control and estimation is developed in Section IV and Section V then discusses methods for estimating the model parameters on-line. Section VI poses the resource allocation problem in terms of the model parameters and Section VII explains the incremental algorithm used to solve it in real-time. Details regarding an implementation are provided in Section VIII. The paper concludes with execution and simulation results, which are presented in Section IX, and some thoughts on future work in Section X.

III. Related Research

The ideas presented in this paper rely on the existence of a reservation based scheduling layer, used to partition the CPU-resources predictably. Theory for reservations can be derived from traditional EDF scheduling, resulting in e.g. the Constant Bandwidth Server (CBS) formulation by Abeni & Buttazzo in [3]. The concept has been extended to include constrained resource situations through Elastic Reservations in [8] and [6]. 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 an optimization problem, with a prominent example in Rajkumar's work on Q-RAM, which was originally described in [20]. This has since been extended to include multi-resource cases in [13]. 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.

Resource allocation is often seen as knapsack- or bin-packing problems (see e.g. [15]), but the difficulty of solving these types of problems makes the formulation ill suited for use in embedded systems.

An approach, using convex optimization in realtime, has been discussed in recent publications by Grant and Boyd[14]. 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 problem structure 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 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 relevant for the problem studied in this paper, but more focused on computer server farms than the embedded space.

IV. System Model

Consider a system of software components executing on the same CPU. The capacity of the CPU is shared between the components through preemptive scheduling. Through the use of reservation based scheduling techniques, the CPU computational capacity can be split among the components. Each component will experience the execution platform as a private but slower environment, where the execution of one component does not influence the timing behavior of the other components. This property of a scheduling mechanism is called *temporal isolation* and makes it possible to approximate the multiplexed CPU as a divisible, fluid resource.

The type of components modeled in this paper execute in a cyclic manner, where the completion of each cycle results in the output of some form of result. Examples include media decoders that compute video images, control components that compute control signals or network communication components that send and receive messages. Figure 2 shows how each component is modeled.

Generally speaking, the component accumulates the incoming resource, in this case CPU execution, until some quantity has been achieved. At this point it outputs the result and commences a new cycle. The details of the model are:

- $u_i(t)$ is the share of $\alpha(t)$ assigned to component i and is a dimensionless number.

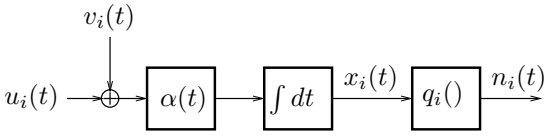

Fig. 2. A cyclic computation software component. $u_i(t)$ is the CPU resource share assigned, $v_i(t)$ is the disturbance on the assigned share caused by the environment, $\alpha_i(t)$ is the execution speed of the CPU, $x_i(t)$ is the accumulated execution, $q_i(t)$ is a staircase function mapping $x_i(t)$ onto $n_i(t)$, which signifies the number of completed cycles.

- $v_i(t)$ is a zero-mean disturbance of this share caused by the scheduler as experienced by component i. The two major sources for this disturbance is the error in the approximation that the CPU is really fluidly divisible and certain system events that interrupt normal execution, such as hardware interrupts or virtual memory handling.
- $\alpha(t)$ denotes the speed at which the CPU executes instructions and has the unit of completed instructions per time unit.
- $x_i(t)$ is the accumulated number of executed instructions for component i.
- q_i is a staircase function defined as

$$q_i(x) = \max k, s.t. \sum_{j=1}^{k} \rho_i^j \leq x_i(t) \qquad (1)$$

  for some sequence $< \rho_i^k >$ that describes the amount of execution it takes component i to complete computation cycle k.
- $n_i(t)$ signifies the number of cycles completed by i.

The objective is to control the growth rate of $n_i(t)$ through $u_i(t)$.

In a traditional real-time setting, the objective would be to guarantee an upper bound on the time between two completed cycles. The key to do this lies in assuming knowledge of bounds on ρ_i, $\alpha(t)$ and $v_i(t)$ (see e.g. [18] and [7] for some classic results).

This paper focuses on cases where such bounds are either unavailable or too conservative and instead make the following assumptions:

(a) the sequence ρ_i^k is unknown and changes slowly over time,

(b) while $\alpha(t)$ cannot be directly measured, it is possible to measure $x_i(t)$ and thereby

$$\int_{t_0}^{t_1} \alpha(t)dt$$

(c) The completion of a cycle is observable through an event, defined as the tuple $(t_i^k, x_i(t_i^k))$, where t_i^k is the time when component i completed cycle k.

