An Optimal Real-Time Scheduling Algorithm for Multiprocessors

Hyeonjoong Cho*, Binoy Ravindran*, and E. Douglas Jensen‡

*ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{hjcho,binoy}@vt.edu

‡The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

We present an optimal real-time scheduling algorithm for multiprocessors — one that satisfies all task deadlines, when the total utilization demand does not exceed the utilization capacity of the processors. The algorithm called LLREF, is designed based on a novel abstraction for reasoning about task execution behavior on multiprocessors: the Time and Local Execution Time Domain Plane (or T-L plane). LLREF is based on the fluid scheduling model and the fairness notion, and uses the T-L plane to describe fluid schedules without using time quanta, unlike the optimal Pfair algorithm (which uses time quanta). We show that scheduling for multiprocessors can be viewed as repeatedly occurring T-L planes, and feasibly scheduling on a single T-L plane results in the optimal schedule. We analytically establish the optimality of LLREF. Further, we establish that the algorithm has bounded overhead, and this bound is independent of time quanta (unlike Pfair). Our simulation results validate our analysis on the algorithm overhead.

1 Introduction

Multiprocessor architectures (e.g., Symmetric Multi-Processors or SMPs, Single Chip Heterogeneous Multiprocessors or SCHMs) are becoming more attractive for embedded systems, primarily because major processor manufacturers (Intel, AMD) are making them decreasingly expensive. This makes such architectures very desirable for embedded system applications with high computational workloads, where additional, cost-effective processing capacity is often needed. Responding to this trend, RTOS vendors are increasingly providing multiprocessor platform support — e.g., QNX Neutrino is now available for a variety of SMP chips [12]. But this exposes the critical need for real-time scheduling for multiprocessors — a comparatively undeveloped area of real-time scheduling which has recently received significant research attention, but is not yet well supported by the RTOS products. Consequently, the impact of cost-effective multiprocessor platforms for embedded systems remains nascent.

One unique aspect of multiprocessor real-time scheduling is the degree of run-time migration that is allowed for job instances of a task across processors (at scheduling events). Example migration models include: (1) *full migration*, where jobs are allowed to arbitrarily migrate across processors during their execution. This usually implies a global scheduling strategy, where a single shared scheduling queue is maintained for all processors and a processor-wide scheduling decision is made by a single (global) scheduling algorithm; (2) *no migration*, where tasks are statically (off-line) partitioned and allocated to processors. At run-time, job instances of tasks are scheduled on their respective processors by processors' local scheduling algorithm, like single processor scheduling; and (3) *restricted migration*, where some form of migration is allowed—e.g., at job boundaries.

The partitioned scheduling paradigm has several advantages over the global approach. First, once tasks are allocated to processors, the multiprocessor real-time scheduling problem becomes a set of single processor real-time scheduling problems, one for each processor, which has been well-studied and for which optimal algorithms exist. Second, not migrating tasks at run-time means reduced run-time overhead as opposed to migrating tasks that may suffer cache misses on the newly assigned processor. If the task set is fixed and known a-priori, the partitioned approach provides appropriate solutions [3].

The global scheduling paradigm also has advantages, over the partitioned approach. First, if tasks can join and leave the system at run-time, then it may be necessary to reallocate tasks to processors in the partitioned approach [3]. Second, the partitioned approach cannot produce optimal real-time schedules — one that meets all task deadlines when task utilization demand does not exceed the total processor capacity — for periodic task sets [14], since the partitioning problem is analogous to the bin-packing problem which is known to be NP-hard in the strong sense. Third, in some embedded processor architectures with no cache and simpler structures, the overhead of migration has a lower

impact on the performance [3]. Finally, global scheduling can theoretically contribute to an increased understanding of the properties and behaviors of real-time scheduling algorithms for multiprocessors. See [10] for a detailed discussion on this.

Carpentar *et al.* [4] have catalogued multiprocessor real-time scheduling algorithms considering the degree of job migration and the complexity of priority mechanisms employed. (The latter includes classes such as (1) *static*, where task priorities never change, e.g., rate-monotonic (RM); (2) *dynamic but fixed within a job*, where job priorities are fixed, e.g., earliest-deadline-first (EDF); and (3) *fully-dynamic*, where job priorities are dynamic.)

The Pfair class of algorithms [2] that allow full migration and fully dynamic priorities have been shown to be theoretically optimal—i.e., they achieve a schedulable utilization bound (below which all tasks meet their deadlines) that equals the total capacity of all processors. However, Pfair algorithms incur significant run-time overhead due to their quantum-based scheduling approach [7, 14]: under Pfair, tasks can be decomposed into several small uniform segments, which are then scheduled, causing frequent scheduling and migration.

