Distributed Management of CPU Resources for Time-Sensitive Applications

Georgios Chasparis, Martina Maggio, Karl-Erik Årzén, Enrico Bini
Lund University, Sweden

***Abstract*— The number of applications sharing the same embedded device is increasing dramatically. Very efficient mechanisms (resource managers) for assigning the CPU time to all demanding applications are needed. Unfortunately existing optimization-based resource managers consume too much resource themselves.**

In this paper, we address the problem of distributed convergence to efficient CPU allocation for time-sensitive applications. We propose a novel resource management framework where both applications and the resource manager act independently trying to maximize their own performance measure and according to a utility-based adjustment process. Contrary to prior work on centralized optimization schemes, the proposed framework exhibits adaptivity and robustness to changes both in the number and nature of applications, while it assumes minimum information available to both applications and the resource manager. It is shown analytically that efficient resource allocation can be achieved in a distributed fashion through the proposed adjustment process. Experiments using the TrueTime Matlab toolbox show the validity of our proposed approach.

I. Introduction

A current trend in embedded computing is that the number of applications that should share the same execution platform increases. The reason for this is the capacity increase of new hardware platforms, e.g., through the use of multi-core techniques, as well as the increase of user demands. An example includes the move from federated to integrated system architectures in the automotive industry.

As the number of applications increases, the need for better mechanisms for controlling the execution behavior of the applications becomes apparent. Increasingly often, virtualization or resource reservation techniques [1], [2] are used. According to these techniques, each reservation is viewed as a virtual processor executing at a fraction of the speed of the physical processor, i.e., the *bandwidth* of the reservation, while the tasks in the different reservations are temporally isolated from each other. Another trend in embedded computing is the increase in temporal uncertainty, both due to the increased hardware complexity, e.g., shared cache hierarchies, and the increased chip density. Hence, using dynamic adaptation is crucial. In the resource reservation case this means that the bandwidth assignment is decided on-line based on feedback from the applications.

An orthogonal dimension along which the application performance can be tuned is the selection of the *service level* of the application. It is assumed that an application is able to execute at different service levels, where a higher service level implies a higher quality-of-service at the price of higher resource consumption. Examples are adjustable video resolutions, the amount of data sent through a socket channel to render a web page, and the possibility to execute a controller at different sampling rates.

The typical solution to this problem is the implementation of a *resource manager* (RM), which is in charge of:

- assigning the resources to each application;
- monitoring the resource usage and possibly adjusting the assignment based on the actual measurements;
- assigning the service levels to each application, so that an overall delivered quality is maximized.

This is often done through the use of *optimization* and *feedback* from the application.

Resource managers that are based on the concept of feedback monitor the progress of the application and adjust the resource management based on some measurements [3], [4]. In these early approaches, however, quality adjustment was not considered. Cucinotta et al. [5] proposed an inner loop to control the resource allocation nested within an outer loop that controls the overall delivered quality.

Optimization-based resource managers also received a considerable attention [6], [7]. These approaches, however, rely on the solution of a centralized optimization problem that determines the amount of assigned resource and sets the service levels of all applications. If, instead, the service levels are assigned by the applications, the RM can certainly be more lightweight. In the context of networking, Johansson et al. [8] modeled the service provided by a set of servers to workloads belonging to different classes as a utility maximization problem. However, there is no notion of adjustment of the service level of the application.

An example of combined use of optimization and feedback was developed in the ACTORS project [9], [10]. In this project, applications provide a table to the RM describing the required amount of CPU resources and the expected QoS achieved at each supported service level [9], [10]. In the multi-core case, applications are partitioned over the cores and the amount of resources is given for each individual partition. Then, the RM decides the service level of all applications and how the partitions should be mapped to physical cores using a combination of ILP and first-fit bin-packing. However, such approaches have several drawbacks. On-line centralized optimization can be inefficient and a proper assignment of service level requires application knowledge, i.e., it is something that is better made by the application itself rather than the RM.

The research leading to these results was supported by the Linneaus Center LCCC, the Swedish VR project n.2011-3635 "Feedback-based resource management for embedded multicore platform", and the Marie Curie Intra European Fellowship within the 7th European Community Framework Programme.

To this end, distributed optimization schemes have also attracted a considerable attention. Subrata et al. [11] considered a cooperative game formulation for job allocation to several service providers in grid computing. Job arrivals follow a Poisson process and a centralized optimization problem is formulated for computing Nash bargaining solutions. Wei et al. [12] proposed a non-cooperative game-theoretic formulation to allocate computational resources to a given number of tasks in cloud computing. Tasks have full knowledge of the available resources and try to maximize their own utility function. Similarly, Grosu and Chronopoulos [13] formulated the load balancing problem among different users as a non-cooperative game and then studied the properties and the computation of Nash equilibria.

In this paper, the problem differs significantly from the grid computing setup of [11] or the load balancing problem of [13], [12] in cloud-computing services. In particular, we are concerned with the problem of allocating the CPU resource among several applications, while applications are able to adjust their own service levels. Under the proposed scheme, both applications and the RM act independently trying to maximize their own performance measure (utility) according to a utility-based adjustment process. Naturally, this framework can be interpreted as a strategic-interaction (or game) among applications and the RM. It is shown analytically that efficient resource allocation can be achieved in a distributed fashion through the proposed adjustment process. Experiments using the TrueTime Matlab toolbox show the validity of our proposed approach.

Below we start by introducing the overall framework.

II. Framework

A. Resource manager & applications

The overall framework is illustrated in Figure 1. A set $\mathcal{I}$ of n *applications* are competing among each other for CPU resources. Since we allow applications to dynamically join or leave, the number n is not constant over time. The resource is managed by a *resource manager* (RM) making sure that the overall allocated resource does not exceed the available one (i.e. the number of cores).

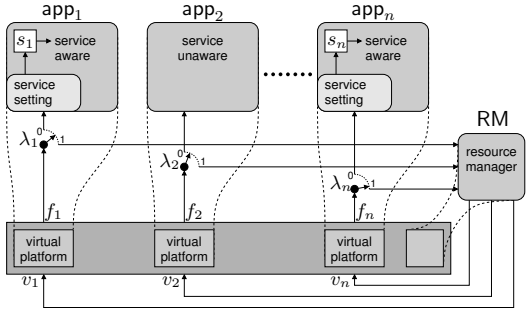

Fig. 1. Resource management framework.

The RM allocates resource through a Constant Bandwidth Server (CBS) [2] with period P_i and budget Q_i. Hence, app_i is assigned a virtual platform with bandwidth $v_i = Q_i/P_i$ corresponding to a fraction of the computing power of a single CPU. The quantity v_i can also be interpreted as speed (relative to the CPU full speed) at which the app_i is running. Obviously not all speeds v_i are feasible, since the sum of them cannot exceed the number m of available CPUs. Formally, we define the set of feasible speed assignments $(v_1, \ldots, v_n)$, as

$$\overline{\mathcal{V}} = \Big\{ \mathbf{v} = (v_1, ..., v_n) \in [0,1]^n : \sum_{i=1}^{n} v_i \leq m \Big\}. \quad (1)$$

Each application $i \in \mathcal{I}$ may change its *service level* $s_i \in \mathcal{S}_i \triangleq [\underline{s}_i, \infty)$, where $\underline{s}_i > 0$ is the minimum possible service level of app_i. Examples of service levels are: the accuracy of an iterative optimization routine, the details of an MPEG player, the sampling frequency of a controller, etc. The service level s_i is an internal state of app_i. Hence it is written/read by app_i only. We denote by $s = (s_1, ..., s_n)$ the profile of service level of all applications evolving within $\mathcal{S} \triangleq \mathcal{S}_1 \times ... \times \mathcal{S}_n$. The framework also allows applications that do not adjust their service level (e.g., app_2 in Figure 1).

