CROSS-STACK PREDICTIVE CONTROL FRAMEWORK FOR MULTICORE REAL-TIME APPLICATIONS

by

Guangyi Cao

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2014

Approved by:

____

Dr. Arindam Mukherjee

____

Dr. Arun Ravindran

____

Dr. Bharat Joshi

____

Dr. Anthony Wilkinson

ABSTRACT

GUANGYI CAO. Cross-stack predictive control framework for multicore real-time applications. (Under the direction of DR. ARINDAM MUKHERJEE)

Many of the next generation applications in entertainment, human computer interaction, infrastructure, security and medical systems are computationally intensive, always-on, and have soft real time (SRT) requirements. While failure to meet deadlines is not catastrophic in SRT systems, missing deadlines can result in an unacceptable degradation in the quality of service (QoS). To ensure acceptable QoS under dynamically changing operating conditions such as changes in the workload, energy availability, and thermal constraints, systems are typically designed for worst case conditions. Unfortunately, such overdesigning of systems increases costs and overall power consumption.

In this dissertation we formulate the real-time task execution as a Multiple-Input, Single-Output (MISO) optimal control problem involving tracking a desired system utilization set point with control inputs derived from across the computing stack. We assume that an arbitrary number of SRT tasks may join and leave the system at arbitrary times. The tasks are scheduled on multiple cores by a dynamic priority multiprocessor scheduling algorithm. We use a model predictive controller (MPC) to realize optimal control. MPCs are easy to tune, can handle multiple control variables, and constraints on both the dependent and independent variables. We experimentally demonstrate the operation of our controller on a video encoder application and a computer vision application executing on a dual socket quadcore Xeon processor with a total of 8 processing cores. We establish that the use of DVFS and application quality as control variables enables operation at a lower power operating point while meeting real-time constraints as compared to non cross-stack control approaches. We also evaluate the role of scheduling algorithms in the control of homogeneous and heterogeneous workloads. Additionally, we propose a novel adaptive control technique for time-varying workloads.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Dr. Mukherjee for opening up the door of research for me. I greatly appreciate his inspiring guidance, great patience and invaluable encouragement through the entire process of my doctoral study. I am deeply indebted to my committee member Dr. Ravindran who provided me timely and generous help when I was most uncertain about direction of my research. The impression on his enthusiasm and insight about research will continuously inspire me in my future professional endeavor. I would like to thank my committee member Dr. Joshi and Dr. Wilkinson for reviewing my work and providing helpful suggestions. I also would like to thank Dr. Ferguson for his encouragement and kindly support.

I would like to thank my parents Mr. Xinzhi Cao and Ms. Yinhua Pang for their unconditional love. They have cherished with me every great moment and supported me whenever I needed it. Without them, this dissertation would never have been written.

I would like to thank my dear friends Changshu Zhang, Libin Bai, Kushal Datta and Zhe Dang for all the great precious times that we have shared.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Motivation and Objective

The next decade of computing is expected to be driven by the increasing pervasiveness of personal mobile computing devices and cyber physical systems. Many of the next generation applications in entertainment, human computer interaction, infrastructure, security and medical systems are computationally intensive, always-on, and are characterized by periodic tasks with Soft Real-Time (SRT) requirements. While failure to meet deadlines is not catastrophic in SRT systems, missing deadlines can result in an unacceptable degradation in the Quality of Service (QoS). To ensure acceptable QoS under dynamically changing operating conditions such as changes in the workload, energy availability, and thermal constraints, systems are typically designed for worst case conditions. Unfortunately, such overdesigning of systems increases costs and overall power consumption. A possible solution to this problem is run-time adaptation of the system to handle dynamically changing operating conditions. Previous research on cross-stack run-time adaptation has focused on open-loop control where the output has no effect on the system input and hence can only counteract against disturbances for which it has been designed. In contrast in closed loop control, feedback is used to determine if real-time requirements are in met in the presence of unmodeled disturbances. However, existing research on closed loop control for real-time workloads have been limited to the use of control inputs derived from a single layer of the computing stack such as processor DVFS or scheduling policies.

In this dissertation we show that a higher overload capacity and better energy efficient operation of the system is possible if a closed loop control uses control inputs derived from all parts of the computing stack. We note that in many of the applications described above,

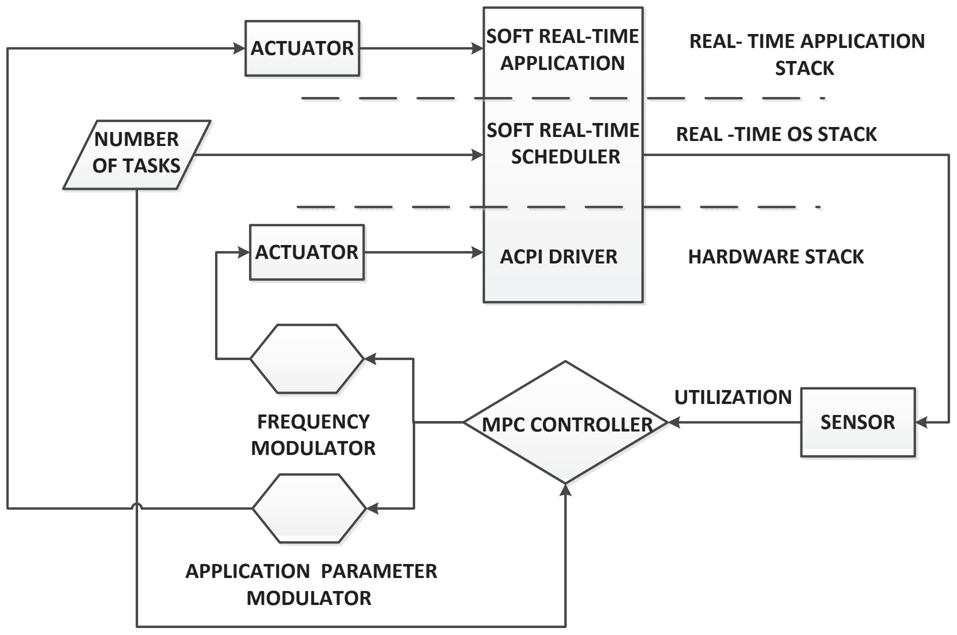


Figure 1.1: Schematic representation of our closed loop cross-stack predictive control framework

although deadlines need to be met to provide QoS guarantees, other quality parameters of the application (for example, visual quality in video processing) can be tuned in conjunction with hardware parameters (for example, DVFS) to give acceptable performance under overload conditions. We formulate the real-time task execution as a Multiple-Input, Single-Output (MISO) optimal control problem involving tracking a desired system utilization set point with control inputs derived from across the computing stack. We assume that an arbitrary number of SRT tasks may join and leave the system at arbitrary times. The tasks are scheduled on multiple cores by a dynamic priority multiprocessor scheduling algorithm. Note that utilization above the set-point results tasks missing deadlines while utilization under the set-point results in energy inefficient operation.

Our cross-stack control framework is shown in Fig 1.1. We use a model predictive controller (MPC) to realize optimal control. MPCs use an internal system model to predict the future trajectory of the output variables. Based on a history of past control moves, a constrained optimization is solved on-line to determine the future input trajectory such that the output variables track a reference trajectory over a receding horizon. MPCs are easy to tune, can handle multiple control variables, and constraints on both the dependent and independent variables.

1.2 Contributions

Homogeneous task control framework: Existing research on closed loop control for real-time workloads have been limited to the use of control inputs derived from a single layer of the computing stack such as processor DVFS or scheduling policies. In order to improve overload capacity and power efficiency of real-time multicore computing system, we propose a cross-stack control framework for homogeneous real-time workloads. We formulate this real-time multicore computing system as an Multiple Input Single Output (MISO) state space model. We use a Model Predictive Controller (MPC) to handle this MISO model since MPC can handle multiple control variables and set constraints on both the dependent and independent variables . MPC uses an internal system model to predict the future trajectory of the output variable. This model is derived by carrying out System Identification (SI) based on data collected on our experimental platform. For every control period, the controller reads system utilization from a sensor, calculates control variables based on control law of MPC, oversamples values of control variables with appropriate modulators and finally writes the oversampled values to application and hardware stack through actuators. We use DVFS technology to adapt operational frequency dynamically in hardware stack. Application quality are updated by writing through global variables protected by a real-time read-write lock.

We apply this homogeneous control framework to on a video encoder application (*x264*) and a computer vision application (*bodytrack*). The experimental platform is a dual socket quadcore Xeon processor with a total of 8 processing cores. We create the state space model using Matlab system identification toolbox and design the MPC controller with help of Matlab model predictive control toolbox. We manually tune controller parameters one at a time to obtain a good step response for the controller. All tasks including the application and the controller are scheduled by real-time dynamic task schedulers implemented in Linux using the Litmus-RT patch. Our results shows better system controllability can be achieved if the control inputs are derived from all parts of the computing stack: the cross-stack controller is able to maintain constant frame rate while DVFS-only or application quality-only control fails to do so at heavy workload (task number over 8)for both *bodytrack* and *x264*. The controller is able to track utilization set-point with a settling time less than 5 seconds in response to a 50 % step change in number of tasks for both *bodytrack* and *x264*. For a pseudo-random number of input tasks, our model predictive control approach shows an energy saving of 31 % compared to the non-control implementation at the highest frequency and application precision for *x264* and an energy saving of 26 % for *bodytrack*. The overall control overhead is less than 0.4 percent of one control period for both *bodytrack* and *x264*.

Heterogeneous control framework: In order to accommodate the more commonly scenario of a server running multiple distinct real-time workloads simultaneously, we propose a cross-stack predictive control framework for heterogeneous workloads. We adopt a cluster control approaches to deal with the problem where different types of workloads are partitioned into different clusters and each cluster is handled by its own controller. In this control approach we use the Cluster-Earliest Deadline First (C-EDF) scheduling algorithm as the task scheduling algorithm.

We applied the two different approaches mentioned above to a workload with a combination of *x264* and *bodytrack* tasks. Experimental results show that cluster control approaches can guarantee real-time constraints on heterogeneous workloads and show acceptable performance in terms of peak overshoot, settling time and jitter value. Due to superior load balancing capability, control with G-EDF performs better with an unbalanced workload. However, for a balanced but heavy workload with large number of tasks for both applications, load balancing is less of any issue. For this case, C-EDF with its better data locality performs better.

Adaptive control framework: We propose an adaptive cross-stack predictive control framework to maintain desired level of performance for dynamic workloads.

We employ a gain scheduling approach for our adaptive cross-stack predictive control framework that uses multiple fixed models identified based on $a\ priori$ workload characterization. During run-time, a supervisor periodically determine the model that is most suitable for actual and switches to the controller associated with the selected model. We select *x264* encoder as the workload to demonstrate the operation of our adaptive cross-stack predictive control framework since *x264* exhibits distinct visual and temporal features if videos from different video genres are used as its encoding input. We select 4 video genres: cartoon, entertainment, news report and sports. We initially subdivide 20 video files used for our experiments into the four genres, according to the subject of the video. We create model predictive controllers as well as their state space models derived by system identification for each genre. To determine which video genre should be chosen in real-time, we choose average frame execution time as video temporal feature and implement a video genre classifier using Kolmogorov-Smirnov test algorithm.

The video genre classifier achieves satisfactory validation results with no less than 90% success rate for each genre. Adaptive controller outperforms non-adaptive controller in terms of smaller steady state error. We also show that the adaptive control incurs a minimal overhead, which accounts for 0.086 % of each control period.

1.3 Organization

The remainder of the dissertation is organized as follows. Chapter 2 provides background on real-time system, control theory, adaptive computing technique and power model that form the foundation of our cross-stack control framework. Chapter 3 describes the cross-stack predictive control framework that improves overload capacity and energy efficiency for homogeneous real-time workloads. Chapter 4 discusses the extension to heterogeneous cross-stack control framework by exploring the impact of different scheduling policies: G-EDF and C-EDF on the performance of the controller. Chapter 5 describes the implementation of the the adaptive cross-stack predictive control framework for dynamic workloads. Chapter 6 concludes with a summary of the work presented in this dissertation and with the discussion of how future work could extend the research work presented in this dissertation. All chapters are relatively self-contained with the necessary background and related work.

CHAPTER 2: BACKGROUND

2.1 Introduction

In this chapter we briefly review basic concepts and terminology of real-time systems and model predictive control that form the foundation of our cross-stack control framework. We also provide a brief overview of the power-performance tuning capabilities of modern software and hardware. Before considering these aspects in detail, we first provide an overview of our cross-stack control framework.

As shown in Fig 2.1, our cross-stack control framework can be considered as a closed loop interconnected system consisting of the following components- hardware stack, real-time Operating System (OS) stack, real-time application stack, controller, sensor, and actuator. Multicore real-time computing system provides users their needed services through a collaboration of its three computing stacks: real-time application stack, real-time operating system stack, and hardware stack. The real-time application stack handles execution of the real-time workload. Real-time operating system stack is a collection of system software that provides common service to users' applications and serves as a middleman between application stack and hardware stack. These services usually include task management, time management, memory management and inter-process synchronization and communication [96]. Besides these basic services, real-time operating system stack employs real-time scheduling and synchronization algorithms to determine how the hardware resources are shared among the systems processes such that the real-time requirements can be guaranteed. Hardware stack is the physical entity realizing the functionality of data processing, storage and transfer in a computing system. Major elements of the hardware stack include one or more multicore processors, memory, I/O devices, and hard disk. Architecture details

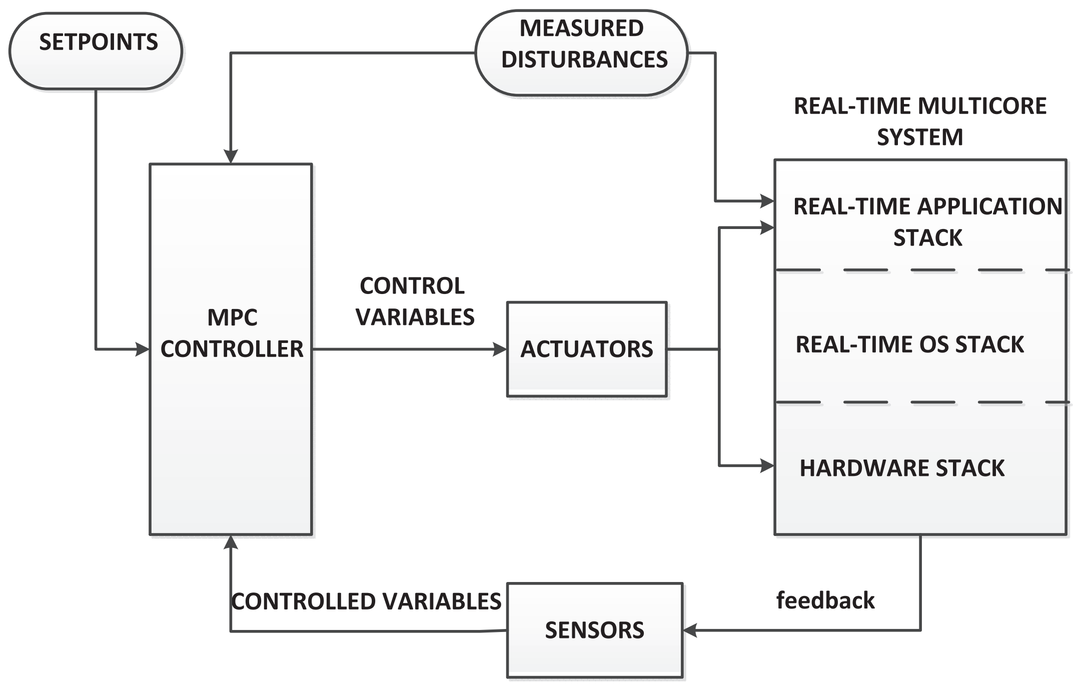


Figure 2.1: Cross-stack control framework

of of multicore processors will be covered in section 2.2.2.

According to Fig 2.1, the control goal is to strive for a desired system performance by dynamically tracking a desired operating set-point through the control of one or more system parameters (control variables). The system performance and set-point are defined by the user. In our work, we choose system utilization as a metric of system performance based on recent development of real-time system theory. To achieve the control goal, the controller follows a three-step procedure. For each control period, the controller first updates the measured system performance by reading the sensed system output. The controller then derives the error by subtracting measured system performance from the set-point. The error is used to update the control variables following the control law. In our work, the control variables are derived from both hardware stack and real-time application stack while the output sensor is realized in real-time OS stack.

The rest of this chapter is organized as follows. Section 2.2 briefly reviews basic concepts and terminology of real-time systems. Section 2.3 introduces background of model predictive control and how to formulate and solve our model predictive control problem, and Section 2.4 reviews adaptive computing techniques.

2.2 Real-time Systems

The distinguishing characteristic of a real-time system in comparison to a non-real-time system is the inclusion of timing requirements in its specification [21]. To guarantee correctness of a real-time system, not only does the system need to produce a logically correct result, but also timing constraints have to be met. Process control systems, air traffic control systems, and multimedia systems are some examples of real-time systems.

Timing requirements and constraints in real-time systems are commonly conveyed as deadlines within which activities should finish execution. Consider an example such as video conferencing, which allows two or more locations to communicate by simultaneous two-way video and audio transmissions. Holding video conferencing requires the system to perform following high-level activities of tasks: sample and capture raw image using a camera, compress it with an encoder, send it to destination through the internet, decode and display video frames received from the internet. Each of these tasks should be invoked repeatedly at a certain frequency, and each invocation should complete execution within a specified time or deadline. Failure to fulfill this timing requirement will result in degradation of service quality. Another characteristic of a real-time system is its predictability. It should always be able to check, prove or verify that the timing requirements are met under assumptions made on certain features of workloads [98]. Note that temporal correctness of real-time system may be impaired by phenomena such as priority inversions and deadlocks (details of priority inversions and deadlocks will be covered in subsection 2.2.5) which result in unpredictable task blocking time. A real-time system must thus incorporate a real-time synchronization protocol that avoids deadlock and allows the maximum length of priority inversion to be bounded.

Based on the consequences resulting from failure of not meeting deadlines, real-time systems are usually divided into two classes - hard and soft. A real-time system is said to be hard if missing its deadline may cause catastrophic consequences on the environment under control. Industrial process-control systems, robots, controllers for automotive systems, and air-traffic controllers are some examples of hard real-time systems. A real-time system is said to be soft if meeting its deadline is desirable for performance reasons, but missing its deadline does not cause serious damage to the environment and does not jeopardize correct system behavior. Multimedia systems and virtual-reality systems are some examples of soft real-time system.

Real-time system design includes four important components: real-time system models including task models and resource models; scheduling algorithms, which determine how the hardware resources are shared among the system's threads and/or processes; validation tests that determine whether a real-time system's timing requirement will be met by a specified scheduling algorithm; and real-time synchronization protocol assuring that deadlock and priority inversion will not harm temporal correctness of real-time system.

2.2.1 Hard Real-time Task Model

A real-time task model is used to describe the workload and the timing requirements associated with it. In real-time terminology, a piece of sequential work that has to be finished before its deadline is referred to as a job. So a simple task model of a real-time system can be a set of jobs, each of which is associated with an arrival or release time, a deadline, and a Worst-Case Execution Time (WCET). The release time of a job denotes the time after which the job is ready to execute. WCET is the maximum length of time a job could take to execute on a specific hardware platform. A review of WCET calculation and analysis is give in [106].

Many real-time systems are made up of one or more sequential chunks of code, each of which is executed repeatedly and each of whose execution should reach its completion within a specified amount of time. Each repeatedly invoked code segment is commonly encapsulated into a different process and is referred to as a task. Here job is an invocation of a task. Tasks can be initiated in response to (a) external activities which interacts with the system, (b) activities taking place in other tasks or (c) a timer. A task is long-lived and can be invoked an indefinite number of times unless its termination is explicitly specified. Hence, many real-time systems can be modeled as a set of N recurrent tasks denoted $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$. Each task τ_i is a sequential program described by three parameters: a WCET ($e_i > 0$), a minimum inter-arrival time ($p_i \geq e_i$) and a relative deadline ($D_i \geq e_i$). p_i denotes the minimum time that should elapse between two consecutive job invocations or arrivals of τ_i. D_i denotes the amount of time within which each job of τ_i should complete execution after its release. A recurrent task with the characteristics as described is referred to as a sporadic task and a task system consisting of sporadic tasks is referred to as a sporadic task system. A periodic task is a special case of a sporadic task in which any two consecutive job arrivals are separated by exactly p_i time units, and a task system whose tasks are all periodic is referred to as a periodic task system.

For a periodic or a sporadic task system, the k^{th} job, where $k \geq 1$, of τ is denoted $\tau_{i,k}$. The release time of $\tau_{i,k}$ and its absolute deadline are denoted $r_{i,k}$ and $d_{i,k}$ respectively. Here $d_{i,k} = r_{i,k} + D_i$. A job$'$s absolute deadline is the absolute or actual time by which the job should complete execution. If $D_i = p_i$ holds, then τ_i and its jobs are said to have implicit deadlines. A task system in which $D_i = p_i$ holds for every task is said to be an implicit-deadline system. Similarly a task system in which $D_i < p_i$ holds for every task is said to be an constrained-deadline system. Unless otherwise specified, all tasks are assumed to have implicit deadlines in this dissertation, and the notation $\tau_i(e_i, p_i)$ will be used to denote the parameters of τ_i concisely.

The ratio of the WCET to the period of a task is referred to as its $utilization$. The utilization of task τ_i is denoted $u_i = e_i/p_i$. Utilization of a task represents the fraction of a processor's computation power that is devoted to execution of this task in the long run. A task is said to be heavy if its utilization is at least 1/2, and light otherwise. The sum of the utilization of all tasks in τ is referred to as the total system utilization of τ and is denoted

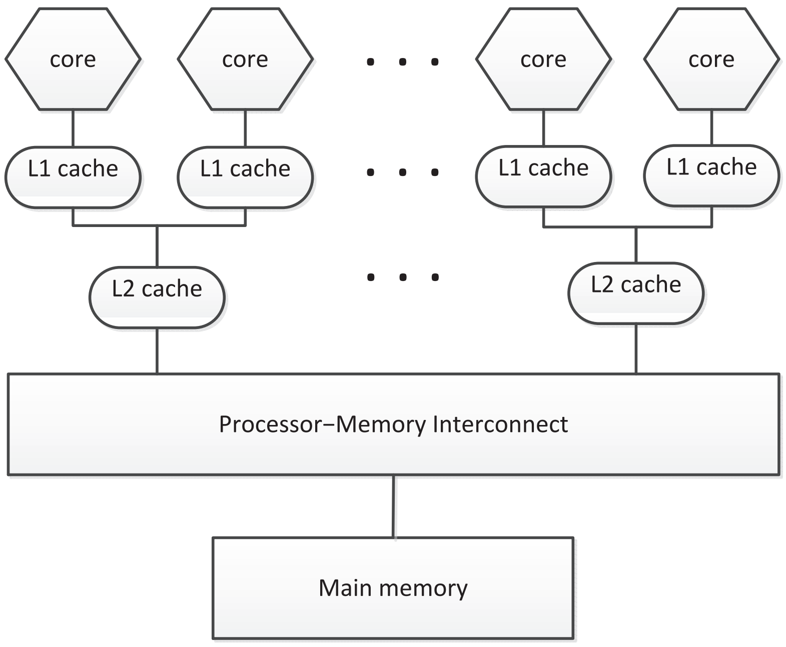


Figure 2.2: Structure of multicore processors with private L1 caches and shared L2 caches

$U_{sum}(\tau)$. $U_{sum}(\tau)$ denotes the total processing needs of τ. A sporadic task system τ is said to be concrete, if the release time and actual execution time (which is at most the WCET) of every job of each of its tasks is specified, and non-concrete, otherwise. Unless specified, actual job execution times are taken to be equal to their worst-case execution times.

2.2.2 Resource Model

Multicore processors are a special class of multiprocessors where multiple indentical processing cores are manufactured on a single integrated circuit chip to exploit increases in transistor density. By identical we mean that (a) all processing cores possess exact same computation capacity, which means, a task's execution is not affected by the processing core's identity, (b) main memory is shared among all the cores and a processing core can access each memory location with the same maximum latency. Such systems are called uniform memory access (UMA) architectures, and (c) each processing core is provided with one or more levels of identical caches to expedite access to frequently accessed addresses or addresses that are spatially close. Fig 2.2 illustrates the structure of a multicore processor with a two level cache hierarchy. Ideally, tasks should execute on every processing core identically with no restriction on the processing cores that a task may execute upon. However, due to the impact of cache, execution time of a job is likely to be more if a job migrates among multiple processing cores. To lower migration overheads, a scheduling algorithm may choose to restrict executing a task or a job to one or a subset of processing cores, even though the system model imposes no restriction.

Tasks can be interdependent on each other due to three factors: synchronization constraints imposed by producer/consumer relationships [47], a need to access shared data resources in a mutually-exclusive manner, and precedence constraints, which restrict the order in which tasks may execute. Under any one of the three scenarios, tasks will be blocked, adding extra overhead to the system. In this dissertation, we assume that tasks from the same application are independent. However, a producer/consumer relationship is imposed between the controller and the applications since in each control period, the controller updates the application parameters. Real-time synchronization used in our work to enforce this producer/consumer relationship is discussed in Section 2.2.5.

2.2.3 Real-time Scheduling Algorithms

A scheduling algorithm allocates processor computation power to tasks by assigning certain slots of a processor's execution time to certain tasks. Scheduling algorithms for general purpose operating systems are commonly employed in order to maximize overall throughput while ensuring fairness among all tasks. A typical example is complete fairness scheduler(CFS) [78] which is adopted by Linux since its 2.6.23 release. In contrast, scheduling strategies used in real-time systems is driven by the need to meet timing constraints. Real-time scheduling algorithms commonly assign priority to each job and select the M highest priority jobs to execute on a M processor system as long as constraints on migrations, preemptions, concurrency, and mutually-exclusive executions are not violated. Before getting into the details of real-time scheduling algorithms, we define terms and metrics commonly used in characterizing real-time scheduling algorithms and in comparing different algorithms.

Feasibility, schedulability, and optimality: A task system τ is said to be feasible on a hardware platform if there is some way of scheduling and meeting all the deadlines of τ on that platform. τ is said to be schedulable on a hardware platform by algorithm $\mathcal{A}$, if $\mathcal{A}$ is capable of correctly scheduling τ on that platform, i.e., can meet all the deadlines of τ. $\mathcal{A}$ is said to be an optimal scheduling algorithm if $\mathcal{A}$ can correctly schedule every feasible task system on every hardware platform. In pratice, optimality is often restricted to a subset of task systems (such as periodic or sporadic task systems) or to a class of scheduling algorithms or a hardware platform class (uniprocessors and multiprocessors).

Schedulable utilization bound: A commonly used metric for comparing effectiveness of different scheduling algorithms in meeting deadlines of a task system is schedulable utilization bound. Formally, if $\mathcal{U}_{\mathcal{A}}(M)$ is a schedulable utilization bound [76] for scheduling algorithm $\mathcal{A}$, then on M processors, $\mathcal{A}$ can correctly schedule every recurrent task system τ with $U_{sum}(\tau) \leq \mathcal{U}_{\mathcal{A}}(M)$. In addition, if there exists at least one task system with total utilization slightly over $\mathcal{U}_{\mathcal{A}}(M)$ by an infinitesimal amount and has a deadline miss under $\mathcal{A}$ on M processors, $\mathcal{U}_{\mathcal{A}}(M)$ is called worst-case schedulable utilization . Furthermore, if no task system with total utilization exceeding $\mathcal{U}_{\mathcal{A}}(M)$ can be scheduled correctly by $\mathcal{A}$, then $\mathcal{U}_{\mathcal{A}}(M)$ is said to be the optimal utilization bound of $\mathcal{A}$ for M.

