2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) | 978-1-6654-4188-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/RTCSA52859.2021.00010

Processor and Bus Co-scheduling Strategies for Real-time Tasks with Multiple Service-levels

Sanjit Kumar Roy, Arnab Sarkar, and Rahul Gangopadhyay

***Abstract*—Cyber-Physical Systems, including those in the automotive domain, are often designed by assigning to each task an appropriate criticality-based reward value which is acquired by the system on its successful execution. Additionally, each task may have multiple implementations designated as service-levels, with higher service-levels producing more accurate results and contributing to higher rewards for the system. This work proposes strategies for co-scheduling a set of periodic tasks with multiple service-levels, on homogeneous processors and system buses. The problem is modeled as a *Multi-dimensional Multiple-Choice Knapsack formulation* (MMCKP) with the objective of maximizing overall system level rewards. A *Dynamic Programming* (DP) solution is proposed to solve the MMCKP. It was observed that although the DP based solution produces optimal results, its complexity is highly sensitive to the number of tasks, processors, buses as well as to the number of task service-levels, which severely restricts scalability of the strategy. Therefore, we have also proposed a fast yet efficient heuristic algorithm called *Accurate Low Overhead Level Allocator* (ALOLA), which attempts to achieve the same objective. Our simulation based experimental evaluation shows that even on moderately large systems consisting of 90 tasks with 5 service-levels each, 16 processors and 4 buses, while MMCKP incurs a run-time of more than 1 hour 20 minutes and approximately 68 GB main memory, ALOLA takes only about 196 μs (speedup of the order of 10^6 times) and less than 1 MB of memory. Moreover, while being fast, ALOLA is also efficient being able to control performance degradations to at most 13% compared to the optimal results produced by MMCKP. We use an automated flight control system employed in modern avionic systems, a real-world application to illustrate the general applicability of our proposed scheme.**

***Index Terms*—Real-Time Processor Scheduling; Real-Time Bus Scheduling; Optimal Service-level Selection; Cyber-Physical Systems; Automotive Systems; Scheduling**

I. Introduction

Safety-critical Cyber-Physical Systems (CPS) like those in the automotive and avionics domains, smart grids, nuclear plants, etc., often consist of multiple physical sub-systems, each with its own dedicated processing platforms. These processing platforms are generally used to compute and transmit necessary commands/actuation outputs that control the behavior of the physical sub-system based on sensed performance parameters. For example, automotive systems consist of a set of components such as chassis control, anti-lock braking system, fuel injection, adaptive cruise control, collision avoidance, airbag control, air conditioning control, power window, infotainment systems, etc. Most of these components typically read their input parameters via sensors. The sensed inputs are then transmitted as messages over buses to one or more dedicated processing elements (often called ECUs) where corresponding control outputs are computed. The outputs in turn, are communicated to actuators as messages via buses.

However, such federated architectures where sub-components are associated with their own dedicated set of processing elements, result in systems with a large number of processors. Many of them remain severely underutilized. Such under utilization may lead to the deployment of more resources than is necessary when diverse functionalities are executed in an integrated fashion on a small consolidated number of processing elements. Thus, federated architectures may result in higher design costs compared to more integrated execution of functionalities on smaller consolidated platforms. Most of these functionalities are modeled as real-time periodic tasks and are classified either as safety-critical (like anti-lock braking system, fuel injection, adaptive cruise control, collision avoidance, airbag control etc.) or non-safety critical (like air conditioning control, power window, infotainment systems etc.). Given a set of processing elements connected through buses, the successful execution of tasks on processing elements and transmission of messages through buses, satisfying resource constraints and deadlines, is essentially a real-time co-scheduling problem.

Traditionally, scheduling in real-time multiprocessor systems uses either partitioned or global approaches [5], [9], [10]. With partitioning, the multiprocessor scheduling problem is transformed into uniprocessor scheduling problem, where a task is assigned to a designated processor and gets executed entirely on that processor. Well known optimal uniprocessor scheduling algorithms include Rate-monotonic (RM) (static priority) and Earliest Deadline First (EDF) (dynamic priority), proposed by [19]. However, a major drawback of partitioning is that, upto half of the system capacity may remain unutilized in order to ensure timing constraints of a given task set [4]. Unlike the fully partitioned approaches, more global schedulers like Pfair [6], PD^2 [3], ERFair [2], Boundary Fair (BF) [24], SA [13], LLREF [8], RUN [23], DP-Fair [17] can achieve very high utilization of the system capacity by allowing migrations of tasks among processors.

The Proportional fair (Pfair) scheduler proposed by [6] is known to be the first optimal global real-time scheduler on multiprocessor systems for tasks with implicit deadlines. Based on Pfair, [2] proposed a work-conserving multiprocessor scheduling algorithm called the Early-Release fair (ERfair)

Sanjit Kumar Roy, Dept. of CSE, IIT Guwahati, Assam, India. E-mail: sanjit.roy@iitg.ac.in

Arnab Sarkar, ATDC, IIT Kharagpur, West Bengal, India. E-mail: arnab@atdc.iitkgp.ac.in

Rahul Gangopadhyay, Dept. of Mathematics & Computer Science, St. Petersburg State University, Russia. E-mail: rahulincxtint@gmail.com

978-1-6654-4188-9/21/$31.00 ©2021 IEEE
DOI 10.1109/RTCSA52859.2021.00010

scheduler. However both the above schemes attempt to maintain proportional fairness at each time slot and incur unrestricted preemption/migration overheads due to this. Recently, [17] proposed a semi-partitioned approximate proportional fair optimal scheduler called DP-Fair with much lower and bounded context switching overheads.

A significant amount of research work on resource allocation schemes has also been carried out for systems where applications may have multiple QoS/operational modes [7], [11], [14]. An analytical model (called Q-RAM) for QoS management in large distributed systems was proposed in [11], [15], [16], [21]. All these Q-RAM based schemes endeavor to meet application specific needs along multiple quality dimensions such as timeliness, reliable delivery schemes, data quality etc. Rajkumar et al. [21] primarily worked with a continuous QoS dimension and assumed that the utility function is concave. They used a continuous optimization technique to solve the problem. Lee et al. [16] on the other hand solved multi-QoS dimension resource allocation problem considering QoS to be discretely varying.

In [22], Real and Crespo developed scheduling strategies referred to as Mode Change Protocol (MCP), which are marked by the need to carefully manage workloads during transition intervals when the system alternates between multiple operation modes. In [1], [7], authors proposed a resource adaptation mechanism for real-time applications with multiple service-levels where each service-level has an associated resource demand and *quality of service* (QoS). For a system with limited computational resources, the objective is to judiciously multiplex available resources by choosing an appropriate service-level for each application such that aggregate system level QoS is maximized.

In distributed Cyber-Physical Systems (like automotive and avionic systems, smart grids, industrial automation systems etc.), allocation of resources only to application tasks is not sufficient as there is a need to schedule associated message transmissions. Scheduling functionalities which require multiple types of resources for their execution, have also received considerable attention of the researchers. In [20], Minaeva et al. proposed the jitter constrained co-scheduling of tasks and messages on a system consisting of homogeneous multicores connected through crossbar switches. In this work, they employed a non-preemptive scheduling scheme which may significantly reduce system utilization in many scenarios.

