Processor Thermal Control Using Adaptive Bandwidth Resource Management

Romero Segovia, Vanessa; Kralmark, Mikael; Lindberg, Mikael; Årzén, Karl-Erik

2011

Processor Thermal Control Using Adaptive Bandwidth Resource Management

Vanessa R. Segovia * **Mikael Kralmark** * **Mikael Lindberg** *
Karl-Erik Årzén *

* *Department of Automatic Control, Lund University, Sweden,*
vanessa,mikael.kralmark,lindberg,karlerik@control.lth.se

Abstract: An adaptive resource management system combined with a thermal controller is presented. The aim of the system is to dynamically allocate computing resources to applications competing for the same computing resources in such a way that the system is not overheated. The approach has been implemented on a mobile robot. Experimental results are presented showing the feasibility of the approach.

Keywords: resource management, thermal control, embedded systems, mobile systems.

1. INTRODUCTION

Dealing with resource constraints is a key challenge in designing mobile embedded systems. Power, communication bandwidth and computational resources are all instrumental to performance in fields such as mobile robotic or media capable mobile phones. These resources are all typically limited, i.e., the processor has a limited capacity or the entire system is battery-driven.

Traditionally, embedded computing systems have been designed using static worst case assumptions on availability of resources, but for systems with large variability in use cases or which are executing on uncertain executing platforms this is increasingly difficult. An alternative approach is to instead allocate resources to different applications or systems dynamically, based on measurements of resource consumption, resource availability, and the generated quality of service. This is also referred to as *adaptive resource management* and is particularly suited for applications of a soft real-time nature, e.g., media processing applications. However, many control and service robotic applications also fall within this domain.

The task of the adaptive resource manager is to decide how resources should be allocated to different applications on the system. If the total computational load is too high the system will consume more power, which will also cause the temperature of the computer chip to increase. In order to prevent failures due to overheating, active cooling can be used. However, this will also contribute to the power consumption. An alternative is then to instead limit the computer load, or utilization

In this paper an approach that combines a PI controller for thermal control of the chip and an adaptive resource management system is presented. The output from the PI controller is used as the maximum allowed utilization of the system, i.e., the maximal amount of CPU bandwidth that the resource manager is allowed to distribute among the applications executing on the system. In the paper an adaptive resource manager from the EC ACTORS project is used. The platform used is a Pioneer mobile robot with an Intel PC computer, but the approach can be considered for any system where energy or space constraints make the use of active cooling infeasible. Such systems include mobile phones and embedded controllers.

2. RELATED RESEARCH

Limiting temperature in the CPU by controlling utilization has been proposed by Fu et al. (2010), but they do not discuss the actual software performance. Software performance metrics for a thermal control case was introduced e.g. by Lindberg and Årzén (2010) but were only evaluated through simulations.

The resource manager used in this work is based on the resource manager developed within the ACTORS project (Segovia et al. (2010)). The combined feedforward/feedback structure employed in the ACTORS resource manager is based on the AQuoSA architecture, Abeni et al. (2005), employed in the EC FRESCOR project. The way to model the resource requirements and the generated quality of service of the applications is based on the MATRIX project (Rizvanovic and Fohler (2007)). The thermal dynamics model used is based on Ferreira et al. (2007).

3. SYSTEM MODEL

In order to prevent CPU processor overheating, which could cause performance degradation of all the applications executing on the system and even system failure, we propose a system model that combines feedback and feedforward techniques to control both the temperature and the utilization of the processor (CPU) through adaptive bandwidth allocation. Figure 1 shows the proposed system model, which combines the features of a thermal controller, that keeps the temperature of the system bounded to a desirable temperature acceptable to the processor, and a resource manager, that dynamically allocates resources to each application on the system. Here, T and T_R are the current temperature of the system, and the reference

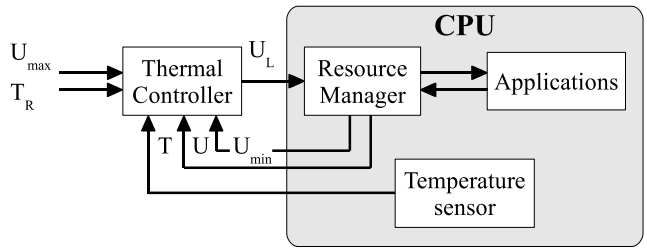

Fig. 1. System model

temperature respectively, it is assumed that this last value is specified by the system designer. The values U, U_{min}, U_{max} and U_L correspond to the utilization of the system, the lower and upper utilization bounds and the utilization limit defined by the thermal controller respectively.