Schedulability tests: Simple and fast validation tests and online admission-control tests for the algorithms are designed based on schedulable utilization bound for schedulability checking. With known schedulable utilization bound $\mathcal{U}_{\mathcal{A}}(M)$ of algorithm $\mathcal{A}$ and a task system τ, an O(N)-time schedulability test for τ under $\mathcal{A}$ can be performed to verify whether $U_{sum}(\tau)$ is no greater than $\mathcal{U}_{\mathcal{A}}(M)$. However, sometimes optimal schedulable utilization bound is not known for certain scheduling algorithms, making the tests only sufficient but not necessary. This can lead to pessimistic conclusion that deadlines will be missed.

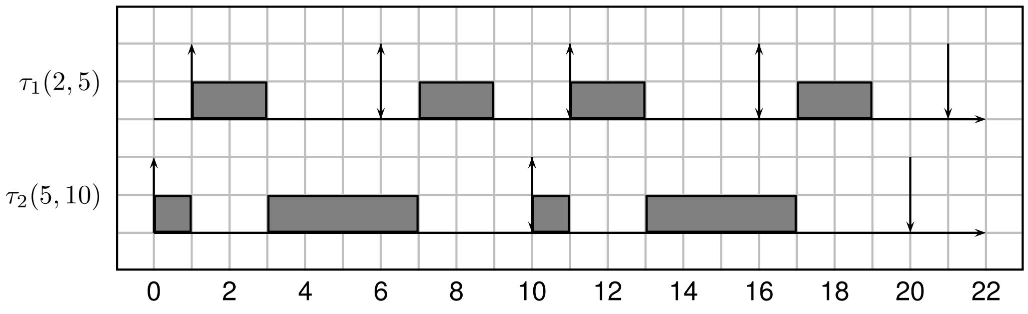
2.2.3.1 Scheduling on Uniprocessors

To ensure that timing requirements are met, priority in a real-time system can be given to: jobs with earlier deadlines; those with smaller slack times (at any time t, the slack time of a job $\tau_{i,k}$ with deadline $d_{i,k}$ is equal to $d_{i,k} - t - e_l$. e_l is the time required to complete the remaining portion of the job); or jobs of tasks with shorter periods. We will consider real-time scheduling on uniprocessor in this section followed by scheduling on multiprocessors.

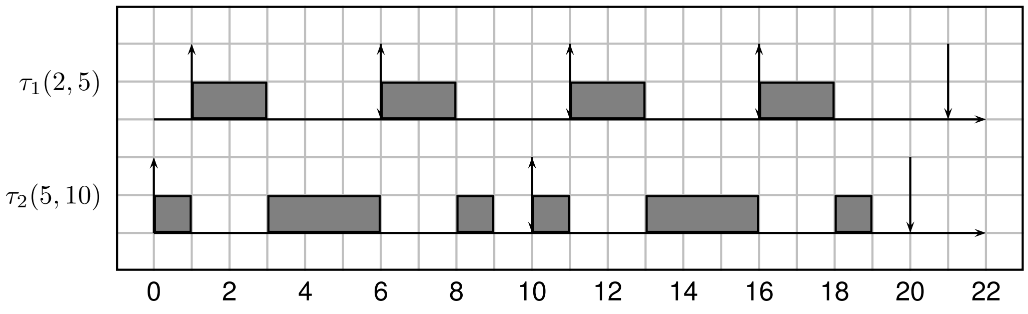
The algorithm using the first strategy is called Earliest-Deadline-First (EDF) [75], which is optimal for scheduling sporadic tasks on a uniprocessor [60]. The second strategy is adopted in a algorithm called *Least-Laxity-First* (LLF) [77], which is also an optimal scheduling algorithm for scheduling sporadic tasks on a uniprocessor. The well-known *Rate-Monotonic* (RM) scheduling algorithm [54] adopts the third strategy to advocate those tasks with short periods over those with longer periods. The three algorithms differ in their computation complexities and their abilities to meet job deadlines. Fig 2.3 shows an example where two tasks are scheduled under the three strategies separately on a uniprocessor system. Based on how job priorities are assigned and updated under each algorithm, a priority-based classification for real-time scheduling algorithm is proposed in [25]. Before describing that classification, we briefly describe two other ways of classifying scheduling algorithms.

Preemptive and non-preemptive algorithms: Under preemptive algorithms, a job can be blocked during its execution before completion. A job may get preempted only if another job with a higher priority arrives and the scheduler decides to run the job on the same processor. Under non-preemptive algorithms, a job may not be interrupted once it starts execution. All jobs from other tasks cannot occupy this processor until the job reaches its completion.

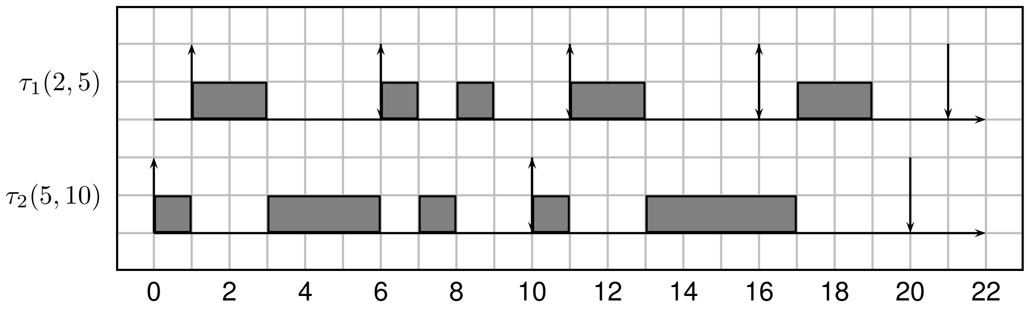
Work-conserving and non-work-conserving algorithms: An algorithm is considered as work conserving if no processor stays in the idle state when one or more jobs are ready to execute, and non-work-conserving, otherwise. The reason of idling the processor intentionally is to improve schedulability. For instance, under non-preemptive algorithms,



(a)



(b)



(c)

Figure 2.3: Uniprocessor schedules under (a) EDF, (b) RM, and (c) LLF for a task system with two tasks.

idling may prevent binding a job prematurely to a processor, and hence, has the potential to correctly schedule task systems that are otherwise not schedulable. However, due to high time complexity, non-work-conserving algorithms mainly find their application in off-line scheduling. Scheduling algorithm used in this dissertation are work-conserving.

Based on how job priorities are assigned and updated, scheduling algorithms can be classified into the following three categories.

Static-priority algorithms (P_s): Under static-priority algorithms, the priority of a task is kept unchanged across job executions. For example, task priorities in the RM algorithm mentioned above is determined by the task period. RM is also an optimal static priority scheduling algorithm for sporadic task systems on uniprocessors if relative deadlines are equal to periods as assumed in this dissertation.

Restricted-dynamic-priority algorithms (P_d^r): The algorithms in this class are also referred to as task-level dynamic-priority and job-level fixed-priority algorithms in the literature [25]. Generally, under this class of algorithms, the priority of a job is determined dynamically, so different jobs of a task can be assigned different priorities. However, the priority of a job cannot be changed during its execution. EDF is an algorithm in this class.

Unrestricted-dynamic-priority algorithms (P_d^u): Under algorithms in this class, a job is allowed to change its priority during its execution. LLF is one example of an algorithm in this class.

Although the worst-case time complexity in selecting the highest priority job is $O(logN)$ (where N is the number of tasks) for each of RM, EDF, and LLF [61], extra overhead resulting from updating priorities is incurred in EDF as absolute deadlines change from a job to the other and need to be computed at each job activation. Such a runtime overhead is not present under RM, since periods are typically constant. This overhead in LLF is even worse since priority of job may change multiple times before its completion. The maximum possible number of job preemptions, as a function of the total number of jobs, is asymptotically comparable for the static and restricted dynamic-priority classes. However, in practice, the actual number can be higher for RM than for EDF [20]. Number of preemptions in the unrestricted-dynamic priority class is much higher than number in other two classes depending on the rate at which job priorities change. Preemptions not only generate context switches but also have cache-related overheads. Thus it is desirable that their number be minimized.

On a single processor, the worst-case schedulable utilization of RM for a sporadic task systems is $U_{RM} = N.(2^{1/N} - 1)$ [60], which converges to $ln2 \approx 0.69$ as $N \longrightarrow \infty$. It should be mentioned that the utilization-bound-based schedulability test for RM is only a sufficient test and can be pessimistic. Previous research [54] showed that the schedulable utilization bound of RM can be up to 88% with a more accurate and complex test.

On the other hand, the schedulable utilization bound of EDF is 1.0 for all N [60]. This test is both necessary and sufficient for any task system on a single processor. So, without considering scheduling overheads, using EDF algorithm instead of RM can significantly improve schedulability. Besides, no task system with utilization over 1.0 is feasible on a uniprocessor, EDF is optimal not only for its class, but also across all the classes of scheduling algorithms applied to uniprocessors.

Since EDF is universally optimal, the extra flexibility in job priorities changing provided in algorithms belonging to unrestricted-dynamic-priority class (such as LLF) does not bring additional benefits for task systems on uniprocessors. Moreover due to the high overhead, unrestricted-dynamic-priority algorithms are not popular on uniprocessors.

2.2.3.2 Scheduling on Multiprocessors

Three approaches used in scheduling on multiprocessors are partitioning, clustering and global scheduling.

Under partitioning, a set of tasks are partitioned statically among processors. Tasks under each processor are scheduled by an individual scheduler dedicated to the processor. Each processor also possesses its own ready queue which holds all the active jobs eligible for execution. Schedulers on different processors may or may not be based on the same

Table 2.1: Carpenter *et al.*'s classification of multiprocessor scheduling algorithms presented in [25]. Entries in the table represent known lower and upper bounds on the worst-case schedulable utilization, U, for the different classes of scheduling algorithms. $\alpha = u_{max}$, the maximum utilization of any task in the task system under consideration.

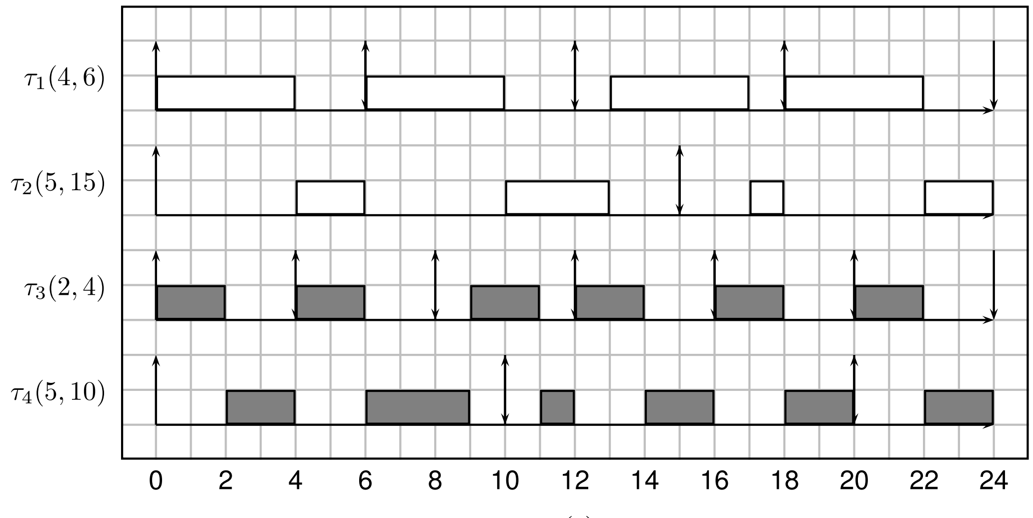
	P_s: static	p_d^r: restricted dynamic	p_d^u: unrestricted dynamic
M_f: full migration	$\frac{M^2}{3M-2} \le U \le \frac{M+1}{2}$	$U = M - \alpha(M-1)$ if $\alpha \le \frac{1}{2}$ $U = \frac{M+1}{2}$, *otherwise*	$U = M$
M_r: restricted migration	$U \le \frac{M+1}{2}$	$U = M - \alpha(M-1)$ if $\alpha \le \frac{1}{2}$ $U = \frac{M+1}{2}$, otherwise	$U = M - \alpha(M-1)$ if $\alpha \le \frac{1}{2}$ $U = \frac{M+1}{2}$, otherwise
M_p: partitioning	$U = \frac{M+1}{2}$	$U = \frac{\beta M+1}{\beta+1}$, where $\beta = \lfloor \frac{1}{\alpha} \rfloor$	$U = \frac{\beta M+1}{\beta+1}$, where $\beta = \lfloor \frac{1}{\alpha} \rfloor$

scheduling algorithm. The algorithm used for task partitioning must ensure that the sum of task utilization in each processor does not exceed the utilization bound.

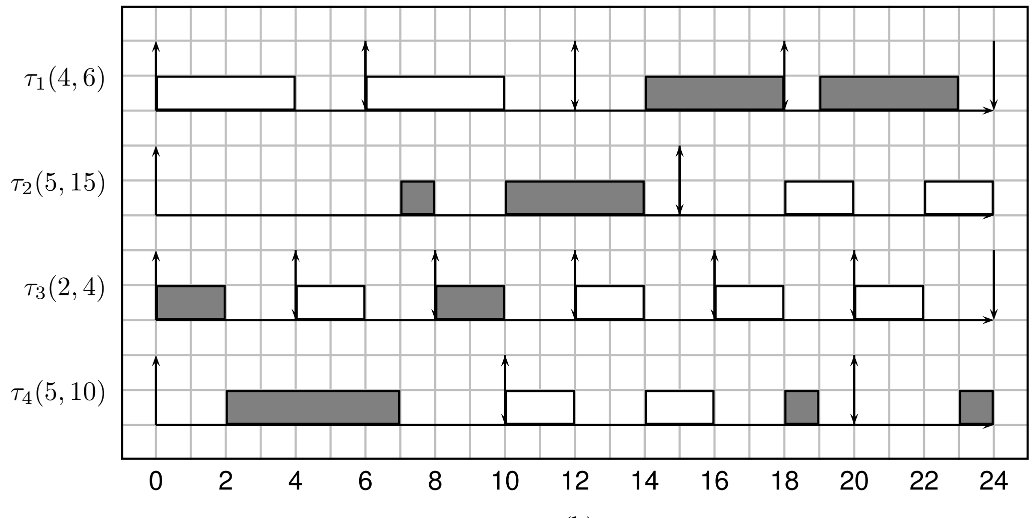
In contrast to partitioning, global scheduling employs a unified scheduler and ready queue to handle all the tasks. At any instant, at most M jobs with the highest priorities executes on M processors. There is no restriction imposed on where a task can execute. Not only can a task execute on different processors, but also it can execute on different processors at different times. Fig 2.4 shows an example where the same task system is scheduled under both Partitioned EDF (P-EDF)and Global EDF (G-EDF) scheduling algorithm.

Clustering can be consider as a hybrid approach of global and partition scheduling. Under clustering, M processors are split into several $\lceil \frac{M}{c} \rceil$ disjoint sets (or clusters) of c processors each. Each cluster is associated with a separate scheduler for scheduling tasks assigned to the cluster and a ready queue which holds its active jobs. Schedulers of different clusters may or may not be based on the same scheduling algorithm. Jobs can migrate freely among all the processors belonging to the same cluster similar to the global approach, but inter-cluster migration is prohibited. Cluster scheduling is a generalization of both global and partitioned scheduling: if $c = 1$, then cluster scheduling yields pure partitioned scheduling; if $c = M$, then cluster scheduling is equivalent to global scheduling.

Carpenter *et al.* [25] provides a comparison of the schedulability of different classes based on the the best known schedulable utilization bounds for any algorithm in each class.



(a)



(b)

Figure 2.4: Schedules under (a) partitioned-EDF and (b) g-EDF for four tasks on two processors. White or dark color rectangle indicates execution on processor 1 or processor 2.

The results are summarized in Table 2.1.

The best known lower and upper bounds on the worst-case schedulable utilization of any algorithm in each of the nine classes are provided in Table 2.1. The top, left entry in the table means that there exists some scheduling algorithm in this class that can correctly schedule every task system with utilization at most $\frac{M^2}{3M-2}$; and there exists some other algorithm in the same class that cannot correctly schedule a task system with utilization higher than $\frac{M+1}{2}$. Other entries can be interpreted in the same way. An "equals" operator means that the upper bound and lower bound are the same, thus indicating standing the worst-case schedulable utilization.

As shown on the table, within P_d^r class, both the M_f and M_r classes shows better schedulability than M_p class when α is small(around 0.2) This indicates a trend of allowing task migration tends to improve schedulability. This trend more obviously affects P_d^u class, under which easing the restrictions on migrations can increase schedulability to 100%.

Experimental data shows that execution overheads incurring from preemption and migration generally increases when migration restriction are relaxed. Some significant components of the overhead contributing are discussed below. As discussed in Subsection 2.2.2, a major overhead incurring from migration is due to the loss of cache affinity. With write-back cache, this overhead includes the time to invalidate the related cache-lines on the processor from which the task migrates and the time to load data into the current processor. Another overhead due to migration comes from the updating of a task′s Process Control Block (PCB). As most modern processor support virtual memory, recently-used page table entries of a process may also have to be invalidated and refetched. Note that the overhead mentioned above may be incurred even if a preempted job resumes its execution at a later time on the same processor. This is because its related cache-lines may be evicted by the jobs that execute in the intervening time.

2.2.4 Soft Real-time Systems

A task system with soft real-time constraints is referred to as a soft real-time system.

In this section, we describe the evolution of the task model for soft real-time systems from hard real-time task model introduced in Subsection 2.2.1 ensuring that tardiness of soft real-time system is within its acceptable range.

2.2.4.1 Soft Real-time Task Model

The soft real-time task model modifies the hard real-time task model by adding a tardiness bound with every task in the task system. If δ stands for tardiness bound of a soft real-time task, then any job of this task can miss its deadline by at most δ time unit. Although a job can miss its deadline in soft real-time systems, we assume that missed deadlines do not delay future job releases. For example, in the schedules in Fig 2.5 although jobs of τ_3 miss their deadlines, releases of future jobs are not affected. It should be noted that, based on this assumption, although a job will be released on time no matter the prior job miss its deadline or not, it cannot commence execution until the prior job completes execution due to an implicit precedence relationship. For example, in a video decoding task, while it is desirable that each video frame is decoded within 33 milliseconds, a tardiness of few milliseconds will not degrade video quality as long as the average rate of decoded frames per second stays at 30. Tardiness may introduce jitter to the job execution time, but it is unlikely that a small amount of jitter will be discerned by human eyes.

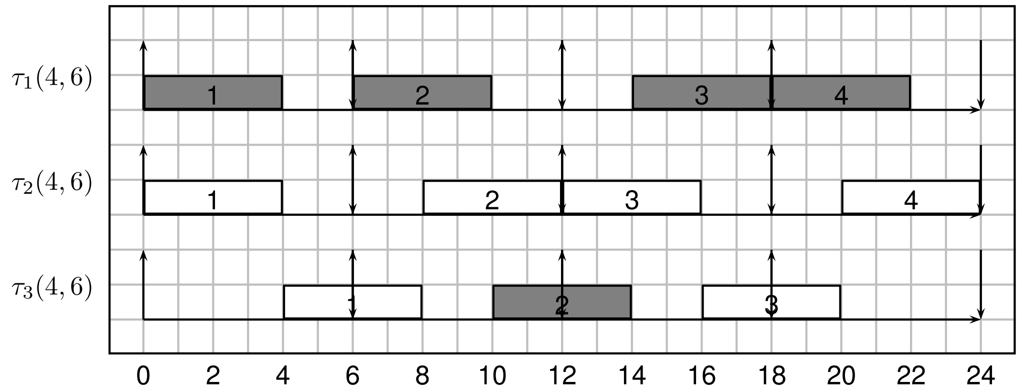


Figure 2.5: Schedules three tasks under two processors under the scheduling algorithm of (P_d^r, M_f) class (i.e. global EDF). White or dark color rectangle indicates execution on processor 1 or processor 2. Jobs of τ_3 miss their deadlines, but releases of future jobs are not affected.

2.2.4.2 Tardiness Bound Assurance

The ultimate objective of our cross-stack control framework is to assure that tardiness is bounded or some performance metric equals to a desirable value(for example, decoded frames per second stays at 30) by dynamically adjusting application and hardware parameters in cross-stack of a dynamic task system, where tasks may join or leave at run-time. To achieve this objective, we have to identify which system parameter should be chosen as controlled variable and how to assign set-point to this controlled variable so that tardiness will be bounded. Devi *et al.*'s work [32] provided a solution to this question. They prove that under preemptive and non-preemptive global EDF, for sporadic real-time task systems on multiprocessors, when the total utilization of a task system is not greater than maximum processing capacity, M, tardiness is bounded. We therefore use system overall utilization as the controlled variable for our control framework.

2.2.5 Real-time Synchronization Protocols

Phenomena including deadlock and priority inversion adversely affects temporal correctness of real-time systems. So synchronization in real-time systems has to prevent race conditions while bounding the maximum duration of priority inversion and avoiding deadlocks. In this subsection, we first introduce the concepts of deadlock and priority inversion. We then introduce major real-time synchronization protocols used in uniprocessors and multiprocessors.

Deadlock and priority inversion: Deadlock [108] is a phenomenon where multiple tasks wait to acquire locks held by other tasks. In deadlock, no task is able to release the lock and no progress can be made on job executions. Fig 2.6 shows an example of deadlock which involves two tasks τ_1,τ_2 and two mutex locks S and R in a uniprocessor system. Note that here we assume task index stands for its priority and smaller value indicates a higher priority, so τ_1 will preempt τ_2 when they are both ready to execute (i.e. $t = 3$). By $t = 8$, τ_1 and τ_2 enter states of deadlock as they all wait each other to release mutex locks. This example shows that lock nesting is a possible cause of deadlocks. Another phenomenon

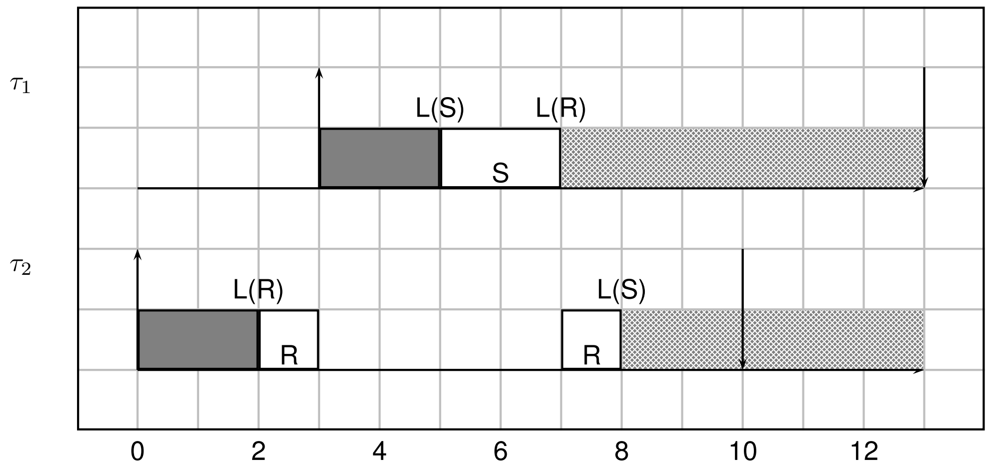


Figure 2.6: Deadlock example that two tasks τ_1,τ_2 and two mutex locks S and R present in a uniprocessor system. "L(S)" stands for mutex lock attempt on S. Hatched rectangle stands for task being blocked.

which endangers temporal correctness of real-time system is priority inversion [51]. In priority inversion, a medium priority task indirectly preempts a high priority task as if their relative priorities are inverted. Fig 2.7 shows such an example illustrating Priority Inversion blocking (PI blocking). This example involves three tasks, two of which share a resource protected by a mutex lock S. The first task τ_3 acquires the mutex lock which causes a later-arriving and higher priority task τ_1 to be suspended at $t = 3$. Afterwards, at $t = 4$, a medium priority task τ_2 joins in and preempts τ_3. As result, τ_2 indirectly delays τ_1 for its entire execution so that τ_1 missed its deadline by 3 time units.

Uniprocessor real-time synchronization protocols: Priority Inheritance Protocol (PIP) [94] is the most widely-used real-time synchronization protocol for uniprocessors. PIP takes effect when preempting a lower priority task would delay a higher priority task. PIP rejects such preemption by temporarily increasing priority of a task holding the resource to the highest priority of any tasks waiting on the resource. Fig 2.8 shows that how PIP handles the same situation in Fig 2.7. Here priority of τ_3 is praised to 1 at $t = 3$ due to τ_1's attempt to acquire the lock at that time so that medium priority task τ_2 cannot preempt τ_3 at $t = 4$. In this case PIP successfully prevents the highest task τ_1 from suffering PI blocking

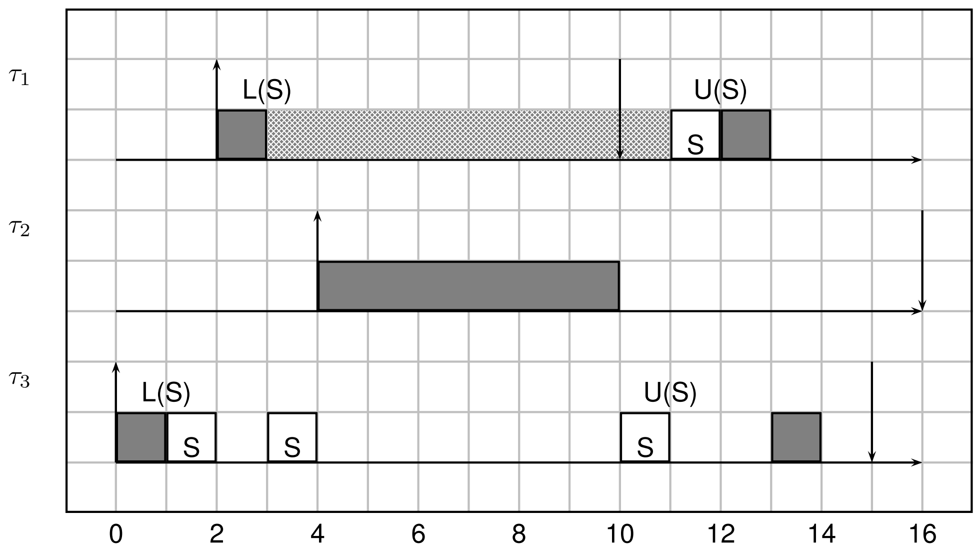


Figure 2.7: Priority inversion example involving three tasks, two of which share a resource protected by a mutex lock S. "L(S)" stands for mutex lock attempt on S. "U(S)" stands for mutex unlock on S. Hatched rectangle stands for task being blocked.

due to the medium priority task τ_2.

However, PIP has the downside of susceptiblity to deadlock. Hence Priority Ceiling protocol (PCP) [95], another procotol which is based on PIP, has been proposed to overcome this problem. PCP can be viewed as PIP with an access test determining whether it is safe to allocate a semaphore S to a job J. In PCP, each resource is assigned a priority ceiling, which is a priority equal to the highest base priority of any task which attempts to acquire the resource. A job J is allowed to lock a semaphore only if J's priority is strictly greater than the priority ceilings of all semaphores locked by other jobs in the system. By doing so, PCP prevents potentially problematic resource requests until deadlock is impossible. [16] provides more detailed analysis about PCP.