Giannopoulou et al. [12] addressed the problem of multi-criticality scheduling of real-time periodic tasks on more than one resource type. The authors employed a time-triggered offline scheduling approach where a schedule for one hyperperiod of the set of tasks is generated. This schedule repeats every hyperperiod during online execution. The duration of a hyperperiod is divided into smaller intervals called *frames*. Execution on all resources synchronize at each frame boundary. A frame is again partitioned into as many sub-frames as the number of criticality levels. Tasks are allocated to sub-frames in decreasing order of their criticalities. That is, the first sub-frame consists only of tasks having the highest criticality level, the second sub-frame contains tasks with the second-highest criticality level, and so on; tasks with the lowest criticality value are allocated to the last sub-frame. A new sub-frame starts only after all tasks in the current sub-frame complete their executions on all processors. The authors used a barrier function to synchronize sub-frames on different processors. This strategy allows the scheme to achieve a criticality based differentiation of the service offered to the tasks in each frame. However, the main drawback of this differentiated service approach is that in the worst case, all processors excepting the one executing tasks may have to idle until the completion of a sub-frame, in spite of the presence of lower criticality ready tasks. This non-work conserving nature may potentially lead to poor resource utilization.

In this work, we consider the optimal off-line allocation of a set of persistent real-time periodic tasks with multiple QoS oriented service-levels on a system consisting of multiple homogeneous processors and shared buses. Each service-level of any task has a distinct computation demand (serviced by one or more processors) and communication demand (serviced by a set of shared buses) with higher service-levels having higher resource demands. Successful execution of a task at a certain service-level is associated with a reward corresponding to that service-level and this reward is proportional to the task's relative importance/criticality.

The objective of the task allocation mechanism is to maximize aggregate rewards such that both computation and communication resource demands of all tasks may be feasibly satisfied. We pose the problem as a *Multi-dimensional Multiple-Choice Knapsack formulation* (MMCKP) and present a *Dynamic Programming* (DP) solution (called MMCKP-DP) for the same. Although DP delivers optimal solutions, it suffers from significantly high overheads (in terms of running time and main memory consumption) which steeply increase as the number of tasks, service-levels, processors and buses in the system grows. Even for a system with a moderately large task set consisting of 90 tasks, it takes approximately 1 hour 20 minutes and consumes a huge amount of main memory space (approximately 68 GB). Such large time and space overheads are often not affordable, especially when multiple quick design iterations are needed during design space exploration and/or powerful server systems are not available at the designer's disposal. Therefore, in addition to the optimal solution approach presented above, we propose an efficient but low-overhead heuristic strategy called ALOLA (*Accurate Low Overhead Level Allocator*) which consumes drastically lower time and space complexities while generating good and acceptable solutions which do not significantly deviate from the optimal solutions. Our simulation based experimental evaluation shows that even on moderately large systems consisting of 90 tasks with 5 service-levels each, 16 processors and 4 buses, while MMCKP incurs a run-time of more than 1 hour 20 minutes and approximately 68 GB main memory, ALOLA takes only about 196 μs (speedup of the order of 10^6 times) and less than 1 MB of memory. Moreover, while being fast, ALOLA is also efficient being able

to control performance degradations to at most 13% compared to the optimal results produced by MMCKP-DP. Both the presented solution strategies (MMCKP-DP and ALOLA) assume *Deadline Partitioning Fair* (DP-Fair) [17], a well known optimal multiprocessor scheduler, as the underlying scheduling mechanism.

The rest of the paper is organized as follows. In the next section, we present a detailed description of the problem considered in this work. The MMCKP based formulation for adaptive resource allocation and the heuristic solution approach (ALOLA) are discussed in Sub-sections II-A and II-B, respectively. Experiments and results are presented in Section III. A case study based on a real-world application is presented in Section IV. Finally, the paper is concluded in Section V. We now present a brief overview on the DP-Fair algorithm before presenting the problem description in the next section.

An Overview of DP-Fair Scheduling: DP-Fair is an optimal two level hierarchical homogeneous multi-resource scheduler. At the outer level, time is partitioned into slices, demarcated by the periods/deadlines of all jobs in the system. At the inner level, within a given time slice of length ts time slots (say), each task T_i is allocated a workload equal to its proportional fair share, $shr_i = (e_i/p_i) * ts$ (where, e_i denotes the execution requirement and p_i, the period of T_i) and assigned to one or more resources (processor/bus resource) for scheduling. DP-Fair is an optimal algorithm which ensures full resource utilization. Given a set of n tasks to be scheduled on M homogeneous resources, DP-Fair guarantees a feasible schedule provided the following constraint is satisfied:

$$\sum_{i=1}^{n} \frac{e_i}{p_i} \leq M \tag{1}$$

Using a combination of techniques including *DP-Wrap*, *mirroring* and *fairness-oblivious intra-slice execution*, DP-Fair is able to ensure atmost $n-1$ context switches and $M-1$ migrations per time slice [17]. These facets make DP-Fair a lucrative resource allocation technique which can efficiently combine the twin benefits of *high resource utilization* and *low scheduling related overheads*.

II. Problem Description

This work considers the problem of off-line scheduling of a set of n persistent real-time periodic tasks with multiple service-levels, on a system consisting of M homogeneous processors and B homogeneous buses. The task deadlines are implicit, that is, deadlines of all tasks are same as their periods. All processors and buses are synchronized in time. Time is represented on a discrete scale and is measured as an integral number of time slots. Each task T_i has $|SL_i|$ alternative service-levels $SL_i = \{sl_{i1}, sl_{i2}, \ldots, sl_{i|SL_i|}\}$, where each service-level sl_{ij} has an associated computation requirement e_{ij} to be completed within a period p_{ij}. Associated with each task, there is also an input message which must be received from a designated sensor prior to the commencement of its computation, via bus. All tasks produce an output message at the completion of their computations which is transmitted over a bus to one or more designated actuators. Thus, corresponding to each service-level sl_{ij}, a task T_i has input and output message transmission demands of mx_{ij} and my_{ij} time slots (over bus(es)) respectively, within every period p_{ij}. We assume both tasks and messages to be preemptive. The computation and total communication resource demands of task T_i at service-level j are denoted by wt_{ij} ($wt_{ij} = \frac{e_{ij}}{p_{ij}}$) and wm_{ij} ($wm_{ij} = \frac{mx_{ij}}{p_{ij}} + \frac{my_{ij}}{p_{ij}}$), respectively. Here, wt_{ij} and wm_{ij} are referred to as the task weight and message weight of T_i for the j^{th} service-level. Without loss of generality, we assume that the period of a task is equal to the sampling period of its input sensor and inverse of the frequency of actuation of its output. Let, T_{ij}^k denote the k^{th} instance of task T_i at service-level sl_{ij}, with mx_{ij}^k being its input message and my_{ij}^k the output message. The designed algorithm must guarantee that mx_{ij}^k (from a sensor) is transmitted and received during the previous period so that the message is available as input to the task at the beginning of the current period. Similarly, the output message my_{ij}^k produced by T_{ij}^k in the current period should be transmitted over the bus to designated actuators during the next period. This pipelined design criterion ensures the completion of one task execution every period, as illustrated in Figure 1.