3.1 Software components

Due to system limitations, as well as performance specifications, it is convenient to keep the temperature of the system bounded to a desired value. From a software point of view, this can be achieved using bandwidth reservation techniques (Abeni and Buttazzo (1998)), which in combination with control theory allow at runtime adaptive allocation of CPU resources provided to the applications. Reservation techniques such as the constant-bandwidth server (CBS), guarantee to each application a certain execution budget every server period, this is also known as virtual processors (VP). In order to achieve this adaptive allocation of resources, we use a modified version of the architecture proposed originally by ACTORS (see e.g. Segovia and Årzén (2010)), which assumes a multicore physical platform. For our particular case we employ a single core physical platform. Figure 2 shows the modified

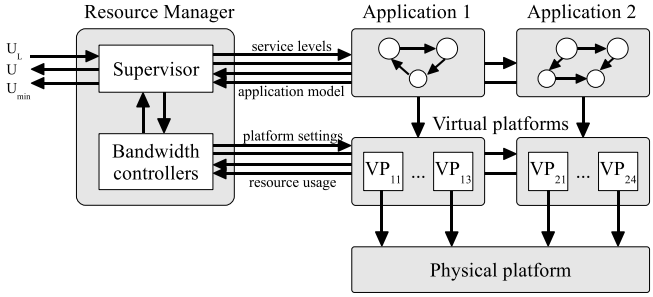

Fig. 2. Modified ACTORS architecture

architecture, which is composed mainly of three components: the applications, the resource manager (RM) and the reservation layer which includes the VPs of each application.

The RM is a daemon application, composed of a centralized supervisor and several bandwidth controllers, these elements will be explained in Section 4. The main tasks of the RM are to accept applications that want to execute on the system, to provide CPU resources to these applications, to monitor their behaviour over time, and to dynamically change the resources provided according to the real needs of the application, and the performance criteria of the system, e.g. to limit the maximum temperature of the system.

The application can be composed of one or several tasks, which may have dependencies between each other. It is assumed that an application has different service levels. The quality-of-service (QoS) provided by the application is associated to the service level at which it executes, the higher the service level the higher the QoS, and the more resources it consumes over time, which implies a higher utilization of the system. It is also assumed that the application can communicate its service level information to the RM. Table 1 shows an example of this information for an application named A1 that has three service levels and three VPs, and for application A2 which has two service levels. In the table SL, QoS,

Application name	SL	QoS [%]	α [%]	Δ [μs]	BWD [%]
A1	0	100	100	28-24-24	40-30-30
	1	80	70	42-48-48	30-20-20
	2	60	40	80-90-90	20-10-10
A2	0	100	60	20	
	1	80	40	50	

Table 1. Service level table for applications A1 and A2, with SL, QoS, α, Δ and BWD as initial model parameters of the applications.

α, Δ, and BWD are a service level index, the quality of service, the total bandwidth, the tolerable application delay, and the bandwidth distribution respectively. The delay Δ represents a measure of the time granularity of the specific service level, typically high QoS levels have a low value of Δ. The bandwidth distribution is an optional value, it is an indication from the application to the RM how this total bandwidth should be distributed over the individual virtual processors. Additionally the RM is informed about the total number of VPs that each application contains and the importance of the application relative to others.

The information provided by the application (see Table 1) represents an initial estimate of the resources required at an specific service level. This information constitutes an initial model of the application, which during run-time and through the control mechanism implemented by the RM will be tuned appropriately.

The reservation mechanism used by the RM is provided by `SCHED_EDF`, Manica et al. (2010), which is a new Linux scheduling class developed in ACTORS that provides support for partitioned hard CBS servers.