Multiprocessor real-time synchronization protocols: A resource is local to a processor if all jobs requesting this resource execute on this processor, and global otherwise. Since local resource can be handled with uniprocessor protocol such as PCP, the major responsibility of multiprocessor real-time synchronization is to tackle global resource. Multiprocessor priority ceiling protocol (M-PCP) [84] was proposed in mid-1990s and is probably the most

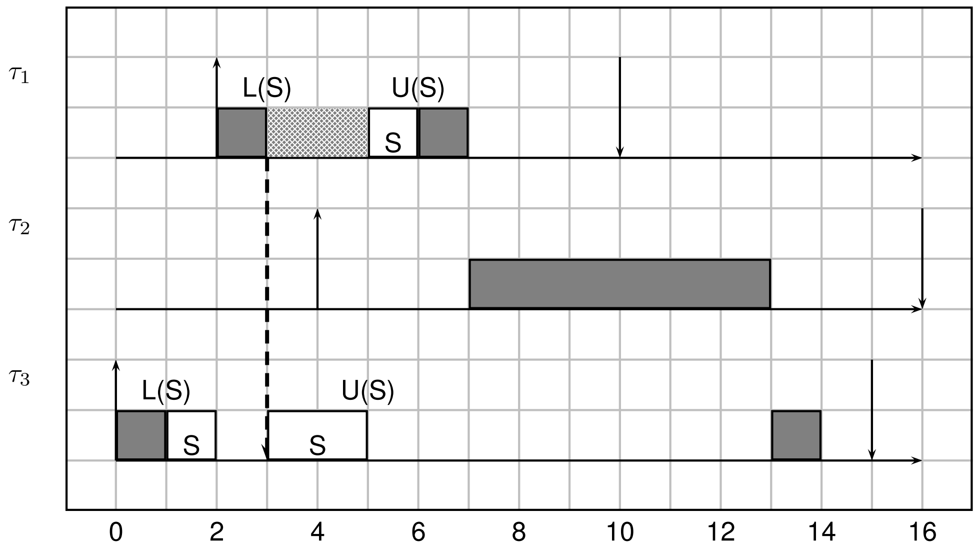


Figure 2.8: PIP schedule of the scenario shown in Fig 2.7. ”L(S)” stands for mutex lock attempt on S. ”U(S)” stands for mutex unlock on S. Hatched rectangle stands for task being blocked.

widely known locking protocol for multiprocessor real-time system. For global resource, M-PCP let jobs holding global resources execute with an effective priority higher than that of any normal task. Competing requests for global resources are served in order of job priority. A requesting job will be suspended if the request cannot be served immediately. Fig 2.9 demonstrates a M-PCP schedule for two-processor system.

However, recent research [14] [17] show that a newly-proposed protocol, flexible multiprocessor locking protocol (FMLP), is superior to them in terms of both performance and flexibility. FMLP is considered as flexible in the sense that it can be used either partitioned or global scheduling, with either static or dynamic task priorities, and it is consistent with both spin lock and semaphore.

FMLP classifies global resources as either ”short” or ”long” type. Short resources are accessed using spin lock and long resources are accessed via a semaphore protocol. To avoid dead lock, FMLP divides resources into groups and only allows one job to access resources in any given group at any time. Two resources are in the same group if they are from the same type (short or long) and requests for those resources may nest with each

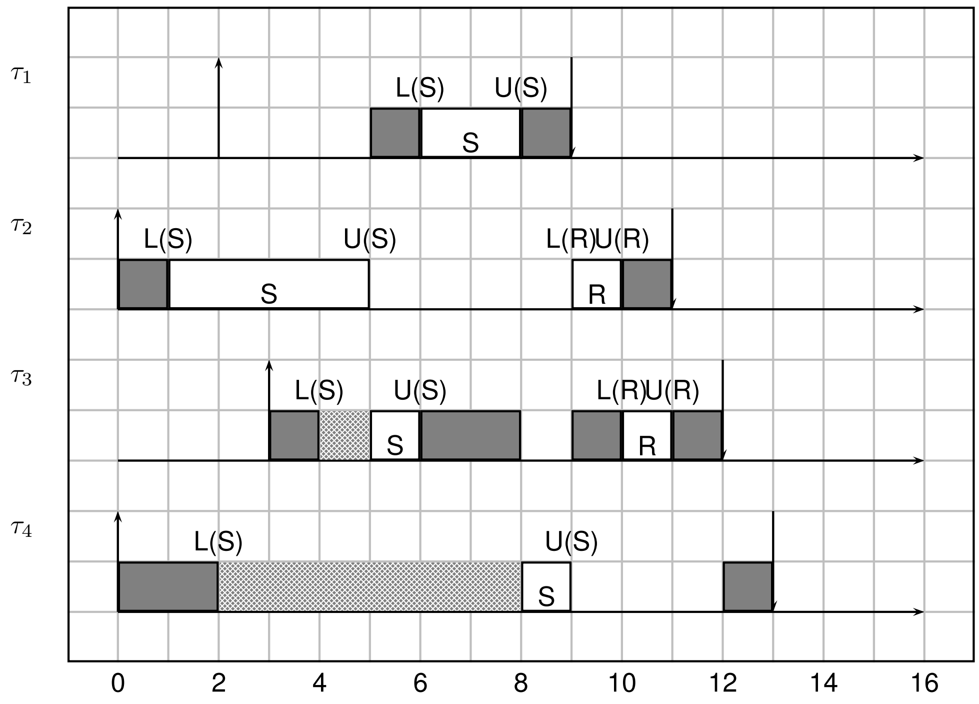


Figure 2.9: M-PCP schedule of four tasks sharing two global resources S and R in a two-processor system. τ_1, τ_2 shares processor 1; τ_3, τ_4 shares processor 2.

other. A group lock is used to realize this mechanism in the way that jobs must first acquire its corresponding group lock before access it.

If request is short and outermost, the corresponding job become non-preemptable and try to hold the group lock (a spin lock). Other blocked jobs busy-wait in FIFO order. The request is satisfied once holding the lock. When the request completes, the job will release the group lock and again become preemptive.

If request is long and outermost, the corresponding job will try to hold the group lock (a semaphore). With a semaphore lock, blocked jobs are added to a FIFO queue and suspend. When the request is satisfied, the job will be executed non-preemptively. This is achieved by boosting its priority to 0. When the request finishes, the job will release the group lock and become preemptive.

Compared with M-PCP, the most significant difference of FMLP is that it serves resource requests in FIFO order instead of job priority and consider distinctions between long and short resource. Brandenburg *et al.* reported FMLP shows better performance than

M-PCP in their work [17] Fig 2.10 depicts FMLP schedule for the same scenario in Fig 2.9 for M-PCP. Note that different from M-PCP, contending requests are satisfied in FIFO order: τ_4's request is satisfied before that of τ_3 at time 5.

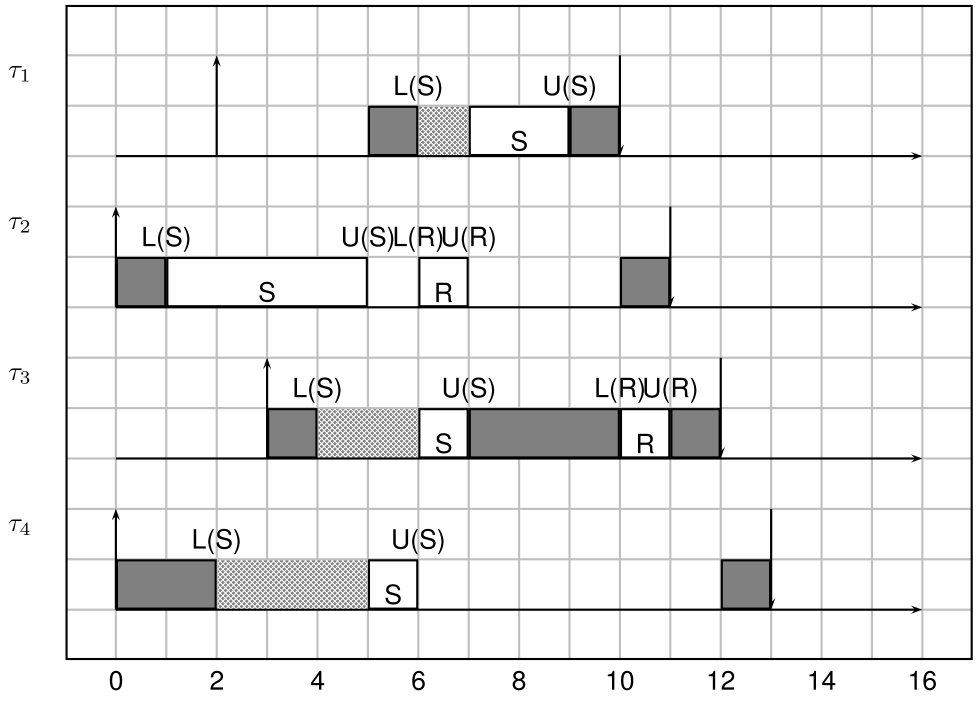


Figure 2.10: FMLP schedule of same scenario in Fig 2.9: four tasks sharing two long global resources S and R in a two-processor system. τ_1, τ_2 shares processor 1; τ_3, τ_4 shares processor 2.

2.3 Model Predictive Control(MPC)

In this section, we first introduce background and general principle of MPC. Then we describe how to formulate our system as a standard linearized, discrete-time, state space model for model predictive control. Finally we explain how to solve this MPC problem.

2.3.1 Background of MPC

Model predictive control is an advanced control methodology which has made a significant impact on industrial control engineering [103]. Compared with other control methodologies, it has four unique features [65]:

- It can conveniently handle multivariable control problems.
- It considers constraints on the manipulated variable.

- It allows for an operating point operations close to the constraints.
- It requires lower control rates, which results in lower control overhead.

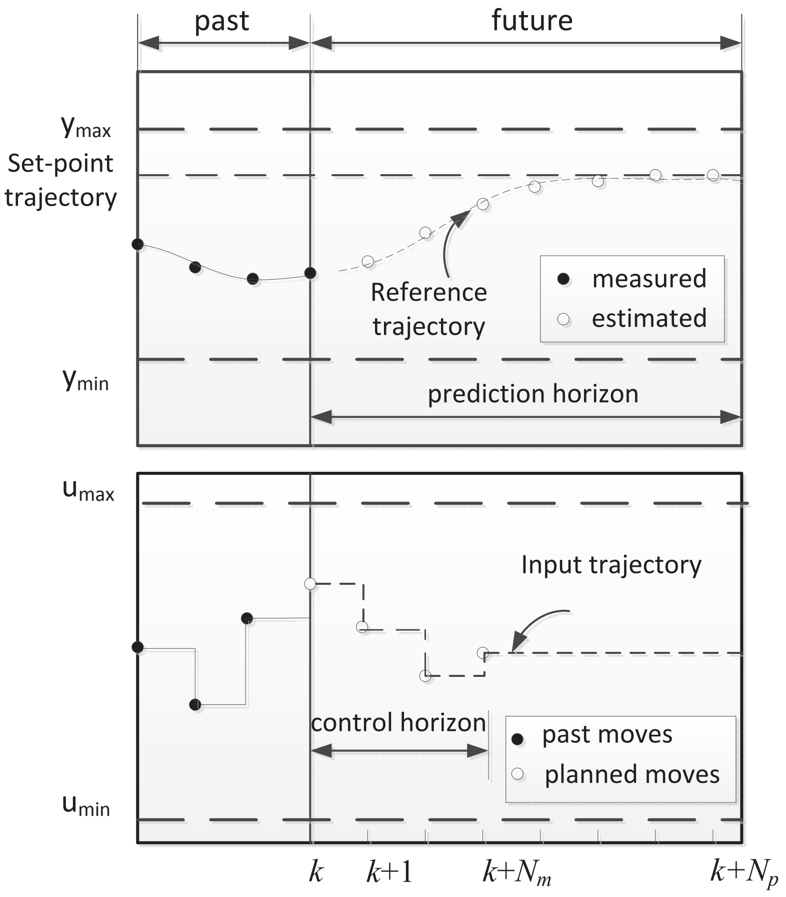


Figure 2.11: A schematic representation of model predictive control strategy. The dotted line crossing with white spots is reference trajectory.

Fig 2.11 shows the basic strategy of model predictive control. In this schematic representation, we assume a discrete-time setting and the current time is labeled as time step k. We also assume that the system operates within its input constraints (u_{min}, u_{max}) and output constraints (y_{min}, y_{max}) . From the current time instant k, based on the dynamic system model, model predictive controller predicts that system output will track to set-point trajectory in N_P time steps [24]. This predicted ideal trajectory along which the plant should return to the set-point trajectory is called reference trajectory. N_P is called prediction horizon. This predicted behavior of system output depends on the input trajectory which are to be applied over N_C time steps. N_C is called control horizon. Usually, N_C is smaller than N_P. Based on a history of past control moves, model predictive control minimizes a cost function J with constraints to determine this future input trajectory. Once the input trajectory is computed, only the first element is applied as control input to the system. At the next control period, the window of prediction horizon and control horizon will move one step forward and the whole process will be repeated. Such a control strategy is also called receding horizon control.

2.3.2 System Model

We model our system as a standard linearized, discrete-time, state space model in the form below [10]:

$$\begin{aligned} x(k+1) &= Ax(k) + B_u u(k) + B_v v(k) + B_d d(k) \\ y_m(k) &= C_m x(k) + D_{vm} v(k) + D_{dm} d(k) \end{aligned} \tag{2.1}$$

In Equation 2.1, $x(k)$ is the N_x-dimensional state vector of the plant, $u(k)$ is the N_u dimensional vector of manipulated variables, $v(k)$ is the N_v dimensional vector of measured disturbances, $d(k)$ is the N_d dimensional vector of unmeasured disturbances entering the system, and $y_m(k)$ is the N_o dimensional vector of measured outputs. The unmeasured disturbance $d(k)$ is modeled as the output of an LTI system

$$\begin{aligned} x_d(k+1) &= \bar{A} x_d(k) + \bar{B} n_d(k) \\ d(k) &= \bar{C} x_d(k) + \bar{D} n_d(k) \end{aligned} \tag{2.2}$$

Where $n_d(k)$ is the random Gaussian noise with zero mean and unit covariance matrix.

2.3.3 MPC Algorithm

The values of the set-points, measured disturbances, and constraints are specified over a finite prediction horizon P; the controller computes future inputs for a control horizon

M $(1 \leq M \leq P)$. Assuming that the estimates $x(k)$ and $x_d(k)$ are available at time k from state estimation, the future inputs at time k are obtained by solving the optimization problem [10]

$$\min_{\Delta u(k|k),\ldots,\Delta u(m-1+k|k),\varepsilon} \{\sum_{i=0}^{p-1}[\sum_{j=1}^{n_y} |w^y_{i+1,j}(y_j(k+i+1|k) \\ - r_j(k+i+1))|^2 + \sum_{j=1}^{n_u} |w^{\Delta u}_{i,j}\Delta u_j(k+i|k)|^2] + \rho_\varepsilon \varepsilon^2\} \quad (2.3)$$

subject to the constraints,

$$\begin{aligned} & u_{jmin}(i) - \varepsilon V^u_{jmin}(i) \leq u_j(k+i|k) \leq u_{jmax}(i) + \varepsilon V^u_{jmax}(i) \\ & \Delta u_{jmin}(i) - \varepsilon V^{\Delta u}_{jmin}(i) \leq \Delta u_j(k+i|k) \leq \Delta u_{jmax}(i) + \varepsilon V^{\Delta u}_{jmax}(i) \\ & y_{jmin}(i) - \varepsilon V^y_{jmin}(i) \leq y_j(k+i|k) \leq y_{jmax}(i) + \varepsilon V^y_{jmax}(i) \\ & , i = 0, ..., p-1 \\ & \Delta u(k+h|k) = 0,\ h = M, ..., P-1\ \varepsilon \geq 0 \end{aligned} \quad (2.4)$$

Here, $r(k)$ is the value of the reference variable at time k, $w^{\Delta u}_{i,j}$, $w^y_{i,j}$ are non-negative weights for the corresponding variables. A smaller w indicates a lower importance of the corresponding variable in the overall cost function. $u_{j,min}$, $u_{j,max}$, $\Delta u_{j,min}$, $\Delta u_{j,max}$, $y_{j,min}$, and $y_{j,max}$ are the lower/upper bounds of the corresponding variables. The weight ρ_ϵ of the variable ϵ penalizes the violation of constraints. The relaxation vectors V^u_{min}, V^u_{max}, $V^{\Delta u}_{min}$, $V^{\Delta u}_{max}$, V^y_{in}, and V^y_{max} represent the penalty for relaxing the corresponding constraints; the larger the V, the softer the constraint. If all bounds are infinite and the slack variables are removed, the problem can be solved analytically; else a Quadratic Programming (QP) solver is used. Since the output constraints are always soft, the QP problem is never infeasible [10]. Note that only $\Delta u(k|k)$ is actually used to compute $u(k)$. The remaining samples $\Delta u(k+i|k)$ are discarded and a new optimization problem based on $y_m(k+1)$ is solved the the next sampling step $k+1$.

Since the states $x(k)$ and $x_d(k)$ are not directly measurable, predictions are obtained from a state estimator. The estimates are computed from the measured output $y_m(k)$ by the linear state observer

$$\begin{bmatrix} \hat{x}(k|k) \\ \hat{x}_d(k|k) \end{bmatrix} = \begin{bmatrix} \hat{x}(k|k-1) \\ \hat{x}_d(k|k-1) \end{bmatrix} + G(y_m(k) - \hat{y}_m(k)) \tag{2.5}$$

$$\hat{y}_m(k) = C_m\hat{x}(k|k-1) + D_{vm}v(k) + D_{dm}\bar{C}\hat{x}_d(k|k-1) \tag{2.6}$$

The gain G is designed using Kalman filtering techniques [10]

2.4 Adaptive Computing Techniques

By making trade-off among performance, accuracy and power consumption, adaptive computing systems have flexibility to meet multiple goals in changing computing environments. A number of technique have been developed for both software [90] [55] [71] and hardware [4] [8] [33]. In this dissertation, control law of controller will determine how to adjust parameters in hardware stack and application stack in achieving of desired trade-off and adaptive computing techniques are employed as components of actuator which in every control period assign new value to those parameters based on the decision of controller.

2.4.1 Adaptive Software Techniques

Adaptive software techniques refer to either static or dynamic alteration of software parameters in response to change in the computing environment. We provide a brief review of adaptive software techniques reported in the literature.

Autotuning technique is used to explore a range of equally accurate implementation alternatives to find the alternative or combination of alternatives that deliver the best performance on the current computational platform. [38] For example, multicore stencil computations are optimized based on system parameters of computation platform such as cache size, number of socket, DRAM bandwidth, thread number, and compiler type [30]

Different from autotuning technique which searches among equally accurate alternatives, dynamic knobs technique [45], along with loop perforation [69] and task skipping [86] trades accuracy of computation for other benefits. In dynamic knobs, static application parameters assigned during program initialization are turned into tunable parameters which can be updated by controller through certain tuning interfaces. Loop perforation transforms loops to perform fewer iterations than the original loop in order to obtain implementations that occupy different points in the performance/accuracy trade-off space. Task skipping works similarly with loop perforation, except that it skips tasks instead of iterations in loops. The three techniques mentioned above sometimes equivalently affects performance and accuracy in the sense that some tunable parameters may affect the number of loop iterations.

2.4.2 Adaptive Hardware Techniques

Adaptive hardware techniques improve power efficiency of hardware by dynamically tuning its parameters during run-time execution to better match varying workload needs. Dynamic Voltage and Frequency Scaling (DVFS) [52] and Dynamic Power Management (DPM) [64] are among those most popular adaptive hardware technique. Advanced Configuration and Power Interface (ACPI) is a standard interface specification closely related to DVFS and DPM.

ACPI [44] is specified by several manufacturers including Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba to establish common interfaces for platform-independent configuration and power management. ACPI specifies solely the interface between hardware and software and the implied requirements of the two. The specification defines what have to be supported and initialised by the hardware and what assumptions can be made on the software side. In the ACPI specification, software is defined as Operating System-directed Power Management (OSPM) [35], an operating system component which is responsible for all power management decisions and actions using the interface defined by the specification. Based on extent of power consumption, ACPI defines four classes of power states as G-states (global states), C-states(processor states), P-states (performance

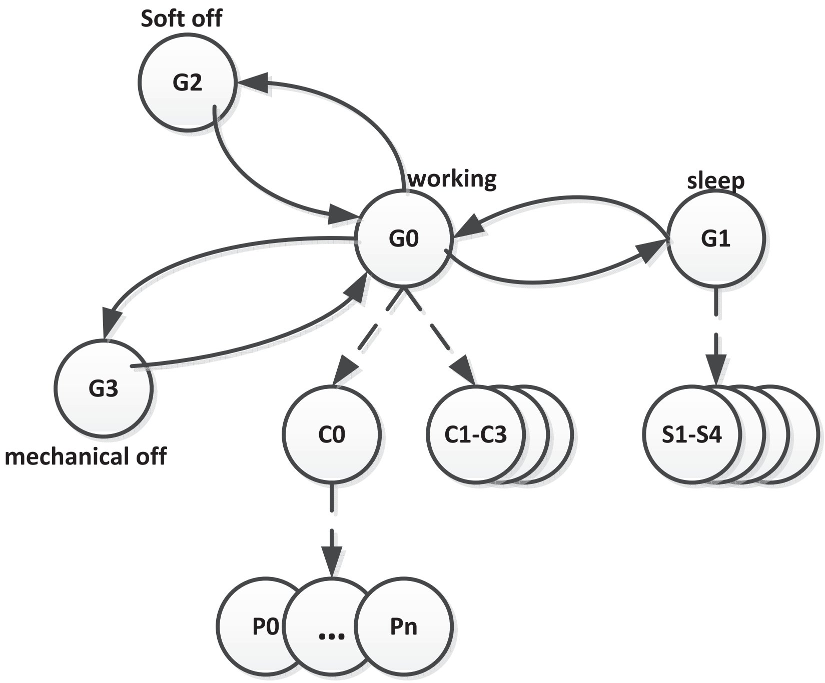


Figure 2.12: ACPI power states

states) and S-states (sleep states). G_0 to G_3 states can be designated in sequence as working state, sleep state, soft off state and mechanical off state. In G_0 state, processor can reside in a state between C_0 to C_3 state where C_0 is operating state and C_1 to C_3 are idle states in which some components of the processor stop working. P-states are a predefined set of frequency and voltage combinations at which the processor can operate when the CPU is under C_0 state. In P-states, P_0 is the highest performance state with highest clock frequency and P_n defines the lowest performance state with lowest clock frequency. DVFS technique dynamically adjusts clock frequency specified in P-states. Relations between different power states in shown in Fig 2.12. DVFS allows processor to switch from different P-states to trade performance for power conservation. Different from DVFS, DPM switches processor to idle or sleep states when they are not used, resulting in a power saving. The deeper the idle or sleep state a processor resides in, the longer the latency will be when switching it back to working state. Compared with state switching time in DPM, which is up to 100 milliseconds to transit from sleep to run [26], the latency in frequency transition in DVFS is significantly smaller [41]. For example, Enhanced Intel SpeedStep technology (EIST) [46] has a maximum transition latency as only 10 us. Since jobs are often released in a small period (e.g. below 40 milliseconds) in soft real-time system , switching processor frequently to idle or sleep states may incur great overhead which may easily cancel the benefit from doing so. Hence in this dissertation we choose DVFS over DPM as the method of adapting hardware for power saving.

Dynamic Cache Repartition (DCR) is another adaptive hardware technique for improving power efficiency. By partitioning the shared cache among tasks at run-time based on the characteristics of tasks (e.g. real-time constraint, private data size), DCR can be used to assure real-time performance while saving power consumption. Although DCR has attracted considerable interest in the research community [4] [99] [104] with a few reported prototypes up till now[4], it is not yet integrated on commerically available processors.

CHAPTER 3: HOMOGENEOUS CONTROL FRAMEWORK

3.1 Introduction

In this chapter, we propose a cross-stack control framework for homogeneous real-time workloads. We note that in many real-time applications, although deadlines need to be met to provide QoS guarantees, application quality (for example visual quality in video processing) can be tuned in conjunction with hardware and system software parameters to improve the controllability of the system. We formulate the real-time task execution as a Multiple-Inputs, Single-Output (MISO) optimal control problem involving tracking a desired task utilization set-point with control inputs derived from across the computing stack as shown in Fig 3.1. We assume that an arbitrary number of soft real-time tasks running in the application stack may join and leave the system at arbitrary times. The tasks are scheduled on multiple cores by global EDF scheduling algorithm in the real-time OS stack since all tasks are identical and belong to same type of workloads. Note that utilization above the set-point results in the task missing deadlines while utilization under the set-point results in sub-optimal power consumption.

As shown in figure 3.1. We use a model predictive controller (MPC) to realize optimal control. MPC uses an internal system model to predict the future trajectory of the controlled variable. This model is derived by carrying out System Identification (SI) based on data collected on our experimental platform. Based on a history of past control moves, a constrained optimization is solved on-line to determine the future input trajectory such that the controlled variable tracks a reference trajectory over a receding horizon. For every control period, the MPC reads system utilization from the sensor, calculates the control variables based on its control law, oversamples values of the control variables with proper

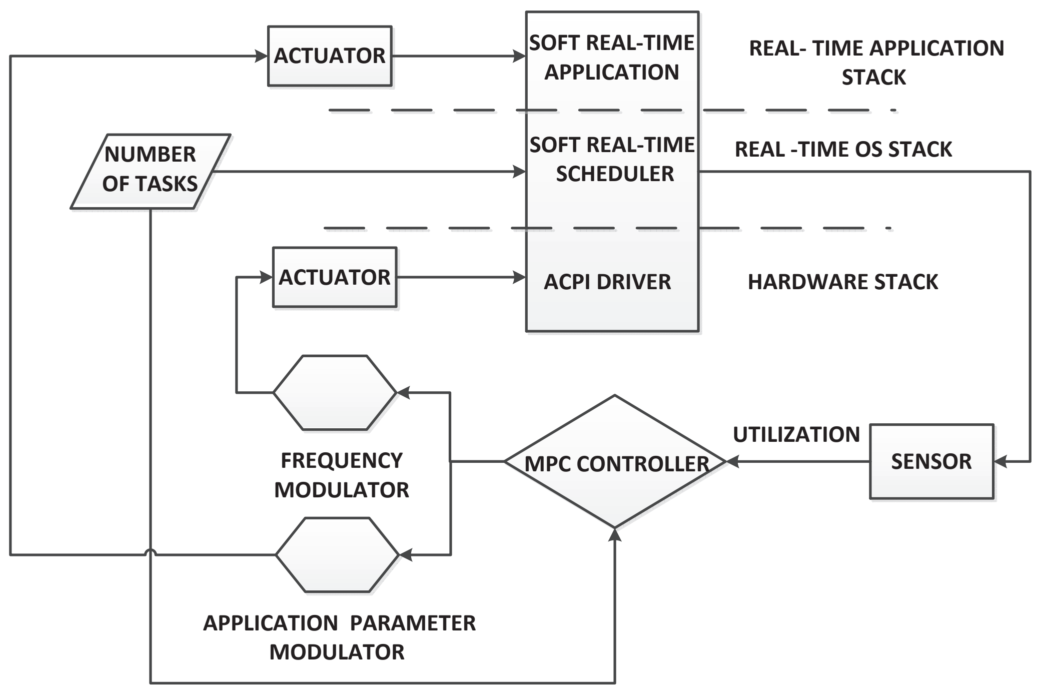


Figure 3.1: Schematic representation of our cross-stack control framework for homogeneous real-time workloads

modulators, and finally writes the oversampled values to application and hardware stack through actuators. As mentioned in subsection 2.4.2 we use DVFS technology to adapt operational frequency dynamically in the hardware stack. Application quality tuning knob is updated by writing through global variables protected by a FMLP read-write lock. Our results shows better system controllability can be achieved if the control inputs are derived from all parts of the computing stack. The controller is able to track utilization set-point in less than 5 seconds in response to a 50 % step change in the number of tasks. For a pseudo-random number of input tasks, our model predictive control approach shows an average power saving up to 31 % compared to a baseline implementation running at the highest frequency and application precision.

The rest of the chapter is organized as follows. In Section 3.2, we introduce various evaluation methodologies used to construct our control framework. In Section 3.3 we present the experimental results which show that better system controllability can be derived if all parts of the computing stack are collaboratively adapted. We review related work in Section 3.4 and conclude the chapter in Section 3.5.

3.2 Evaluation Methodology

In this section we will introduce our experimental platform , the benchmarks we use as soft real-time workloads, each component of our control framework including Global EDF scheduler, sensor and actuator, the metrics we use to evaluate performance of controller and the power model used to account for power consumption.