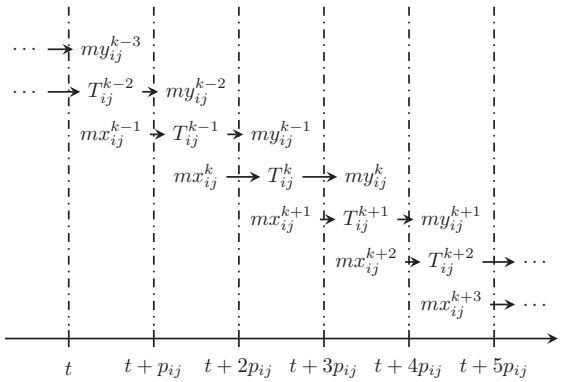

Fig. 1: Pipelined message transmission and execution over a synchronized system of homogeneous processors and buses

Successful execution of task T_i at a certain service-level sl_{ij} delivers a distinct *quality of service* (QoS) which is measured in terms of a numeric reward[1] value QoS_{ij}. Higher the service-level of execution, higher the QoS achieved by a task, but higher also becomes its resource demands in terms of task and message weights. Thus, computational requirements have a monotonically increasing relationship with service-levels. The actual numeric values of reward assigned to the different service-levels of a task depend on the system wide criticality/importance attributed to the task's different service-level.

[1]The terms *QoS* and *reward* mean the same and has been used interchangeably in the rest of the paper.

Prior to schedule generation, an appropriate service-level is selected for each task such that the aggregate QoS obtained by the system is maximized while guaranteeing feasible schedules for both the tasks' computation demands as well as messages (Given the appropriate task service-levels, two different DP-Fair schedulers have been employed in parallel, one of which allocates processor resources to meet computation demands of the tasks while other allocates bus resources to schedule task messages). For any task T_i, we define a binary decision variable x_{ij} corresponding to each of its service-levels. x_{ij} is set to 1, if T_i is executed at sl_{ij}. From Equation 1, it is clear that, given a selected service-level say, sl_{ij} for each task, feasible execution may be guaranteed if the following constraints hold:

$$\sum_{i=1}^{n} \sum_{j=1}^{|SL_i|} x_{ij} * wt_{ij} \leq M \tag{2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{|SL_i|} x_{ij} * wm_{ij} \leq B \tag{3}$$

$$\sum_{j=1}^{|SL_i|} x_{ij} = 1, \qquad \text{for } i = 1, 2, \ldots, n \tag{4}$$

The objective of the problem discussed above may be symbolically expressed as:

$$max \sum_{i=1}^{n} \sum_{j=1}^{|SL_i|} x_{ij} * QoS_{ij} \tag{5}$$

A. An Optimal Solution Approach (MMCKP-DP)

The objective function in Equation 5 along with the constraints in Equations 2, 3 and 4 together constitute a *Multi-Dimensional Multiple-Choice Knapsack Problem* (MMCKP). It may be observed that this MMCKP formulation has an optimal substructure and hence a solution approach based on *Dynamic Programming* (DP) becomes a natural choice. Thus, the above MMCKP may be equivalently represented as a recursive DP formulation as depicted in Equation 6 below:

$$f(k, \beta, \gamma) = \max_{j=1}^{|SL_k|} \begin{cases} f(k-1, \beta - wt_{kj}, \gamma - wm_{kj}) + QoS_{kj}, \\ \quad \text{[If } wt_{kj} \leq \beta, wm_{kj} \leq \gamma \text{ and} \\ \quad f(k-1, \beta - wt_{kj}, \gamma - wm_{kj}) > 0\text{]} \\ 0, \quad \text{[Otherwise]} \end{cases} \tag{6}$$

Here, the function $f(k, \beta, \gamma)$ recursively returns the maximum achievable aggregate QoS/reward considering $1, 2, \ldots, k$ tasks, while limiting computation and communication resource consumption below β ($0 < \beta \leq M$) and γ ($0 < \gamma \leq B$), respectively. The constraint: $f(k-1, \beta - wt_{kj}, \gamma - wm_{kj}) > 0$, in Equation 6, ensures that while determining the service-level for the k^{th} task, all tasks from $1, 2, \ldots, k-1$ have at least been assigned at their minimum service-levels. It may be noted that β and γ are free to take any arbitrary values within the continuous ranges (0, M] and (0, B], respectively. Therefore, to make β and γ applicable in the DP formulation, the above continuous ranges have been discretized in terms of a unit called $tics$. Thus in Equation 6, both β and γ have been measured as integral number of $tics$ ($(1 \leq \beta \leq \frac{M}{tics})$ and $(1 \leq \gamma \leq \frac{B}{tics})$). The necessary base condition corresponding to the above recursion is presented in Equation 7.

$$f(1, \beta, \gamma) = \max_{j=1}^{|SL_1|} \begin{cases} QoS_{1j}, & \text{[If } wt_{1j} \leq \beta, wm_{1j} \leq \gamma\text{]} \\ 0, & \text{[Otherwise]} \end{cases} \tag{7}$$

Time Complexity of MMCKP-DP: An implementation of the above DP formulation requires the generation of partial solutions $f(k, \beta, \gamma)$ for different values of the number of tasks (k), as well as all distinct values (β and γ) considered for processor and communication capacities. For any given value of $\langle k, \beta, \gamma \rangle$, the partial optimal solution is provided by choosing that service-level for the k^{th} task which delivers the maximum QoS. Therefore the overall time complexity becomes,

$$O\left(\sum_{i=1}^{n} |SL_i| \times \frac{M}{tics} \times \frac{B}{tics}\right) \tag{8}$$

where, $|SL_i|$ denotes the number of service-levels corresponding to task T_i.

Space Complexity of MMCKP-DP: At any intermediate stage, an efficient implementation of MMCKP-DP necessitates memorization of all partial solutions corresponding to the $(k-1)^{th}$ task when partial solutions $f(k, \beta, \gamma)$ for the k^{th} task is being derived. As there are $\frac{M}{tics} \times \frac{B}{tics}$ partial solutions for the k^{th} task and each such solution is an enumeration of the service-levels assigned to tasks $T_1, T_2, \ldots, T_{k-1}$, the overall space complexity is given by,

$$O\left(n \times \frac{M}{tics} \times \frac{B}{tics}\right) \tag{9}$$

where, n denotes the number of tasks.

The MMCKP-DP strategy suffers from significantly high overheads which steeply increase as the number of tasks, service-levels, processors and buses in the system becomes larger. The simulation based experimental evaluation shows that even on moderately large systems consisting of 90 tasks with 5 service-levels each, 16 processors and 4 buses, the MMCKP-DP incurs a run-time of more than 1 hour 20 minutes and approximately 68 GB main memory when $tics = 0.001$. Hence, we propose a fast yet efficient heuristic algorithm called *Accurate Low Overhead Level Allocator* (ALOLA), which attempts to achieve the same objective of maximizing aggregate rewards such that both computation and communication resource demands are satisfied.