B. The matching function

The goal of the proposed resource allocation framework if to find, for each application $i \in \mathcal{I}$, a matching between the service level s_i set by app_i and the speed v_i assigned by the RM to app_i.

Definition 2.1 (Matching function): The quality of the matching between a service level s_i and a speed v_i of app_i is defined by the *matching function* $f_i : \mathcal{S}_i \times [0,1] \to \mathbb{R}$ with the following properties:

- if $|f_i(s_i, v_i)| \leq \delta$, then the matching is *perfect*;
- if $f_i(s_i, v_i) < -\delta$, then the matching is *scarce*;
- if $f_i(s_i, v_i) > \delta$, then the matching is *abundant*.

with δ being a system parameter.

A perfect matching between s_i and v_i describes a situation in which app_i has the right amount of resource v_i when it runs at service level s_i. A scarce (resp., abundant) matching describes the situation when either increasing v_i or decreasing s_i (decreasing v_i or increasing s_i) is needed to move toward the perfect matching.

Notice that the service levels are internal states of the applications, while the virtual platforms $(v_1, \ldots, v_n)$ belong to the RM space. Hence, neither the RM nor the applications have a complete knowledge of the matching function f_i. In fact, the matching function is only measured during run-time. The only properties that we require from any implementation of f_i are that

(P1) $s_i \neq 0 \Rightarrow f_i(s_i, 0) < -\delta$, that is, the matching must certainly be scarce if no resource is assigned;

(P2) $s_i \geq s_i' \Rightarrow f_i(s_i, v_i) \leq f_i(s_i', v_i)$, if an application lowers its service level, then the matching function should increase,

(P3) $v_i \geq v_i' \Rightarrow f_i(s_i, v_i) \geq f_i(s_i, v_i')$, if the bandwidth given to an application is decreased, then the matching function should decrease.

For a real-time application, a *time-sensitive* matching function may be considered, defined as follows:

$$f_i = \frac{D_i}{R_i} - 1,$$

where D_i denotes a *soft-deadline* of the application, and R_i denotes the *job response time*, that is the time elapsing from the start time to the finishing time of a job. For a large class of applications the above matching function can be rewritten with respect to s_i and v_i as follows:

$$f_i = \beta_i \frac{v_i}{s_i} - 1. \tag{2}$$

For example, for *multimedia applications*, the soft deadline D_i can be considered constant, while the response time can be defined as $R_i = {}^{C_i}/_{v_i}$, where $C_i = \alpha_i s_i$ is the execution time per job (at a service level s_i) and v_i is the speed of execution. Similarly, in *control applications*, $R_i = {}^{C_i}/_{v_i}$ where C_i denotes nominal time of execution, while the soft deadline D_i is considered inverse proportional to the sampling frequency (or service level) s_i, i.e., $D_i = {}^{\alpha_i}/_{s_i}$. Both cases lead to a matching function with the form of (2).

C. Adjustment weight

When $|f_i| > \delta$ some adjustment is needed to the service level s_i or to the virtual platform v_i in order to establish a better matching. The *weight* $\lambda_i \in [0,1]$ determines the amount of correction made by each application:

- if $\lambda_i = 0$, then the correction is entirely made by app_i through an adjustment of the service level s_i;
- if $\lambda_i = 1$, then the correction is entirely made by the RM through a change in speed of the virtual platform v_i;
- intermediate values of λ_i correspond to a combined correction made by both the app_i and the RM.

III. Adjustment Dynamics

Below, we introduce a learning procedure under which the applications and the RM adapt to possible changes in their "environment" (other applications) trying to improve their own performance.

A. RM adjustment

To simplify the implementation, the RM updates the bandwidth $\tilde{v}_i = {}^{v_i}/_{m}$, normalized w.r.t. the number of cores. The unused bandwidth and its normalized version, are defined respectively by

$$v_r = m - \sum_{i=1}^{n} v_i, \quad \tilde{v}_r = \frac{v_r}{m} = 1 - \sum_{i=1}^{n} \tilde{v}_i. \tag{3}$$

At time $t = 1, 2, ...$, the RM assigns resources according to the following rule:[1]

1) It measures performance $f_i = f_i(t)$ for each $i \in \mathcal{I}$.
2) It updates the normalized resource allocation vector $\tilde{\mathbf{v}} \triangleq (\tilde{v}_1, ..., \tilde{v}_n)$ as follows:
$$\tilde{v}_i(t+1) = \Pi_{\widetilde{\mathcal{V}}_i}\Big[\tilde{v}_i(t) + \epsilon(t) g_{\mathsf{rm},i}(t)\Big] \tag{4}$$
for each $i = 1, ..., n$, where
$$g_{\mathsf{rm},i}(t) \triangleq -\lambda_i f_i(t) + \sum_{j=1}^{n} \lambda_j f_j(t) \tilde{v}_i(t),$$
and $\Pi_{\widetilde{\mathcal{V}}_i}$ denotes the projection onto the feasible set $\widetilde{\mathcal{V}}_i \triangleq [0, {}^1/_m]$. The unused bandwidth is updated according to (3).
3) It computes the original values of bandwidths by setting $v_i(t+1) = m\, \tilde{v}_i(t+1)$.
4) It updates the time $t \leftarrow t+1$ and repeats.

The above algorithm ensures that the virtual platforms v_i are always feasible according to (1). We will also consider the step-size sequence $\epsilon(t) \triangleq {}^1/_{t+1}$. According to the recursion (4), we should expect that v_i increases when app_i performs poorly compared with the group of applications, i.e., when $\lambda_i f_i$ is small compared to $\sum_{j=1}^{n} \lambda_j f_j$.

In some cases, we will use vector notation for (4)

$$\tilde{\mathbf{v}}(t+1) = \widetilde{\Pi}_{\{\widetilde{\mathcal{V}}_i\}}\left[\tilde{\mathbf{v}}(t) + \epsilon(t)\mathbf{g}_{\mathsf{rm}}(t)\right] \tag{5}$$

where $\mathbf{g}_{\mathsf{rm}}(t) \triangleq -\mathbf{\Lambda}\mathbf{f}(t) + \left(\mathbf{1}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{f}(t)\right)\tilde{\mathbf{v}}(t)$, with $\mathbf{\Lambda} \triangleq \mathsf{diag}\,\{\lambda_i\}_i$, $\mathbf{f} \triangleq [f_i]_i$ and $\widetilde{\Pi}_{\{\widetilde{\mathcal{V}}_i\}}[\cdot]$ denotes the combination of projections on $\mathcal{V}_i$'s, i.e.,

$$\widetilde{\Pi}_{\{\widetilde{\mathcal{V}}_i\}}[\tilde{\mathbf{v}}] \triangleq \left(\Pi_{\widetilde{\mathcal{V}}_1}[\tilde{v}_1], ..., \Pi_{\widetilde{\mathcal{V}}_n}[\tilde{v}_n]\right).$$

We will use the notation $\widetilde{\mathcal{V}} \triangleq \widetilde{\mathcal{V}}_1 \times ... \times \widetilde{\mathcal{V}}_n$ to denote the space of $\tilde{\mathbf{v}}$.