3.2.1 Benchmarks

We report our experiment results for two benchmarks, *x264* video encoder and *bodytrack*. Both the benchmarks are taken from PARSEC benchmark suite. The Princeton Application Repository for Shared-Memory Computers (PARSEC) [12] [11] is a benchmark suite created to encompass representative modern workloads for the emerging class of multicore computing systems. It is composed of 10 application and 3 kernel programs selected from different domains including multimedia, data mining, and computer games.

We provide a brief description of the *x264* video encoder and *bodytrack*, identify their application parameters for dynamical tuning, and quantify how application parameters affect application accuracy.

3.2.1.1 *x264* Encoder

The *x264* application is an H.264/AVC (Advanced Video Coding) video encoder [1]. It follows the ITU-T H.264 standard, which is now part of ISO/IEC MPEG-4. H.264 improve encoding quality with several new features such as increased sample bit depth precision, higher-resolution color information, variable block-size motion compensation (VBSMC) or context-adaptive binary arithmetic coding (CABAC). The flexibility of H.264 make it suitable for a wide range of contexts with different requirements. For example, next-generation HD DVD or Blu-ray video players already adopt H.264/AVC encoding as part of their standard.

Before introducing the H.264 video encoder, we describe key terminology important in video encoding:

- Block: A block holds the data of one color channel and is of size 8 by 8 pixel.
- Macroblock: A macroblock contains 16 by 16 pixels and consists of four blocks.
- Slice: A slice holds a number of macroblocks.
- Frame: A frame consists of slices.

The general process of encoding a video frame is shown in figure 3.2. To encode a video frame, each frame is divided into small blocks of pixel size 8 by 8. Each block is then transformed using the Discrete Cosine Transform (DCT). Finally the DCT coefficients are quantized, and encoded with a variable run length coding.

There is considerable redundancy in a typical video which is usually recorded at 25 frames per second. To exploit this redundancy for better encoding efficiency, motion estimation [36] is adopted. For each macroblock of a frame to be encoded, motion estimation searches similar macroblocks in one or two temporally neighbored reference frames. If such a macroblock is found, only the difference between the macroblocks needs to be en-

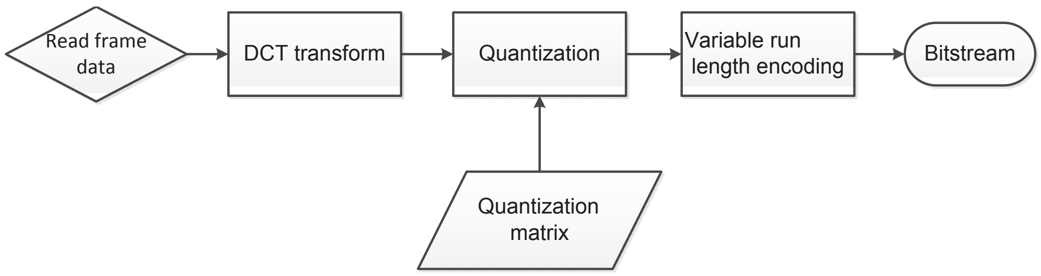


Figure 3.2: General flow of encoding a video frame

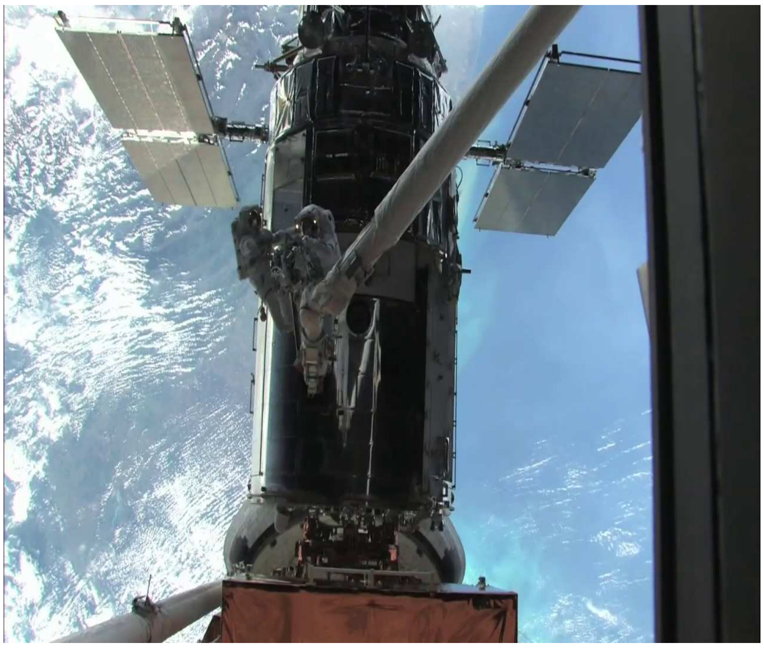
coded. Based on the frame dependency during encoding, video frames can be divided into three class:

- I-frames (Intra Coding): These frames are independent of other frames. They contain the entire image information and do not use any reference frames.
- P-frames (Predictive Coding): These frames contain only the changed part of an image from previous I- or P-frame. The macroblocks of this type of frame can be encoded using a previous I-frame or P-frame as a reference frame for motion estimation.
- B-frames (Bidirectional Coding): The macroblocks of these frame type can use a previous and a following I-frame or P-frame as reference frames for motion estimation. B-frames cannot be used as a reference frame and can be compressed much more than other frame types.

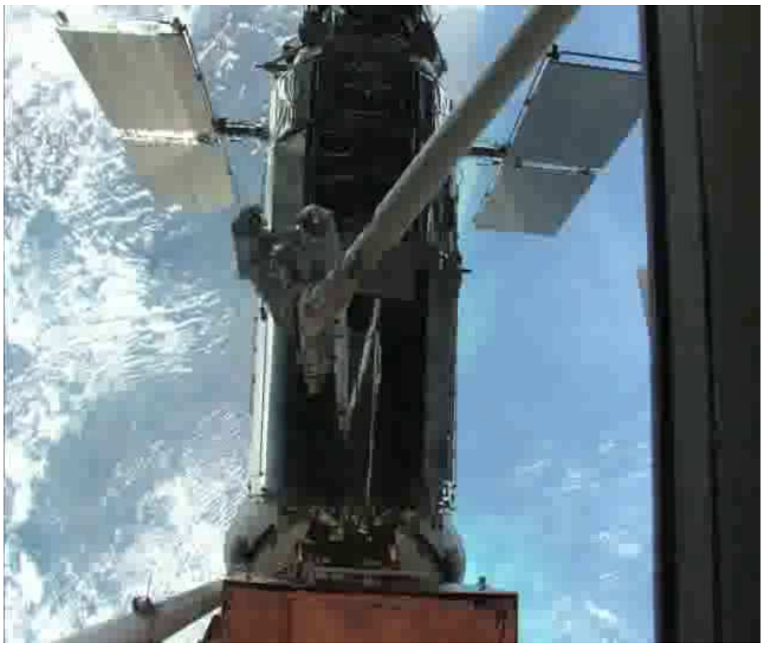
In this dissertation, each video encoding task is mapped to a single thread and is independent of other encoding tasks. The encoder grabs video frames periodically at 25 fps and encodes based on the MP4 video format specification. The video frame resolution level ranges from $\frac{1}{4}$ HD (230,400 Pixels per image) to full HD (921,600 Pixels per image) and and is chosen as the application quality tuning knob determining the visual quality. Fig 3.3 shows a snapshot image of the encoded video in full HD and $\frac{1}{4}$ HD. It is assumed that visual quality is linear with number of pixels per image.

3.2.1.2 *bodytrack*

The *bodytrack* computer vision application is an Intel RMS workload which tracks a



(a) HD



(b) $\frac{1}{4}$ HD

Figure 3.3: Snapshots of encoded video entitled ”Hubble: 20 Years of Discovery” in full HD and $\frac{1}{4}$ HD resolution

humans movement through an image sequence with multiple cameras [7] [31]. An annealed particle filter is used to track the movement of a human through the scene. The graphic output of *bodytrack* generates conic cylinders to represent 10 body components including torso, head and limbs as shown in Fig 3.4. *bodytrack* is widely used in areas such as video surveillance, and character animation.

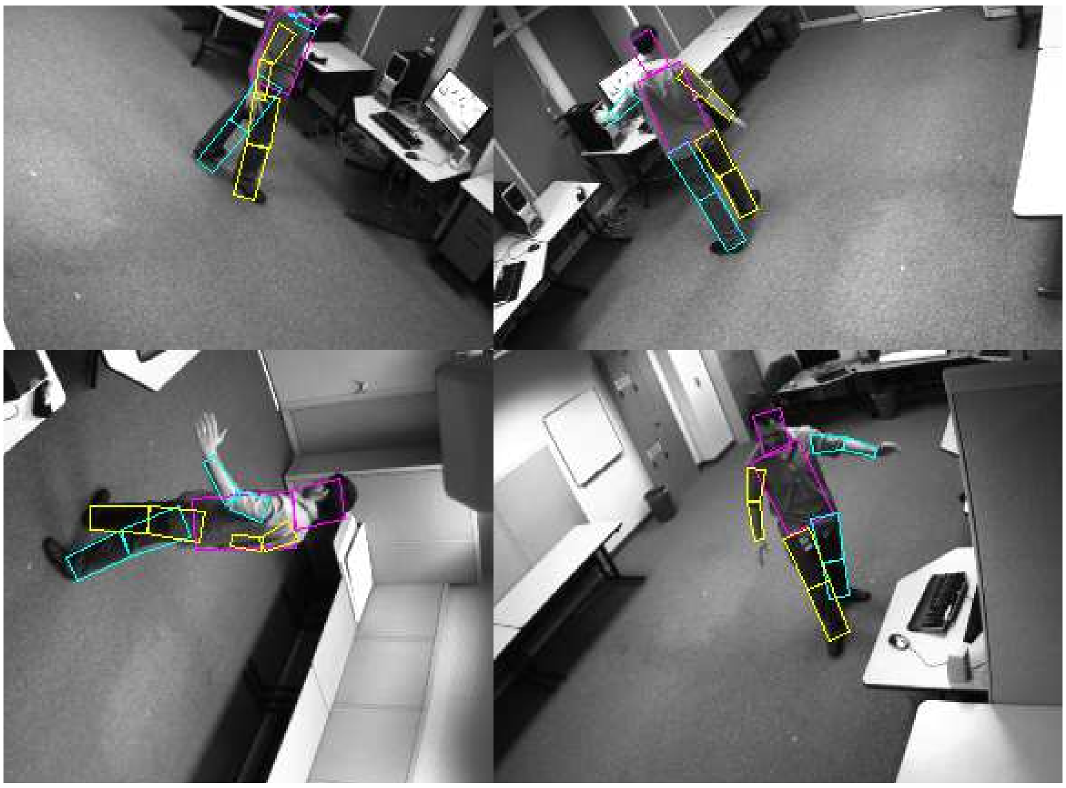


Figure 3.4: Output of *bodytrack* on the third image in PARSEC's inputs

The number of annealing layers ranging from 1- 5, and the number of particles ranging from 100 - 4000, are chosen as the application quality tuning knobs. As a measure of visual quality, the relative mean square error in the magnitude of the position vectors of the body parts for different values of the tuning knobs is used [45].

In this dissertation, we assume that each *bodytrack* task executes at 20 fps. Due to non-linearity between visual quality and values of tuning knob, we have to manually choose combinations of application quality tuning knobs as discrete steps for run time manipulation. For all the chosen combinations of application quality tuning knobs, the relative mean square error is less than 56%. To set visual quality between two steps, actuator use a Pulse Width Modulator (PWM) which will be described in Section 3.2.3.2.

3.2.2 Implementation of G-EDF Scheduling Algorithm

Since all tasks are identical in the case of homogeneous workloads, we use G-EDF algorithm for task scheduling. $LITMUS^{RT}$ project [23] [15] implements several different real-time scheduling plugins including G-EDF on top of original Linux kernel and we use their implementation as a component of our control framework. In this subsection we introduce how G-EDF scheduling algorithm is implemented in $LITMUS^{RT}$

As shown in Fig 3.5, G-EDF usees a single unified task scheduler, ready queue and release queue to handle all tasks. At any instant, the task scheduler picks at most M jobs with the earliest deadlines to execute on M processors. There is no restriction imposed on where a task can execute. Ready queue and release queue are mechanisms which store and order ready jobs and jobs for future time-based releases. When a job is released, it must be transferred from release queue to ready queue and the task scheduler will be triggered to check if a preemption will happen. A ready or release queue is a priority queue which can be implemented with different data structures such linked list, heap or tree. To reduce worst case overhead when multiple jobs release at the same time, $LITMUS^{RT}$ uses a binomial heap [16] to implement the ready and the release queue. Compared with other data structures, the benefit of binomial heap is that it can add multiple elements efficiently. So when k jobs are released at the same time, it only takes $O(logn)$ time units (n is number of tasks in ready queue) for a ready queue implemented in binomial heap to insert them.

Fig 3.6 shows the relationship between core functions of G-EDF scheduling algorithm. According to Fig 3.6, scheduling process can be triggered in three scenarios. The first scenario is when one job finishes its execution. Then a kernel function $job\_completion$ will be called, which removes the task from the processor core by calling function $unlink$, puts it back to the release queue by calling function $requeue$ and checks preemption by calling function $check\_for\_preemption$. During each preemption, At most M jobs with

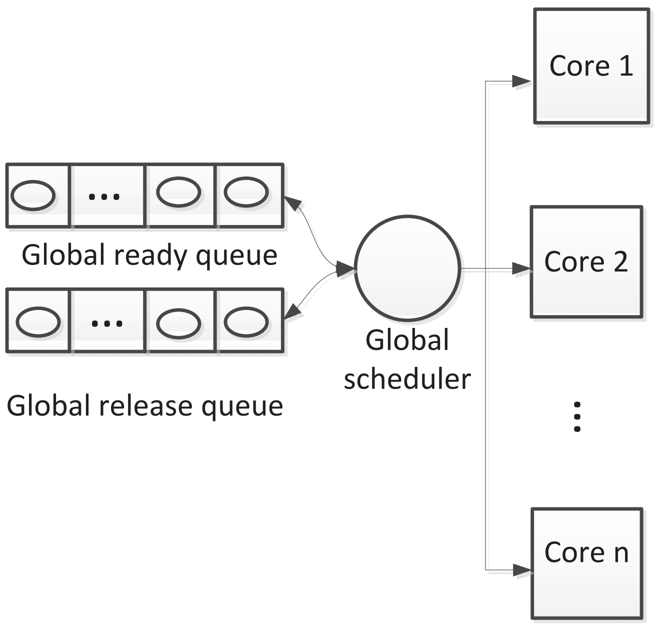


Figure 3.5: Schematic representation of G-EDF algorithm

lower priorities will be put back to the release queue and replaced with M jobs with the highest priorities in the ready queue. In the second case, local timer interrupt handler will check if any job needs to be released based on their periods. If so, they will be moved from release queue to ready queue by calling function EDF_task_wakeup and followed with another preemption checking. The third case is when new tasks arrive, which triggers function EDF_task_new to take care of requeue and preemption.

3.2.3 Actuator

3.2.3.1 Overview

In our control framework, actuators are used to update the processor operational frequency and the application quality tuning knob values. Prior to applying the manipulated variables to the hardware and the application stack, the actuators filter these through a modulator to allow for fine-grained control. We use a first order delta-sigma modulator for the frequency actuator and a pulse width modulator for the application tuning knob actuator. Compared to the pulse width modulator, the first order delta-sigma modulator

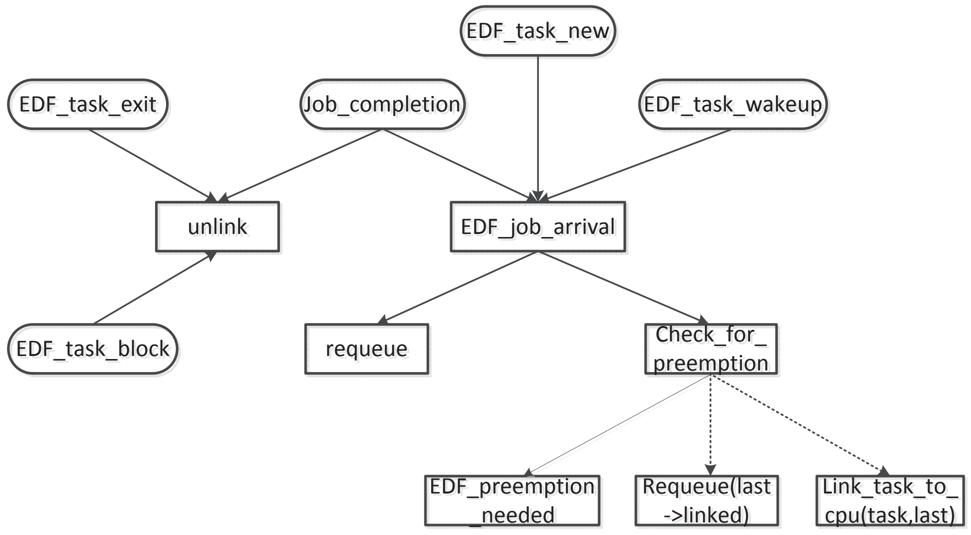


Figure 3.6: Relationship between core functions of G-EDF scheduling algorithm

provides higher accuracy but incurs larger overhead due to oversampling. Hence first order delta-sigma modulators are more suitable for frequency actuators due to the small transition latency for DVFS (10 μs). However, the latency associated with the application tuning knobs may be up to 500 μs, precluding the use of oversampled techniques.

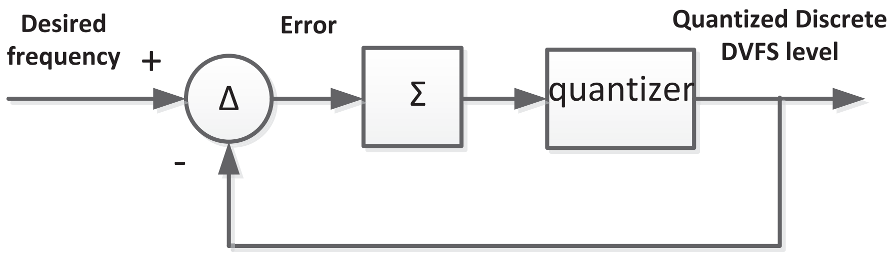


Figure 3.7: Block diagram of first order delta-sigma modulator consisting of a difference stage, an discrete time integrator and a quantizer

3.2.3.2 Modulator

First order delta-sigma modulator: First order delta-sigma modulator approximates a desired value by oversampling it into a series of predefined discrete values at a certain

OverSampling Ratio (OSR) [27]. For example, to approximate 2.86 GHz during a control period, the modulator would output the sequence, 2.67, 3, 3, 2.67, 3, at a $OSR = 5$. As shown in Fig 3.7, the first order delta-sigma modulator employs a feedback loop to calculate the error between the instantaneous input and previous quantized output (hence denoted as delta) [53]. This error is then accumulated by a discrete-time integrator (denoted as sigma). The sum of errors is finally quantized to produce an oversampled output. Since delta-sigma modulators can sharply differentiate input signal and quantized output, it is widely used in high resolution data conversion system. We follow this strategy to implement our first order delta-sigma modulator. Note that oversampled output frequency f_o is quantized based on the inequality below:

$$f_o = \begin{cases} f_L, & \text{if } \sum error > \frac{f_H - f_L}{2} \\ f_H, & \text{if } \sum error \leq \frac{f_H - f_L}{2} \end{cases} \tag{3.1}$$

Here f_L and f_H are two neighbor discrete frequency values which the desired frequency value falls in between.

Pulse Width Modulator (PWM): Pulse width modulator is a simple method to approximates a desired value by modulating duty cycle in each control period [100]. Compared with first order delta-sigma modulator, it doesn't require feedback and only switch its output once during a control period. Suppose that q, is the desired application quality level and q_H and q_L are two neighbor application quality level near it. The duty cycle D equals to:

$$D = \frac{q_H - q}{q_H - q_L} \tag{3.2}$$

If n jobs are executed with in one control period and application quality may be changed at beginning of each job, after n_1 jobs application quality should be switched to q_L

$$n_1 = \lceil D * n \rceil \tag{3.3}$$

3.2.3.3 Frequency Actuator

Frequency actuator utilizes a Linux kernel subsystem called *cpufreq* to dynamically scale value of operational frequency oversampled by the first order delta-sigma modulator.

Cpufreq [79] is a kernel module which has been incorporated to Linux kernel since the kernel version 2.6.0. Most mainstream CPU manufacturers support dynamic frequency scaling, such as Intel Enhanced SpeedStep and AMD PowerNow. However, each manufacturer has their unique way of implementation dealing with underlying APCI interfaces in achieving frequency scaling. *Cpufreq* serves as an uniform software interface which provides functionality of dynamic frequency scaling to users no matter which manufacturer′s technique is actually used.

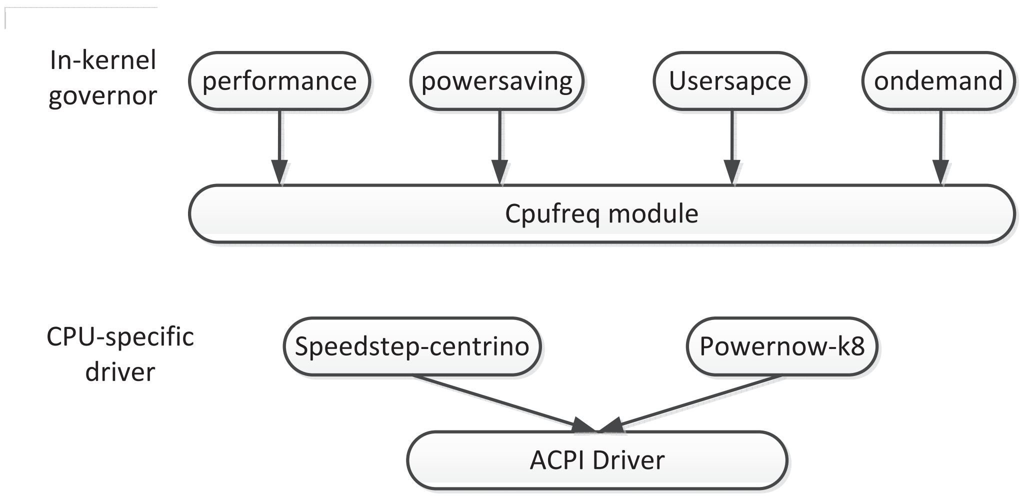


Figure 3.8: Software architecture of cpufreq

As shown in Fig 3.8, *cpufreq* consists of three parts: CPU specific drivers , *cpufreq* module and in-kernel governors. CPU specific drivers incorporate CPU driver code subject to different implementation techniques such as Intel Enhanced SpeedStep and AMD PowerNow. Cpufreq module extracts and encapsulates upper governor policy from underlying implementation techniques. In-kernel governors are used to implement different frequency scaling policies. Governors available in the current Linux kernel are listed below:

- cpufreq_performance: Runs the CPU at maximum clock speed.

- cpufreq_ondemand: Dynamically switches between the CPU available clock speeds based on system load.
- cpufreq_powersave: Runs the CPU at minimum speed.
- cpufreq_userspace: Users can select desired frequency.

We choose cpufreq_userspace as the governor so as to set CPU frequencies according to output of the MPC controller. To set a new frequency value at run time, write it into the system file /sys/devices/system/cpu[i]/cpufreq/scaling_setspeed, where [i] is the core index.

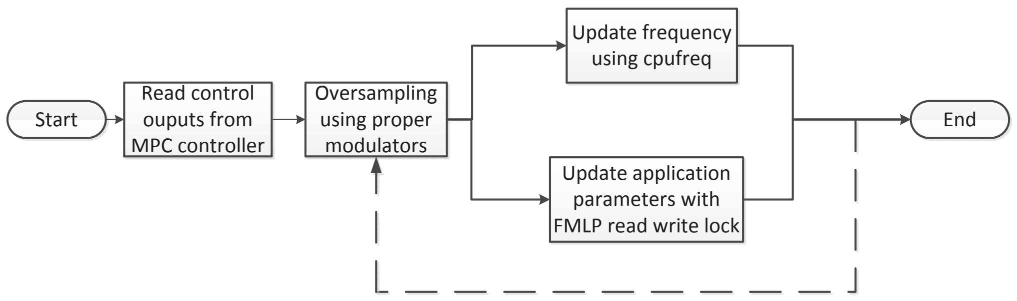

Start
Read control ouputs from MPC controller
Oversampling using proper modulators
Update frequency using cpufreq
Update application parameters with FMLP read write lock
End

Figure 3.9: Flow chart of actuator

3.2.3.4 Application Tuning Knob Actuator

Application tuning knob actuator updates application quality calculated by the MPC controller through global variables protected by a FMLP read-write lock introduced in 2.2.5. The read-write lock is used because it is very likely that there will be more reader processes (task process) than writer process (controller process) under our control framework. FMLP is used to prevent deadlock and priority inversion in multiprocessor systems. For each control period, application tuning knob actuator uses a pulse width modulator to approximate the desired application quality.

3.2.3.5 Flow Chart

Fig 3.9 shows the flow chart of the actuator. For each control period, the actuator will read the desired frequency and application quality from the MPC controller, generate frequency and application tuning knob level using the modulator, and writes these to the hardware and the application stack.

3.2.4 Sensor

For every control period, the MPC controller reads system utilization from a sensor. To calculates system utilization, the sensor reads average per-core execution time over one control period by calling a specially designed system call, and divides it by the control period. Before explaining how to implement the sensor, we first introduce implementation of a system call in Linux and Time management in Linux.

3.2.4.1 Implementation of Linux System Call

Due to considerations of security and stability, functions in user space cannot invoke kernel functions directly because they exist in different memory space. Instead, system call serves as a communication layer between kernel and user-space application. In order to access functions in kernel space, invocation of system call will signal the kernel that the system needs to be switched to kernel mode. The mechanism to signal the kernel is software interrupt which incurs an exception. Then system executes the exception handler, which triggers the execution of exception vector and a switch to kernel mode. In X86 architecture, this exception handler is called *system_call( )* and implemented in assembly in *entry_64.S*.

To add a new system call to the Linux kernel, a system call number needs to be allocated to the newly added system call, followed by the definition of a system call kernel function with an API function to wrap the kernel function.

A system call number is an unique number that is used to reference a specific system call. A system call number can be assigned to a system call by adding two lines of code similar to the one below in file $< asm/unistd.h >$.

```
#define __NR_sensor 200
__SYSCALL(__NR_sensor, sys_sensor)
```

Here we define the system call number of system call *sensor (long * utilizaiton)* to be 200.

utilizaiton is a pointer pointing to a long integer with same name to store overall utilization over one control period. The system call number cannot be changed once it is assigned and it cannot be recycled even if the system call is removed.

A system call kernel function can be defined with zero or more arguments and return a long integer signifying success or error. A negative return value denotes an error and a return value of zero represents success. We define our system call system call *sensor (long *utilization)* in the form below:

```
asmlinkage long sys_sensor (long *utilization)
```

Here the asmlinkage modifier is a directive which tells the compiler to seek function's argument only on the stack. Another point to note is that system call *sensor (long *utilization)* is defined as *sys_sensor (long * utilization)* in the kernel, which is a naming convention followed by all the system call in Linux.

Usually, applications access APIs implemented in user-space instead of accessing system calls directly. This greatly enhances portability of application since the same API can exist on multiple systems and provide the same interface to applications while the implementation of the API itself can differ from system to system. One of most commonly used API is POSIX standard [73]. To add an API function for our system call *sensor (long * utilization)*, we use the codes shown below:

```
long sensor (long *utilization)
{return syscall(__NR_sensor, utilization);}
```

3.2.4.2 Time Management in Linux

The Linux kernel has to the system hardware called system timer to keep track of time. The system timer is actuated periodically by an electronic time source, such as a digital clock or the frequency of the processor. This time period is called tick and its reciprocal is called tick rate which represents how many ticks pass during one second. Whenever the system timer goes off, it issues an interrupt which signals the kernel to respond via executing an interrupt handler service. The kernel only needs to know the number of ticks and the corresponding tick rate to calculate the time interval between any two timer interrupts.