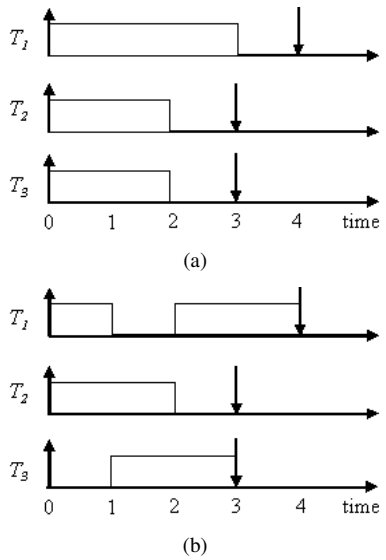

Figure 1: A Task Set that EDF Cannot Schedule On Two Processors

Thus, scheduling algorithms other than Pfair—e.g., global EDF [7–9, 13], have also been intensively studied though their schedulable utilization bounds are lower. Figure 1(a) shows an example task set that global EDF cannot feasibly schedule. Here, task T_1 will miss its deadline when the system is given two processors. However, there exists a schedule that meets all task deadlines; this is shown in Figure 1(b).

Interestingly, we observe in Figure 1(b) that the scheduling event at time 1, where task T_1 is split to make all tasks schedulable is not a traditional scheduling event (such as a task release or a task completion). This simple observation implies that we may need more scheduling events to split tasks to construct optimal schedules, such as what Pfair's quantum-based approach does. However, it also raises another fundamental question: is it possible to split tasks to construct optimal schedules, not at time quantum expiration, but perhaps at other scheduling events, and consequently avoid Pfair's frequent scheduling and migration overheads? If so, what are those scheduling events?

In this paper, we answer these questions.[1] We present an optimal real-time scheduling algorithm for multiprocessors, which is not based on time quanta. The algorithm called LLREF, is based on the fluid scheduling model and the fairness notion. We introduce a novel abstraction for reasoning about task execution behavior on multiprocessors, called the *time and local remaining execution-time plane* (abbreviated as the *T-L plane*). T-L plane makes it possible for us to envision that the entire scheduling over time is just the repetition of T-L planes in various sizes, so that feasible scheduling in a single T-L plane implies feasible scheduling over all times. We define two additional scheduling events and show when they should happen to maintain the fairness of an optimal schedule, and consequently establish LLREF's scheduling optimality. We also show that the overhead of LLREF is tightly bounded, and that bound depends only upon task parameters. Furthermore, our simulation experiments on algorithm overhead validates our analysis.

The rest of the paper is organized as follows: In Section 2, we discuss the rationale behind T-L plane and present the scheduling algorithm. In Section 3, we prove the optimality of our algorithm, establish the algorithm overhead bound, and report simulation studies. The paper concludes in Section 4.

2 LLREF Scheduling Algorithm

2.1 Model

We consider global scheduling, where task migration is not restricted, on an SMP system with M identical processors. We consider the application to consist of a set of tasks, denoted **T**=$\{T_1, T_2, ..., T_N\}$. Tasks are assumed to arrive periodically at their release times r_i. Each task T_i has an execution time c_i, and a relative deadline d_i which is the same as its period p_i. The utilization u_i of a task T_i is defined as c_i/d_i and is assumed to be less than 1. Similar to [1, 7], we assume that tasks may be preempted at any time, and are

[1] There exists a variant of EDF, called *Earliest Deadline until Zero Laxity* (EDZL), which allows another scheduling event to split tasks, but it does not offer the optimality for multiprocessors [11].

independent, i.e., they do not share resources or have any precedences.

We consider a non-work conserving scheduling policy; thus processors may be idle even when tasks are present in the ready queue. The cost of context switches and task migrations are assumed to be negligible, as in [1, 7].

2.2 Time and Local Execution Time Plane

In the *fluid* scheduling model, each task executes at a constant rate at all times [10]. The quantum-based Pfair scheduling algorithm, the only known optimal algorithm for the problem that we consider here, is based on the fluid scheduling model, as the algorithm constantly tracks the allocated task execution rate through task utilization. The Pfair algorithm's success in constructing optimal multiprocessor schedules can be attributed to *fairness* — informally, all tasks receive a share of the processor time, and thus are able to simultaneously make progress. P-fairness is a strong notion of fairness, which ensures that at any instant, no application is more than one quantum away from its due share (or fluid schedule) [2, 5]. The significance of the fairness concept on Pfair's optimality is also supported by the fact that task *urgency*, as represented by the task deadline is not sufficient for constructing optimal schedules, as we observe from the poor performance of global EDF for multiprocessors.