3.2 CPU thermal model

In this paper, we propose a model for the thermal dynamics based on Ferreira et al. (2007). According to this model the dynamics from the CPU consumed power P to the CPU temperature T is on the form

$$\dot{T} = a(T_a - T) + bP + d \tag{1}$$

where a and b are constants that depend on the thermal resistance and heat capacity of the processor and T_a is the ambient temperature, d is a disturbance term which will be assumed to have slow dynamics, such as heat generated by direct sunlight or by being placed on a heated surface. For off-the-shelf CPUs, a and b are in the order of 10^{-4} and 10^{-3} respectively (see e.g. Fu et al. (2010)), making the

dynamics relatively slow. It is therefore assumed that it is possible to filter out measurement noise, which is therefore omitted from the model.

Considering that the relationship between CPU utilization U and the power consumption P is defined by the linear formulation (see Heath et al. (2006))

$$P = P_{idle} + U(P_{max} - P_{idle}) \tag{2}$$

where P_{idle} and P_{max} are the power consumption when the processor is idle and fully utilized respectively.

Equations 1 and 2 show that by limiting the load of the system, the temperature can be controlled even if the CPU is only passively cooled. Sampling the combination of these two equations can be done under ZOH assumptions.

4. CONTROL DESIGN

4.1 Control objectives

The proposed thermal control algorithm together with the resource manager are designed to meet two fundamental requirements: to prevent processor overheating by minimizing the maximum temperature of the system, and to provide desired system performance by maximizing the QoS provided by the running applications subject to the resource limitations.

4.2 Thermal control design

To fulfill the first objective of our control design, we propose the use of a PI algorithm for the thermal controller. As shown in Figure 3, the signal T from the

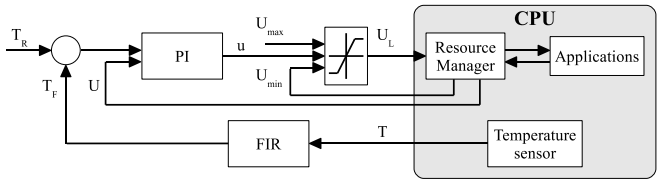

Fig. 3. Thermal controller structure

temperature sensor is passed through a lowpass filter with FIR-structure, in order to reduce measurement noise. The filtered temperature T_F is then compared with the reference temperature T_R, producing the error input of the PI controller. Additionally to this input the controller uses the utilization signal U provided by the resource manager, which represents the current CPU load caused by the applications running at an specific service level on the system. This signal is used by the anti-windup component of the PI controller to prevent wind up of the integral part.

The additional inputs U_{min} and U_{max}, represent the minimum utilization required by the applications to provide the lowest permissible QoS, and the maximal available utilization defined by the employed scheduling policy respectively. U_{min} must be chosen so that thermal constraints are not violated when running the enabled software at the lowest possible QoS. These two inputs provide the lower and upper limits that bound the PI controller output $u(k)$ such that, $u(k) \in [U_{min}(k), U_{max}]$. According to this the control output of the thermal controller, or utilization limit U_L, can be defined as $U_L(k) = sat(u(k), U_{min}(k), U_{max})$, with

$$sat(u(k), U_{min}(k), U_{max}) = \begin{cases} U_{min}(k), & u(k) < U_{min}(k) \\ U_{max}, & u(k) > U_{max} \\ u(k), & otherwise \end{cases}$$

The output of the thermal controller $U_L(k)$, decides the maximum amount of CPU bandwidth available to the RM. The resource manager dynamically allocates CPU resources to the applications based on the utilization limit U_L, the measurements provided by the scheduler for each of the running applications, and the performance criteria of the system, e.g. maximization of the QoS.

4.3 CPU resource allocation

The different elements that constitute the resource manager as shown in Figure 2, implement a control mechanism that combines feedforward and feedback strategies, which allow adaptive allocation of CPU resources at runtime.

The feedforward algorithm is carried out by the supervisor, which responsibilities include acceptance or registration of applications, monitoring of the minimum utilization U_{min} required by the applications to provide and specific QoS, and monitoring and control of the system utilization U, which is subject to the constraints defined by the thermal controller.