B. Application adjustment

The RM provides information to all applications to guide their selection of a proper service level. This information will be closely related to the performance of each application i as measured by the matching function f_i.

Let us assume that at time t the RM measures the matching function $f_i(t)$ and it discovers that the matching is *not* perfect. In response to this deviation the RM sets the virtual platforms. **How should instead app_i react to bring the next matching function $f_i(t+1)$ in the interval between $-\delta$ and δ?**

We will consider an adjustment process for the service levels of each application which has the generic form:

$$s_i(t+1) = \Pi_{\mathcal{S}_i}\left[s_i(t) + \epsilon(t) g_{\mathsf{app},i}(t)\right], \tag{6}$$

where the term $g_{\mathsf{app},i}(t)$ captures an "observation" sent by the RM to the application and depends on the matching function f_i. In particular, *we would like this observation to be zero when the matching function is zero, the most desirable case.*

In several cases, we will use the more compact form:

$$\mathbf{s}(t+1) = \widetilde{\Pi}_{\{\mathcal{S}_i\}}\left[\mathbf{s}(t) + \epsilon(t)\mathbf{g}_{\mathsf{app}}(t)\right]. \tag{7}$$

where $\mathbf{s} \triangleq [s_i]_i$ and $\mathbf{g}_{\mathsf{app}} \triangleq [g_{\mathsf{app},i}]_i$.

Below, we identify two candidates for the observation term $g_{\mathsf{app},i}$.

1) Scheme (a): The first scheme is rather generic and independent of the specific form of the matching function f_i. It simply defines $g_{\mathsf{app},i}(t) \triangleq f_i(t)$. Naturally, in this case, the specifications we set above are satisfied.

2) Scheme (b): The second scheme takes into account the specific form of the matching function (2). Assuming that the app_i could read $v_i(t)$, a natural way for the application to adjust its service level is to simply set $s_i(t+1) = \beta_i v_i(t)$, since, by (2), if s_i matches $\beta_i v_i$, the matching function will

[1] Although the performance f_i of application i is a function of both the service level s_i and the virtual platform v_i, the RM simply receives an instance of this function. Thus, we abuse notation by writing $f_i = f_i(t)$.

become zero. However, setting the next service level $s_i(t+1)$ according to this rule is an open-loop technique that relies on a careful estimation of β_i, which may be unavailable. From the (possibly non-zero) measurement $f_i(t)$ at time t we can actually estimate β_i as

$$\beta_i = (1 + f_i(t))\frac{s_i(t)}{v_i(t-1)}$$

so that the service level update rule becomes

$$s_i(t+1) = (1 + f_i(t))\frac{v_i(t)}{v_i(t-1)}s_i(t). \tag{8}$$

The above recursion may exhibit large incremental differences, $s_i(t+1) - s_i(t)$, which may lead to instability. Hence, we introduce a smoother update rule for the service level s_i that exhibits the same stationary points of (8), by setting:

$$g_{\mathsf{app},i}(t) \triangleq (1 + f_i(t))\frac{v_i(t)}{v_i(t-1)}s_i - s_i. \tag{9}$$

In words, we should expect that s_i decreases when $f_i < 0$ and $v_i(t)/v_i(t-1) < 1$, i.e., when the application is doing poorly and the assigned resources have been decreased. The term $v_i(t)/v_i(t-1)$ provides a look-ahead information to the application about the expectation over future available resources.

C. Resource allocation game

Briefly, we would like to note that the above framework naturally introduces a strategic-form game (cf., [14]) between the applications and the RM. Note that a strategic-form game is defined as a collection of: (i) a set of players $\mathcal{P}$, which here is defined as the set of applications and the RM, i.e., $\mathcal{P} \triangleq \mathcal{I} \cup \mathsf{RM}$; (ii) a set of available *actions*, $\mathcal{A}_p$, for each player $p \in \mathcal{P}$, i.e., service level for each application and allocation of virtual platforms for the RM; and (iii) a set of *utilities* or *performance measures*, $u_p : \mathcal{A}_p \to \mathbb{R}$, for each player p. The selection of these utility functions is in fact open-ended. A candidate selection, motivated by the dynamics presented above could be:

- for each app_i, $u_{\mathsf{app},i} : \mathcal{S}_i \times [0,1] \to \mathbb{R}$, such that
$$\nabla_{s_i} u_{\mathsf{app},i}(s_i, v_i) = f_i(s_i, v_i);$$
- for the RM, $u_{\mathsf{rm}} : \mathcal{S} \times \overline{\mathcal{V}} \to \mathbb{R}$, such that for each i:
$$\nabla_{v_i} u_{\mathsf{rm}}(s, v) = -\lambda_i f_i(s_i, v_i) + \sum_{j=1}^{n} \lambda_j f_j(s_j, v_j)\tilde{v}_i$$

Under such strategic-form formulation, each application would prefer to select s_i that makes the matching function f_i equal to zero, while the RM would prefer to select a virtual platform allocation that would be "*fair*" to all applications. Furthermore, under this framework the adjustment dynamics presented before introduce a natural way for searching for a *Nash equilibrium* allocation (cf. [14]).

IV. Convergence Analysis

In this section, we analyze the asymptotic behavior of the RM (5) and the applications (7). *For the remainder of the paper,* we will consider scheme (b) for the applications adjustment.

For the sake of analysis, we will abuse notation by taking the observation signals $g_{\mathsf{rm},i}$ and $g_{\mathsf{app},i}$ as functions of the service levels and virtual platforms. In particular, for the RM update, the observation signal $g_{\mathsf{rm},i}(t)$ is replaced by $g_{\mathsf{rm},i} : \mathcal{S} \times \mathcal{V}^n$ such that:

$$g_{\mathsf{rm},i}(\mathbf{s}, \tilde{\mathbf{v}}) \triangleq -\lambda_i f_i(s_i, v_i) + \sum_{j=1}^{n} \lambda_j f_j(s_j, v_j)\tilde{v}_i,$$

where $v_i = m\,\tilde{v}_i$. Likewise, the observation signal $g_{\mathsf{app},i}(t)$ in the application i's adjustment is replaced by the function $g_{\mathsf{app},i} : \mathcal{S}_i \times \mathcal{V}$ such that:

$$g_{\mathsf{app},i}(s_i, \tilde{v}_i, y_i) = (1 + f_i(s_i, v_i))\frac{\tilde{v}_i(t)}{y_i(t)}s_i(t) - s_i(t),$$

where

$$y_i(t+1) = y_i(t) + \epsilon(t)\left(\tilde{v}_i(t) - y_i(t)\right). \tag{10}$$

The reason for introducing the new state variable $\mathbf{y} \triangleq (y_1, ..., y_n)$ is to deal with the different time indices in (9).