B. Accurate Low Overhead Level Allocator (ALOLA)

ALOLA is a greedy but balanced heuristic service-level allocation approach that proceeds level by level so that a high aggregate QoS may be acquired by the system at much lower complexity compared to the optimal MMCKP-DP strategy. The mechanism starts by storing all tasks in a max-heap (based on a key $cost_i$) and assigning base service-levels to all tasks. The algorithm then proceeds by repeatedly extracting the task at the root of the heap, incrementing its service-level by 1, updating its $cost$ value and reheapifying it, until residual resources are completely exhausted, or all the tasks have been assigned their maximum possible service-levels. Algorithm 1 depicts a step-wise description of ALOLA.

The effectiveness of the solution hinges on the design of the prioritization key, $cost_i$. In order to obtain good and acceptable solutions, ALOLA must not only consider individual QoS gains ($\Delta QoS_i^{j,l}$) during service-level enhancements (from level j to l), but also the amount of incremental consolidated resource ($\Delta CR_i^{j,l}$) required during such an enhancement process. Therefore, the optimization objective gets transformed from QoS as in MMCKP-DP to ($\Delta QoS/\Delta CR$) in ALOLA. ΔCR produces a measure of the incremental consolidated resource consumption combining both computation and communication resource demands and is defined as follows:

$$\Delta CR_i^{j,l} = (1-\alpha) * \Delta wt_i^{j,l} + \alpha * \Delta wm_i^{j,l} \tag{10}$$

Here, $\Delta wt_i^{j,l}$ and $\Delta wm_i^{j,l}$ are respectively the extra computation and communication resource demands of task T_i corresponding to service-level enhancement from level j to l. The term α is a constant and is defined as the ratio of the relative mean approximate bus utilization (ABU) of any task with respect to the mean approximate total resource utilization combining processors (APU) and buses (ABU). Hence, α is symbolically represented as follows:

$$\alpha = \frac{ABU}{APU + ABU} \tag{11}$$

where, $\left[APU = \left(\sum_{i=1}^{n} \frac{1}{|SL_i|}\left(\sum_{j=1}^{|SL_i|} wt_{ij}\right)\right)\Big/ M\right]$ and $\left[ABU = \left(\sum_{i=1}^{n} \frac{1}{|SL_i|}\left(\sum_{j=1}^{|SL_i|} wm_{ij}\right)\right)\Big/ B\right]$. It may be observed that the parameters α and $(1-\alpha)$ provides a measure of the relative approximate bus utilization and processor utilization for any task considering all possible service-level assignments over all tasks. This makes α independent of specific service-levels assigned to the tasks at any time and thus, its value remains unchanged over the entire execution of the algorithm. Such a static definition of α helps us to control the overall complexity of the ALOLA algorithm. The key $cost_i$ which determines the urgency of each task T_i (towards service-level upgrade) within the heap, is as follows:

$$cost_i = max\left\{\frac{\Delta QoS_i^{j,j+1}}{\Delta CR_i^{j,j+1}}, \frac{\Delta QoS_i^{j,|SL_i|}}{\Delta CR_i^{j,|SL_i|}}\right\} \tag{12}$$

Here, $[\Delta QoS_i^{j,j+1}/\Delta CR_i^{j,j+1}]$ denotes the additional QoS received by the system per unit consolidated resource consumption, if T_i is upgraded from current service-level j to level $(j+1)$ and $[\Delta QoS_i^{j,|SL_i|}/\Delta CR_i^{j,|SL_i|}]$ represents the QoS gain per unit additional resource consumption, if T_i is enhanced from level j to its highest service-level $|SL_i|$.

It was observed that $cost_i$ (Equation 12) is able to provide an appropriate balance between the immediate gain obtained through a service-level enhancement from j to $(j+1)$ and the overall obtainable gain for task T_i is $(\Delta QoS_i^{j,|SL_i|}/\Delta CR_i^{j,|SL_i|})$. A situation where the consideration of such overall gains may be useful is as follows: Let us consider the relative urgency of two tasks T_i and $T_{i'}$ currently allocated service-level j and j' respectively (say), at an arbitrary intermediate stage of the resource allocation process using ALOLA. Assume, $[\Delta QoS_i^{j,j+1}/\Delta CR_i^{j,j+1}]$ is lower than $[\Delta QoS_{i'}^{j',j'+1}/\Delta CR_{i'}^{j',j'+1}]$. However,

$$\frac{\Delta QoS_i^{j,|SL_i|}}{\Delta CR_i^{j,|SL_i|}} >> \frac{\Delta QoS_{i'}^{j',|SL_{i'}|}}{\Delta CR_{i'}^{j',|SL_{i'}|}}.$$

In such a situation, if overall gain in QoS/reward is not considered as part of the key, $T_{i'}$ will be selected for upgradation by one level over T_i even if its overall gain $(\Delta QoS_i^{j,|SL_i|}/\Delta CR_i^{j,|SL_i|})$ is much greater than

$$max\left\{\frac{\Delta QoS_{i'}^{j',j'+1}}{\Delta CR_{i'}^{j',j'+1}}, \frac{\Delta QoS_{i'}^{j',|SL_{i'}|}}{\Delta CR_{i'}^{j',|SL_{i'}|}}\right\}.$$

A more severe case is that, if T_i's immediate gain $[\Delta QoS_i^{j,j+1}/\Delta CR_i^{j,j+1}]$ is relatively very low, T_i may be indefinitely starved in spite of potentially handsome overall gains. Defining the key $cost_i$ as,

$$max\left\{\frac{\Delta QoS_i^{j,j+1}}{\Delta CR_i^{j,j+1}}, \frac{\Delta QoS_i^{j,|SL_i|}}{\Delta CR_i^{j,|SL_i|}}\right\},$$

appropriately handles the situation.

Time Complexity of ALOLA: The complexity related to the assignment of the minimum service-level to all tasks and computation of their $cost_i$ values in lines 1-3, is $O(n)$. Building the max-heap in line 4 also takes $O(n)$ time. The *while* loop in lines 6-16 iterates at most $|SL_i|$ times for each task T_i and during each iteration, the heapify operation (line 16) incurs $O(\log n)$ time. So, the overall time complexity of the ALOLA algorithm becomes $O(\sum_1^n |SL_i| \times \log n)$.

Space Complexity of ALOLA: The space complexity of ALOLA is upper bounded by the memory required to store *task weights*, *message weights* and *QoS* values corresponding to all service-levels for each task. So, the overall space complexity of the ALOLA is $O(n \times \sum_1^n |SL_i|)$.