During registration, each application communicates its service level information (see Table 1) to the RM, in particular to the supervisor. Based on this information the supervisor assigns the service level at which each application must execute. This assignment can be formulated as an integer linear programming (ILP) optimization problem, which objective is to maximize the global QoS of the current applications running on the system including the new application, under the constraint that the total amount of resources is limited. The boolean variable y_{ij} is 1 if application i is assigned QoS level j, it is 0 otherwise. For each application i, q_{ij} denotes the quality at level j, and α_{ij} the bandwidth requirement. The problem can now be stated as follows

$$\begin{aligned} &\max \sum_i w_i \sum_j q_{ij} y_{ij} \\ &\sum_i \sum_j \alpha_{ij} y_{ij} \leq C \\ &\sum_j y_{ij} \leq 1 \qquad \forall i \end{aligned} \tag{3}$$

where C is the total assignable bandwidth of the system, which corresponds to the utilization limit U_L value defined by the thermal controller, and w_i is the weight (importance) of application i relative to other applications. The importance values are assumed to be decided by the system designer. The last constraint implies that, if necessary, some low important applications might be turned off, in order to allow the registration of more important applications.

After the service level assignment of each application, the supervisor calculates the reservation parameters of each VP. Hence, it creates the VPs for the tasks of each application by defining the budget Q, and the period P

of each VP. The calculation of the budget and the period of the server is based on the corresponding (α, Δ) (see Mok et al. (2001)) parameters described by the following equation

$$Q = \alpha P \qquad P = \frac{\Delta}{2} + Q \tag{4}$$

The service level assignment of the applications running on the system, is carried out not only during registration, but also when the thermal controller redefines the utilization limit U_L. This could be the result of abrupt temperature increments in the system caused by internal factors, such as a high computational load of the running applications, or by external ones such as overheated adjacent equipment. Any of these situations would trigger a new service level assignment for all applications.

The service level assigned to each application running on the system, sets an initial upper limit for the assigned budget also known as AB_L, this value can be directly calculated from the information provided by the application (see Table 1).

The feedback mechanism is implemented by the bandwidth controllers of the VPs of each application. They check, whether or not the tasks within the VPs make optimal use of the bandwidth provided, or the assigned budget (AB), and take actions to ensure this without degrading the performance of the application. The bandwidth controllers are executed periodically with a period that is a multiple of the period of the VP that they are controlling.

The bandwidth controllers measure the actual resource consumption using two measurements provided by the scheduler. The used budget (UB) is the average used budget over the sampling period of the controller at the current assigned service level. Considering that the linux scheduler `SCHED_EDF` supports hard reservations, the UB is always less than or equal to the budget that has been assigned to the VP by the RM. The hard reservation (HR) is a value that tells the percentage of server periods over the last sampling period that the task in the VP consumed its full budget. This is an indicator of the number of deadlines missed.

The bandwidth controllers have a cascade structure shown in Figure 4. The hard reservation set point (HR_{SP}) corresponds to the maximum percentage of deadlines misses that can be allowed in each sampling period. Based on the difference between the HR_{SP} and the HR values, the outer controller C_1 defines the new values of the set point for the used budget (UB_{SP}), which in this case corresponds to upper and lower bounds within which the UB measurement should reside. The inner controller C_2, which corresponds to an exponential controller, requires that the UB lays within the bounds, in case any of these bounds are violated, C_2 recalculates and adjusts the assigned budget AB of the VP, subject to the limitation defined by the supervisor. The value of HR_{SP} can be related to the performance of the application.

5. IMPLEMENTATION

The thermal control algorithm has been implemented together with the modified ACTORS framework. The implementation was done on a Pioneer mobile robot [MobileRobots Inc (2006)] with an internal Intel Pentium III based computer [Versalogic Corporation (2007)]. The thermal sensor used is a National Semiconductor LM83 chip with an accuracy of ±3°C [National Semiconductor Corporation (1999)], and sample period of 2 seconds. The D/A-conversion takes approximately 500 ms and the temperature measurement is updated by the sensor driver every two seconds [Delvare (2010)]. In order to avoid aliasing effects, the sampling period of the bandwidth controllers is set to a multiple of the application granularity that is higher than the A/D conversion time.