The asymptotic behavior of the overall adjustment process described by (5), (7) and (10), can be characterized as follows:

Proposition 4.1: The overall recursion (5), (7) and (10) is such that the sequence $\{(\mathbf{s}(t), \tilde{\mathbf{v}}(t), \mathbf{y}(t))\}$ converges[2] to some limit set of the ODE:

$$(\dot{\mathbf{s}}, \dot{\tilde{\mathbf{v}}}, \dot{\mathbf{y}}) = \mathbf{g}(\mathbf{s}, \tilde{\mathbf{v}}, \mathbf{y}) + \mathbf{z}(t), \tag{11}$$

where $\mathbf{g} \triangleq (\mathbf{g}_{\mathsf{app}}, \mathbf{g}_{\mathsf{rm}}, \tilde{\mathbf{v}} - \mathbf{y})$ and $\mathbf{z} \triangleq (\mathbf{z}_{\mathsf{app}}, \mathbf{z}_{\mathsf{rm}}, 0)$ is the minimum force required to drive $\tilde{v}_i(t)$ to $\tilde{\mathcal{V}}_i$ and $s_i(t)$ back to $\mathcal{S}_i$. Finally, if $E \subset \mathcal{S} \times \tilde{\mathcal{V}} \times [0,1]^n$ is a locally asymptotically stable set in the sense of Lyapunov[3] for (11) and $(\mathbf{s}(t), \tilde{\mathbf{v}}(t), \mathbf{y}(t))$ is in some compact set in the domain of attraction of E, then $(\mathbf{s}(t), \tilde{\mathbf{v}}(t), \mathbf{y}(t)) \to E$.

Proof: The proof is based on Theorem 2.1 in [16]. ■

The above proposition relates the asymptotic behavior of the overall discrete-time recursion with the limit sets of the ODE (11). Since the *stationary points*[4] of the vector field $\mathbf{g}$ are invariant sets of the ODE (11), then they are also candidate attractors for the recursion. In the following sections, we analyze the convergence properties of the recursion with respect to the stationary points of the ODE (11).

A. Stationary points

Lemma 4.1 (Stationary Points): Any stationary point of the ODE (11) satisfies all the following conditions:

(C1) $\sum_i \lambda_i f_i(s_i^*, v_i^*)\tilde{v}_j^* = \lambda_j f_j(s_j^*, v_j^*)$, or $\{\tilde{v}_j^* = 1/m \wedge f_j(s_j^*, v_j^*) \leq 0\}$;
(C2) $f_j(s_j^*, v_j^*) = 0$, or $\{s_j^* = \underline{s}_j \wedge f_j(s_j^*, v_j^*) \leq 0\}$,
(C3) $y_j^* = \tilde{v}_j^*$,

for all $j \in \mathcal{I}$. Furthermore, the set of stationary points is non-empty.

Proof: Condition (C1) is an immediate consequence of setting $g_{\mathsf{rm},j}(s^*, \tilde{v}^*) + z_{\mathsf{rm},j} = 0$, $j \in \mathcal{I}$. Likewise, conditions

[2] By $x(t) \to A$ for a set A, we mean $\lim_{t\to\infty} \mathrm{dist}(x(t), A) = 0$.
[3] See [15, Definition 3.1].
[4] The stationary points of an ODE $\dot{x} = \mathbf{g}(x)$ are defined as the points in the domain D for which $\mathbf{g}(x) = 0$.

(C2) and (C3) follow directly from setting $g_{\mathsf{app},j}(s^*, \tilde{v}^*, y_i) + z_{\mathsf{rm},j} = 0$ and $y_j = \tilde{v}_j$, $j \in \mathcal{I}$.

Regarding existence of stationary points, and without loss of generality, we restrict attention to the allocations $(\mathbf{s}, \tilde{\mathbf{v}})$ for which $f_i(s_i, v_i) \leq 0$ for all $i \in \mathcal{I}$ (since, if there exists app_i for which $f_i(s_i, v_i) > 0$, then app_i may always increase s_i to match v_i without affecting the matching functions of the other applications). Under this restriction, we consider two cases: (a) there exists $\mathbf{s}^* \in \mathcal{S}$ and $\tilde{\mathbf{v}}^* \in \widetilde{\mathcal{V}}$ such that $f_j(s_j^*, v_j^*) = 0$ for all $j \in \mathcal{I}$; and (b) there exists at least one $j \subset \mathcal{I}$ such that $f_j(s_j, v_j) < 0$ for all $s_j \in \mathcal{S}_j$ and $\tilde{v}_j \in \widetilde{\mathcal{V}}_j$. In case (a), $(\mathbf{s}^*, \tilde{\mathbf{v}}^*, \tilde{\mathbf{v}}^*)$ is a stationary point of the ODE (11). In case (b), $\sum_i \lambda_i f_i(s_i, v_i) < 0$ for the allocations under consideration. Then, condition (C1) gives:

$$\tilde{\mathbf{v}}^* = \mathbf{h}(\mathbf{s}^*, \tilde{\mathbf{v}}^*) \triangleq \frac{\mathbf{\Lambda f}(\mathbf{s}^*, \mathbf{v}^*)}{\mathbf{1}^{\mathrm{T}} \mathbf{\Lambda f}(\mathbf{s}^*, \mathbf{v}^*)}, \tag{12}$$

where $\mathbf{v}^* = m\,\tilde{\mathbf{v}}^*$. Since the function $\mathbf{h}(\mathbf{s}^*, \cdot)$ is continuous on a convex and compact set $\widetilde{\mathcal{V}}$, by Brower's fixed point theorem (cf., [17, Corollary 6.6]) $\mathbf{h}(\mathbf{s}^*, \cdot)$ has a fixed point. If the fixed point suggests $\tilde{v}_j^* > 1/m$ for some $j \in J \subseteq \mathcal{I}$, then set $\tilde{v}^* \equiv 1/m$ for all $j \in J$ and resolve (12) to compute a stationary point for the rest of applications $\mathcal{I} \backslash J$. This process will define a stationary point. ∎

In words, the above lemma states that at a stationary point, app_i is either performing sufficiently good (i.e., $f_j(s_j^*, v_j^*) = 0$), or it performs poorly but i) its service level s_i cannot be decreased any further (i.e., $f_j(s_j^*, v_j^*) \leq 0$, $s_j^* = \underline{s}_j$), and/or ii) its virtual platform v_i cannot be increased any further (i.e., $f_j(s_j^*, v_j^*) \leq 0$, $\tilde{v}_j^* = 1/m$).

Note that any pair $(\mathbf{s}, \mathbf{v})$ for which $f_i(s_i, v_i) = 0$ is a stationary point of the ODE (11). This multiplicity of stationary points complicates the convergence analysis, however, in several cases, uniqueness of the stationary point can be shown. Next, we identify a few such special cases.