Several different system timers are available in the X86 architecture including Programmable Interrupt Timer (PIT), local APIC timer, and Time Stamp Counter timer (TSC) [28]. TSC is the most commonly-used system timer among them, present on all X86 processors since the Pentium. The benefit of TSC lies in that it can acquire system time information in high resolution(nanosecond units) with very low overhead. Thus in our work TSC is used to keep track of execution time of each job. To do this, a long integer variable *exec_time* is defined in job parameters structure which is updated by referring to TSC whenever the job is released, preempted, and finished. We then read the aggregation of all the jobs execution time by calling a specially designed system call at the end of each control period.

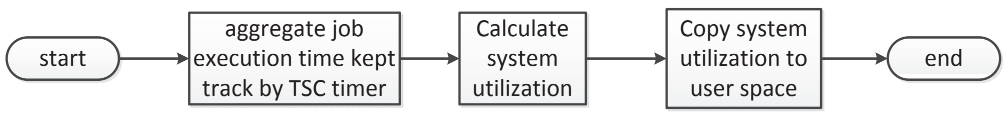

start
aggregate job execution time kept track by TSC timer
Calculate system utilization
Copy system utilization to user space
end

Figure 3.10: Flow chart of sensor

3.2.4.3 Flow Chart

Fig 3.10 shows the flow chart of the sensor implementation. When the system call *sensor (long *utilization)* is triggered, it first derives the execution time of all the jobs over one control period by accumulating completed job execution times on all the cores. Then overall utilization is then calculated by dividing the execution time by the control period in nanoseconds. Finally, the overall utilization is copied back to user space using kernel function *copy_to_user*.

3.2.5 Performance Metrics of the Controller

Several metrics are used throughout this dissertation to quantitatively evaluate our controller design. These standard performance metrics are usually defined in terms of the step response of a system as shown in Fig 3.11.

- Rise time T_r: Rise time is used to measure the swiftness of the response and is defined as the time taken by a signal to change from 10% and 90% of the step height.
- percent overhoot: The percent overshoot is defined as

$$P.O. = \frac{M_{Pt} - f_v}{f_v} \times 100\% \tag{3.4}$$

  Where M_{Pt} is the peak value of the time response, and f_v is the final value of the response.
- settling time T_s: The settling time is defined as the time required for the system to settle within a certain percentage δ of the set-point amplitude. In our measurement, δ is set to 5 percent.
- steady state error: The difference between final value and set-point.

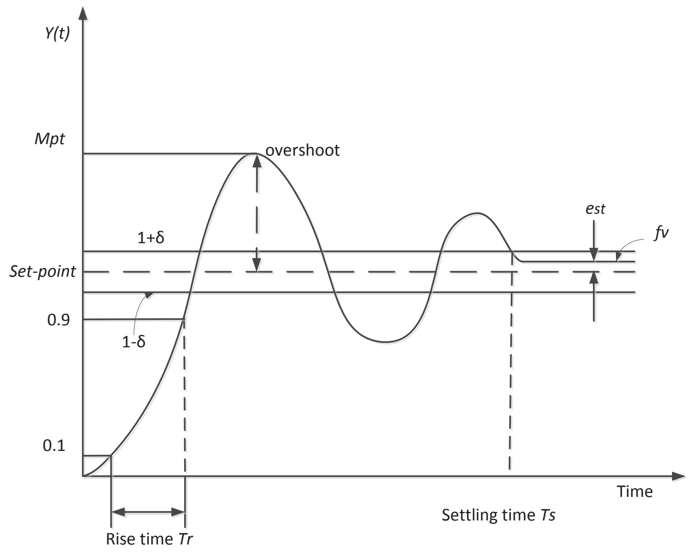


Figure 3.11: Performance metrics of a control system

3.2.6 Power Model

Our experimental machine contains two quadcore Intel Xeon X5365 processor sharing a 16 GB main memory. As shown in equation 3.5, the total power consumption of each processor consists of its dynamic power $p_{dynamic}$ and its static power p_{static}.

$$P_{total} = p_{dynamic} + p_{static} \tag{3.5}$$

$P_{dynamic}$ can be expressed in terms of the operating voltage V_{dd}, the operational frequency f, and the switching capacity c_l as follows:

$$P_{dynamic} = c_l.V_{dd}^2.f \tag{3.6}$$

P_{static} is caused by leakage current [88], which flows even while no instructions are being executed. Expression of P_{static} is shown in equation 3.7, where I_{leak} is leakage current.

$$P_{static} = I_{leak}.V_{dd} \tag{3.7}$$

As we are interested in profiling the power consumption during the control phase, an analytical model is needed to account for power components of both $p_{dynamic}$ and p_{static}. It should be noted that $p_{dynamic}$ can be modeled as proportional to the cube of operational frequency f since voltage can be approximated as linear of frequency [67]. Based on this assumption, Fu *et al.* [40] proposed a power consumption model considering both dynamic power and static power for Intel Xeon X5365 quad-core processor as shown below:

$$P_{total} = 95 \times f_r^3 + 25 \tag{3.8}$$

Notice that here f_r is relative frequency which is normalized to the highest processor frequency of 3 GHz. DVFS technology allows users to switch from several discrete power states of processor during run-time by change the operational frequency f. Intel Xeon X5365 processor has 4 different power states with frequency at 2 GHz, 2.33 GHz, 2.66

Table 3.1: Power consumption of Intel Xeon X5365 processor under different operational frequencies

frequency f(GHz)	power consumption (watt)
2	53
2.33	69
2.67	91
3	120

GHz and 3 GHz. The corresponding power dissipation of states are calculated using equation 3.8 and are in Table 3.1.

3.3 Experimental Results

In this section, we derive system models by carrying out system identification and analyze their stability. We then use the models to create MPC controller, design the controller by optimizing its parameters, test the controller's step response and evaluate controllers' capability in power saving and measure the associated overhead.

3.3.1 Experimental Setup

We experimentally demonstrate the operation of our cross-stack predictive control framework for homogeneous soft real-time workload on a dual socket quad-core Intel Clovertown server. This server is equipped with Intel Xeon processor X5365 with 8MB on-die L2 cache 1.333 GHz FSB and a 16GB main memory. The processor supports four DVFS level: 3.0 GHz, 2.67 GHz, 2.33 GHz and 2.0 GHz. The operating system is Linux 2.6.36 kernel patched with Litmus-RT. The soft real-time workloads considered in this section are soft real-time applications from video processing and machine vision. Each soft real-time task is mapped to a single thread and is independent of other tasks. In our work, *x264* encoder grabs video frames periodically at 25 fps and *bodytrack* processes a new frame at 20 fps

3.3.2 System Identification

We carried out system identification using *n4sid* algorithm from MATLAB system identification Toolbox. For each application, we obtain the utilization for randomly generated combination of inputs for 400 periods. We use the first half of working data for data modeling and the other half of data for validation. We apply *n4sid* system identification algorithm to generate state space models given in Equation 2.1 with order of one for both *x264* and *bodytrack*. For our model, $N_x = 1$, $N_u = 2$(frequency and application quality), $N_v = 1$ (number of tasks), $N_d = 1$ (job level variations in the execution time), and $N_o = 1$ (system utilization). Fig 3.12 shows the plot of model validation where the measured data superposed over the predicted data. Validation results show the model fit is 84.8% for *x264* and 87.4% for *bodytrack*. Table 3.2 shows the values of the coefficient matrices A, B, and C.

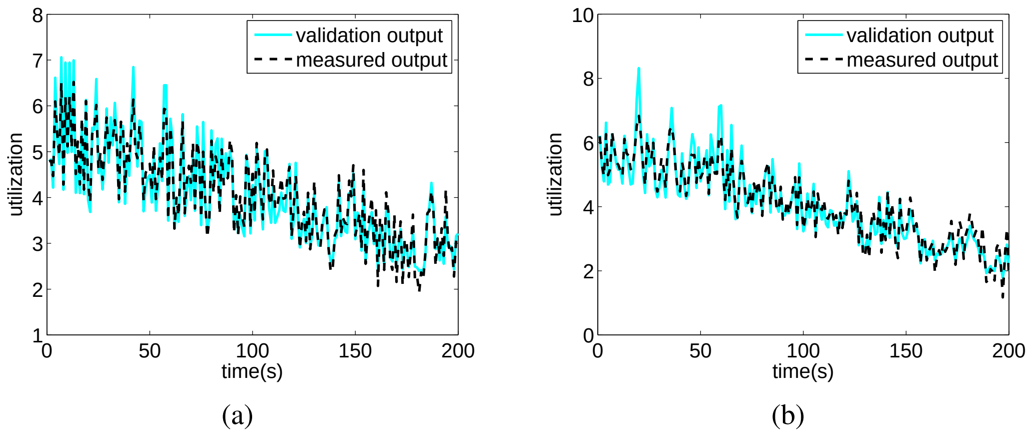


Figure 3.12: Model validation for (a) *x264* and (b) *bodytrack*. The model fit is 84.8% for *x264* and 87.4% for *bodytrack*.

3.3.3 Stability Analysis

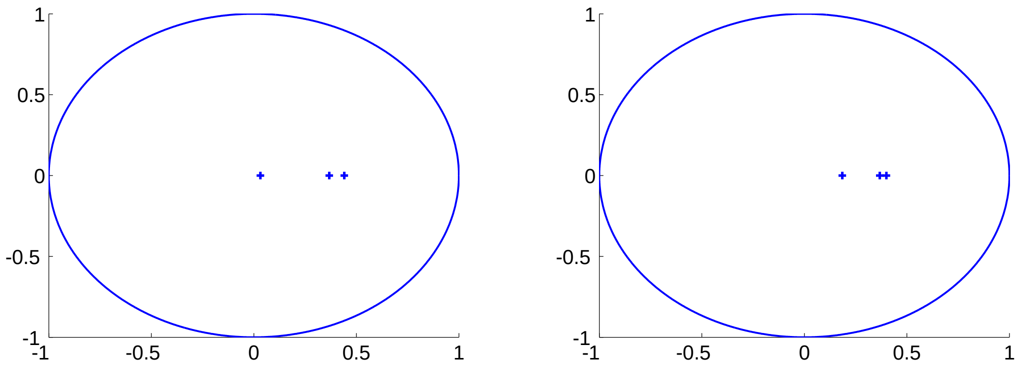
System stability is directly related to the location of closed loop poles. In a discrete system of unconstrained MPC controller, if all poles are located inside the unit circle in the complex space, the controller system is stable. As shown in Fig 3.13, Both *x264* and *bodytrack* have three closed loop poles of the unconstrained MPC controller lying within unit circle, indicating good stability.

3.3.4 Controller Design

In order to achieve good performance, the Model Predictive Controller is designed using the MATLAB MPC Toolbox to optimize controller parameters. Common tunable parameters of the MPC controller include control horizon, prediction horizon, input and

Table 3.2: A,B,C COEFFICIENT MATRICES of *x264* and *bodytrack*

	x264	*bodytrack*
A	0.0441	0.199
B	$\begin{bmatrix} -0.133 & 0.24 & 0.074 & -0.014 \end{bmatrix}$	$\begin{bmatrix} -0.12 & 0.23 & 0.041 & 74e-5 \end{bmatrix}$
C	0.65	0.84



(a) (b)

Figure 3.13: A stable unconstrained MPC controller is indicated by poles (blue plus sign) within the unit circle for (a) *x264* and (b) *bodytrack*.

output weights, blocking mode and disturbance model. We tune these parameters based on performances of controller's step response in terms of the metrics introduced in Subsection 3.2.5. Only one parameter is optimized at a time with other parameters at their default settings shown in Table 3.3. We take *x264* as example to show this work flow.

Table 3.3: Default settings for parameters of MPC controller

control horizon	2
prediction horizon	10
input weight	0,0
output weight	1
blocking mode	non-blocking
disturbance model	$\frac{1}{s}$

Control horizon: As shown in Table 3.4 and Table 3.5, by varying control horizon from 2 to 10 with an increment of 2, we check controller's responses to step change in the number of tasks and step changes of the set-point. We observe that the control horizon has minimal effect on the controller performance.

Table 3.4: Step response to changes on number of task from 8 to 12 for *x264* under different control horizons. Set-point is set as 4.

control horizon	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
2	0.83	3.54	8.4	0
4	0.83	3.54	8.4	0
6	0.83	3.54	8.4	0
8	0.83	3.54	8.4	0
10	0.83	3.54	8.4	0

Table 3.5: Step response to changes of set-point from 4 to 4.8 with number of task at 10 for *x264* under different control horizons. number of task is set as 10.

control horizon	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
2	1.48	3.51	0	0
4	1.49	3.6	0	0
6	1.49	3.6	0	0
8	1.49	3.6	0	0
10	1.49	3.6	0	0

Prediction horizon: We then check the impact of prediction horizon on the performance of the controller as shown in Table 3.6 and Table 3.7. Similar to the control horizon, the prediction horizon has minimal effect on the controller performance.

Table 3.6: Step response to changes in the number of tasks from 8 to 12 for *x264* under different prediction horizons. Set-point is set as 4.

prediction horizon	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
5	0.83	3.52	8.4	0
10	0.84	3.52	8.4	0
15	0.84	3.51	8.4	0
20	0.84	3.51	8.4	0
25	0.84	3.51	8.4	0

Blocking: By default the controller optimizes the first N_c moves of the prediction horizon, after which the manipulated variable remains constant for rest of the prediction horizon as shown in Fig 2.11. Alternatively, N_c planned moves can be distributed evenly along the prediction horizon. The time slice during which the manipulated variables are kept constant

Table 3.7: Step response to change of set-point from 4 to 4.8 for *x264* under different prediction horizons. Number of tasks is set as 10.

prediction horizon	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
5	1.48	3.54	0	0
10	1.48	3.51	0	0
15	1.47	3.50	0	0
20	1.47	3.50	0	0
25	1.47	3.50	0	0

is called a block. As shown in Table 3.8 and Table 3.9, by varying block length, we check the impact of blocking on the controller's step responses. It can be observed that increasing the block length significantly improves the performance of controller. The overshoot reduces up to 71 percent compared with non-blocking mode. We set blocking length as 5.

Table 3.8: Step response to changes on number of task from 8 to 12 for *x264* under different blocking length. Set-point is set as 4.

blocking length	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
1	0.88	3.54	8.4	0
2	0.83	2.96	5.4	0
3	0.61	2.89	3.8	0
4	0.60	2.85	2.9	0
5	0.60	2.82	2.4	0

Table 3.9: Step response to changes of set-point from 4 to 4.8 for *x264* under different blocking lengths. Number of tasks is set as 10.

blocking length	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
1	1.49	3.58	0	0
2	1.19	2.97	0	0
3	0.99	2.91	0	0
4	0.91	2.90	0	0
5	0.85	2.87	0	0

Input and output weights: The output weights let you dictate the accuracy with which each output must track its set-point. Specifically, the controller predicts deviations for each output over the prediction horizon. It multiplies each deviation by the output's weight value, and then computes the weighted sum of squared deviations, $S_y(k)$, as follows,

$$S_y(k) = \sum_{i=1}^{P} \sum_{j=1}^{n_y} \{w_j^y [r_j(k+i) - y_j(k+i)]\}^2 \tag{3.9}$$

Where k is the current sampling interval, $k+i$ is a future sampling interval (within the prediction horizon), P is the number of control intervals in the prediction horizon, w_j^y is the output weight, and the term $r_j(k+i) - y_j(k+i)$ is a predicted deviation for output j at interval $k+1$.

If a particular weight is large, deviations for that output dominate $S_y(k)$. One of the controller's objectives is to minimize $S_y(k)$. Thus, a large weight on a particular output causes the controller to minimize deviations in that output (relative to outputs having smaller weights).

The controller also minimizes the weighted sum of manipulated variable deviations from their nominal values, computed according to,

$$S_u(k) = \sum_{i=1}^{M} \sum_{j=1}^{n_{mv}} \{w_j^u [u_j(k+i) - \bar{u}_j(k+i)]\}^2 \tag{3.10}$$

Where w_j^u is the input weight and $\bar{u}_j(k+i)$ is the nominal value for input j. Since tracking manipulated variable to their nominal values is not require for our control problem, input weights use their default value 0.

We compare the performance of controller under different settings of output weights. Table 3.10 and Table 3.11 show that controller with default setting of input and output weight yields the best performance, which means the input weight and output weight should stick to 0 and 1 respectively.

Disturbance model: The disturbance model is obtained by low-pass filtering a Gaussian white noise. An aspect of controller design to determine the filter's cut-off angular

Table 3.10: Step response to changes in number of task from 8 to 12 for *x264* for different input and output weights. Set-point is set as 4.

input weights	output weight	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
0,0	1	0.83	3.54	8.4	0
0,0	0.8	0.82	2.58	14.7	0.4
0,0	0.6	0.81	2.48	20.0	0.6
0,0	0.4	0.81	2.45	22.8	0.7
0,0	0.2	0.79	2.40	25.4	0.9

Table 3.11: Step response to changes of set-point from 4 to 4.8 for *x264* for different input and output weights. Number of tasks is set as 10.

input weights	output weight	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
0,0	1	1.48	3.51	0	0
0,0	0.8	1.23	2.98	0	0.6
0,0	0.6	0.88	2.96	0	0.7
0,0	0.4	0.87	2.93	0	0.9
0,0	0.2	0.85	2.87	0	1.0

frequency. We select an appropriate cut-off angular frequency based on simulation. Results in Table 3.12 and Table 3.13 shows that cut-off angular frequency minimally effects the rise time, settling time and overshoot. However a cut-off angular frequency less than 1 rad/s will generate steady state error. Therefore, the cut-off angular frequency is set to 1 rad/s to avoid steady state error. The disturbance model thus is $\frac{1}{1+s}$.

Table 3.12: Step response to changes in the number of task from 8 to 12 for *x264* for different bandwidths of the disturbance model. Set-point is set as 4.

cut-off angular frequency	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
0	0.87	3.59	8.2	-0.2
1	0.85	3.63	8.21	0
10	0.85	3.58	8.15	0
100	0.86	3.59	8.23	0
1000	0.86	3.60	8.19	0

We follow a similar procedure to identify controller parameters for *bodytrack*. The

Table 3.13: Step response to changes in set-point from 4 to 4.8 for *x264* for different bandwidths of the disturbance model. Number of tasks is set as 10.

cut-off angular frequency	rise time (sec)	settling time (sec)	overshoot (percent)	steady state error
0	1.46	2.58	0	-0.2
1	1.51	2.62	0	0
10	1.50	2.63	0	0
100	1.47	2.57	0	0
1000	1.49	2.60	0	0

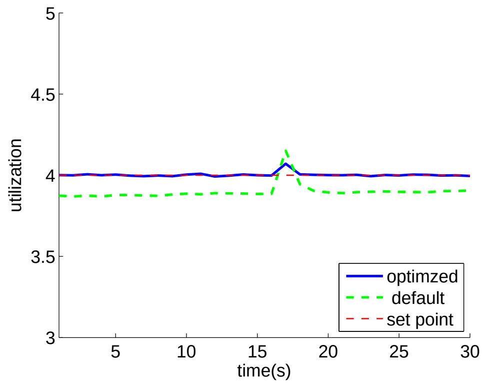
Table 3.14: Optimized parameter settings of the MPC controller

	x264	bodytrack
control horizon	2	4
prediction horizon	10	12
input weight	0, 0	0, 0
output weight	1	1
blocking	5	3
disturbance model	$\frac{1}{s+1}$	$\frac{1}{s+10}$

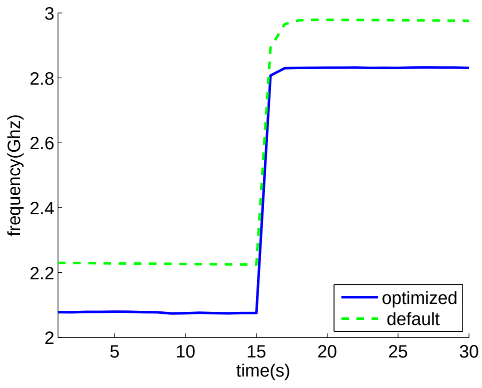
optimized controller parameters for both *x264* and *bodytrack* are shown in Table 3.14.

To evaluate the effect of the above controller optimization we compare the performance of the controller under default parameters setting to the optimized parameters. In Fig 3.14, we compare the controller's step response to changes in the number of tasks for *x264* under default and optimized parameter settings. Fig 3.14a shows the system response to a 50% step change that number of task changes from 8 to 12. The system utilization under optimized setting is able to track the set-point whereas under the default setting, a steady state error of 5.5 percent to the set-point is present. Compared with the default setting, the optimized setting shows quicker response with 30% less rise time, 22% less settling time and 68% less overshoot. Fig 3.14b and 3.14c show that in order to track set-point, the controller adapts the manipulated variable by increasing the frequency and decreasing video frame resolution.

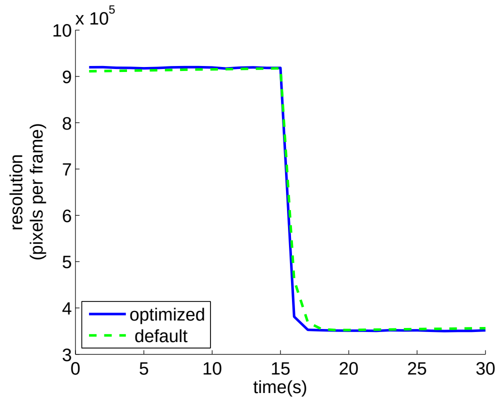
Fig 3.15 compares the controller's step response to changes in the set-point for *x264* under default and optimized parameter settings. Fig 3.15a shows that although there are



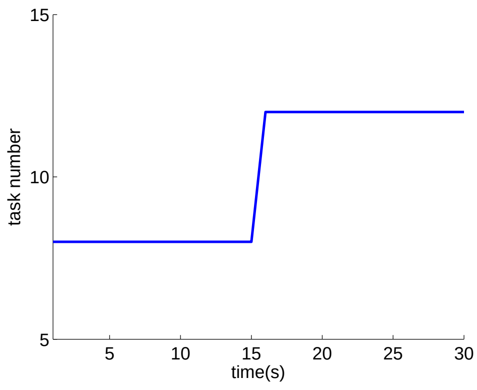
(a) Utilization versus time



(b) Processor operating frequency versus time

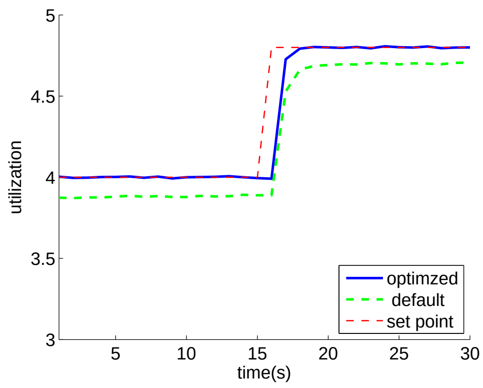


(c) Video frame resolution versus time

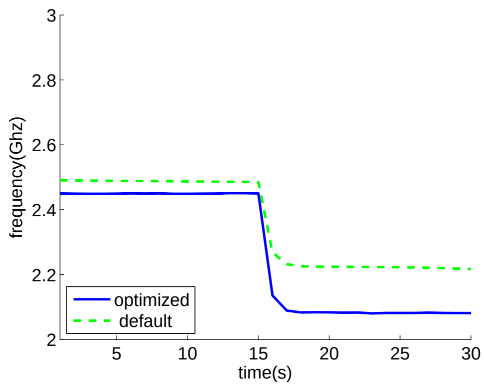


(d) Task number versus time

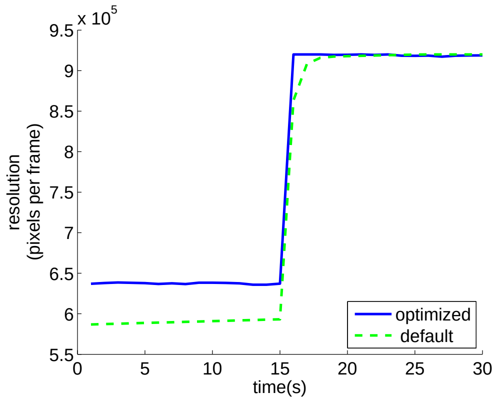
Figure 3.14: Comparison of simulated model predictive control system step response for *x264* under default and optimized setting. At t = 15s, the number of tasks changes from 8 to 12



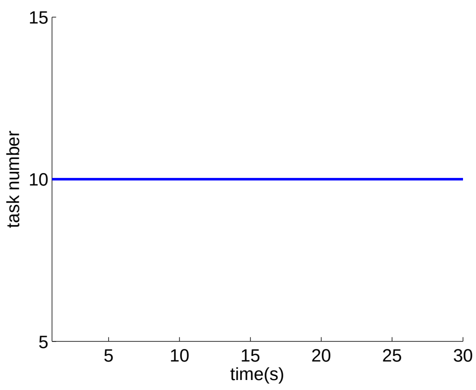
(a) Utilization versus time



(b) Processor operating frequency versus time

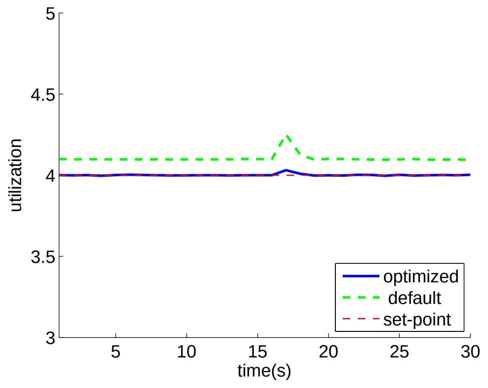


(c) Video frame resolution versus time

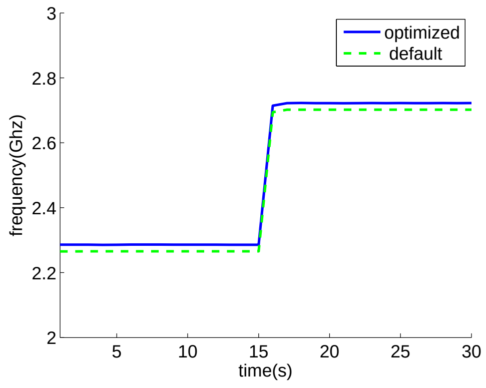


(d) Task number versus time

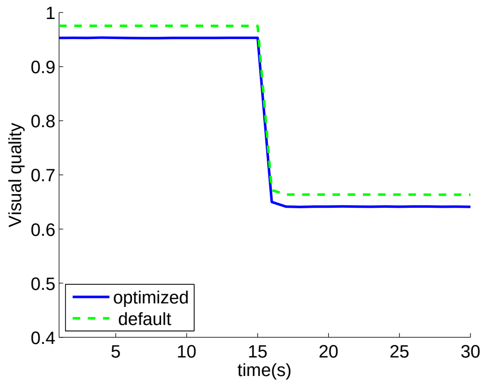
Figure 3.15: Comparison of simulated model predictive control system step response for *x264* under default and optimized setting. At t = 15s, the set-point changes from 4 to 4.8



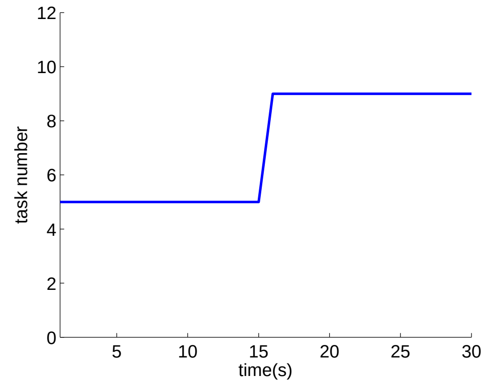
(a) Utilization versus time



(b) Processor operating frequency versus time

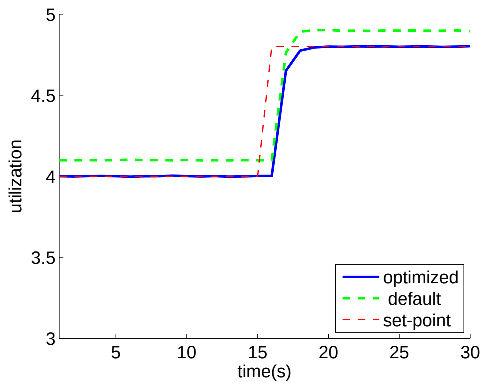


(c) Visual quality versus time

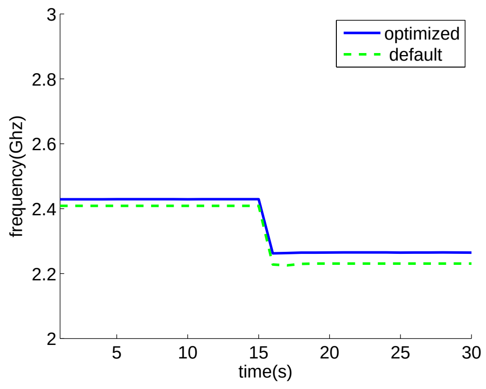


(d) Task number versus time

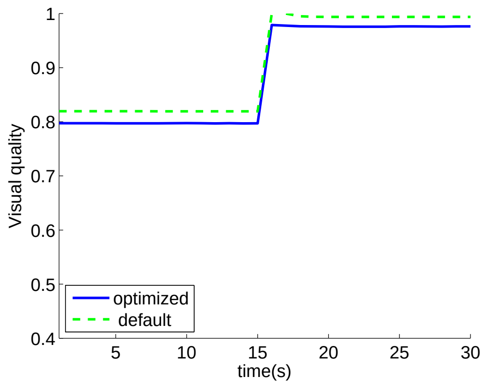
Figure 3.16: Comparison of simulated model predictive control system step response for *bodytrack* under default and optimized settings. At t = 15s, the number of tasks changes from 5 to 9.