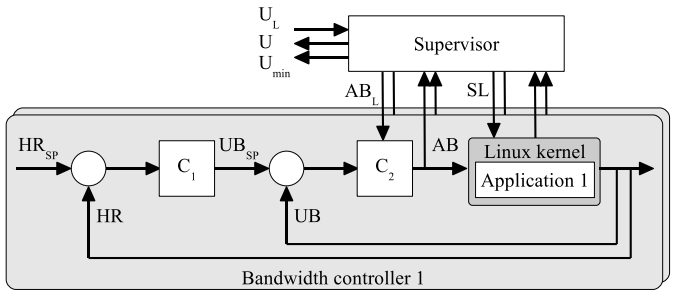

Fig. 4. Resource manager controller structure

A PI controller, designed as discussed in Section 4.2 is used to calculate the utilization limit parameter, that keeps the temperature of the system around a reference value provided by the user. To limit the measurement noise, the temperature signal is passed through a FIR filter with a rectangular window of one minute. The thermal controller is set to run as often as new data is available from the sensor, i.e. every two seconds, this is done to get as much data as possible to improve the filtering. The utilization limit U_L calculated by the controller sets the upper bound C of one of the constraints of the service level assignment problem defined by equation 3. In order to solve the ILP optimization problem, the RM uses the GLPK linear programming toolkit(Makhorin (2000)).

The RM is implemented in C++. It consists of two threads that execute within the same fixed-size reservation in one of the cores. The RM communicates with the applications through a DBus interface and with the underlying `SCHED_EDF` scheduler using the control groups API of Linux. The first thread handles incoming DBus messages containing, e.g., the service level table information which is sent when an application registers, and notifications that an application has terminated. The second thread periodically samples the VPs, measures the resource consumption, and invokes the bandwidth controllers.

The reservation mechanism is provided by the Linux scheduling class `SCHED_EDF`. The measured system utilization value only considers the applications that register with the resource manager, and not the RM itself which has a fixed amount of resources allocated by the system. For this implementation the maximum utilization U_{max} was set to 80%. Every time that the utilization limit changes in any direction the system utilization will be temporarily higher, this occurs while the RM is solving the ILP optimization problem. To reduce this effect and to limit the influence of the noise that might still be

present after filtering, the new calculated utilization limit is passed to the RM only if it has changed by more than five percentage points with respect to its previous value.

6. EXPERIMENTAL RESULTS

6.1 Thermal model validation and PI controller design

In order to validate the model structure presented by the equations 1 and 2, a step response experiment was carried out on the experimental platform. According to the results

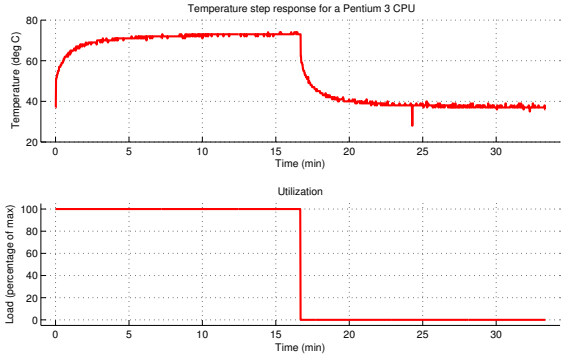

Fig. 5. Step response experiment performed on a Pioneer mobile robot.

shown in Figure 5, the dynamics between utilization U and chip temperature T can be roughly modeled by the first order system with time delay

$$\frac{T(s)}{U(s)} = \frac{K_P}{Ts+1}e^{-\tau s} \tag{5}$$

where the gain of the system corresponds to $K_P = 0.37$, the time constant to $T = 32.54\ s$ and the dead time to $\tau = 31.1\ s$. A better fit could likely be achieved with a more advanced model, as the step responses in Figure 5 do not correspond exactly to those of a first order system, but it would not change the approach significantly.

According to this model, and the internal model control (IMC) approach, Daniel E. Rivera and Skogestad (1986), the tuning constants of the PI controller correspond to $K = 4.1792$ and $T_i = 48.09$.

6.2 Experimental setup

In order to see the performance of the proposed algorithm under normal and overloaded conditions, two different experiments were carried out. In the first experiment the reference temperature was set to 55°C for a period of 10 minutes, and then changed to 45°C for another 10 minutes. For the second experiment the reference temperature was kept constant at 50°C.