1) Fixed Service Levels: The following result characterizes the set of stationary points when the service levels are *fixed*, i.e., when each application has only one service level. First, define the following constants:

$$\Theta_s \triangleq \inf_{\tilde{\mathbf{v}} \in \widetilde{\mathcal{V}}} \left| \sum_i \lambda_i f(s_i, v_i) \right| \geq 0$$

and

$$K \triangleq \sup_{i, s_i \in \mathcal{S}_i, \tilde{v}_i \in \widetilde{\mathcal{V}}_i} |f(s_i, v_i)| < \infty.$$

Let also $\gamma > 0$ be such that $\frac{\beta_i m}{s_i} \leq \gamma$, for all i and $s_i \in \mathcal{S}_i$.

Proposition 4.2 (Uniqueness of Stationary Points): For some given $\mathbf{s} \in \mathcal{S}$, let f_i be defined by (2). If $\Theta_s > 0$ and $\rho_s \triangleq K\gamma \left(\sum_i \lambda_i\right)^2 / \Theta_s^2 < 1$, then, the ODE (11) exhibits a unique stationary point.

Proof: We will first consider the unconstrained case where $\widetilde{\mathcal{V}}_i = [0, 1]$ for each $i \in \mathcal{I}$, i.e., there is only one core ($m = 1$). (We will revisit this assumption later.) Under this assumption, a stationary point $\tilde{\mathbf{v}}^*$ satisfies $\tilde{\mathbf{v}}^* = \mathbf{h}(\mathbf{s}, \tilde{\mathbf{v}}^*)$ for some constant vector $\mathbf{s}$. We will find a sufficient condition for uniqueness of the stationary point by computing a sufficient condition under which the mapping $\mathbf{h} : \widetilde{\mathcal{V}} \to \widetilde{\mathcal{V}}$ defines a contraction (cf., [18, Definition 5.1-1]). More specifically, we have,

$$\begin{aligned} & h_j(\mathbf{s}, \tilde{\mathbf{v}}') - h_j(\mathbf{s}, \tilde{\mathbf{v}}) \\ = \; & \frac{\lambda_j f(s_j, v_j')}{\sum_i \lambda_i f(s_i, v_i')} - \frac{\lambda_j f(s_j, v_j)}{\sum_i \lambda_i f(s_i, v_i)} \end{aligned}$$

Note that, $\left|\sum_i \lambda_i f(s_i, v_i') \sum_i \lambda_i f(s_i, v_i)\right| \geq \Theta_s^2 > 0$. Thus,

$$\begin{aligned} |h_j(\mathbf{s}, \tilde{\mathbf{v}}') - h_j(\mathbf{s}, \tilde{\mathbf{v}})| \;\leq\; & \frac{\lambda_j}{\Theta_s^2} \cdot \sum_i \lambda_i \left| f(s_j, v_j') f(s_i, v_i) - \right. \\ & \left. f(s_j, v_j) f(s_i, v_i') \right|. \end{aligned}$$

Also, we have:

$$\begin{aligned} & \left| f(s_j, v_j') f(s_i, v_i) - f(s_j, v_j) f(s_i, v_i') \right| \\ \leq \; & K \left(|f(s_j, v_j') - f(s_j, v_j)| + |f(s_i, v_i) - f(s_i, v_i')| \right) \\ \leq \; & K \left(\frac{\beta_j m}{s_j} |\tilde{v}_j' - \tilde{v}_j| + \frac{\beta_i m}{s_i} |v_i - v_i'| \right) \\ \leq \; & K\gamma \|\tilde{\mathbf{v}}' - \tilde{\mathbf{v}}\|_1 \end{aligned}$$

where $\|\cdot\|_1$ denotes the ℓ_1 norm. Thus, we conclude that

$$|h_j(\mathbf{s}, \mathbf{v}') - h_j(\mathbf{s}, \mathbf{v})| \leq \frac{\lambda_j}{\Theta_s^2} \cdot \sum_i \lambda_i K\gamma \|\tilde{\mathbf{v}}' - \tilde{\mathbf{v}}\|_1,$$

which also implies that

$$\|\mathbf{h}(\mathbf{s}, \mathbf{v}') - \mathbf{h}(\mathbf{s}, \mathbf{v})\|_1 \leq \frac{K\gamma \left(\sum_i \lambda_i\right)^2}{\Theta_s^2} \cdot \|\tilde{\mathbf{v}}' - \tilde{\mathbf{v}}\|_1.$$

We conclude that $\mathbf{h}$ is a contraction if $\rho_s \triangleq K\gamma \left(\sum_i \lambda_i\right)^2 / \Theta_s^2 < 1$, and therefore by Banach Fixed Point Theorem (cf., [18, Theorem 5.1-2]) $\mathbf{h}$ has a unique fixed point.

If, instead, $\widetilde{\mathcal{V}}_i = [0, 1/m]$, for all $i \in \mathcal{I}$, we proceed as follows: We set $\tilde{v}_j^* = 1/m$ for all $j \in J \subseteq \mathcal{I}$ such that the unconstrained solution suggests $\tilde{v}_j^* > 1/m$. Then, we proceed as in the unconstrained case for all $i \in \mathcal{I} \backslash J$. ∎

The condition of Proposition 4.2 is not restrictive. In fact, there are some cases when uniqueness of the stationary point can be shown. The following corollary discusses two special cases.

Corollary 4.1 (Two special cases): Consider the matching function defined by (2). For some given $\mathbf{s} \in \mathcal{S}$, let us also consider either one of the following hypotheses:[5]

(H1) β_i / s_i is sufficiently small for all i;

(H2) $\lambda_i \frac{\beta_i}{s_i} \approx \lambda \frac{\beta}{s}$ for all i and for some constants $\lambda \in (0, 1)$, $\beta > 0$ and $s > 0$.

Then, the ODE (5) has a unique stationary point $\tilde{\mathbf{v}}^*$. Furthermore, there exists some index set $J \subseteq \mathcal{I}$, such that,

– under (H1),

$$\begin{cases} \tilde{v}_j^* = 1/m, & j \in J \\ \tilde{v}_j^* \approx \lambda_j / \sum_i \lambda_i, & j \notin J \end{cases}; \tag{13}$$

– under (H2),

$$\begin{cases} \tilde{v}_j^* = 1/m, & j \in J \\ \tilde{v}_j^* \approx \lambda_j / \left(\sum_i \lambda_i + \lambda \frac{\beta}{s} m \tilde{v}_r^*\right), & j \notin J \end{cases}. \tag{14}$$

Proof: (H1) If β_i / s_i is sufficiently small for all i, then $\rho_s \approx \gamma < 1$. Therefore, from Proposition 4.2, there exists a

[5]Here we abuse notation by interpreting the symbol "≈" as "sufficiently close in the Euclidean norm."

unique stationary point. Furthermore, the stationary point can be computed approximately from condition (C1) and it can be verified in a straightforward manner that satisfies (13).