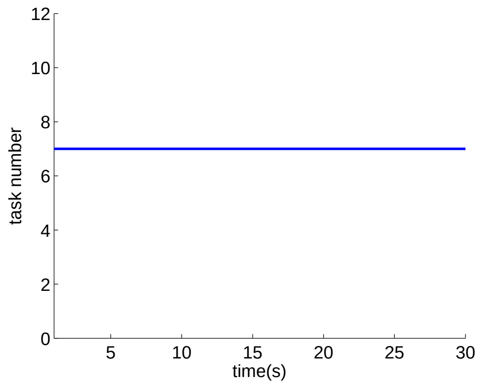
(a) Utilization versus time



(b) Processor operating frequency versus time



(c) Visual quality versus time



(d) Task number versus time

Figure 3.17: Comparison of simulated model predictive control system step response for *bodytrack* under default and optimized setting. At t = 15s, the set-point changes from 4 to 4.8.

no overshoots in both cases, the controller under the optimized setting is able to track the set-point while the one under default setting, settles with a steady state error of 5.7 percent to the set-point. Fig 3.15b and 3.15c show that in order to track the set-point, the controller adapts the manipulated variable by decreasing the frequency and increasing video frame resolution. Comparisons of *bodytrack* under default and optimized parameter settings are shown in Fig 3.16 and Fig 3.17, where the controller under optimized parameter setting also outperforms the default parameter setting for rise time, settling time, overshoot, and steady state error.

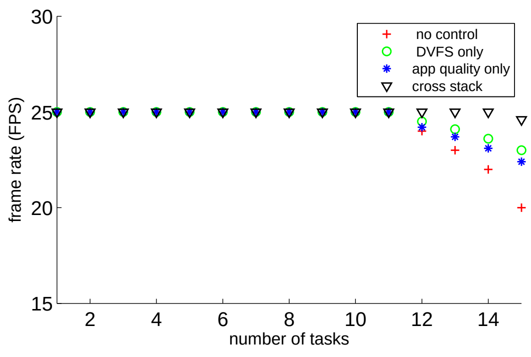
3.3.5 Need for Control

We demonstrate the need for the controller by measuring the average frame rate both with and without feedback control, as the workload is varied from light to heavy. A G-EDF scheduler is used to schedule all the tasks. As seen from Fig 3.18, in the absence of the controller, the frame rate drops beyond 8 tasks for both *x264* and *bodytrack*. For our 8 core system, the lowered frame rate indicates that the system is in overload. However, unlike the non-control case, the feedback controller is able to maintain a constant frame rate by automatically adjusting the processor frequency and the application quality. Fig 3.18 also shows the advantage of a cross-stack approach to feedback control as compared to deriving the control variables from a single layer of the computing stack. For both *x264* and *bodytrack* using DVFS-only or application quality-only as the control variable, results in a sharper drop in the frame rate with a heavier task load as compared to the cross-layer control.

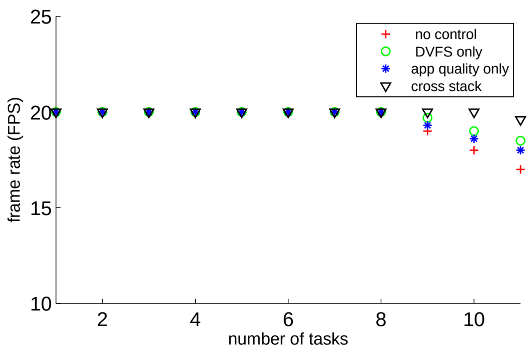
3.3.6 Step Response

In this subsection, with the optimized controller parameters shown in Table 3.14, we experimentally evaluate performance of the controller's response to an input step change in the number of tasks and an output step change of the set-point.

We define the power-efficient operating point as where all the cores run at the minimum frequency and all the tasks run at the maximum visual quality. Similarly, we define the



(a) x264

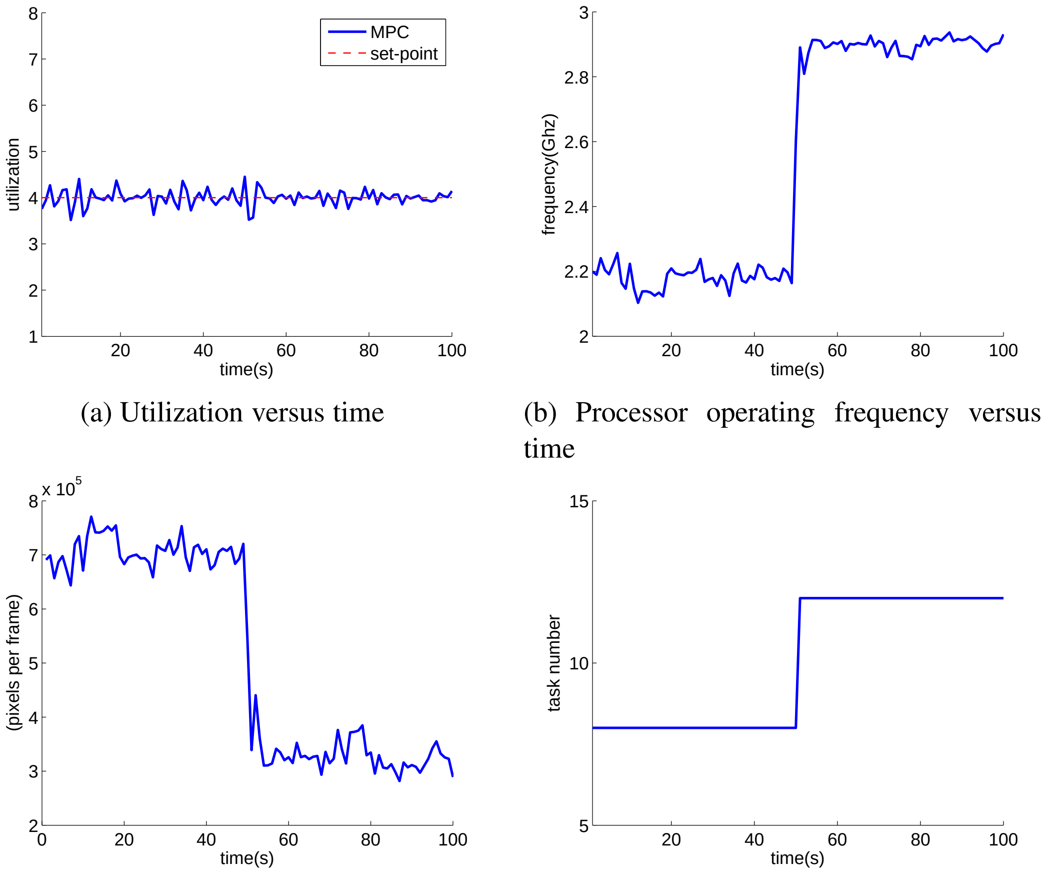


(b) bodytrack

Figure 3.18: Average FPS versus number of tasks - under no control, MPC control with DVFS-only, MPC control with application quality-only, and cross-layer MPC control

power-maximum operating points as where all the cores run at the maximum frequency and all the tasks run at the minimum application quality. As discussed in Subsection 2.2.4.2, for m cores, soft-real time tasks deadlines can be met with bounded tardiness at a utilization of m. Four our $m = 8$ system, we set the task utilization set-point to 4. The choice of this utilization set-point of 4 ensures that enough processor capacity is available for non-real time background tasks. At this utilization point, we experimentally determine power-efficient and power-maximum operating points for *x264* for 8 and 12 tasks. Due to the heavier workload per task, power-efficient and power-maximum operating points for *bodytrack* in terms of task numbers is 5 and 9 respectively. When evaluating the controller's response to a step change of set-point, we fix the task number as the average of power-efficient and power-maximum operating points, which are 10 for *x264* and 7 for *bodytrack*.

Fig 3.19 shows the controller's step response to a change in the number of tasks from 8 to 12 for *x264*. From Fig 3.19a we note that the controller settles to within 10% of the utilization set-point in 4.3 seconds and the input step change cause a peak overshoot of 11.2%. In Fig 3.19b and 3.19c, as expected, the controller responds to a higher load by increasing the frequency while decreasing the video frame resolution. The jitter seen in Fig 3.19band 3.19c represents unmodeled disturbances (noise) resulting from motion estimation in *x264*. Despite the noise, the controller is able to maintain the set-point within



(a) Utilization versus time

(b) Processor operating frequency versus time

(c) Video frame resolution versus time

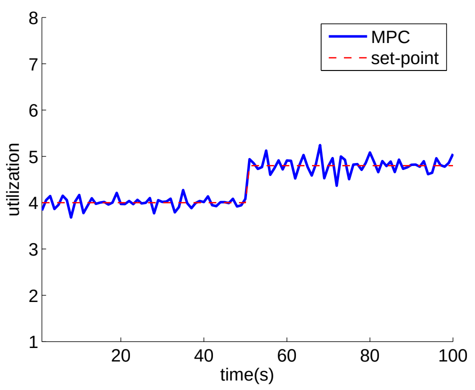
(d) Task number versus time

Figure 3.19: Experimental evaluation of model predictive control system step response for *x264*. At t = 50s, the number of tasks changes from 8 to 12.

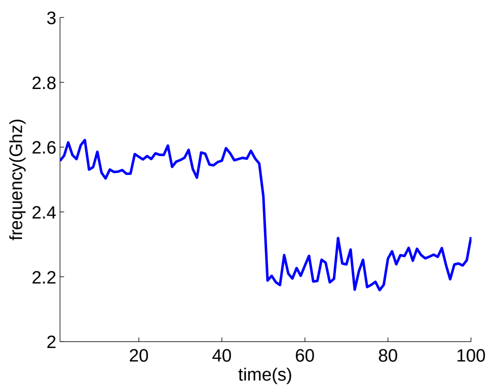
10%.

Fig 3.20 shows the controller's step response to changes in set-point from 4 to 4.8 for *x264*. From Fig 3.20a, we see that the controller is able to follow this set-point change with a peak overshoot of 7.2 % and settling time of 2.1 seconds. Fig 3.20b and 3.20c shows that in order to maintain the new set-point the controller decreases the frequency while increasing the video frame resolution. The controller is able to maintain the set-point within 10%.

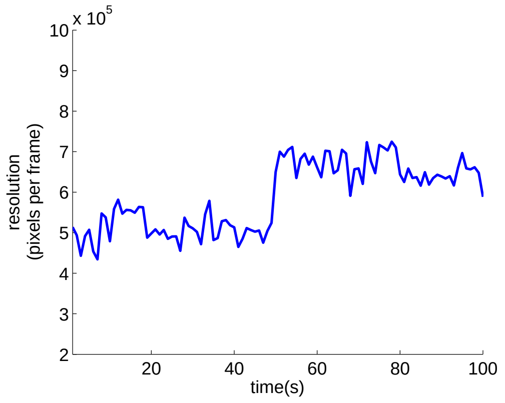
Fig 3.21 shows the controller's step response to changes in the number of tasks from 5 to 9 for *bodytrack*. By adapting operational frequency and visual quality, the controller for *bodytrack* is able to maintain this set-point within 5% with a peak overshoot of 29.7% and



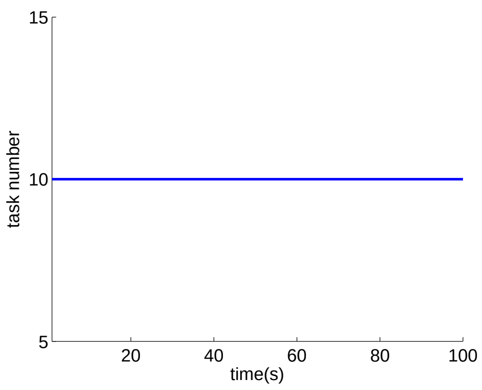
(a) Utilization versus time



(b) Processor operating frequency versus time

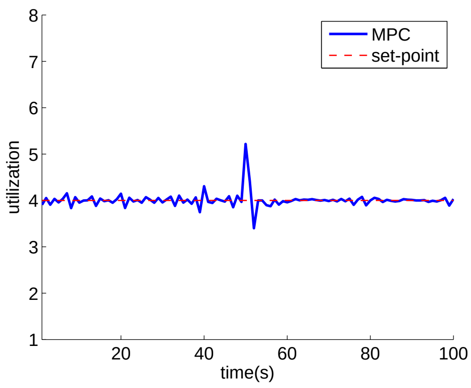


(c) Video frame resolution versus time

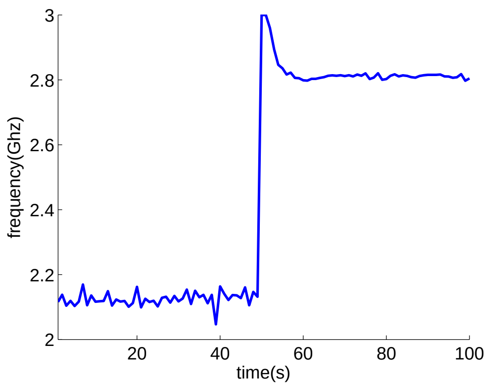


(d) Task number versus time

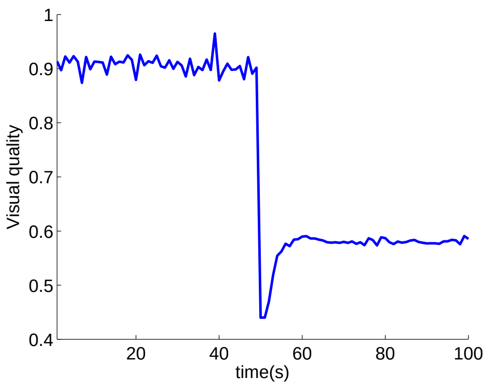
Figure 3.20: Experimental evaluation of model predictive control system step response for *x264*. At t = 50s, the set-point changes from 4 to 4.8.



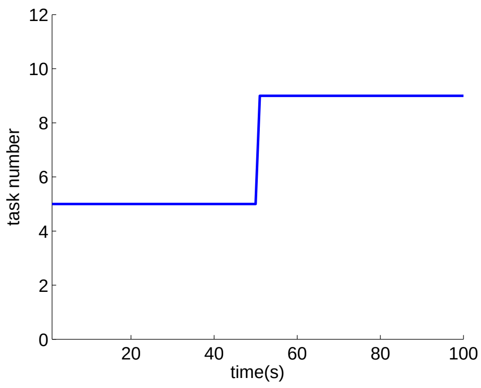
(a) Utilization versus time



(b) Processor operating frequency versus time

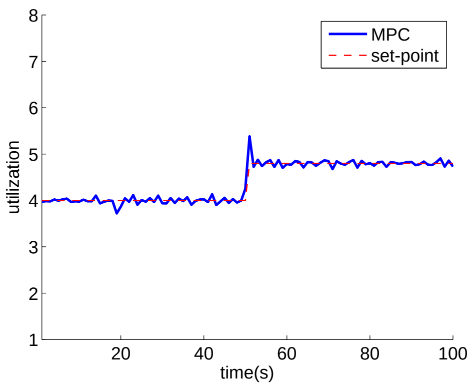


(c) Video frame resolution versus time

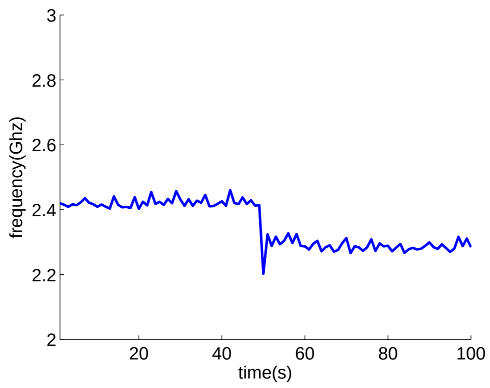


(d) Task number versus time

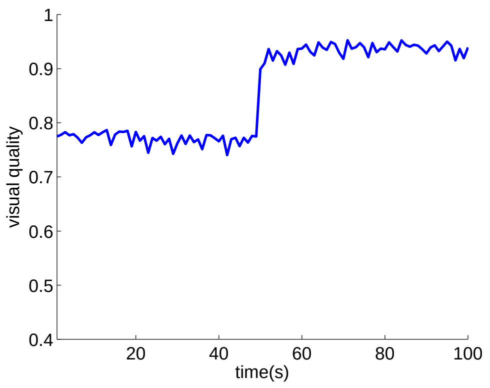
Figure 3.21: Experimental evaluation of model predictive control system step response for *bodytrack*. At t = 50s, the number of tasks changes from 5 to 9.



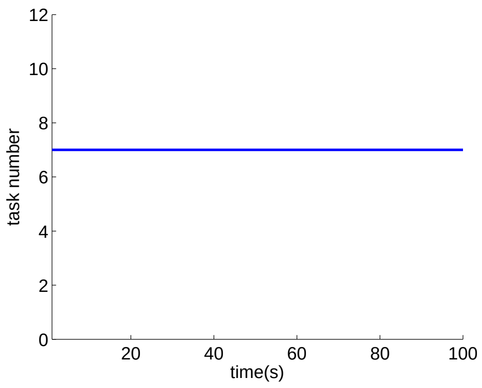
(a) Utilization versus time



(b) Processor operating frequency versus time



(c) Video frame resolution versus time



(d) Task number versus time

Figure 3.22: Experimental evaluation of model predictive control system step response for *bodytrack*. At t = 50s, the set-point changes from 4 to 4.8.

a settling time of 4.7 seconds. Fig 3.22 shows the controller's step responses to changes in set-point from 4 to 4.8 for *bodytrack*. The controller for *bodytrack* is able to follow the set-point change within 6.4% with a peak overshoot of 14.5% and a settling time of 1.8 seconds.

Compared with *x264*, *bodytrack* shows a much smaller jitter in the system output. This smaller jitter value indicates that the workload per frame for *bodytrack* has less variation compared to *x264*.

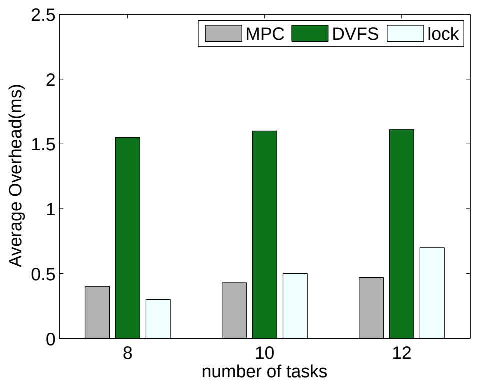
3.3.7 Power Saving

To evaluate the power savings obtained by our cross-stack control approach, we compare the average power consumption of the controller to a baseline implementation with the cores running at maximum frequency and the tasks operating at maximum visual quality. The power savings are evaluated based on the power model described in Subsection 3.2.6. For a pseudo-random number of homogeneous input tasks, our model predictive control approach shows an energy saving of 31% compared to the baseline implementation for *x264* and an energy saving of 26% for *bodytrack* . The energy saving is obtained at an average video frame resolution of 70% for *x264* and at an average visual quality of 65% for *bodytrack*.

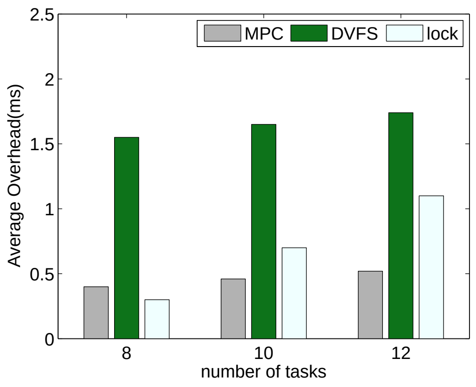
3.3.8 Controller Overhead

Three factors contribute to overhead of the controller, 1) computation cost of the MPC controller, 2) overheads due to DVFS and 3) real-time synchronization cost in modifying shared global variables in the application. Core computation of MPC controller is a quadratic programming (QP) solver whose computation complexity is polynomial time in the product of control outputs and prediction horizons. In one control period (1 second in our experiments), the core frequency is changed 20 times by the sigma-delta modulator. The overall DVFS overhead is accumulated through all the subintervals within a control period. The synchronization occurs when the application quality tuning knobs are updated. Fig 3.3.8 shows the different overhead components in milliseconds in one control period.

As we can see, increasing the number of tasks has minimal effect of the computation cost of the controller and the DVFS overhead. However, synchronization cost increases linearly with number of tasks. The overall overhead is less than 0.4 percent of one control period for both *x264*and *bodytrack* .



(a) Cross stack control overheads for *x264*



(b) Cross stack control overheads for *body-track*

3.4 Related Work

A large amount of prior work exists for run-time power management for unicore and multicore processors. Most of these works utilize either DVFS [87, 82, 97, 81] and/or application QoS [83, 3, 49, 2, 80, 89, 92, 56, 68, 74] to achieve energy efficiency. Since our work targets a cross stack energy optimization at run-time for real time applications, here we review pervious work on energy optimization for real time workloads. Aydin et. al. [6] have used DVFS and formulated the power optimization as a non-linear optimization problem with processor utilization and frequency as constraints for synthetic soft real time workloads. Seo proposed an energy-efficient scheduling algorithm called dynamic repartition for real-time tasks on a multi-core processor which dynamically balances dynamic utilization of cores by migrating tasks among them [93]. The Illinois Grace project [107] uses a hierarchical adaptation at all system layers including application (frame rate and dithering for video decoding), soft real time scheduling (CPU time allocation) and CPU (DVFS) for power optimization. The optimization problem involves maximizing quality and minimizing power with energy, processor utilization, frequency and quality of service as constraints. Unlike these works in which the optimization step operates in open loop, we use a close loop feedback controller. Cucinotta, et al [29] have proposed an adaptive resource allocation mechanism organized in two feedback loops. The internal loop is responsible for updating execution budget for soft real-time multimedia task so that timing constraints of the application are satisfied. The external loop operates on the QoS level of the applications and on the power level of the resources in open loop to strike a good trade-off between the global QoS and the energy consumption. Our work is fundamentally different in that we use one closed loop model predictive controller to meet timing constraints while adapting power and QoS. Block et al. [13] proposed a PI controller to change the processors share of tasks to meet real time constraints without any consideration of power optimization. Fu et. al. [39] used a model predictive controller similar to what we propose to optimize energy using process frequency and L2 cache size partitions as control variables for a synthesized real time work load consisting of media processing benchmarks running on a simulated quadcore processor. However, unlike our cross-stack approach their controller has control variables exclusively from the hardware stack.

3.5 Conclusions

In this chapter, we designed a model predictive controller that tracks the overall task utilization set-point with processor frequency and application quality as the control variables for soft real-time workloads such as *x264* and *bodytrack*. we have experimentally demonstrated that a cross-stack predictive control approach for homogeneous workloads is able to handle of dynamically changing operating conditions such that real-time constraints are satisfied. We demonstrated that the use of DVFS and application quality as control variables allows operation at a lower power operating point while meeting real-time constraints as compared to non cross stack control approaches.

CHAPTER 4: HETEROGENEOUS CONTROL FRAMEWORK

4.1 Introduction

In this chapter, we extend our cross-stack predictive control framework for homogeneous real-time workloads presented in Chapter 3 to the scenario where multiple heterogeneous real-time tasks execute on the same server simultaneously. The ability to run multiple dissimilar workloads on the same server enables workload consolidation. This allows aggressive power saving by powering down idle servers and consolidating the load on a smaller subset of servers.

To achieve our goal of building a cross-stack predictive control framework for heterogeneous real-time workloads, we proposed a cluster approach where different types of workloads are partitioned into different clusters and each cluster has its own controller. In this approach we utilize the cluster-EDF real-time scheduling algorithm to handle task scheduling.

We applied the proposed heterogeneous control frameworks to a workload of mixed *x264* and *bodytrack* tasks. We first demonstrate the controllability of the control approaches by evaluating its step responses. We present a comparison of performance and power consumption under G-EDF and C-EDF scheduling algorithm by evaluating their the average frame rate as the workload is varied from light to heavy. Finally, we evaluate the control overheads for this approach.

The rest of the chapter is organized as follows in Section 4.2 we describe our cross-stack predictive control framework for heterogeneous real-time workloads in detail. We present the the evaluation results in Section 4.3. We review related work on heterogeneous control framework in Section 4.4 and we conclude the chapter in Section 4.5.

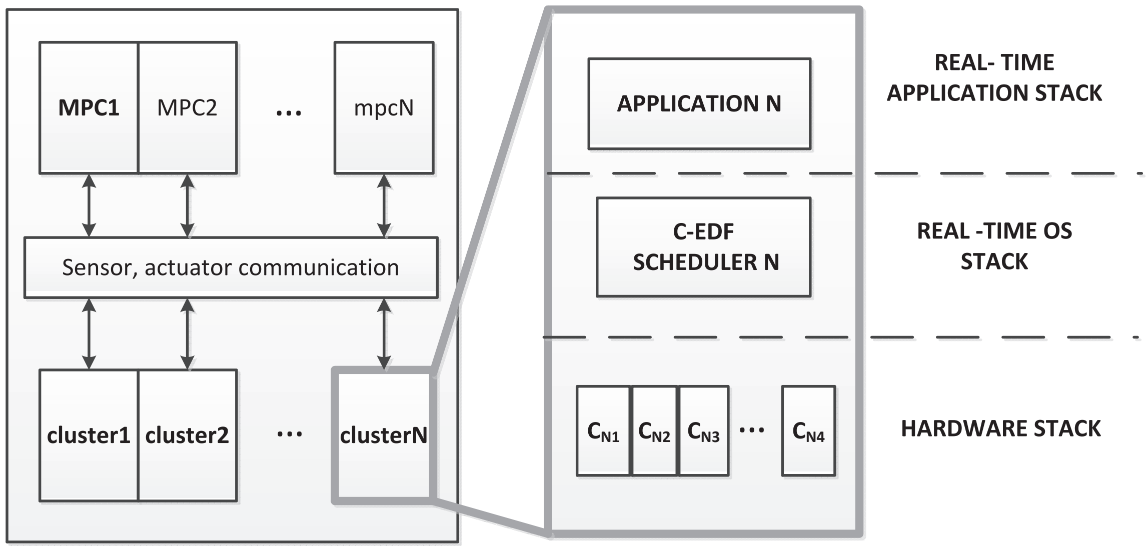


Figure 4.1: Schematic representation of cross-stack control framework for heterogeneous real-time workloads in cluster control approach

4.2 Framework

In this chapter, as mentioned in the introduction, we propose the cluster control approaches to handle heterogeneous soft real-time workloads.