A. Components with rate-based utility

For media applications, the quality strongly connected to 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. Using resource management terminology, it can be said that these applications have *rate-based utility*. Rate is here taken as the number of completion cycles over period of time.

It has been demonstrated that feedback control is robust to jitter in sampling and setting of control signal (see e.g. [10]). In much the same way, video playback can suffer both jitter and the occasional frame loss without significant degradation in quality ([11]). In this paper, applications of this type is considered timing sensitive rather than hard real-time. Under this assumption, the requirement that all deadlines must be met is relaxed and the formal objective becomes to minimize the rate error

$$e_i = r_i - y_i(t) = r_i - \frac{1}{h_i}(n_i(t) - n_i(t-h_i)) \qquad (2)$$

where r_i is the *rate set-point* for component i and h_i is a time interval chosen depending on r_i. It is assumed from here on that $h_i \gg r_i^{-1}$. Using assumption (a), ρ_i^k is approximated by its mean value $\bar{\rho}_i$ and $y_i(t)$ written as as

$$\begin{aligned} y_i(t) =& \frac{1}{h_i}\frac{x_i(t) - x_i(t-h_i)}{\bar{\rho}_i} = \\ & \frac{u_i(t-h_i)}{\bar{\rho}_i}\frac{\int_{t-h_i}^{t}\alpha(t)dt}{h_i} = u_i(t-h_i)\frac{\bar{\alpha}}{\bar{\rho}_i} \end{aligned} \qquad (3)$$

assuming u_i is constant in the interval $[t-h_i, t)$. As $\bar{\alpha}/\bar{\rho}_i$ is an unknown quantity it will be treated as one unknown parameter k_i and can be seen as the static gain from u_i to y_i, resulting in the simple linear model

$$y_i(t) = k_i u_i(t-h_i) \qquad (4)$$

A slight complication arises from the fact that most components of this type will limit its execution rate once it reaches r_i. A more complete model would then be

$$y_i(t) = \begin{cases} k_i u_i(t-h_i) & 0 \le u_i(t-h_i) \le \frac{r_i}{k_i}, \\ r_i & u_i(t-h_i) > \frac{r_i}{k_i} \end{cases} \qquad (5)$$

Figure 3 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 controlling the CPU share available to the player below what is required for full rate playback, it starts to skip frames to keep up.

Estimating the parameters k_i in (5) would be straight forward if the rate y_i 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 filtering arrival times[4]. It is worth noting that a benefit from measuring the rate through the completion events is that this poses a 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 below.

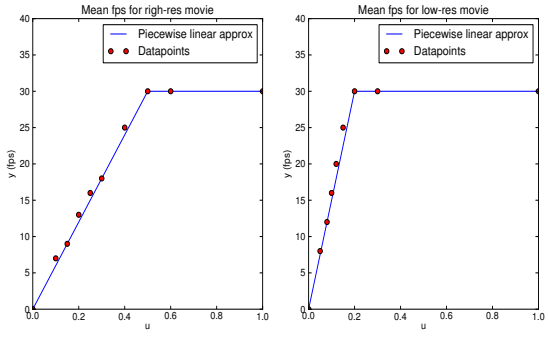

Fig. 3. Experimental results of controlling CPU-share for the MPlayer decoder using Linux 2.6.27 and Control Groups. The diagrams show how the frame rate per second (fps) depends on the amount of CPU share allocated to the decoder. The rate increases linearly with share until the movie can be played back at encoded rate.

V. Event based estimation

This section will discuss two approaches to form an estimate of the rate y_i and the parameter k_i from (5). Important considerations for an estimation scheme are:

- that estimates are calculated in predictable time/space,
- how sensitive the estimate is to noise and
- how fast the scheme can detect a change in rate

The following assumptions will be made on the component

- The rate is constant over the interval of time h_i
- k_i can in h_i be considered to be generated by 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 needs 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 view the estimation of y_i as a prediction problem, where the objective is to predict the time until the next event arrives. If $\Delta(n)$ denotes the time between event n and $n-1$, using the assumed stationarity stochastic properties of n, a predictor can be written on the discrete time shift operator form

$$\hat{\Delta}(n+1) = \frac{B(q^{-1})}{A(q^{-1})}\Delta(n) \qquad (6)$$

$$\hat{y_i}(n) = \frac{1}{\Delta(n+1)} \qquad (7)$$

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 $t > \hat{\Delta}(n+1)$, 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_n denotes the arrival time of event n and $\Delta_e(n)$ denotes the extended sequence $\{..., \Delta(n-1), \Delta(n), (t-t_n)\}$, the resulting estimator for $y(t)$ would then be

$$\begin{aligned}\hat{\Delta}(n+1) =& \frac{B(q^{-1})}{A(q^{-1})}\Delta(n)\\ \hat{\Delta}_e(n+1) =& \frac{B(q^{-1})}{A(q^{-1})}\Delta_e(n)\\ \hat{y_i}(t) =& \frac{1}{\max(\hat{\Delta}(n+1), \hat{\Delta}_e(n+1))}\end{aligned} \tag{8}$$

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 noisy estimate.