C. Example - Service-level Assignment

Consider $(n = 3)$ tasks T_1, T_2, and T_3 with $M = 2$ processors and $B = 1$ bus. The $\langle wt_{ij}, wm_{ij}, QoS_{ij}\rangle$ values

ALGORITHM 1: Accurate Low Overhead Level Allocator (ALOLA)

Input: A set of n tasks, M homogeneous processors and B homogeneous buses.
Output: Assignment of service-levels to tasks.

```
Assign minimum service-level to all tasks
forall tasks T_i do
    Compute cost_i using Equation 12
Create max-heap based on cost_i
Insert all tasks into max-heap
while max-heap is non-empty do
    Select root node T_i
    if Extra resource demands of T_i ≤ available resources then
        Upgrade service-level of T_i to the next one
        if current service-level < |SL_i| then
            Recompute cost_i using Equation 12
        else
            remove T_i from max-heap
    else
        remove T_i from max-heap
    Heapify max-heap
```

corresponding to different service-levels of each task are given in Table I. For these values, $APU = 0.95$, $ABU = 0.933$ and $\alpha = 0.496$. The overall residual computation and communi-

Task	SL_i	wt_{ij}	wm_{ij}	QoS_{ij}
T_1	sl_{11}	0.3	0.1	2
	sl_{12}	0.6	0.2	4
	sl_{13}	0.7	0.3	5
T_2	sl_{21}	0.6	0.3	4
	sl_{22}	0.7	0.4	7
	sl_{23}	0.8	0.4	8
T_3	sl_{31}	0.5	0.2	2
	sl_{32}	0.7	0.3	6
	sl_{33}	0.8	0.6	7

TABLE I: Tasks parameters

cation capacities after allocating the minimum service-level sl_{i1} to each task T_i become: $(M - \sum_{i=1}^{n} wt_{i1} =)$ 0.6 and $(B - \sum_{i=1}^{n} wm_{i1} =)$ 0.4, respectively. The initial key values corresponding to the three tasks are $cost_1 = 9.96$, $cost_2 = 30$, $cost_3 = 26.59$. A Max Heap (H) is built using these key values. T_2 with the highest key value (currently, at the root of the max-heap) is extracted from the heap, its service-level is enhanced from sl_{21} to sl_{22}, $cost_2$ is updated ($cost_2$ becomes 19.82) and then T_2 is re-heapifyed. Now, T_3 with the currently highest key value is extracted from the heap, its service-level is upgraded to sl_{32}, the key value is recomputed ($cost_3$ becomes 5.02) and T_3 is re-heapifyed. After this, T_2 again becomes the task with the highest key value and hence, it is upgraded to sl_{23} subsequent to its extraction from the heap. T_2, which has reached its highest enhancement level, is removed from further consideration and so, it is not reinserted into the heap. Now, T_1 which is the task with the highest key, is extracted from the heap but its service-level cannot be enhanced further to sl_{12} as the additional resources necessary for this enhancement is more than the currently remaining residual resources. T_1 is not considered further. Due to the same reason, T_3 also cannot be considered for further enhancement. The heap becomes empty and the algorithm terminates. The final service-levels for T_1, T_2 and T_3 are sl_{11}, sl_{23} and sl_{32}, respectively and the aggregate QoS fetched by the system is 16.

D. Offline Schedule Generation

The ALOLA algorithm provides the final service-levels at which each task should be executed. Subsequent to this, the DP-Fair scheduling technique is employed to allocate tasks on processors and messages on buses, for the entire duration $\mathcal{H}$ corresponding to the system's hyperperiod. These offline schedules are then used for online execution and repeat every hyperperiod.

We explain the generation of DP-Fair schedules by continuing with the illustrative example presented in section II-C. For the service-levels selected by ALOLA, the task weights of T_1, T_2 and T_3 are 0.3, 0.8 and 0.7 respectively, while the corresponding message weights are 0.1, 0.4 and 0.3 respectively. The tasks are first partitioned into the available processors and buses by: (i) lining them up over two separate scales based on task and message weights, as shown in Figures 2 and 5 respectively, and (ii) extracting unit length chunks from designated scales for allocation on to distinct processors and buses, as depicted by Figures 3 and 6. It may be observed from Figure 3 that after partitioning, task T_2 becomes a migrating task with one part (0.7-out-of-0.8) allocated to processor M_1 and the other part (0.1-out-of-0.8) allocated to M_2.

Let the execution times, message transmission times and periods ($\langle e_{ij}, m_{ij}, p_{ij} \rangle$) for the tasks (at their selected service-levels) be: T_1 $\langle 9, 3, 30 \rangle$, T_2 $\langle 12, 6, 15 \rangle$, T_3 $\langle 7, 3, 10 \rangle$. To construct a DP-Fair schedule, the hyperperiod of duration $\mathcal{H}$ = 30 (LCM(30, 15, 10), where LCM is the *least common multiple*), is divided into slices $ts_1 = 10, ts_2 = 5, ts_3 = 5$ and $ts_4 = 10$, demarcated by the periods/deadlines of all jobs in $\mathcal{H}$. The tasks and messages are then allocated resource shares in each time slice in proportion to their allocated weights on appropriate processors and buses, as designated by the partitioning process. For example, based on the task weight partition obtained in Figure 3, T_1 and T_3 are allocated execution shares 3 and 7 on M_1 and M_2 respectively, in time slice ts_1. Also T_2, the migrating task, receives shares 7 and 1 on M_1 and M_2 in ts_1. Figure 4 depicts the schedule of task execution over all the four time slices in $\mathcal{H}$. It may be observed from the figure that the scheduling sequences in consecutive time slices are mirrored in order to avoid task migrations at time slice boundaries. The system is able to limit context switches to 2 and number of migrations to 1 per time slice on our example three-task-two-processor system. The message schedule is obtained in a way very similar to the generation of the task schedule using the DP-Fair algorithm. Figure 7 depicts the obtained message schedule for the entire hyperperiod $\mathcal{H}$.

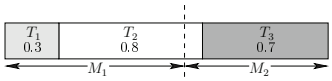

Fig. 2: Tasks lined up on a scale based on task weights. The dotted line partitions the chunks of at most unit length

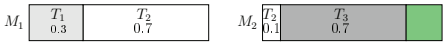

Fig. 3: Tasks partitioned on to processors M_1 and M_2. The green colored block in M_2 represents unused capacity

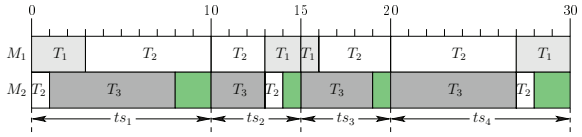

Fig. 4: Task schedule for hyperperiod $\mathcal{H}$

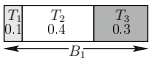

Fig. 5: Message weight partitioning

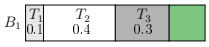

Fig. 6: Allocated message weights to bus

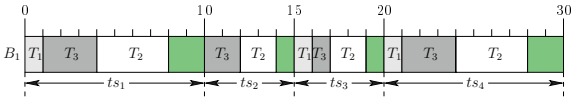

Fig. 7: Message schedule for hyperperiod $\mathcal{H}$

III. Experiments and Results

In this section, we experimentally evaluate the performance of the ALOLA algorithm and compare it against the optimal MMCKP-DP strategy in terms of both performance (total *QoS* acquired) and execution time required. The proposed work has been evaluated using simulation based experiments