Fig 4.1 schematically shows our cross-stack control framework for heterogeneous real-time workloads in the cluster control. This control approach is based on the cluster-EDF scheduling algorithm.In Cluster-EDF, all cores that share a specific cache level (L2 or L3) are defined to be a cluster (size of cluster in the same level are identical for symmetric multiprocessor); tasks are allowed to migrate within a cluster, but not across clusters; tasks assigned to a cluster are scheduled globally within the cluster under EDF algorithm. Obviously, one important benefit of clustering comes from lower overhead costs due to prohibition of task migration among same level of caches. Based on feature of cluster-EDF scheduling algorithm, we make the following assumptions for our controlled system in cluster control approach: only one type of application is assigned to one cluster; each cluster is equipped with its own cluster-EDF scheduler; cores in the same cluster share same operational frequency, frequency of cores from different clusters can be different. Separate cross-stack controllers need to be designed for each task which are capable of tracking the individual task utilization by independently controlling the cluster frequency and the per-task quality. Note that since tasks from the same type of application may occupy several clusters, it is possible that one controller is shared by different clusters. In each control period, the controller for each cluster reads per cluster utilization with sensor associated with that cluster, calculates cluster frequencies and the per-task quality, and inputs these back to the actuators associated with that cluster.

4.3 Experimental Results

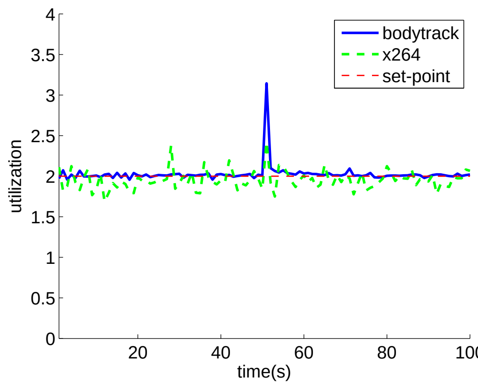
In this section, we demonstrate controllability of the cluster control approach by evaluating its step responses. We compare performance and power consumption between G-EDF and C-EDF by evaluating their the average frame rate as the workload is varied from light to heavy. Finally we evaluate the control overheads for this approach.

4.3.1 Experimental Setup

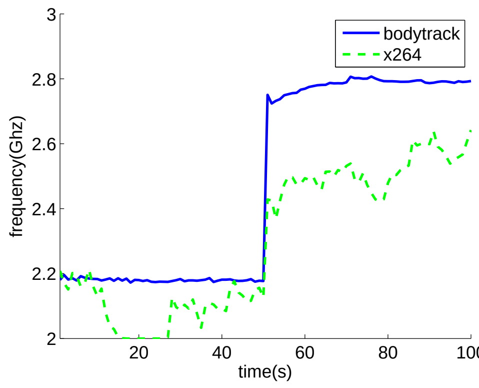
We experimentally demonstrate the operation of our cross-stack predictive control framework for heterogeneous soft real-time workload on the dual socket quad-core Intel Clovertown server described in subsection 3.3.1. The heterogeneous workload under evaluation is a mix of *bodytrack* and *x264*. For the cluster control approach, since two cores share a L2 cache in the Intel Xeon processor X5365, we group the dual-socket quad-core server into 4 clusters. We assign two clusters each to *bodytrack* and and *x264* with an MPC controller per application.

4.3.2 Step Response

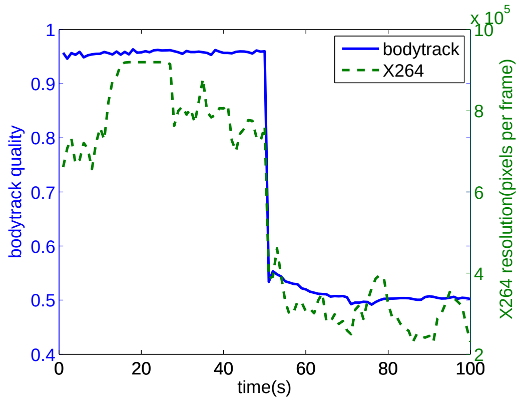
In this subsection, we experimentally evaluate the step responses of the heterogeneous control system for the global and cluster control approaches. As each application take half of server's computing capacity, we experimentally determine power-efficient and power-maximum operating points as 4 and 6 tasks for both *x264* and *bodytrack*. When evaluating the controller's response to a step change of set-point, we fix the task number as the average of power-efficient and power-maximum operating points, which are 5 for both applications.



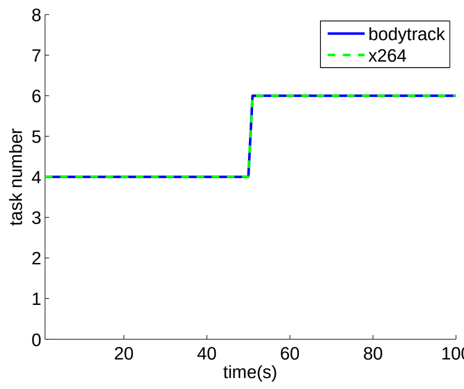
(a) Utilization versus time



(b) Processor operating frequency versus time



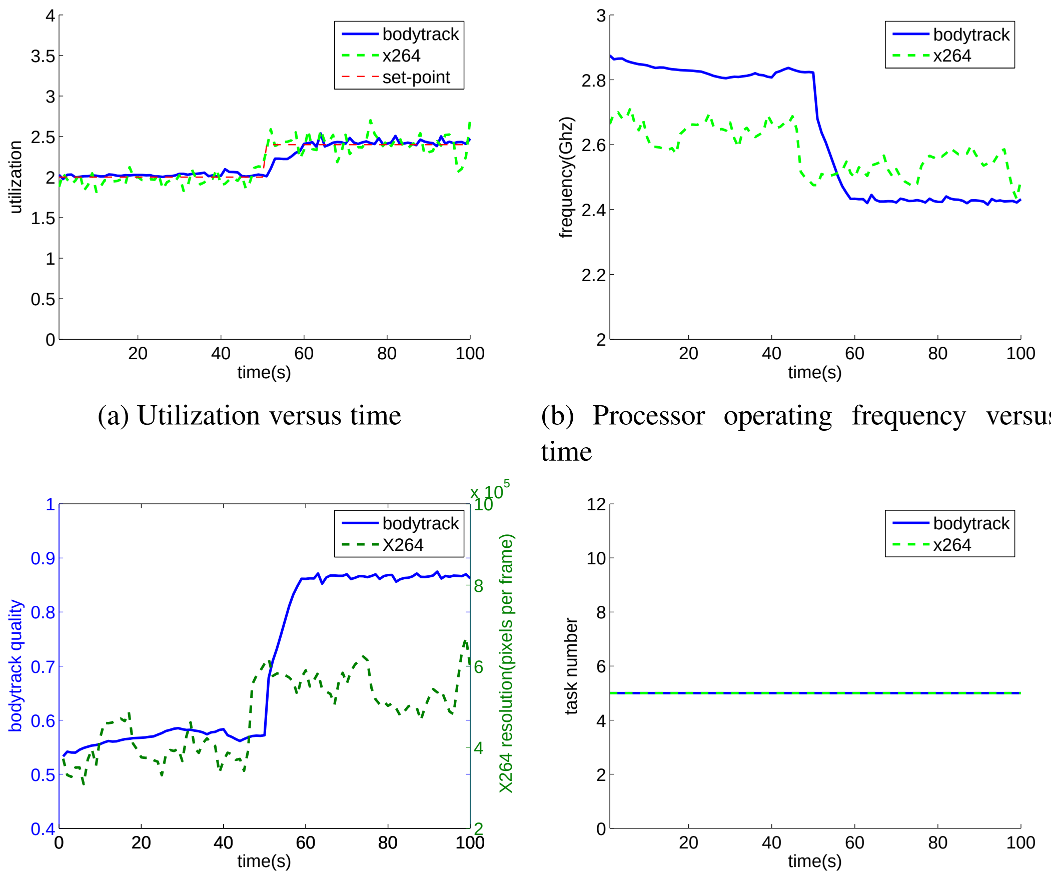
(c) Visual quality versus time



(d) Task number versus time

Figure 4.2: Experimental evaluation of cluster control approaches' responses to step changes on task number. At $t = 50s$, task number for both applications changes from 4 to 6.

Fig 4.2 shows the controller's response to step changes in the number of tasks from 4 to 6 for both *x264* and *bodytrack* at the 50^{th} seconds under the cluster control approach. Since *bodytrack* and *x264* each are assigned four cores, the utilization set-point for each application is set to half that of the global control approach. Fig 4.2a shows that, the *x264* cluster controller settles to within 14.2% of the utilization set-point with a peak overshoot of 14.0% and a settling time of 2.9 seconds; the *bodytrack* cluster controller settles to within 4.5% of the utilization set-point with a peak overshoot of 56.0% and a settling time of 4.2 seconds. The smaller jitter observed for *bodytrack* compared to *x264* is consistent with the observation in Chapter 3. Fig 4.2b and 4.2c show how the controllers adapt CPU frequency and application quality in order to track set-point: CPU frequency for *264* and *bodytrack* increases to around 2.55 GHz and 2.80 GHz respectively; visual quality for *x264* and *bodytrack* decrease to around 2.8×10^5 pixels per frame and 49 percent respectively.



(a) Utilization versus time

(b) Processor operating frequency versus time

(c) Visual quality versus time

(d) Task number versus time

Figure 4.3: Experimental evaluation of cluster control approaches' responses to step changes on set-points. At $t = 50s$, set-point of each application changes from 2 to 2.4.

Fig 4.3 shows the controller's response to step changes in the set-point from 2 to 2.4 for both applications under cluster control approach. From Fig 4.3a, we note that the controller for *x264* is able to follow this set-point change with a peak overshoot of 9.2 % in 2.4 seconds; the controller of *bodytrack* is able to follow this set-point change in 6.5 seconds with no overshoot. Fig 4.3b and 4.3c shows that in order to follow the new set-point, the controller for *x264* decreases the CPU frequency from 2.62 GHz to 2.48 GHz. Simultaneously the video frame resolution increases from 3.2×10^5 to 8.0×10^5 pixels per frame.

Meanwhile, the controller for *bodytrack* decreases the CPU frequency from 2.8 GHz to around 2.4 GHz and increases the visual quality from around 56 percent to 88 percent.

Table 4.1: Average FPS under C-EDF and G-EDF scheduler for a heterogeneous workload.

number of tasks		FPS of *x264*		FPS of *bodytrack*	
x264	*body-track*	C-EDF	G-EDF	C-EDF	G-EDF
2	2	25	25	20	20
2	8	25	25	15.8	20
10	2	20.1	25	20	20
8	6	25	23.1	20	18.3

4.3.3 Comparison of C-EDF and G-EDF Scheduling Algorithm

We also investigate the choice of the real-time scheduling algorithms on the performance of the the cross-layer feedback controller when the system hosts heterogeneous tasks from multiple applications (*x264* and *bodytrack*). In global-EDF scheduling, tasks from both the applications are scheduled globally across all 8 cores. In clustered-EDF scheduling, the applications run on separate clusters with tasks from a single application assigned to a cluster of 4 cores sharing the L2 cache. In both cases, separate controllers are designed for the two applications. Unfortunately, our hardware platform does not allow independent control of the core frequencies, limiting us to use only the application quality as the control variable for this experiment. Table 4.1 compares the average frame rate for G-EDF and C-EDF for different combination of number of tasks. For a balanced but light workload, both C-EDF and G-EDF achieve the targeted average FPS of 25 and 20 for *x264* and *bodytrack* respectively. For an unbalanced workload, where the applications have dissimilar number of tasks, we note that G-EDF with its superior load balancing capability performs better. However, for a balanced but heavy workload with large number of tasks for both applications, load balancing is less of any issue. For this case, C-EDF with its better data locality performs better.

4.3.4 Control Overhead

For control overhead we consider the same three components introduced in Subsection 3.3.8. Fig 4.4 shows the different average overhead components in milliseconds in one control period. As we can see, both DVFS cost (1.60 milliseconds for C-EDF, 1.65 milliseconds for G-EDF), synchronization cost (0.84 milliseconds for C-EDF, 0.81 milliseconds for G-EDF) and computation cost of MPC controller (0.73 milliseconds for C-EDF, 0.7 milliseconds for G-EDF) are very close for both task scheduling algorithms. The overall overheads are less than 0.4 percent of one control period.

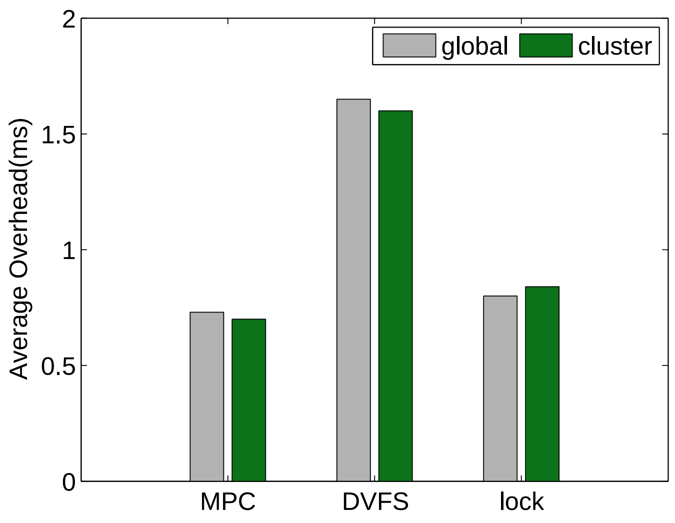


Figure 4.4: Control overhead under global and cluster control approach

4.4 Related Work

To the best of our knowledge, we present the first control framework which can power efficiently operate a heterogeneous soft real-time computing workloads while meeting real-time constraints. We review previous works related to the development of real time global and cluster scheduling algorithms.

Brandenburg et al. [18] implemented global-EDF scheduling algorithm as part of $LITMUS^{RT}$ testbed. Their evaluation platform is the Sun's Niagara multiprocessor. This empirical study is carried out to evaluate implementation tradeoffs including choice of ready queue implementation, quantum-driven vs. event driven scheduling, and interrupt handling strategy. Their results show that implementation tradeoffs can significantly impact schedulability and the scalability and that global-EDF is suitable for real-time scheduling on large scale multicore platforms.

Calandrino et al. [22] first proposed the cluster approach for scheduling real-time tasks on large-scale multicore platforms with hierarchical shared caches. This work is evaluated under the SESC processor microarchitecture simulator which simulates 64 core processor with 4 levels of shared caches. They showed that the cluster approach performs better on large-scale platforms than the global approach does in terms of lower task migration overhead, lower scheduling overhead, and higher schedulability.

Bastoni et al. [9] presented an empirical comparison of global, partitioned, and cluster EDF scheduling algorithms on a 24 core Intel multicore platform. Scheduling algorithms were compared in terms of real-time schedulability. They adopted a new aggregate performance metric to compare schedulability results for a wide range of cache-related delays and to clearly identify the range of data locality and migration overhead in which a particular scheduler shows better performance. Their results showed that, global-EDF is not a practical choice for hard real-time systems. The cluster EDF approaches were found to be superior to all other evaluated algorithms in terms of kernel overhead, data locality overhead, and schedulability.

4.5 Conclusions

In this chapter, we proposed cluster approach to implement cross-stack predictive control framework for heterogeneous real-time workloads and experimentally evaluate their performance tradeoffs. Experimental results show that cluster control approaches can guarantee real-time constraints on heterogeneous workloads and show acceptable performance in terms of peak overshoot, settling time and jitter value. Due to superior load balancing capability, control with G-EDF performs better with an unbalanced workload. However, for a balanced but heavy workload with large number of tasks for both applications, load balancing is less of any issue. For this case, C-EDF with its better data locality performs better.

CHAPTER 5: ADAPTIVE CONTROL FRAMEWORK

5.1 Introduction

In this chapter, we present an adaptive cross-stack predictive control framework which maintains the desired level of performance for dynamic workloads. One major limitation of conventional controllers is that their performance might be not always acceptable under an unknown environment due to a number of possible factors including external disturbances, changes in subsystem dynamics, and parameter variations [72]. The primary reason for this limitation is because that they employ a fixed controller structure which is not suitable for the entire range of the operation. On the contrary, an adaptive controller is able to achieve a desired level of performance of the control system by dynamically adjusting its structure in response to a changing environment.

In our work, we employ a gain scheduling methodology for our adaptive cross-stack predictive control framework using multiple fixed models identified based on $a\ priori$ available information. At run-time, a supervisor will periodically check which model is the most suitable for the actual system with respect to certain desired features and switch to the controller associated with the selected model.

We select *x264* encoder as the workload to demonstrate the operation of our adaptive cross-stack predictive control framework. *x264* exhibits distinct visual and temporal features if videos from different video genres are used as its encoding input. A computing system executing such a workload is prone to exhibit large performance variations, which may lead to significant degradation on performance of a controller with a fixed structure. Hence it is essential to apply adaptive control for such a system. We initially subdivide 20 video files used for our experiments into four genres, according to the subject of the video. The four categories we use are cartoon, entertainment, news report, and sports. We create model predictive controllers as well as their state space models derived by system identification for each genre. To determine which model should be chosen in real-time, we implement a video genre classifier which employs Kolmogorov-Smirnov (K-S) test to statistically calculate similarity between current system and established models.

For the experiment part, we first evaluate effectiveness of our video genre classifier using 100 different video files from the 4 video genres. Then we compare performance of our adaptive controller with non-adaptive controller in terms of their steady state error. Finally we present overhead analysis of the adaptive cross-stack predictive control framework.

The rest of the chapter is organized as follows in Section 5.2 we describe adaptive cross-stack predictive control framework in detail. In Section 5.3, we introduce how to select an experimental set of video genres for our research and introduce the K-S test algorithm used in our work. We then present the results evaluating the performance of the adaptive cross-stack predictive control framework in Section 5.4. We review related work in Section 5.5 and conclude the chapter in Section 5.6.

5.2 Framework

In Chapter 3, we constructed a cross-stack predictive control framework for homogeneous real-time workloads. In this chapter we extend our work to adaptive cross-stack predictive control framework by employing a methodology using multiple fixed models and switching among these at run-time. The structure of our adaptive cross-stack predictive control framework is shown in Fig 5.1. The computing system consists of three different stacks, real-time application stack, real-time OS stack and the hardware stack. In each control period, the controller reads system the utilization from a sensor implemented in the kernel space, calculates the desired values of CPU frequency and visual quality according to the MPC algorithm, and sends these measurements to the hardware and application stack by means of the corresponding modulator and actuator. Additionally, a supervisor is used to arbitrate which controller will be selected in real-time. The supervisor reads informa-

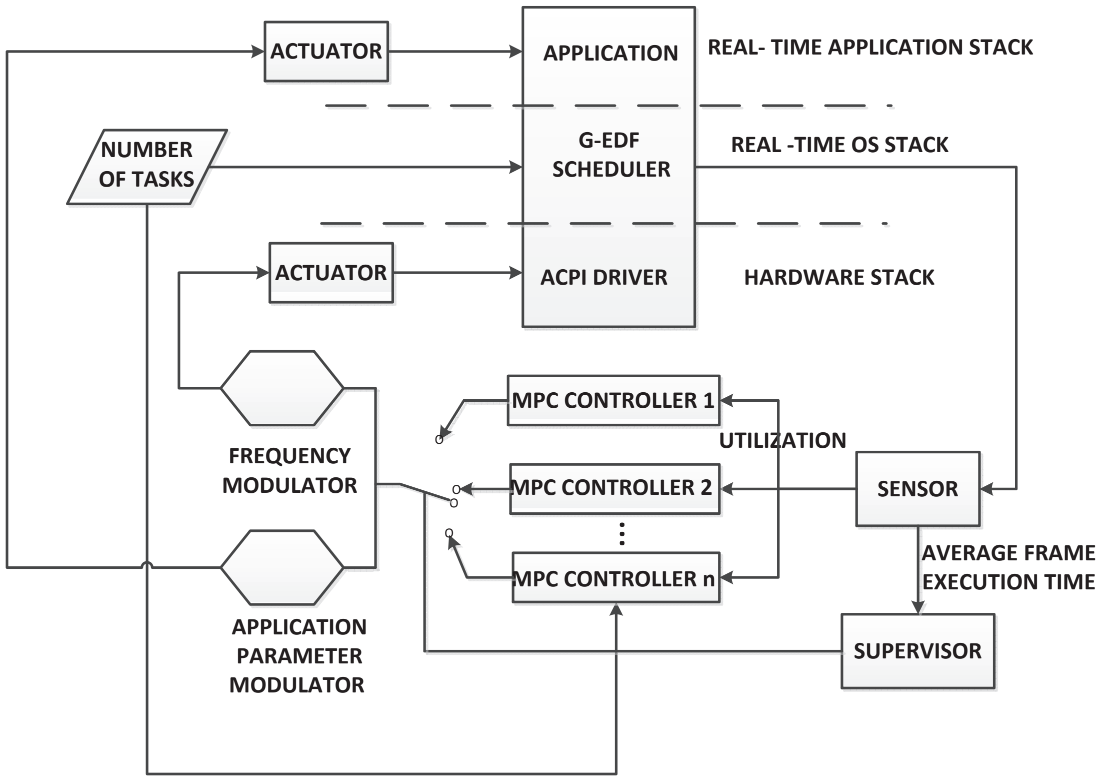


Figure 5.1: Schematic representation of our adaptive cross-stack predictive control framework

tion of average frame execution time from the sensor and saves it to a buffer of video genre classifiers at every control period. Then for a fixed number of control period, the supervisor calls the video genre classifier to determine which video category should be selected based on the buffered data. Stored with data information about average frame execution time of all video genres, the video genre classifier will calculate each significance level which determine whether the buffered data and preloaded data from each video genre are from the same data distribution using Kolmogorov-Smirnov test algorithm. The video genre associated with the largest significance level will be selected. Finally the controller corresponded to the video genre is selected for better control performance.

5.3 Video Genre Classification

In this section, we present our approach for video genre classification. We first explain how to determine an experimental set of video genres based on previous research. Then we introduce the classification approach used in our research. Finally we describe how to implement K-S test algorithm for our video genre classifier.

5.3.1 Video Genres

The genre of a video is a general class to which it belongs, such as sports, news, cartoon etc. The determination of a genre is made by watching the video content. Due to subjective opinions and semantic subtleties, sometimes determination of a genre can be controversial and prone to error. To correctly decide genre of a video, two issues are worth noting. First, a genre may consist of other genres as well but genre at the same level should be mutually exclusive. For example, a sports live broadcast on a basketball match belongs to genre basketball which in turn is a member of the genre sports. However a basketball broadcast cannot belong to genre baseball or belong to genre cartoon as opposed to sports. Secondly, the genre of video sometimes undergoes phase change where different genres exclusively appear in series. Consider an example when a news program reports on a recent classical music concert and plays highlight clip of this concert: although it is broadcast in the context of a news program, this concert highlight clip should be treated as a distinct video genre. Hence a successful video genre classifier should be able to capture this phase change.

Although there is no standard video genre set for video classification research, some genres are common to classfiications proposed by different research groups. As reported in the literature [37, 63, 59, 62, 43, 101, 102, 34, 50, 91], the most popular genres proposed include: sports, news, cartoon and entertainment. We therefore choose these four genres as our experimental set for video genre classification.

5.3.2 Classification Metric

Most work on video genre classification adopts approaches which try to differentiate video genres based on their different visual and audio features. For example, the scenes which are extracted from a horror film may contain much more dark-lighted scenes compared with scenes from a comedy movie; scenes from action movies may contain much more explosive, noisy sounds than a romantic movie. Commonly used visual features include color-based features, shot-based features, object-based features and motion-based features [19]; time-domain features and frequency-domain features are popular in audio [19].

However, all these approaches require decompressing the video sequence beforehand and analyzing the video frame by frame. Since most videos in broadcast is compressed for higher efficiency in storage and network transmission, those approaches could incur enormous overhead. Hence they are not suited for the real-time operation of our control system. Conversely, research on real-time video genre classification adopts a different methodology since video in compression has very little information available for classification except for temporal knowledge [58]. In this work, we adopt a temporal approach which classify video genre based on data sets of their average frame execution time and uses Kolmogorov-Smirnov test algorithm to determine if these data sets coming from same the statistical distribution.

5.3.3 Kolmogorov-Smirnov (K-S) Test Algorithm

K-S test is a statistical hypothesis test to determine if two data sets follow the same statistical distribution. The K-S test is applicable to continuous data as function of a single independent variable where each data point can be represented by a single value.

As shown in Fig 5.2, a measured distribution of values in x (shown as N dots on the lower abscissa) is to be compared with a theoretical distribution whose cumulative probability distribution is plotted as $P(x)$. D is the greatest distance between the two cumulative distributions. To apply K-S test, the list of data points should be firstly converted to a step-function cumulative probability distribution $S_N(x)$. If N data points are located at values x_i, $i = 1, 2, ..., N$, $S_N(x)$ keeps constant between consecutive x_i's and increases by the same constant $1/N$ at each x_i. $S_N(x)$ is constant between consecutive (i.e., sorted into ascending order) x_i's, and jumps by the same constant $1/N$ at each x_i as shown in Fig 5.2. Since sets of data from different distribution function give different cumulative distribution

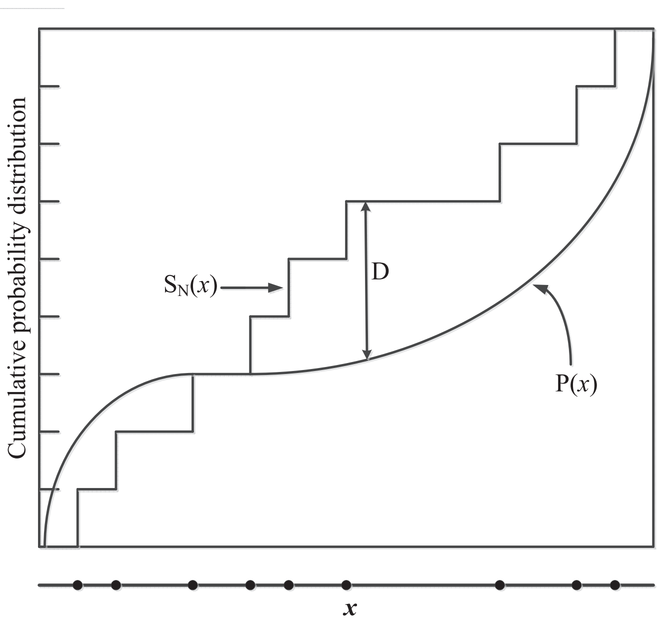


Figure 5.2: Illustration of Kolmogorov-Smirnov test algorithm

function estimators, K-S test uses the maximum value of the absolute difference between two cumulative distribution functions as the K-S statistic D. To compare two different cumulative distribution functions $S_{N_1}(x)$ and $S_{N_2}(x)$ for two data sets, the K-S statistic is defined as:

$$D = \max_{-\infty<x<\infty} |S_{N_1}(x) - S_{N_2}(x)| \tag{5.1}$$

D between $S_{N_1}(x)$ and $S_{N_2}(x)$ can be iteratively derived by stepping through each x_i in the two data sets.

The significance level of an observed value of D is given approximately by the equation 5.2. The larger significance level on K-S statistic is, the more likely that the two data sets come from the same cumulative probability distribution. According to [66], the threshold value of significance level on K-S statistic is set to 5%.

$$Probability(D > observed) = Qks([\sqrt{N_e} + 0.12 + 0.11/\sqrt{N_e}]D) \tag{5.2}$$

Where

$$Q_{KS}(\lambda) = 2\sum_{j=1}^{\infty}(-1)^{j-1}e^{-2j^2\lambda^2} \tag{5.3}$$

which is a monotonic decreasing function with limiting values

$$Q_{KS}(0) = 1 \qquad Q_{KS}(0) = \infty \tag{5.4}$$

and N_e is the effective number of data points, for equation 5.2

$$N_e = \frac{N_1 N_2}{N_1 + N_2} \tag{5.5}$$

where N_1 is the number of data points in the first distribution and N_2 is the number in the second. The approximation of equation 5.2 becomes asymptotically accurate when the N_e becomes large and reasonably accurate when $N_e \geq 4$.