C. k-parameter estimation

Given an estimate of the current execution rate $\hat{y_i}(t)$, falling back on (5) results in the following estimate:

$$\hat{k_i}(t) = \frac{\hat{y_i}(t)}{u_i(t-h_i)} \tag{9}$$

Unfortunately, this estimate does not take the disturbance w_i into account. As it is possible to measure $x_i(t)$ directly, a better estimator would be

$$\hat{k_i}(t) = \frac{\hat{y_i}(t)}{x_i(t_1) - x_i(t_0)} \tag{10}$$

if t and t_0 and t_1 are such that the events used to form $\hat{y_i}(t)$ occur in the interval (t_0, t_1).

VI. 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. If each component is associated with a utility function [5, pp 130], a natural choice would be an aggregate of these. 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 is that it should fit the convex framework presented in Section VI-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

$$\begin{aligned}\min J(u) =& \sum_{i=1}^{N} w_i J_i(u_i)\\ u \geq& 0\\ 1^T u \leq& u_{tot}\end{aligned} \tag{11}$$

under the restriction that $J_i(u_i)$ is a convex differentiable function. In the examples presented in this paper, $J_i(u_i) = e_i^2$, a common quadratic programming (QP) structure. Note, however, that this is only an example. The solver algorithm presented below allows for heterogenous mixes of convex differentiable utility functions. This can be exploited by system designers to design policies which distinguishes between different classes of software, e.g. system services or 3rd party add-ons. Even if the quadratic form is used, general QP solvers are difficult to implement using fixed point arithmetics. Code generation [14] and automatic tools [16] can remove some of these issues, but are not suitable when the problem structure is not known beforehand.

Let $\bar{y_i}(u_i) = k_i u_i$ denote the steady state rate as a function of assigned resource. The cost function then becomes

$$J = \sum_{i=1}^{N} w_i e_i^2 = \sum_{i=1}^{N} w_i (r_i - \bar{y_i}(u_i))^2 \tag{12}$$

This formulation is much like the water filling problem (see [5, pp 245]) used for power allocation in communications theory, with the difference that the set of utility functions can here 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) quickly accounts for changing parameters,
3) produces at least partial results in predictable time and memory, and
4) can improve upon a previous allocation even if aborted before optimum was computed.

VII. 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 $J^k_{i,j}$ 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

$$\begin{aligned} \min_{\delta} J^{k+1}_{i,j} =& w_i J_i(u^k_i + \delta) + w_j J_j(u^k_j - \delta) \\ & s.t. - u^k_i \leq \delta \leq u^k_j \end{aligned} \tag{13}$$

This ensures that

$$J^{k+1} \leq J^k \tag{14}$$

In other words, by in each step solving a subproblem, performance will improve incrementally. Solving this minimization problem for general convex functions $J_i(u_i)$ can be done by modifications to unconstrained methods such as Newton-Rhapson or even bisection. In the case of components modeled by (5), near closed form expressions can be obtained for some common cost function, see [17] for examples.

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. [5]). Posing (11) on standard form, the Lagrangian becomes

$$L(u, \lambda, \nu) = \sum_{i=1}^{N} w_i J_i(u_i) - \lambda^T u + \nu(1^T u - u_{tot}) \tag{15}$$

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

$$\frac{\partial L(u_i, \lambda, \nu)}{\partial u_i} = w_i \frac{\partial J_i(u_i)}{\partial u_i} - \lambda_i + \nu = 0 \tag{16}$$

it can be seen that in an optimal point, either $u_i = 0$ or $-w_i \partial J_i(u_i)/\partial u_i = \nu$. Let $\psi_i(u_i) = -w_i \partial J_i(u_i)/\partial u_i$. Going back to the water filling analogy, $\psi_i(u_i)$ would represent the water level in container i. If $u_i = 0$ and therefore $\lambda_i > 0$, then $\psi_i(u_i)$ must be less than ν. In other words, a point where $\psi_i(u_i) > \psi_j(u_j)$ and $u_j > 0$ does not minimize (13).