Since the objective of these experiments is to show the performance of the thermal controller working together with the resource manager, we define the service level tables of applications A1 and A2 such that, the ILP optimization problem defined by Equation 3 always finds a feasible solution. The infeasible solution case which is handled by a bandwidth compression algorithm, and the tuning of the values on the service level tables are outside the scope of this paper.

For the first experiment, we used a pipeline application A1 consisting of two tasks T_1^1 and T_2^1 with random execution times. As described in Section 4.3, during registration the application provides its service level information (see Table 2) to the RM. Since there are enough resources in the system, the RM assigns service level 0 to A1.

Application name	SL	QoS [%]	α [%]	Δ [ms]	BWD [%]
A1	0	100	60	24-32	40-20
	1	90	30	32-36	20-10
	2	75	20	36-36	10-10

Table 2. Service level table for application A1

The first plot in Figure 6 shows the behavior of the filtered system temperature (green) with respect to the reference temperature (red). The second plot displays the measured utilization (green) and the utilization limit (red), which is the output of the thermal controller. The third plot shows the service levels of the application A1. The changes are done to compensate for the reference temperature change. The last plot of Figure 6 depicts the process variables of the bandwidth controller (see Figure 4), i.e., the used budget (green) and the hard reservation (blue) values, and the assigned budget (red) which is the output of the bandwidth controller.

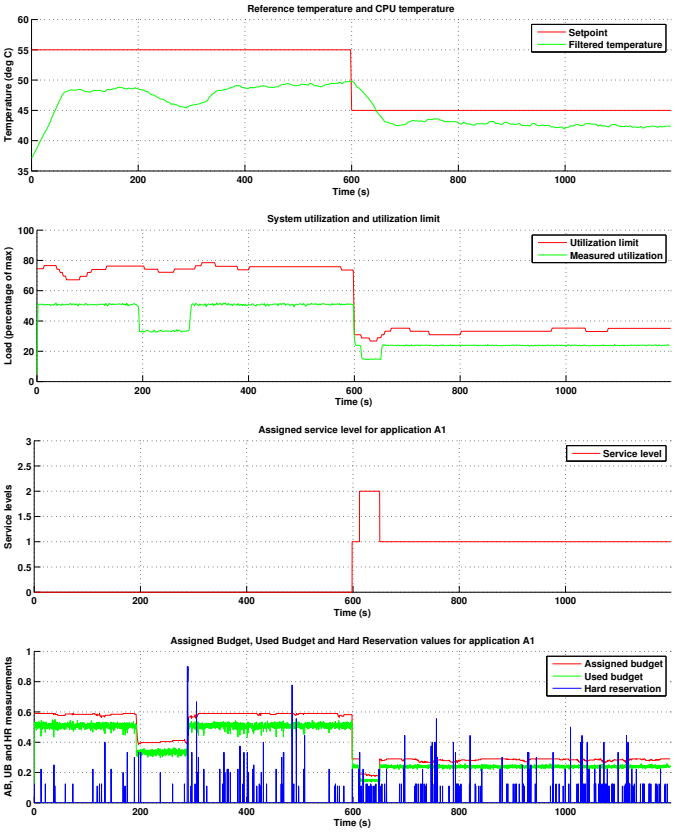

Fig. 6. Performance results under normal conditions. One application running on the system subject to changes in the execution time, and in the system reference temperature.

As can be seen in Figure 6, after registration the bandwidth controllers read the HR and UB values of each of the application VPs and adjust them according to their

set points, the HR_{SP} is defined as 0.1, that is, up to 10% deadline misses are allowed within each sampling period. During the entire execution of the application, the bandwidth controllers keep adapting the AB of the application. At time $t = 200s$ the execution time of the application decreases around 10%, the bandwidth controllers react reducing the assigned budget AB. A new execution variation can be seen at time $t = 300s$, this causes the HR value to equal 0.9, which is quickly compensated by the cascade controller.