A. Data Generation Framework

The experimentation framework is used as follows: The data sets consist of randomly generated hypothetical periodic tasks whose periods (p_{i1}), task weights (e_{i1}/p_{i1}) and message weights (m_{i1}/p_{i1}) for lowest/base service-level have been generated from normal distributions with mean $\langle mean(\mu),\ standard\ deviation(\sigma)\rangle$ values being $\langle 2000, 400\rangle$, $\langle 0.2, 0.1\rangle$ and $\langle 0.2, 0.1\rangle$, respectively. The reward values (QoS_{i1}) of the tasks are generated from a uniform distribution within the range from 20 to 200. Given n tasks, M processors and B buses, the processor utilization PU and bus utilization BU corresponding to a data set generated using the above mentioned distributions are given by:

$$PU = \frac{1}{M}\sum_{i=1}^{n} wt_{i1} \qquad \textit{and,} \qquad BU = \frac{1}{B}\sum_{i=1}^{n} wm_{i1} \quad (13)$$

Experiments have been conducted with data sets having different PU and BU values varying within the range from 0.6 to 1. To obtain a desired and fixed value of PU and BU corresponding to a given data set, the generated weights wt_{i1} and wm_{i1} are appropriately scaled. All tasks are assumed to have 5 service-levels. The weights (wt_{ij} or wm_{ij}) for non-base service-levels (starting from level 2) of the tasks are assigned uniform random values bounded between 110% and 120% of the weights ($wt_{i(j-1)}$ or $wm_{i(j-1)}$) corresponding to their immediately lower service-levels. The rewards (QoS_{ij}) for any task T_i increase monotonically as service-levels become higher. The values of the rewards have been chosen randomly from the range 20 to 200, while ensuring that the random reward value for a task at a given service-level is higher than the reward values at lower service-levels.

We have conducted experiments with data sets having 15, 30, 45 and 60 tasks, #processors varying between 2 to 8 and #buses varying between 1 to 4. All data points in the plots presented below are obtained by taking the average of the results from 50 different runs of a given experiment, executed with distinct data sets for a fixed set of parameters. All experiments have been carried out on a 2.5 GHz core, with 128 GB of physical memory.

B. QoS Measurements

The performance of ALOLA and MMCKP-DP have been evaluated and compared using a metric called *NSQP* (Normalized System-level QoS Percentage). NSQP is defined as the ratio of the total reward acquired through a given service-level assignment with respect to the maximum achievable reward (when the highest service-level is always assigned), in percentage. That is,

$$NSQP = \frac{\sum_{i=1}^{n} QoS_{ij}}{\sum_{i=1}^{n} QoS_{i|SL_i|}} \times 100 \quad (14)$$

where, j denotes the service-level selected for the i^{th} task. Two categories of experiments have been conducted. The first category conducts performance analysis in scenarios where bus resources are unlimited, that is, the bus capacity remains underloaded even when all tasks are assigned highest service-levels. For the second category, we assume unlimited processor resources.

Results (Category 1): Figure 8 presents the NSQP plots obtained for both ALOLA and MMCKP-DP in systems consisting of 15, 30, 45 or 60 tasks, 8 processors and processor utilization with respect to minimum service-levels PU (refer Equation 13) varying between 0.6 and 1. It may be observed from the figure that as is obvious, QoS based rewards decrease monotonically as PU increases. This is because the probability of attaining higher service-levels is reduced with the increase in PU. The optimal algorithm is seen to significantly outperform the heuristic strategy ALOLA (by about 5% to 10%) for values of PU between ~ 0.6 and ~ 0.85. However, as system load increases further and PU approaches $\sim$ 1, both the optimal and heuristic strategies

select only the lowest service-level for each task. This is indicated in Figure 8, which shows that both MMCKP-DP and ALOLA deliver the same NSQP values for the same number of tasks, when $PU = 1$. In Table II, we present the average NSQP values acquired by both the algorithms for a system consisting of 45 tasks, five distinct values of PU (0.6, 0.7, 0.8, 0.9 and 1) and four distinct number of processors (2, 4, 6 and 8). The table shows that the performance of both algorithms degrade as system load increases and that MMCKP-DP outperforms ALOLA in all cases except fully loaded systems ($PU = 1$), conforming the observations made for Figure 8. However, the NSQP values are not seen to significantly vary with increase in the number of processors for any given value of PU.

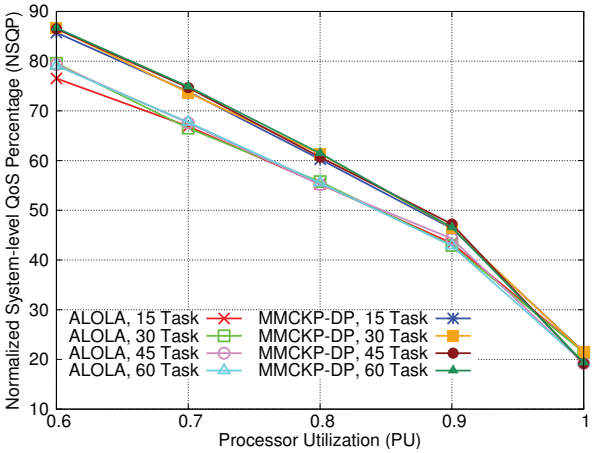

Fig. 8: Processor Utilization (PU) Vs. NSQP

PU	Algorithm	#Processors			
		1	2	3	4
0.6	MMCKP-DP	86.94	86.88	86.2	86.51
	ALOLA	78.25	79.11	78.84	79.29
0.7	MMCKP-DP	74.5	74.24	73.91	74.68
	ALOLA	66.93	66.71	67.11	67.57
0.8	MMCKP-DP	61.93	60.64	61.84	60.68
	ALOLA	54.8	54.88	55.91	55.14
0.9	MMCKP-DP	47.29	46.44	46.58	47.22
	ALOLA	43.03	42.84	43.52	44.3
1	MMCKP-DP	21.16	19.55	19.82	19.19
	ALOLA	21.16	19.55	19.82	19.19

TABLE II: QoS (NSQP) of MMCKP-DP and ALOLA for varying Processor Utilization (PU) and #Processors

Results (Category 2): Experiments for category 2 have been conducted for similar scenarios as category 1, although here, we vary bus utilization BU (refer Equation 13) from 0.6 to 1, instead of processor utilization PU. As depicted in Figure 9 and Table III, the trends of the results corresponding to this category are almost identical to those obtained for category 1.