- If the algorithm tries to select C_i, C_j so that $\psi_i(u_i) > \psi_j(u_j)$ and $u_j > 0$, solving (13) results in $J^{k+1} < J^k$.
- If there is no such pair to select, then that point satisfies the KKT-conditions of (11) 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 (13) it is evident that the size of the gain depends on

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

The two criteria can be in conflict, there are large variations in k_i. It will in this paper be assumed that the components require resources of the same magnitude.

An intuitive strategy would be to sort the components according to $\psi_i(u_i)$ and select the two furthest apart, skipping the ones with zero resources on the lower end. If $\psi_i(u_i)$ is taken as the water level, it makes sense to equalize the two extremes. 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. [12] 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 $(r_i, k_i, u_i, \partial J_i/\partial u_i)$, 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

$$\min J_{0,1} = (25 - 40(1 - \delta))^2 + (15 - 20\delta)^2 \tag{17}$$

$$1 \geq \delta \geq 0 \tag{18}$$

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	(u_0, ψ_0)	(u_1, ψ_1)	(u_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.

VIII. Implementation Aspects

For experimental purposes, an implementation of the framework has been done for Linux 2.6 using the CFS scheduler and control groups [19] for resource allocation. The execution platform was a 2.40 Ghz Intel Pentium(R) 4 D based computer with 512Mb of memory and all software was compiled with gcc 4.3.2 using the -O3 compiler flag. The resulting experiment platform can be said to consist of three major parts.

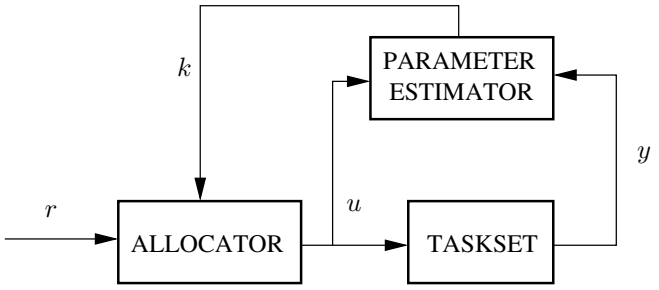

Fig. 4. Proposed control structure.

A. Component Set

The components are 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 packet 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 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 executes 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 VII.

IX. 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 5 and Figure 6. 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. (8) suggests that the filter could have other structures, but that would require a model of the process noise and sensor dynamics to exploit.

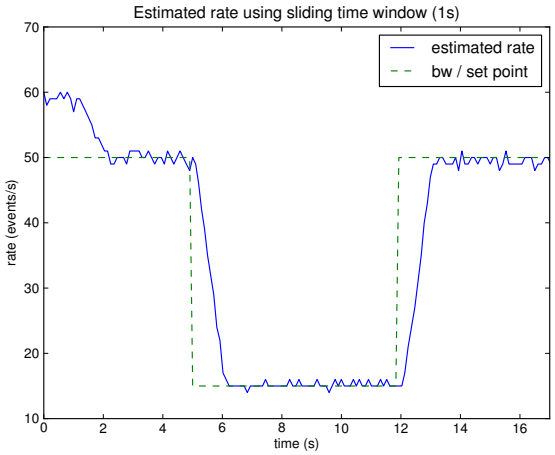

Fig. 5. Event based estimation using a sliding time window. The estimated parameter k is used in feedforward control to show that the model can be used to accurately control the process.

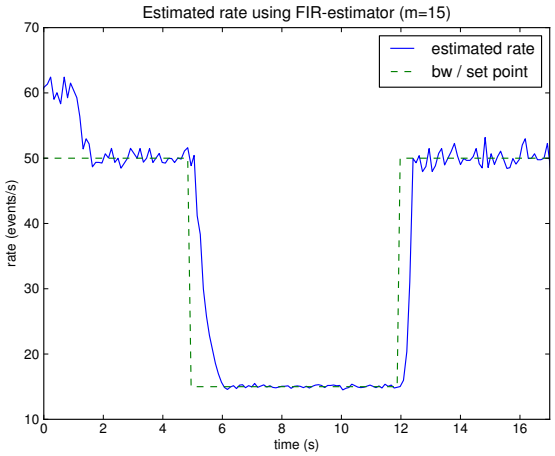

Fig. 6. Event based estimation using an event filter with FIR structure. The estimated parameter k is used in feedforward control to show that the model can be used to accurately control the process.