(H2) Note that for any $j \in \mathcal{I}$, we have:

$$\begin{aligned} -\lambda_j f_j(s_j, v_j) + \sum_{i\in\mathcal{I}} \lambda_i f_i(s_i, v_i)\tilde{v}_j \\ \approx \; -\lambda\frac{\beta}{s} m\tilde{v}_j + \lambda_j + \lambda\frac{\beta}{s}\tilde{v}_j m \sum_{i\in\mathcal{I}} \tilde{v}_i - \tilde{v}_j \sum_{i\in\mathcal{I}} \lambda_i \\ = \; \lambda_j - \tilde{v}_j \sum_{i\in\mathcal{I}} \lambda_i - \lambda\frac{\beta}{s} m\tilde{v}_r\tilde{v}_j. \end{aligned}$$

By setting the above quantity equal to zero, we derive the stationarity condition (14). ∎

The above proposition also provides an answer to how the stationary point changes with respect to the weight parameters $\{\lambda_i\}$. In particular, from (13), we conclude that if either one of the hypotheses (H1) or (H2) is valid, then the percentage of resources $\tilde{v}_j^*$ of application j will increase at the stationary point if λ_j is also increased, unless some bandwidth reached the maximum at $\tilde{v}_j^* = 1/m$.

2) Non-fixed Service Levels: Note that the conclusions of Proposition 4.2 and the special cases of Corollary 4.1 continue to hold when the service levels are also adjusted based on (7) as long as the corresponding hypotheses are satisfied for all $\mathbf{s} \in \mathcal{S}$. The following corollary identifies one such case.

Corollary 4.2: Consider the matching function defined by (2). If hypothesis (H1) holds for all $\mathbf{s} \in \mathcal{S}$, then the overall dynamics (11) exhibits a unique stationary point $(\mathbf{s}^*, \tilde{\mathbf{v}}^*)$ such that $s_i^* = \underline{s}_i$ and $\tilde{\mathbf{v}}^*$ satisfies property (13).

Proof: If hypothesis (H1) holds for all $\mathbf{s} \in \mathcal{S}$ and $\tilde{\mathbf{v}} \in \widetilde{\mathcal{V}}$, then the conclusions of Corollary 4.1 apply. Furthermore, $s_i^* = \underline{s}_i$ for all $i = 1, \ldots, n$, since $f_i(s_i, v_i) < 0$ for all $s_i \in [\underline{s}_i, \infty)$ and all $\tilde{\mathbf{v}} \in \widetilde{\mathcal{V}}$. ∎

As we have already pointed out, in the more general case where hypothesis (H1) does not hold, there is a multiplicity of stationary points including (if exist) any pair $(\mathbf{s}^*, \tilde{\mathbf{v}}^*)$ for which $f_i(s_i^*, v_i^*) = 0$.

B. Local Asymptotic Stability & Convergence

The following proposition characterizes locally the stability properties of the stationary points under the hypotheses of Corollaries 4.1–4.2.

Proposition 4.3 (LAS): Under the hypotheses of either Corollary 4.1 or 4.2, the unique stationary point of the dynamics (11) is a locally asymptotically stable point in the sense of Lyapunov.

Proof: Let us define the non-negative function

$$W(\tilde{\mathbf{v}}) = \frac{1}{2}(\tilde{\mathbf{v}} - \tilde{\mathbf{v}}^*)^{\mathrm{T}}(\tilde{\mathbf{v}} - \tilde{\mathbf{v}}^*) \geq 0$$

and let us consider the unconstrained case at which $\widetilde{\mathcal{V}}_i = [0, 1]$ for all $i \in \mathcal{I}$. (We will revisit this assumption later on.) In this case,

$$\begin{aligned} \dot{W}(\tilde{\mathbf{v}}) &= (\tilde{\mathbf{v}} - \tilde{\mathbf{v}}^*)^{\mathrm{T}} \mathbf{g}_{\mathsf{rm}}(\mathbf{s}, \tilde{\mathbf{v}}) \\ &= (\tilde{\mathbf{v}} - \tilde{\mathbf{v}}^*)^{\mathrm{T}} \left[-\mathbf{\Lambda f}(\mathbf{s}, \mathbf{v}) + \mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, \mathbf{v})\tilde{\mathbf{v}}\right]. \end{aligned}$$

Let us consider the following perturbed allocation $\tilde{\mathbf{v}} = (1-\varepsilon)\tilde{\mathbf{v}}^* + \varepsilon\mathbf{w}$ for some $\mathbf{w} \in \widetilde{\mathcal{V}}$ and $\varepsilon > 0$. Then, we have:

$$\begin{aligned} \dot{W}(\tilde{\mathbf{v}}) &= \varepsilon(\mathbf{w} - \tilde{\mathbf{v}}^*)^{\mathrm{T}} \left[-\mathbf{\Lambda f}(\mathbf{s}, (1-\varepsilon)\tilde{\mathbf{v}}^* + \varepsilon\mathbf{w}) + \right. \\ &\quad \left. \mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, (1-\varepsilon)\tilde{\mathbf{v}}^* + \varepsilon\mathbf{w})((1-\varepsilon)\tilde{\mathbf{v}}^* + \varepsilon\mathbf{w})\right]. \end{aligned}$$

Given that $\mathbf{f}(\mathbf{s}, \tilde{\mathbf{v}})$ is linear with respect to $\tilde{\mathbf{v}}$, we also have

$$\mathbf{f}(\mathbf{s}, (1-\varepsilon)\tilde{\mathbf{v}}^* + \varepsilon\mathbf{w}) = (1-\varepsilon)\mathbf{f}(\mathbf{s}, \tilde{\mathbf{v}}^*) + \varepsilon\mathbf{f}(\mathbf{s}, \mathbf{w}).$$

Thus,

$$\begin{aligned} \dot{W}(v) &\approx \varepsilon^2\|\mathbf{w} - \tilde{\mathbf{v}}^*\|_2^2 \mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, \tilde{\mathbf{v}}^*) + \\ &\quad \varepsilon^2(\mathbf{w} - \tilde{\mathbf{v}}^*)^{\mathrm{T}}\left[-\mathbf{\Lambda f}(\mathbf{s}, \mathbf{w}) + \mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, \mathbf{w})\tilde{\mathbf{v}}^*\right] \end{aligned}$$

plus higher order terms of ε.

(H1) If β_i/s_i is sufficiently small for all i, then the first term of the RHS dominates the second term. This is due to the fact that as β_i/s_i approaches zero for all i, the second term approaches zero while the first term is bounded away from zero and it is strictly negative. Therefore, from [15, Theorem 3.1], the stationary point $\tilde{\mathbf{v}}^*$ is locally asymptotically stable.