C. Time Measurements: Results

The performance of the designed algorithms with respect to incurred running times have been evaluated by measuring

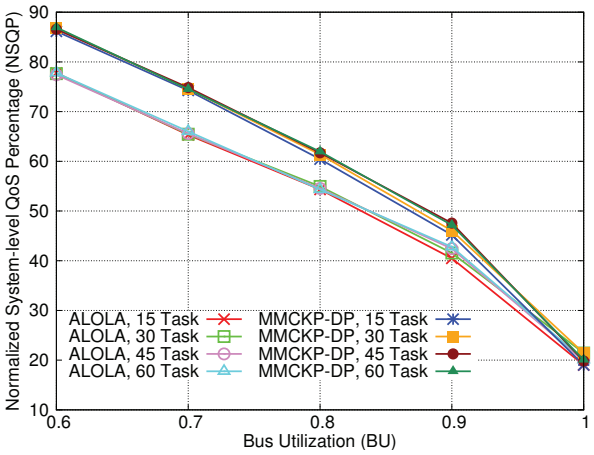

Fig. 9: Bus Utilization (BU) Vs. NSQP

BU	Algorithm	#Buses			
		1	2	3	4
0.6	MMCKP-DP	87.38	87.02	87.13	86.55
	ALOLA	76.06	77.49	77.76	77.4
0.7	MMCKP-DP	75.25	74.96	75.16	74.83
	ALOLA	65.51	65.55	65.6	65.58
0.8	MMCKP-DP	62.69	62.23	61.63	61.71
	ALOLA	53.71	53.83	54.37	54.54
0.9	MMCKP-DP	46.72	47.39	47.41	47.57
	ALOLA	40.39	41.35	42.16	42.43
1	MMCKP-DP	22.7	21.02	19.99	19.82
	ALOLA	22.7	21.02	19.99	19.82

TABLE III: QoS (NSQP) of MMCKP-DP and ALOLA for varying Bus Utilization (BU) and #Buses

their actual average run-times over various data sets (here, run-time denotes only service-level selection overheads and does not include running time of the scheduled tasks on the processors/buses). Then, we have determined the *speedup* achieved by ALOLA over MMCKP-DP. That is,

$$speedup = \frac{\text{Running time of MMCKP-DP}}{\text{Running time of ALOLA}} \tag{15}$$

Speedups have been measured for both the categories of experiments for which performance evaluation in terms of QoS was conducted.

Results: Figures 10 and 11 depict results for the running times incurred by ALOLA for category 1 (which considers 15 to 60 tasks, 4 and 8 processors, 2 buses, with PU being varied between 0.6 and 1) and category 2 (which considers 15 to 60 tasks, 8 processors, 2 and 4 buses, with BU being varied between 0.6 and 1) experiments, respectively. From both figures, it may be seen that run-time of ALOLA decreases as PU (category 1: Figure 10) or BU (category 2: Figure 11) increases. This is obvious because with the increase of utilization, residual capacities reduce and exhaust more quickly during the execution of ALOLA, thus giving the strategy less opportunity to select higher service-levels. For any given utilization, running times increase with the number

of tasks. This is also expected because the time complexity of ALOLA is directly proportional to the number of tasks and service-levels. However, for any given utilization and number of tasks, the run-times remain unaffected by the change in number of processors (category 1: Figure 10) or number of buses (category 2: Figure 11). In Table IV, we present the speedups achieved by ALOLA over MMCKP-DP for 15 to 90 tasks, number of processors varying from 2 to 16 (with #buses B = 2, PU = 0.7; underloaded bus capacity) and number of buses varying between 1 to 4 (with #processors M = 16, BU = 0.7; underloaded processor capacity). The actual speedups are 10^6 times the values (say, x) shown in the table (that is, actual speedup = $x \times 10^6$ times). It may seen that speedups increase with the number of tasks, processors and/or buses. The reason may be attributed to the complexity of MMCKP-DP which is highly sensitive to the number of tasks, their service-levels, as well as the number of processors and buses (refer Equation 8). In comparison, the complexity of ALOLA exhibits significantly lower sensitivity to the number of tasks and service-levels (refer Section II-B). ALOLA's running time is also dependent on the residual processor and bus capacities. Therefore, run-time of ALOLA does not significantly vary with changes in the number of processors and/or buses, being only indirectly sensitive to them.

Number of Tasks	Speedup ($x \times 10^6$)								
	#Processors					#Buses			
	2	4	6	8	16	1	2	3	4
15	0.12	0.7	1.13	1.66	2.89	1.46	2.9	4.52	6.33
30	0.62	1	1.55	2.14	4.06	2.52	5.92	8.11	9.25
45	0.75	1.51	2.36	3.02	6.03	2.88	7.19	10.53	13.58
60	0.89	1.97	2.81	3.78	8.08	3.54	8.52	10.95	14.71
75	1.11	2.26	3.12	4.45	8.14	5.98	9.79	14.34	18.32
90	1.21	2.58	4.19	4.7	10.07	5.9	11.48	17.4	24.23

TABLE IV: Speedup of ALOLA with respect to MMCKP-DP

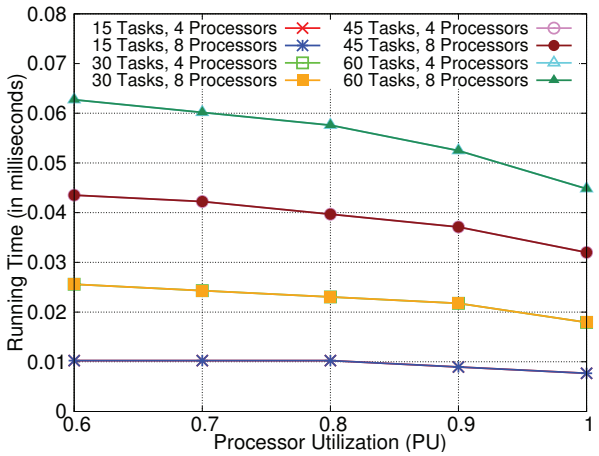

Fig. 10: ALOLA: Processor Utilization (PU) Vs. Running Time

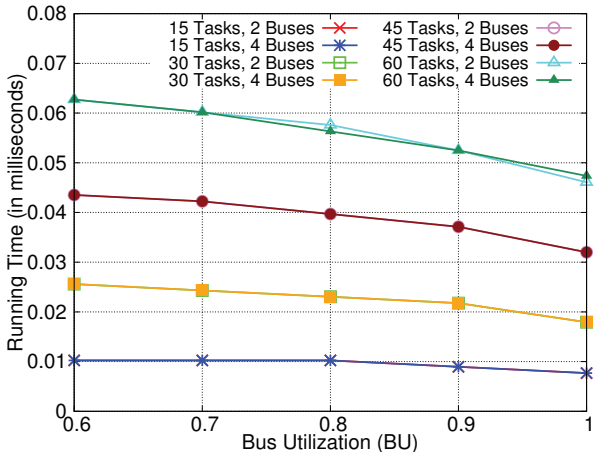

Fig. 11: ALOLA: Bus Utilization (BU) Vs. Running Time

IV. Case Study

To illustrate the generic applicability of our proposed strategy to real world designs, we present a case study using an automated flight control system employed in modern avionic systems. The *Flight Management System* (FMS) in an aircraft performs several flight control functions like flight planning, navigation, guidance and control [18]. FMS employs four separate tasks, namely, *Guidance*, *Control*, *Slow Navigation* and *Fast Navigation*, to control the aircraft during flight. The *Guidance* task (say T_1) sets the reference trajectory of the aircraft with respect to altitude and heading. The *Control* task (say T_2) executes closed loop control functions that compute actuator commands based on reference trajectory and navigation sensor values. Actuator commands for numerous components including elevator, ailerons, rudder and throttle are executed by the control task in order to achieve desired reference altitude and heading for the aircraft. The *Slow Navigation* task (say T_3) reads data from sensors at low frequencies which are used by the less critical *Guidance* task to determine high-level altitude and heading commands. The *Fast Navigation* task (say T_4) on the other hand, is used to read data from sensors at higher frequencies which are used by the higher critical *Control* task. In addition to these basic tasks (Guidance, Control, Slow Navigation and Fast Navigation), a fighter aircraft like F-16 performs an additional task, *Missile Control* (say T_5), which monitors the aircraft radar to detect enemy targets and if a target has been detected, it fires a missile to destroy the target.