B. Optimization solver performance

The optimization problem solver was implemented in ANSI C using an off 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 7 The algorithm has been benchmarked using large sets of random components. The algorithm was run 10 times for each set. The fluctuations in completion times is most likely due to sorting artifacts and cache misses. Figure 8 shows how the iteration time increases as the number of components in the problem grows, while Figure 9 displays how long it takes to complete the optimization. Even for a fairly large tasks sets, the time is reasonable and running it as part of a periodic controller is deemed reasonable.

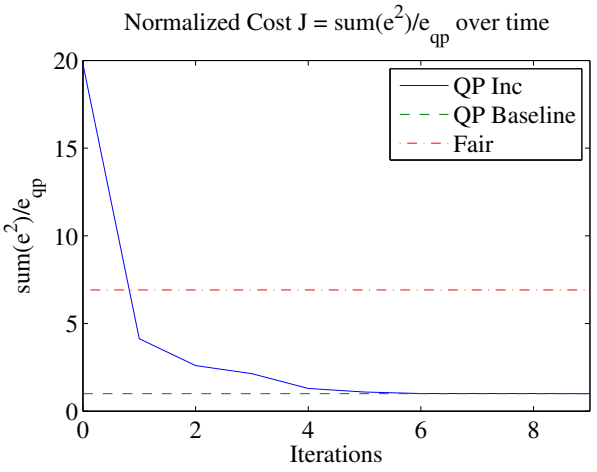

Fig. 7. Solver running over a set of 3 random components with the *quadprog* baseline solution computed with MATLAB as a baseline and the fair allocation provided for comparison.

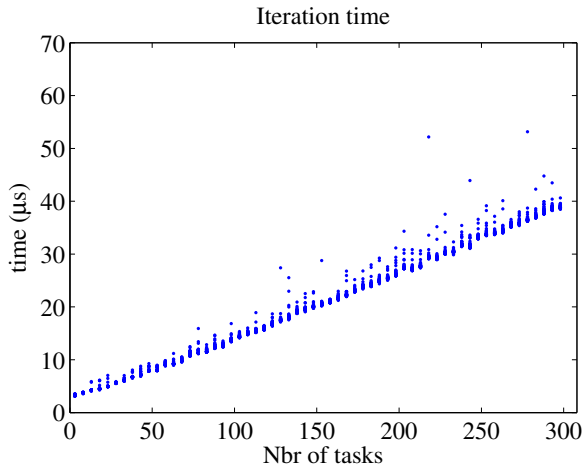

Fig. 8. Iteration time as a function of components in problem.

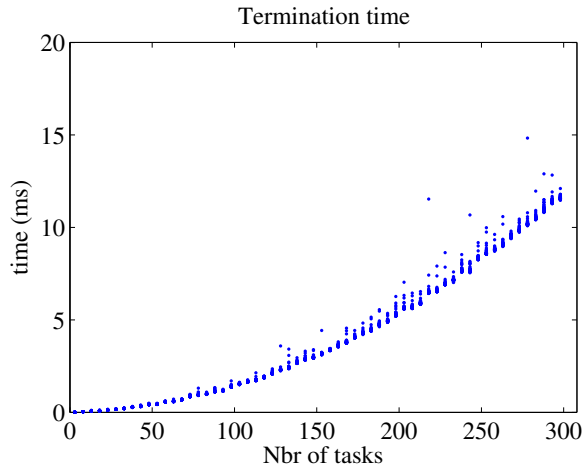

Fig. 9. Optimization time as a function of components in problem.

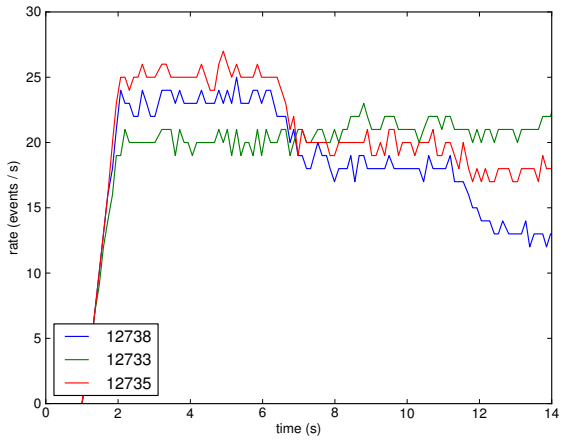

Fig. 10. 3 random components running in a constrained resource environment. Each component changes its dynamics every 3 seconds after which it takes about 1 second for the estimator to converge and a new allocation is calculated.