(H2) In this case, note that for any j, we have:

$$-\lambda_j f_j(s_j, w_j) + \sum_{i\in\mathcal{I}} \lambda_i f_i(s_i, w_i)\tilde{v}_j^* \approx -\lambda\frac{\beta}{s} m(w_j - \tilde{v}_j^*)$$

where the last approximation is due to the fact that $\tilde{v}_j^*$ is a stationary point and satisfies $\lambda_j - \tilde{v}_j^* \sum_{i\in\mathcal{I}} \lambda_i = 0$. Thus,

$$-\mathbf{\Lambda f}(\mathbf{s}, \mathbf{w}) + \mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, \mathbf{w})\tilde{\mathbf{v}}^* \approx -\lambda\frac{\beta}{s} m(\mathbf{w} - \tilde{\mathbf{v}}^*)$$

and $\mathbf{1}^{\mathrm{T}}\mathbf{\Lambda f}(\mathbf{s}, \tilde{\mathbf{v}}^*) \approx \lambda\frac{\beta}{s} m - \sum_{i\in\mathcal{I}} \lambda_i$. We conclude that:

$$\begin{aligned} \dot{W}(\tilde{\mathbf{v}}) &\approx \varepsilon^2\|\mathbf{w} - \tilde{\mathbf{v}}^*\|_2^2 \left(\lambda\frac{\beta}{s} m - \sum_{i\in\mathcal{I}} \lambda_i\right) - \\ &\quad \varepsilon^2\lambda\frac{\beta}{s} m\|\mathbf{w} - \tilde{\mathbf{v}}^*\|_2^2 \\ &= -\varepsilon^2\|\mathbf{w} - \tilde{\mathbf{v}}^*\|_2^2 \sum_{i\in\mathcal{I}} \lambda_i \end{aligned}$$

plus higher order terms of ε. Thus, the stationary point $\tilde{\mathbf{v}}^*$ is a locally asymptotically stable stationary point of $\mathbf{g}_{\mathsf{rm}}$.

Finally, in case $\widetilde{\mathcal{V}}_i = [0, 1/m]$, the unique stationary point may assign $\tilde{v}_i^* = 1/m$ for some applications i. In this case, it is straightforward to check that the vector field $\mathbf{g}_{\mathsf{rm}}$ points outwards, which implies that the conclusions of the unconstrained case continue to hold. ∎

From Proposition 4.1, we conclude that the stationary points of the ODE (11), which satisfy the hypotheses of Proposition 4.3, are local attractors of the overall recursion.

V. Experimental evaluation

To investigate the assignment of the bandwidth and the values of the application service levels, the resource management scheme was implemented both in Matlab and in TrueTime [19].

A. Experiment with synthetic applications

In the first Matlab experiment, the applications are not executed, they are simply abstracted by their characteristic parameters. We have three applications running over two cores. Applications are all the same, except for the values of

the weights: $\lambda_1 = 0.9$, $\lambda_2 = 0.5$ and $\lambda_3 = 0.1$. As explained in Section II-C, λ_i determines the amount of effort that is taken by the RM to achieve a perfect matching for app_i (i.e., f_i close to zero). For example, $\lambda_1 = 0.9$ implies that the adjustment is mainly done by the RM through adjusting the bandwidth v_1, while $\lambda_3 = 0.1$ implies that the adjustment is mainly done by app_3 through adjusting its service level s_3. In accordance with the time-sensitive application model of Section II-B, each application has $D_i = 2500$ and $\alpha_i = 2000$. In Figure 2, we show the bandwidth v_i, the matching

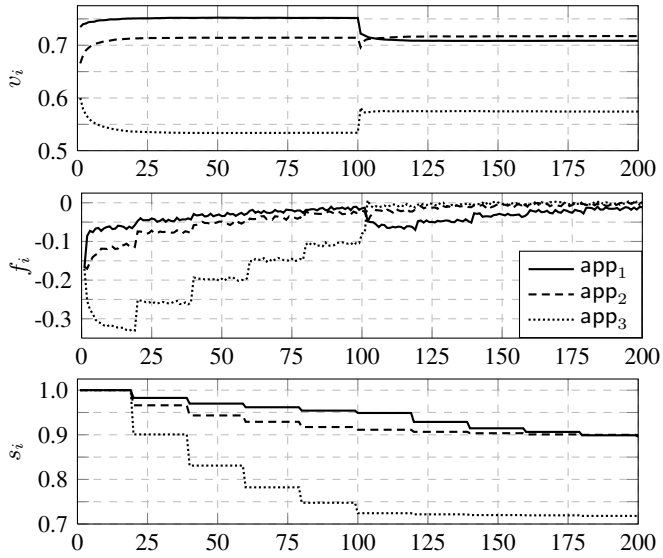

Fig. 2. Simulation results of three applications over two cores.

function f_i, and the service levels s_i of the three applications under the adjustment dynamics of (5), (7) and (10). Some noise is added to the matching functions f_i, to account for the inaccuracy of the real measures and to show the robustness of the method. Also, the service level adaptation is performed once every twenty steps of the RM execution, to resemble some real behavior, where applications are adjusting at a slower rate with respect to the resource allocation. At time 100, the weights of the applications are changed to $\lambda_1 = 0.1$, $\lambda_2 = 0.5$ and $\lambda_3 = 0.9$ and the RM is reinitialized.

At time 0, in response to an equally scarce matching between the bandwidth and the service levels, app_1 is assigned more resource, while app_3 has to significantly lower its service level s_3. This observation complies with the prediction of Corollary 4.2 and Proposition 4.3 under assumption (H1), however we should not expect to observe the exact allocation (13), since condition (H1) partially holds at the beginning of the simulation (when f_i are quite negative). As the weights are changed by the RM, app_3 receives more resource, but not as predicted by (13) since the matching function is already quite close to zero.

B. Experiments with real applications

TrueTime [19] is a Matlab/Simulink-based tool that allows simulation of tasks executing within real-time kernels and communication over networks, embedded within Simulink. Among other things, it supports simulation of CBS-based [2]

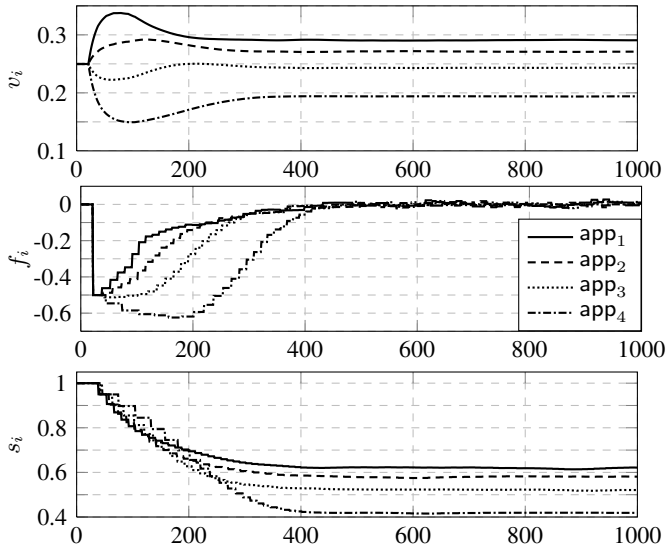

Fig. 3. TrueTime simulation results of a single core machine with four applications.

task execution. The policy allows to adjust the CPU time allocated to the running applications, in the same exact way as in a real-time computing system. Moreover, TrueTime offers the ability to simulate memory management and protection, therefore being a perfect match to simulate our resource management framework. A TrueTime kernel simulates a single CPU that hosts the execution of RM and of the CBS servers that contain the applications. A shared memory segment is initialized and both the RM and the applications have access to the memory area reporting their execution data. The RM reads the matching function, f_i, of each application i and computes the new reservations v_i. Then, it updates the parameters of the CBS server and writes in the shared memory values to be read by the applications. The execution time of the RM is a parameter of the simulation. Both the applications and the RM are coded in a way that is very close to a real implementation and the resulting simulation data are generally very close to the data that would have been obtained on a real execution platform.