We have used the task set corresponding to FMS simulation of F-16 fighter aircraft, from [1]. Table V shows the execution times (e_i), message transmission times (m_i), periods (p_i), computation demands (wt_i), communication demands (wm_i) and QoS (QoS_i) for the different service-levels (SL_i) of all the five tasks mentioned above. Both algorithms MMCKP-DP and ALOLA run successfully on the task set in Table V. In this particular case, the heuristic algorithm ALOLA assigns the same service-levels to the tasks as the optimal algorithm MMCKP-DP. The assigned service-levels to tasks T_1, T_2, T_3, T_4, and T_5 by both ALOLA and MMCKP-DP are $sl_{13}, sl_{23}, sl_{32}, sl_{43}$, and sl_{52}, respectively. The aggregate QoS fetched by the

Tasks	SL_i	e_i [ms]	m_i [ms]	p_i [sec]	wt_i	wm_i	QoS_i
T_1 (Guidance)	sl_{11}	100	80	10	0.01	0.008	10
	sl_{12}	100	80	5	0.02	0.016	15
	sl_{13}	100	80	1	0.1	0.08	20
T_2 (Controller)	sl_{21}	80	100	5	0.016	0.02	1
	sl_{22}	60	80	1	0.06	0.08	100
	sl_{23}	80	100	1	0.08	0.1	104
	sl_{24}	60	80	0.2	0.3	0.4	120
	sl_{25}	80	100	0.2	0.4	0.5	124
T_3 (Slow Navigation)	sl_{31}	100	120	10	0.01	0.012	10
	sl_{32}	100	120	5	0.02	0.024	20
	sl_{33}	100	120	1	0.1	0.12	25
T_4 (Fast Navigation)	sl_{41}	60	70	5	0.012	0.014	1
	sl_{42}	60	70	1	0.06	0.07	100
	sl_{43}	60	70	0.2	0.3	0.35	120
T_5 (Missile Control)	sl_{51}	500	200	10	0.05	0.02	1
	sl_{52}	500	200	1	0.5	0.2	200

TABLE V: Task set of *Flight Management Systems* [1]

system due to this assignment is 464.

V. Conclusion

Many cyber-physical systems including those in the automotive domain often execute applications which necessitate combined scheduling on both processors and buses. In this work, we consider independent periodic tasks with multiple service-levels, where each instance of a task also needs to receive data from sensor(s) and produce data to actuator(s) through one or more system buses. The scheduling problem is modeled as a *Multi-dimensional Multiple-Choice Knapsack Problem formulation* and shown that conventional *Dynamic Programming* based solution approaches are very expensive. Therefore, we have proposed a heuristic strategy called *Accurate Low Overhead Level Allocator* (ALOLA), which is very fast producing speedups of the order of 10^6 times with respect to the optimal *Dynamic Programming* solution and also efficient, being able to control performance degradations to at most 13% compared to the optimal solutions.

References

[1] TF Abdelzaher, Ella M Atkins, and Kang G Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11):1170–1183, 2000.

[2] James H Anderson and Anand Srinivasan. Early-release fair scheduling. In *Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on*, pages 35–43. IEEE, 2000.

[3] James H Anderson and Anand Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 76–85. IEEE, 2001.

[4] Björn Andersson and Jan Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 33–40. IEEE, 2003.

[5] Hyeongboo Baek, Jinkyu Lee, and Insik Shin. Multi-level contention-free policy for real-time multiprocessor scheduling. *Journal of Systems and Software*, 137:36–49, 2018.

[6] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[7] Enrico Bini, Giorgio Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Karl-Erik Arzen, Vanessa Romero Segovia, and Claudio Scordino. Resource management on multicore systems: The ACTORS approach. *IEEE Micro*, pages 72–81, 2011.

[8] Hyeonjoong Cho, Binoy Ravindran, and E Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 101–110. IEEE, 2006.

[9] Hoon Sung Chwa, Hyoungbu Back, Jinkyu Lee, Kieu-My Phan, and Insik Shin. Capturing urgency and parallelism using quasi-deadlines for real-time multiprocessor scheduling. *Journal of Systems and Software*, 101:15–29, 2015.

[10] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[11] Sourav Ghosh, Ragunathan Rajkumar, Jeffery Hansen, and John Lehoczky. Scalable QoS-based resource allocation in hierarchical networked environment. In *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, pages 256–267. IEEE, 2005.

[12] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–15. IEEE, 2013.

[13] Ashok Khemka and RK Shyamasundar. An optimal multiprocessor real-time scheduling algorithm. *Journal of parallel and distributed computing*, 43(1):37–45, 1997.

[14] Anne Koziolek, Danilo Ardagna, and Raffaela Mirandola. Hybrid multi-attribute QoS optimization in component based software systems. *Journal of Systems and Software*, 86(10):2542–2558, 2013.

[15] Chen Lee, John Lehoczky, Dan Siewiorek, Ragunathan Rajkumar, and Jeff Hansen. A scalable solution to the multi-resource QoS problem. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 315–326. IEEE, 1999.

[16] Chen Lee, John Lehoezky, Ragunathan Rajkumar, and Dan Siewiorek. On quality of service optimization with discrete QoS options. In *Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE*, pages 276–286. IEEE, 1999.

[17] Greg Levin, Shelby Funk, Caitlin Sadowski, Ian Pye, and Scott Brandt. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, pages 3–13. IEEE, 2010.

[18] Sam Liden. The evolution of flight management systems. In *Digital Avionics Systems Conference, 1994. 13th DASC., AIAA/IEEE*, pages 157–169. IEEE, 1994.

[19] 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.

[20] Anna Minaeva, Benny Akesson, Zdenek Hanzalek, and Dakshina Dasari. Time-triggered co-scheduling of computation and communication with jitter requirements. *IEEE Transactions on Computers*, 2017.

[21] Ragunathan Rajkumar, Chen Lee, John Lehoczky, and Dan Siewiorek. A resource allocation model for QoS management. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 298–307. IEEE, 1997.

[22] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-time systems*, 26(2):161–197, 2004.

[23] Paul Regnier, George Lima, Ernesto Massa, Greg Levin, and Scott Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 104–115. IEEE, 2011.

[24] Dakai Zhu, Daniel Mossé, and Rami Melhem. Multiple-resource periodic scheduling problem: how much fairness is necessary? In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 142–151. IEEE, 2003.