Since during the first 10 minutes the system temperature keeps below 55°C, the U_L value set by the thermal controller does not force a service level change in the application A1. At time $t = 600s$ the reference temperature changes to 45°C, here the thermal controller sets the U_L according to the algorithm described in Section 4.3. The changes in U_L trigger the feedforward mechanism of the RM, which assigns a new service level to A1. In order to compensate for the large temperature change, 3 service level changes are carried out, from 0 to 1, from 1 to 2 and finally from 2 to 1, where it remains. Notice that after time $t = 600s$ the filtered temperature T_F does not drop as rapidly as one could expect, this is the effect of solving the optimization problem that leads to a new service level assignment, and which increases momentarily the load on the system.

For the second experiment, we used a new pipeline application A1 and a simple application A2 consisting of one task T_1^2, where application A1 has a higher importance than application A2. Table 3 shows the service level information provided by applications A1 and A2. At the beginning the

Application name	SL	QoS [%]	α [%]	Δ [ms]	BWD [%]
A1	0	100	40	28-36	30-10
	1	90	30	32-36	20-10
	2	75	20	36-36	10-10
A2	0	100	20	32	20
	1	85	10	72	10
	2	35	5	152	5

Table 3. Service level table for applications A1 and A2

only running application is A1, to which the RM assigns the service level 0. At time $t = 300s$ application A2 registers with the RM, which assigns service level 0 for A2 and keeps A1 at service level 0.

Figure 7 shows the behavior of both of the applications when it is required to keep the system temperature bounded to 50°C. This figure contains the same variables as described for the first experiment, together with the additional measurements corresponding to the second application A2. This can be seen specifically in the third plot, which shows the service levels for A1 (red) and A2 (green), and in the fifth plot which represents the measurement variables and the controller output of the bandwidth controller of the application A2.

When the application A2 registers with the RM, the utilization of the system increases causing an increment on the system temperature. Around time $t = 720s$ the thermal controller sets the utilization limit U_L to a value that requires a new service level change from the RM. This is carried out for both of the applications, but since application A1 has a higher importance than A2, the RM reduces the service level of A2 from 0 to 2. Once the system temperature gets below 50°C, the thermal controller increases the value of U_L, this causes a new service level assignment for application A2, from 2 to 1. The bandwidth controllers for both of the applications are also shown.

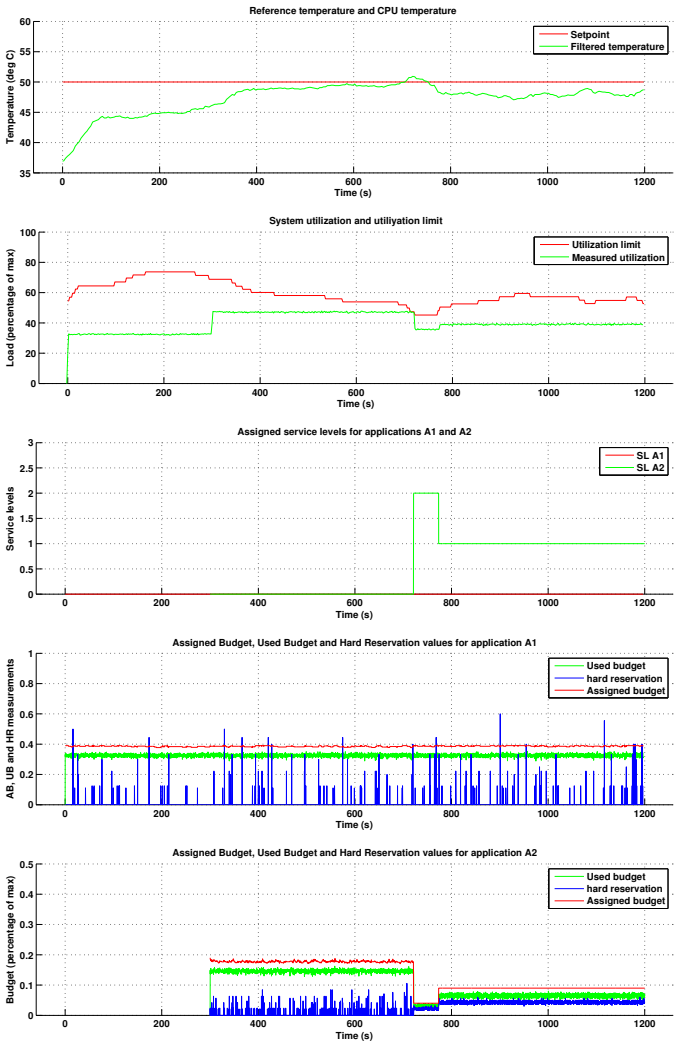

Fig. 7. Performance results under overloaded conditions. Two applications running on the system subject to system temperature constraints.