The first experiment with TrueTime considers four applications and the RM, which employ the adjustment process of (5), (7) and (10). Figure 3 shows the allocated bandwidths v_i, the matching functions f_i and the service levels s_i. For these applications, we take $D_i/\alpha_i = 2$ as explained in the derivation of (2) for multimedia applications. The weights are $\lambda_1 = 0.8$, $\lambda_2 = 0.6$, $\lambda_3 = 0.4$ and $\lambda_4 = 0.2$. Some randomness is also introduced in the execution times to show the effect of disturbances generated by lock acquisition, resource contention, memory management, etc. At the beginning of the simulation and when the matchings are quite scarce (i.e., (H1) is satisfied), the RM distributes the CPU as predicted by Corollary 4.2 and Proposition 4.3. However, as the service levels also adjust and the matching functions approach zero, we observe a deviation from the exact resource allocation predicted by (13), which is anticipated since condition (H1)

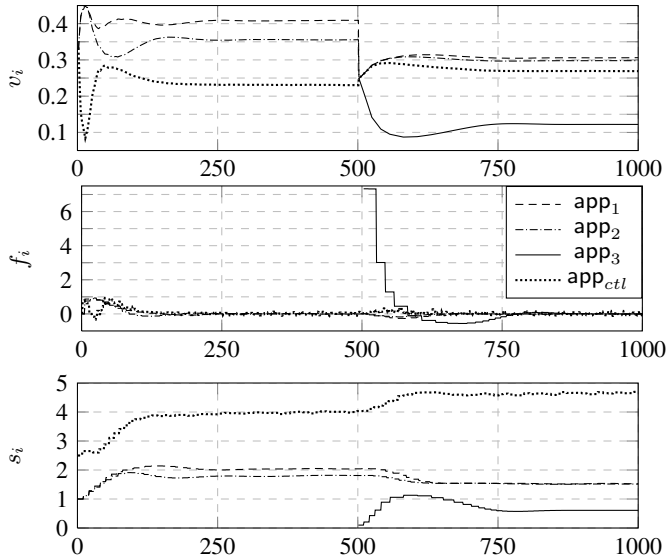

Fig. 4. TrueTime simulation results of a single core hardware with two applications and a control task. Another application is arriving at time 500.

no longer applies.

In the second experiment, we show how four heterogeneous applications are handled by the RM. Three of them are alive from the very beginning while the fourth one enters the system at time 500. Among the three applications that are alive from the beginning, one is a LQG controller controlling an inverted pendulum which is simulated in Simulink. The controller, modeled as a time-sensitive application (2), has a deadline set to $0.85T_s$ where T_s is its sampling period. Its service level s_i is simply set equal to $1/T_s$, because of the natural observation that faster sampling can provide better performance. The applications' weights are set as $\lambda_1 = 0.64$, $\lambda_2 = 0.73$, $\lambda_3 = 0.41$ and $\lambda_{ctl} = 0.41$. Every five controller jobs, the service level of the controller, i.e., the sampling frequency, is adjusted. A new sampling period is chosen and the controller is redesigned taking into account the measured sampling period and the measured input/output latency. The latency, i.e., the average amount of time between the sensor measurement and the actuation, depends on the amount of bandwidth assigned to the controller by the resource manager. In a real system the on-line redesign would be replaced by look-up in a table consisting of pre-calculated controller parameters for different sampling periods and latencies.

Figure 4 shows the quantities involved in the simulation, the CPU bandwidth allocated to the applications and to the controller, the performance function and the service level. Notice that when a new application is introduced the resource manager is reinitialized, and the CPU is redistributed.

VI. Conclusion and future work

We proposed a distributed management framework for the CPU resource. Being distributed, our scheme has only a linear time complexity in the number of demanding applications. We exploited a game-oriented logic, where all applications competes for the assignment of the CPU time. The validity of the framework (such as the existence of stationary points and stability of the resource assignment) is theoretically proved and experimentally validated.

Currently, we are working on an implementation of the framework within the Linux kernel. Also we will extend the resource management scheme to be robust against potential malicious behaviors of the applications, which may lead to an incorrect resource assignment.

References

[1] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Boston, MA, U.S.A., May 1994, pp. 90–99.

[2] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec. 1998, pp. 4–13.

[3] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, Feb. 1999.

[4] J. Eker, P. Hagander, and K.-E. Årzén, "A feedback scheduler for real-time controller tasks," *Control Engineering Practice*, vol. 8, no. 12, pp. 1369–1378, Jan. 2000.

[5] T. Cucinotta, L. Palopoli, L. Abeni, D. Faggioli, and G. Lipari, "On the integration of application level and resource level qos control for real-time applications," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 479–491, Nov. 2010.

[6] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for QoS management," in *Proceedings of the IEEE Real Time System Symposium*, 1997.

[7] C. Lee, J. P. Lehoczky, D. Sieworek, R. Rajkumar, and J. Hansen, "A scalable solution to the multi-resource QoS problem," in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, Phoenix, AZ, Dec. 1999, pp. 315–326.

[8] B. Johansson, C. Adam, M. Johansson, and R. Stadler, "Distributed resource allocation strategies for achieving quality of service in server clusters," in *Proceedings of the 45^{th} IEEE Conference on Decision and Control*, Dec. 2006, pp. 1990–1995.

[9] E. Bini, G. C. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Årzén, R. Vanessa, and C. Scordino, "Resource management on multicore systems: The ACTORS approach," *IEEE Micro*, vol. 31, no. 3, pp. 72–81, 2011.

[10] K.-E. Årzén, V. Romero Segovia, S. Schorr, and G. Fohler, "Adaptive resource management made real," in *Proc. 3rd Workshop on Adaptive and Reconfigurable Embedded Systems*, Chicago, IL, USA, Apr. 2011.

[11] R. Subrata, A. Y. Zomaya, and B. Landfeldt, "A cooperative game framework for QoS guided job allocation schemes in grids," *IEEE Transactions on Computers*, vol. 57, no. 10, pp. 1413–1422, Oct. 2008.

[12] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *The Journal of Supercomputing*, vol. 54, no. 2, pp. 252–269, Nov. 2010.

[13] D. Grosu and A. T. Chronopoulos, "Noncooperative load balancing in distributed systems," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1022–1034, 2005.

[14] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, MA: MIT Press, 1994.

[15] H. Khalil, *Nonlinear Systems*. Prentice-Hall, 1992.

[16] H. J. Kushner and G. G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*, 2nd ed. Springer-Verlag New York, Inc., 2003.

[17] K. Border, *Fixed Point Theorems with Applications to Economics and Game Theory*. Cambridge University Press, 1985.

[18] E. Kreyszig, *Introductory Functional Analysis with Applications*. John Wiley & Sons, 1978.

[19] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, "How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime," *IEEE Control Systems Magazine*, vol. 23, no. 3, p. 1630, Jun. 2003.