C. Online allocation

Figure 10 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 11 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 12 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. 50 experiments were run for each ratio and the plot shows the average of the results. This demonstrates that it is 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.

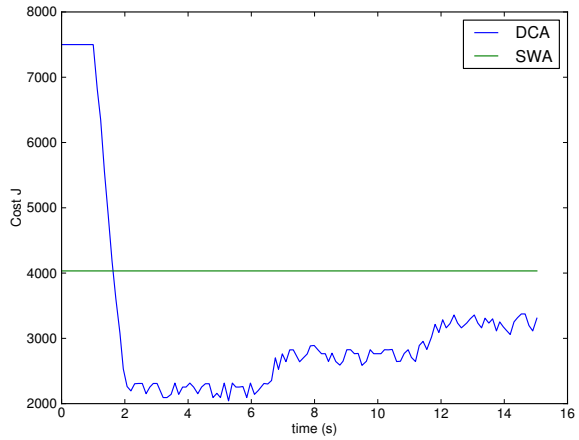

Fig. 11. The performance cost function over time for Dynamic Convex Allocation (DCA) compared with the Static Worst-case Allocation (SWA) baseline.

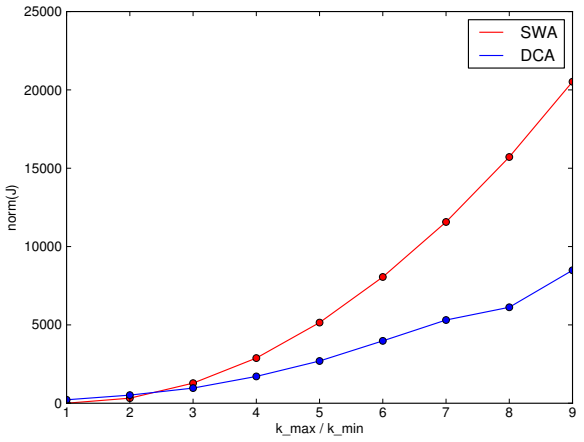

Fig. 12. A comparison between DCA and SWA for different variability of execution time.

X. 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. One important direction therefore must be to see how to model a system with components that consume hardware resources (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, how to bootstrap new components onto a running system must be investigated. A newly arriving component will have unknown parameters and the utility function is therefore unknown except perhaps in structure. One possibility would be to include the uncertainty of the estimates in the cost function so that unknown components would be allowed to run in order to collect information about them in a structured way.

References

[1] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu. Introduction to control theory and its application to computing systems. In Z. Liu and C. H. Xia, editors, *Performance Modeling and Engineering*. Springer US, 2008.

[2] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002.

[3] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*, pages 4 – 13, Madrid, Spain, 1998.

[4] D. R. Barr, A. G. Glen, and H. F. Graf. The "straightforward" nature of arrival rate estimation? *The American Statistician*, 52(4):346–350, 1998.

[5] S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, 2004.

[6] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1):7–24, 2002.

[7] G. C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1997.

[8] G. C. Buttazzo and L. Abeni. Elastic task model for adaptive rate control. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*, pages 286–295, Madrid, Spain, 1998.

[9] A. Cervin, J. Eker, B. Bernhardsson, and K. Årzén. Feedback–feedforward scheduling of control tasks. *Real-Time Systems*, 23:23–53, 2002.

[10] A. Cervin, B. Lincoln, J. Eker, K.-E. Årzén, and G. 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 (RTCSA 2004)*, Göteborg, Sweden, 2004. Best paper award.

[11] M. Claypool and J. Tanner. The effects of jitter on the peceptual quality of video. In *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, pages 115–118, 1999.

[12] T. H. Cormen. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[13] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS 2004)*, pages 12 – 22, Lisbon, Portugal, 2004.

[14] M. Grant and S. P. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`, 2010.

[15] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer-Verlag Berlin, Heidelberg, Germany, 2004.

[16] S. Kim, K.-I. Kum, and W. Sung. Fixed-point optimization utility for c and c based digital signal processing programs. *IEEE Transactions on Circuits and Systems*, 45(11):1455 – 1464, 1998.

[17] M. Lindberg. Constrained online resource control using convex programming based allocation. In *Proceedings of the 4th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2009)*, San Francisco, CA, USA, 2009.

[18] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2000.

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

[20] R. Rajkumar, C. Lee, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 298 – 307, San Francisco, CA, USA, 1997.