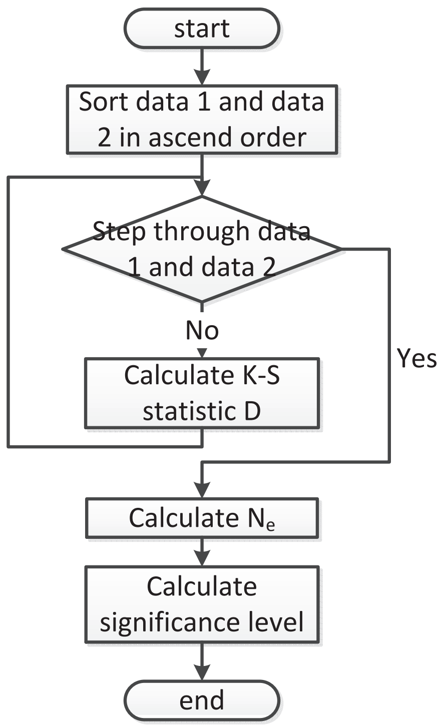

start
Sort data 1 and data 2 in ascend order
Step through data 1 and data 2
No
Yes
Calculate K-S statistic D
Calculate N_e
Calculate significance level
end

Figure 5.3: Flow chart of K-S test

We implemented the K-S test in C++. Fig 5.3 shows the flow chart. The two data sets are initially sorted in ascending order to generate their cumulative distribution function $S_N(x)$. Then the K-S statistic D is iteratively derived by stepping through every element of the two data sets. Finally, significance level between the two data sets can be calculated as shown in Equation 5.2 with calculated D and N_e.

5.4 Experimental Results

In this section, we first validate the effectiveness of our video genre classifier with 100 HD mp4 video inputs from 4 video genres. Then we experimentally evaluate the performance of our adaptive cross-stack predictive control system in terms of its steady state error to specific set-point and compare it with the non-adaptive counterpart. Finally we quantitatively analyze the execution overheads of the adaptive controller.

5.4.1 Selection of Classification Period

The video genre classifier is invoked every classification period. This duration of this classification period should not be either too long or too short: too long classification period will undermine the responsiveness of the adaptive controller; too short classification period will not be able to buffer enough data points of average frame execution time , leading to inaccurate result of video genre classification. Taking these two aspects into consideration, we set this classification period to be 3 seconds and set control the period to be 0.3 second. The video genre classifier is thus called upon every 10 control periods. On one hand, this setting gives the latency of capturing a phase change by the adaptive controller around the same length as the classification period. We note that this is an acceptable latency even for some fast phase change scenarios such as commercial breaks where each video genre may last from tens of seconds to several minutes. On the other hand, 10 data points of average frame execution time during each classification period with a preloaded data size set to 100 will make N_e in Equation 5.5 equals to 9.04. This N_e value is greater than 4, which guarantees accurate video classification according to Subsection 5.3.3.

5.4.2 Experimental Setup

We experimentally demonstrate the operation of our adaptive cross-stack predictive control framework for soft real-time workload on the dual socket quad-core Intel Clovertown server described in Subsection 3.3.1. We choose *x264* encoder as the soft real-time workload under evaluation for reasons explained in Section5.1. We classify HD mp4 video inputs in four video genres: sports, news, cartoon and entertainment. Our adaptive controller incorporates four fixed predictive controllers based on the corresponding four video genres and uses a supervisor which calls upon a video genre classifier to determine which controller should be chosen in real-time. 100 data points of average frame execution time for each video genres are preloaded in the video genre classifier. The video genre classifier is called upon by the supervisor every 3 seconds.

5.4.3 Validation of Video Genre Classifier

For the purpose of validating the effectiveness of our video genre classifier, we gathered 100 HD mp4 video inputs which include videos from 4 video genres, news, sports, cartoon and entertainment. The video from genre entertainment is based on different movie trailers and music live videos. The video of sports consist of video clips mainly from basketball and soccer matches. We execute *x264* encoder with every videos for 30 seconds while simultaneously running the video genre classifier. Since we set classification period to be 3 seconds, it generate a total of 10 classification results in 30 seconds. We claim the classification is valid if at least 8 out of 10 classification results are correct. The video genre classifier achieves satisfactory validation results with no less than 90% success rate for each genre.

5.4.4 Performance Evaluation

We experimentally evaluate the performance of our adaptive cross-stack predictive control system and compare it with the non-adaptive counterpart introduced in chapter 3. The video inputs for this evaluation are 10 highly viewed test videos drawn from YouTube with content ranging from music, sports, news and do-it yourself.

Table 5.1: Performance comparison of adaptive control with non-adaptive control

video name	non-adaptive control		adaptive control	
	steady state error	significance level of K-S statistic	steady state error	significance level of K-S statistic
1 music video	8.6%	31.3%	7.2%	34.8%
2 music video	7.5%	36.7%	6.9	37.9%
3 news report	9.1%	28.9%	8.1%	32.4%
4 photography hacks	22.5%	0.015%	9.1%	25.3%
5 cooking	8.2%	32.5%	7.5%	30.2%
6 sports	25.7%	0.006%	8.7%	31.3%
7 news report	9.7%	24.3%	7.2%	36.4%
8 hiring program	8.9%	29.4%	8.3%	27.3%
9 movie clip	9.9%	19.4%	8.2%	31.9%
10 about champagne	9.5%	24.1%	8.6%	30.8%

Table 5.1 compares the adaptive controller's performance with non-adaptive controller in terms of its steady state error to the set-point for the 10 videos mentioned above at medium workload(10 *x264* encoding tasks) and at set-point of 4. We observe that 8 videos out of 10 the controller shows acceptable performance in terms of small steady state error (less than 10%) under the non-adaptive controller. This is because the model used to generate the non-adaptive controller shows similar feature with the 8 videos as indicated by their large significance levels of K-S statistic shown on Table 5.1. However, It performs poorly on video No.4 and No.6 in terms of large steady state error (greater than 20%). On the contrary, adaptive controller shows acceptable performance on each video input. This demonstrates that the adaptive controller achieves superior performance compared to non-adaptive controller by dynamically selecting appropriate video genres.

5.4.5 Control Overhead

In addition to the control overhead contributed by the MPC computation, DVFS, and real-time synchronization, analyzed in Subsection 3.3.8, the additional overhead of our adaptive control framework stems from the video classifier, whose core computation is dominated by the K-S test. The computational overheads associated with the K-S test depends on the size of video classifier buffer ($n1$) and the size of preloaded data set for each video genre ($n2$) since the computation complexity of K-S test is $O(n1.logn1 + (n1 + n2))$ [42]. We experimentally determine that the average overhead of K-S test in this adaptive control framework is 2.6 milliseconds. Since the video classifier is called every 10 control periods and each control periods is 300 milliseconds, the classifier accounts for 0.086 % of each control period.

5.5 Related Work

Numerous approaches and techniques have been proposed in the area of adaptive control [5]. Here we limit our review to those applying techniques of adaptive control into computing system design and optimization.

Wang et al. [105] proposed an algorithm which controls the power consumption of a multicore processor to the desired set-point while maintaining the temperature of each core below a specified threshold. Their experimental platform is an Intel Xeon X5365 Quad Core processor with 8MB on-die L2 cache and 1333 MHz FSB. They model the power consumption of multicore processor in term of operational frequency of each core. The controller is implemented with the lsqlin solver in Matlab. To adaptively capture the model variation, they use a recursive least square estimator with directional forgetting to update the parameter matrix in the system model.

Reed et al. [85] proposed an adaptive controller for the Apache web server to enforce metrics that affect the user experience in a client machine such as HTTP reply time. Their simulation platform is Apache v2.2 running on a Linux kernel 3.0.0-14 x64 workstation with a dual core Intel T2400 and 2GB of RAM They adopt linear Auto-Regressive modeling with exogenous input to model the throughput of Apache web server. A modified recursive least squares algorithm is employed to adaptively identify system dynamics. A minimum degree pole placement controller is used to adjust the maximum number of concurrent connections.

Kalyvianaki et al. [48] proposed a resource management strategy that combines the Kalman filter with feedback controllers to dynamically allocate CPU resources to server applications hosted by virtual machines for better system throughput. They carried out their experiments in a virtualized cluster consisting of three machines connected by gigabit ethernet and each running the Xen 3.0.2 hypervisor which hosts the Rubis server application. They model the time-varying CPU usage of application as a linear stochastic difference equation. They adopt an adaptive MIMO process noise covariance controller to self-configures its parameters and self-adapts to changing workload conditions.

Li et al. [57] proposed a feedback-based strategy to maximize the platform performance of vSphere using a gradient based hill climbing algorithm. Their experimental platform is a cluster consisting of three ESX hosts, each of which has a Dell PowerEdge R610 machine with dual quad-core Intel Xeon 3.0 GHz processor and 1649GB of RAM with 540GB local disk. They implemented a vSphere adaptive task management system which combines feedback control and adaptive hill-climbing algorithm to determine the maximum throughput for any given vSphere environment and control the number of tasks assigned to the system, while adapting to changes in the environment.

Different from above references which assume a linear system, our work demonstrates use of gain scheduling for non-linear workloads.

5.6 Conclusions

In this chapter, we have presented an adaptive cross-stack predictive control framework which maintains desired level of performance in the presence of non-linearities in the workload using gain scheduling. We implement our adaptive controller by adopting a supervisor which dynamically switches among several fixed structure controllers to improve control performance. To illustrate the effectiveness of our adaptive controller, we choose *x264* encoder as workload. We classify this workload based on different genres of its video inputs and implement a video genre classifier based on Kolmogorov-Smirnov test algorithm. We then incorporate this video genre classifier into our adaptive control framework. Experimental results shows adaptive controller outperforms non-adaptive controller in terms of smaller steady state error. We also show that adaptive control only requires a small overhead, which accounts for 0.086 % of each control period.

CHAPTER 6: CONCLUSIONS

The goal of the dissertation was to improve the overload capacity and power efficiency of real-time multicore computing systems by establishing a cross-stack predictive control framework. We established that the use of DVFS and application quality as control variables enables operation at a lower power operating point while meeting real-time constraints as compared to non cross-stack control approaches. We also evaluated the role of scheduling algorithms in the control of homogeneous and heterogeneous workloads. Additionally, we proposed a novel adaptive control technique for dynamic workloads.

6.1 Summary of Results

We implemented a cross-stack control framework for homogeneous real-time workloads. The real-time multicore computing system was modeled as a MISO state space model using system identification. We used a model predictive controller (MPC) to implement the control framework since MPC can handle multiple control variables and constraints on both the dependent and independent variables. Our results showed that better control performance can be achieved if the control inputs are derived from all parts of the computing stack: the cross-stack controller was able to maintain constant frame rate while DVFS-only or application quality-only control failed to do so at heavy workload (task number over 8 on an 8 core system)for both *bodytrack* and *x264* workloads.

We then extended our cross-stack predictive control framework for heterogeneous workloads by adopting a cluster control approach with a C-EDF scheduler. Our experimental results demonstrated that due to superior load balancing capability, control with G-EDF performs better with an unbalanced workload. However, for a balanced but heavy workload with large number of tasks for both applications, C-EDF with its better data locality performs better.

To handle dynamic workloads where the execution characteristics change significantly with time, we proposed an adaptive cross-stack predictive control approach using gain scheduling. For the *x264* video encoding workload, the adaptive controller was found to outperform the non-adaptive controller with a smaller steady state error.

6.2 Future Work

Our work can be extended in several directions as described below:

- One direction for improvement is the replacement of the Matlab MPC Toolbox with a custom MPC controller, allowing more flexibility in incorporating a variety of adaptive control algorithms.
- We can also explore the incorporation of additional control variables from the hardware stack and the OS stack into our control framework for better power efficiency and control performance. For example, dynamic cache repartitioning is another adaptive hardware technique for improving power efficiency; in some reservation-based schedulers, bandwidth (the fraction of per job CPU time over period) can be dynamically allocated to each job.
- In our work, the controller and the real-time scheduling algorithms are implemented in userland. While this approach allows better real-time performance, the need to modify the kernel may limit its applicability due to security concerns. An alternate approach that could be explored involves implementing both control and scheduling algorithms in the userspace. Here we could take advantage of the recent work by Millson and Anderson in implementing real-time scheduling in Linux using userspace libraries [70].
- Our control framework was demonstrated for the case of a single multicore server. Power efficient execution of soft real-time workloads is an increasing requirement in a data center. Future work could involve exploring the scalability of the proposed control framework using a hierarchical control approach to enable scaling to hun-

dreds of servers.

BIBLIOGRAPHY

[1] Ffmpeg general documentation. `http://ffmpeg.org/general.html`.

[2] Luca Abeni and Giorgio Buttazzo. Hierarchical qos management for time sensitive applications. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 63–72. IEEE, 2001.

[3] Luca Abeni, Tommaso Cucinotta, Giuseppe Lipari, Luca Marzario, and Luigi Palopoli. Qos management through adaptive reservations. *Real-Time Systems*, 29(2-3):131–155, 2005.

[4] David H Albonesi, Rajeev Balasubramonian, SG Dropsbo, Sandhya Dwarkadas, Eby G Friedman, Michael C Huang, Volkan Kursun, Grigorios Magklis, Michael L Scott, Greg Semeraro, et al. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, 2003.

[5] Karl Johan Åström. Theory and applications of adaptive controla survey. *Automatica*, 19(5):471–486, 1983.

[6] Hakan Aydin, Vinay Devadas, and Dakai Zhu. System-level energy management for periodic real-time tasks. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 313–322. IEEE, 2006.

[7] Alexandru O Balan, Leonid Sigal, and Michael J Black. A quantitative evaluation of video-based 3d person tracking. In *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*, pages 349–356. IEEE, 2005.

[8] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 245–257. ACM, 2000.

[9] Andrea Bastoni, Björn B Brandenburg, and James H Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 14–24. IEEE, 2010.

[10] Alberto Bemporad, Manfred Morari, and N. Lawrence Ricker. *Model Predictive Control Toolbox*. The MathWorks, 2005.

[11] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[12] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[13] Aaron Block, Björn Brandenburg, James H Anderson, and Stephen Quint. An adaptive framework for multiprocessor real-time system. In *Real-Time Systems, 2008. ECRTS'08. Euromicro Conference on*, pages 23–33. IEEE, 2008.

[14] Aaron Block, Hennadiy Leontyev, Björn B Brandenburg, and James H Anderson. A flexible real-time locking protocol for multiprocessors. In *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, pages 47–56. IEEE, 2007.

[15] B Brandenburg, A Block, J Calandrino, U Devi, Hennadiy Leontyev, and J Anderson. Litmusrt: A status report. In *Proceedings of the 9th real-time Linux workshop*, pages 107–123, 2007.

[16] Björn B Brandenburg. *Scheduling and locking in multiprocessor real-time operating systems*. PhD thesis, University of North Carolina, 2011.

[17] Björn B Brandenburg and James H Anderson. A comparison of the m-pcp, d-pcp, and fmlp on litmusrt. In *Principles of Distributed Systems*, pages 105–124. Springer, 2008.

[18] Björn B Brandenburg and James H Anderson. On the implementation of global real-time schedulers. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 214–224. IEEE, 2009.

[19] Darin Brezeale and Diane J Cook. Automatic video classification: A survey of the literature. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(3):416–430, 2008.

[20] Giorgio C Buttazzo. Rate monotonic vs. edf: judgment day. *Real-Time Systems*, 29(1):5–26, 2005.

[21] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer, 2011.

[22] John M Calandrino, James H Anderson, and Dan P Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*, pages 247–258. IEEE, 2007.

[23] John M Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C Devi, and James H Anderson. Litmusˆ rt: A testbed for empirically comparing real-time multiprocessor schedulers. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 111–126. IEEE, 2006.

[24] Eduardo F Camacho, Carlos Bordons, Eduardo F Camacho, and Carlos Bordons. *Model predictive control*, volume 2. Springer London, 2004.

[25] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook on Scheduling Algorithms, Methods, and Models, pages*, pages 30–1, 2004.

[26] WenZhi Chen and Huan Zheng. Analysis of power saving effect for dynamic power management. In *Mechatronic and Embedded Systems and Applications, Proceedings of the 2nd IEEE/ASME International Conference on*, pages 1–4. IEEE, 2006.

[27] James A Cherry and W Martin Snelgrove. Clock jitter and quantizer metastability in continuous-time delta-sigma modulators. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(6):661–676, 1999.

[28] Intel Coorporation. Using the rdtsc instruction for performance monitoring. *Techn. Ber., tech. rep., Intel Coorporation*, page 22, 1997.

[29] Tommaso Cucinotta, Luigi Palopoli, Luca Abeni, Dario Faggioli, and Giuseppe Lipari. On the integration of application level and resource level qos control for real-time applications. *Industrial Informatics, IEEE Transactions on*, 6(4):479–491, 2010.

[30] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 4. IEEE Press, 2008.

[31] Jonathan Deutscher and Ian Reid. Articulated body motion capture by stochastic search. *International Journal of Computer Vision*, 61(2):185–205, 2005.

[32] UmaMaheswari C Devi and James H Anderson. Tardiness bounds under global edf scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189, 2008.

[33] Ashutosh S Dhodapkar and James E Smith. Managing multi-configuration hardware via dynamic working set analysis. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 233–244. IEEE, 2002.

[34] Nevenka Dimitrova, Lalitha Agnihotri, and Gang Wei. Video classification based on hmm using text and faces. In *European Conference on Signal Processing*. Citeseer, 2000.

[35] Loïc Duflot, Olivier Levillain, and Benjamin Morin. Acpi: Design principles and concerns. In *Trusted Computing*, pages 14–28. Springer, 2009.

[36] Ralph Ewerth, Martin Schwalb, Paul Tessmann, and Bernd Freisleben. Estimation of arbitrary camera motion in mpeg videos. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 1, pages 512–515. IEEE, 2004.

[37] Stephan Fischer, Rainer Lienhart, and Wolfgang Effelsberg. Automatic recognition of film genres. In *ACM multimedia*, volume 95, pages 295–304, 1995.

[38] Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

[39] Xing Fu, Khairul Kabir, and Xiaorui Wang. Cache-aware utilization control for energy efficiency in multi-core real-time systems. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 102–111. IEEE, 2011.

[40] Xing Fu and Xiaorui Wang. Utilization-controlled task consolidation for power optimization in multi-core real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 73–82. IEEE, 2011.

[41] Marco ET Gerards and Jan Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4):41, 2013.

[42] Teofilo Gonzalez, Sartaj Sahni, and William R. Franta. An efficient algorithm for the kolmogorov-smirnov and lilliefors tests. *ACM Transactions on Mathematical Software (TOMS)*, 3(1):60–64, 1977.

[43] Alexander G Hauptmann and Michael J Witbrock. Story segmentation and detection of commercials in broadcast news video. In *Research and Technology Advances in Digital Libraries, 1998. ADL 98. Proceedings. IEEE International Forum on*, pages 168–179. IEEE, 1998.

[44] Intel Hewlett-Packard. Microsoft, phoenix, and toshiba. advanced configuration and power interface specification, 2004.

[45] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *ACM SIGPLAN Notices*, volume 46, pages 199–212. ACM, 2011.

[46] Enhanced Intel. Speedstep® technology for the intel® pentium® m processor, 2004.

[47] Kevin Jeffay. The real-time producer/consumer paradigm: A paradigm for the construction of efficient, predictable real-time systems. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, pages 796–804. ACM, 1993.

[48] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, pages 117–126. ACM, 2009.

[49] Shinpei Kato, Ragunathan Rajkumar, and Yutaka Ishikawa. Airs: Supporting interactive real-time applications on multicore platforms. In *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, pages 47–56. IEEE, 2010.

[50] Vikrant Kobla, Daniel DeMenthon, and David S Doermann. Identifying sports videos using replay, text, and camera motion features. In *Electronic Imaging*, pages 332–343. International Society for Optics and Photonics, 1999.

[51] Butler W Lampson and David D Redell. Experience with processes and monitors in mesa. *Communications of the ACM*, 23(2):105–117, 1980.

[52] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8. USENIX Association, 2010.

[53] Charles Lefurgy, Xiaorui Wang, and Malcolm Ware. Server-level power control. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 4–4. IEEE, 2007.

[54] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171. IEEE, 1989.

[55] Julie Letchner, Christopher Ré, Magdalena Balazinska, and Matthai Philipose. Approximation trade-offs in markovian stream processing: An empirical study. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 936–939. IEEE, 2010.

[56] Baochun Li and Klara Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *Selected Areas in Communications, IEEE Journal on*, 17(9):1632–1650, 1999.

[57] Zhichao Li, Aalap Desai, Chirag Bhatt, Rajit Kambo, and Erez Zadok. vatm: vsphere adaptive task management. In *Proceedings of the 7th international workshop on feedback computing*, 2012.

[58] Qilian Liang and Jerry M Mendel. Mpeg vbr video traffic modeling and classification using fuzzy technique. *Fuzzy Systems, IEEE Transactions on*, 9(1):183–193, 2001.

[59] Rainer Lienhart, Christoph Kuhmunch, and Wolfgang Effelsberg. On the detection and recognition of television commercials. In *Multimedia Computing and Systems' 97. Proceedings., IEEE International Conference on*, pages 509–516. IEEE, 1997.

[60] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[61] Jane Liu. *Real-Time Systems*, pages 515–518. 2000.

[62] Zhu Liu, Jincheng Huang, and Yao Wang. Classification tv programs based on audio information using hidden markov model. In *Multimedia Signal Processing, 1998 IEEE Second Workshop on*, pages 27–32. IEEE, 1998.

[63] Zhu Liu, Yao Wang, and Tsuhan Chen. Audio feature extraction and analysis for scene segmentation and classification. *Journal of VLSI signal processing systems for signal, image and video technology*, 20(1-2):61–79, 1998.

[64] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Operating-system directed power reduction. In *Proceedings of the 2000 international symposium on Low power electronics and design*, pages 37–42. ACM, 2000.

[65] Jan Marian Maciejowski. Predictive control with constraints. 1999.

[66] Mathwork. Two-sample kolmogorov-smirnov test. `http://www.mathworks.com/help/stats/kstest2.html`.

[67] Ke Meng, Russ Joseph, Robert P Dick, and Li Shang. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 177–186. ACM, 2008.

[68] Malena Mesarina and Yoshio Turner. Reduced energy decoding of mpeg streams. *Multimedia Systems*, 9(2):202–213, 2003.

[69] Sasa Misailovic, Stelios Sidiroglou, Henry Hoffmann, and Martin Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 25–34. ACM, 2010.

[70] Malcolm S Mollison and James H Anderson. Bringing theory into practice: A userspace library for multicore real-time scheduling. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 283–292. IEEE, 2013.

[71] Ramanathan Narayanan, Berkin Özıkyılmaz, Gokhan Memik, Alok Choudhary, and Joseph Zambreno. Quantization error and accuracy-performance tradeoffs for embedded data mining workloads. In *Computational Science–ICCS 2007*, pages 734–741. Springer, 2007.

[72] Kumpati S Narendra and Jeyendran Balakrishnan. Adaptive control using multiple models. *Automatic Control, IEEE Transactions on*, 42(2):171–187, 1997.

[73] Bradford Nichols, Dick Buttlar, and Jacqueline Farrell. *Pthreads programming: A POSIX standard for better multiprocessing*. O'Reilly Media, Inc., 1996.

[74] Brian D Noble, Mahadev Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R Walker. Agile application-aware adaptation for mobility. In *ACM SIGOPS Operating Systems Review*, volume 31, pages 276–287. ACM, 1997.

[75] Massachusetts Institute of Technology. Project MAC. Engineering Robotics Group and ML Dertouzos. *Control robotics: The procedural control of physical processes*. 1973.

[76] Dong-Ik Oh and Theodore P. Bakker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2):183–192, 1998.

[77] Sung-Heun Oh and Seung-Min Yang. A modified least-laxity-first scheduling algorithm for real-time tasks. In *Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on*, pages 31–36. IEEE, 1998.

[78] Chandandeep Singh Pabla. Completely fair scheduler. *Linux Journal*, 2009(184):4, 2009.

[79] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230. sn, 2006.

[80] Sung I Park, Vijay Raghunathan, and Mani B Srivastava. Energy efficiency and fairness tradeoffs in multi-resource, multi-tasking embedded systems. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 469–474. ACM, 2003.

[81] Padmanabhan Pillai and Kang G Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 89–102. ACM, 2001.

[82] Gang Qu and Miodrag Potkonjak. Energy minimization with guaranteed quality of service. In *Proceedings of the 2000 international symposium on Low power electronics and design*, pages 43–49. ACM, 2000.

[83] Ragunathan Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Photonics West'98 Electronic Imaging*, pages 150–164. International Society for Optics and Photonics, 1997.

[84] Ragunathan Rajkumar, Lui Sha, and John P Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium, 1988., Proceedings.*, pages 259–269. IEEE, 1988.

[85] Erik Reed, Abe Ishihara, and Ole J Mengshoel. Adaptive control of apache web server. 2013.

[86] Martin Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 324–334. ACM, 2006.

[87] Arjun Roy, Stephen M Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, pages 139–152. ACM, 2011.

[88] Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305–327, 2003.

[89] CA Rusu, Rami Melhem, and Daniel Mossé. Maximizing the system value while satisfying time and energy constraints. *IBM Journal of Research and Development*, 47(5.6):689–702, 2003.

[90] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[91] Juan María Sánchez and Xavier Binefa. Automatic digital tv commercials recognition. *Pattern Recognition and Applications*, 56:223, 2000.

[92] Vanessa Romero Segovia, Karl-Erik Årzén, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Johan Eker, and Harald Gustafsson. Adaptive resource management framework for mobile terminals-the actors approach. In *Proceedings of the First International Workshop on Adaptive Resource Management (WARM), Stockholm, Sweden*, 2010.

[93] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *Parallel and Distributed Systems, IEEE Transactions on*, 19(11):1540–1552, 2008.

[94] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *Computers, IEEE Transactions on*, 39(9):1175–1185, 1990.

[95] Lui Sha, Ragunathan Rajkumar, Sang Hyuk Son, and C-H Chang. A real-time locking protocol. *Computers, IEEE Transactions on*, 40(7):793–800, 1991.

[96] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating system concepts*, pages 100–101. J. Wiley & Sons, 2009.

[97] David C Snowdon, Etienne Le Sueur, Stefan M Petters, and Gernot Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 289–302. ACM, 2009.

[98] John A Stankovic and Krithi Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, 1990.

[99] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proceedings of the 45th annual Design Automation Conference*, pages 300–303. ACM, 2008.

[100] Asif Syed, Ershad Ahmed, Dragan MaksimoviC, and Eduard Alarcon. Digital pulse width modulator architectures. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, volume 6, pages 4689–4695. IEEE, 2004.

[101] Ba Tu Truong and Chitra Dorai. Automatic genre identification for content-based video categorization. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 230–233. IEEE, 2000.

[102] Jianguo Wang and Tieniu Tan. A new face detection method based on shape information. *Pattern Recognition Letters*, 21(6):463–471, 2000.

[103] Liuping Wang. *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009.

[104] Weixun Wang, Prabhat Mishra, and Sanjay Ranka. Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 948–953. IEEE, 2011.

[105] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ACM SIGARCH computer architecture news*, volume 37, pages 314–324. ACM, 2009.

[106] Jun Yan and Wei Zhang. Wcet analysis for multi-core processors with shared l2 instruction caches. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 80–89. IEEE, 2008.

[107] Wanghong Yuan, Klara Nahrstedt, Sarita V Adve, Douglas L Jones, and Robin H Kravets. Grace-1: Cross-layer adaptation for multimedia quality and battery energy. *Mobile Computing, IEEE Transactions on*, 5(7):799–815, 2006.

[108] Dieter Zöbel. The deadlock problem: A classifying bibliography. *ACM SIGOPS Operating Systems Review*, 17(4):6–15, 1983.