7. CONCLUSIONS AND FUTURE WORK

An algorithm that combines the features of a processor thermal controller with the qualities of adaptive resource management is described. We present an application model where the application defines the QoS that can be expected at an specific service level. The combination of feedforward and feedback techniques allows adaption at two different levels, at the central level, where all the applications on the system adapt to the temperature constraints of the system, and at a distributed level, where the system adapts to the particular resource requirements of each application.

In future work we plan to extend this approach for multi-core systems, where due to the complexity of the system, the solution could lead to partial or even total migration of applications to other processors on the system.

8. ACKNOWLEDGEMENTS

This work has been supported by the EC ICT FP7 project ACTORS (ICT-216586).

REFERENCES

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

Abeni, L., Cucinotta, T., Lipari, G., Marzario, L., and Palopoli, L. (2005). Qos management through adaptive reservations. *Real-Time Systems*, 29(2-3), 131–155.

Daniel E. Rivera, M.M. and Skogestad, S. (1986). Internal model control 4: Pid controller design. In *Industrial and Engineering Chemestry Research*, 252–265.

Delvare, J. (2010). Kernel driver lm83. `http://www.mjmwired.net/kernel/Documentation/hwmon/lm83`.

Ferreira, A.P., Mosse, D., and Oh, J.C. (2007). Thermal faults modeling using a rc model with an application to web farms. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS 2007)*, 113–124. Pisa, Italy.

Fu, Y., Kottenstette, N., Chen, Y., Lu, C., Koutsoukos, X.D., and Wang, H. (2010). Feedback thermal control for real-time systems. In *Proceedings of the 16th Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, 111–120. Stockholm, Sweden.

Heath, T., Centeno, A.P., George, P., Ramos, L., Jaluria, Y., and Bianchini, R. (2006). Mercury and freon: Temperature emulation and management for server systems. In *Proceedings of the 2006 ASPLOS Conference*, 106–116. San Jose, CA, USA.

Lindberg, M. and Årzén, K.E. (2010). Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties. In *Proceedings of the 31st Real-Time Systems Symposium*. San Diego, California USA.

Makhorin, A. (2000). Gnu linear programming kit. `http://www.gnu.org/software/glpk`.

Manica, N., Abeni, L., Palopoli, L., Faggioli, D., and Scordino, C. (2010). Schedulable device drivers: Implementation and experimental results. In *Proceedings of International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 3–13. Brussels, Belgium.

MobileRobots Inc (2006). *Pioneer 3 Operations Manual*. Amherst, NH, US.

Mok, K., A., Feng, X., and Chen, D. (2001). Resource partition for real-time systems. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, 75–84. Taipei, Taiwan.

National Semiconductor Corporation (1999). *LM83 Triple-Diod Input and Logical Digital Temperature Sensor with Two-Wire Interface, DS101058*.

Rizvanovic, L. and Fohler, G. (2007). The matrix - a framework for real-time resource management for video streaming in networks of heterogenous devices. In *The International Conference on Consumer Electronics 2007*, 1–2. Las Vegas, USA.

Segovia, V.R. and Årzén, K.E. (2010). Towards adaptive resource management of dataflow applications on multi-core platforms. In *Proceedings Work-in-Progress Session of the 22nd Euromicro Conference on Real-Time Systems, ECRTS 2010*, 13–16. Brussels, Belgium.

Segovia, V.R., Årzén, K.E., Schorr, S., Guerra, R., Fohler, G., Eker, J., and Gustafsson, H. (2010). Adaptive resource management framework for mobile terminals - the actors approach. In *Proceedings of Workshop on Adaptive Resource Management, WARM 2010*, 28–33. Stockholm, Sweden.

Versalogic Corporation (2007). *Model VSBC-8 Reference manual*. Eugene, OR, US.