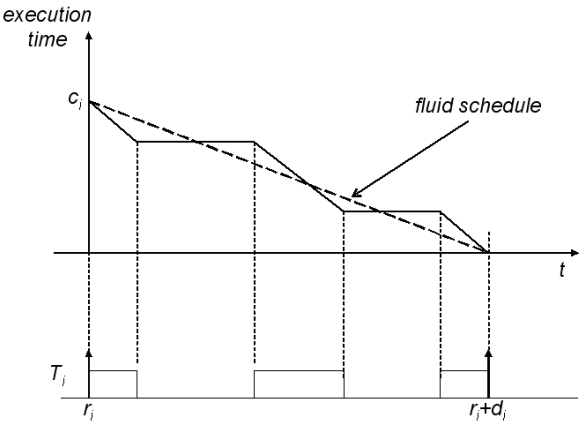

Figure 2: Fluid Schedule versus a Practical Schedule

Toward designing an optimal scheduling algorithm, we thus consider the fluid scheduling model and the fairness notion. To avoid Pfair's quantum-based approach, we consider an abstraction called the *Time and Local Execution Time Domain Plane* (or abbreviated as the T-L plane), where tokens representing tasks move over time. The T-L plane is inspired by the L-C plane abstraction introduced by Dertouzos *et al.* in [6]. We use the T-L plane to describe fluid schedules, and present a new scheduling algorithm that is able to track the fluid schedule without using time quanta.

Figure 2 illustrates the fundamental idea behind the T-L plane. For a task T_i with r_i, c_i and d_i, the figure shows a 2-dimensional plane with time represented on the x-axis and the task's remaining execution time represented on the y-axis. If r_i is assumed as the origin, the dotted line from $(0, c_i)$ to $(d_i, 0)$ indicates the fluid schedule, the slope of which is $-u_i$. Since the fluid schedule is ideal but practically impossible, the fairness of a scheduling algorithm depends on how much the algorithm approximates the fluid schedule path.

When T_i runs like in Figure 2, for example, its execution can be represented as a broken line between $(0, c_i)$ and $(d_i, 0)$. Note that task execution is represented as a line whose slope is -1 since x and y axes are in the same scale, and the non-execution over time is represented as a line whose slope is zero. It is clear that the Pfair algorithm can also be represented in the T-L plane as a broken line based on time quanta.

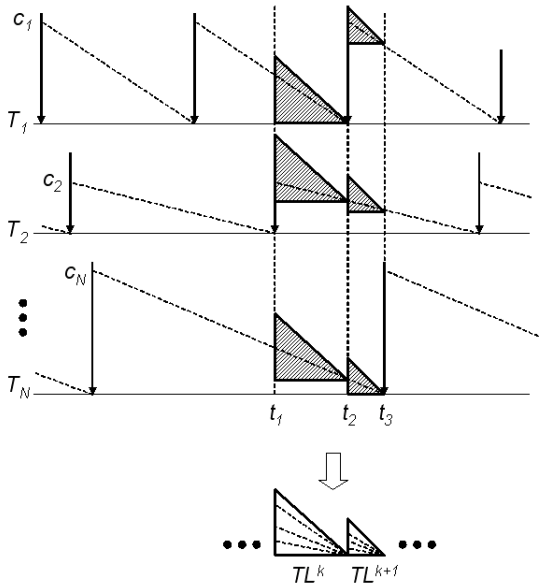

Figure 3: T-L Planes

When N number of tasks are considered, their fluid schedules can be constructed as shown in Figure 3, and a right isosceles triangle for all tasks is found between every two consecutive scheduling events. We call this as the T-L plane TL^k, where k is simply increasing over time. The size of TL^k may change over k. The bottom side of the triangle represents time. The left vertical side of the triangle represents a part of tasks' remaining execution time, which we call the *local remaining execution time*, l_i, which is supposed to be consumed before each TL^k ends. Fluid schedules for each task can be constructed as overlapped in each TL^k plane, while keeping their slopes.

2.3 Scheduling in T-L planes

The abstraction of T-L planes is significantly meaningful in scheduling for multiprocessors, because T-L planes are repeated over time, and a good scheduling algorithm for a single T-L plane is able to schedule tasks for all repeated T-L planes. Here, good scheduling means being able to construct a schedule that allows all tasks' execution in the T-L plane to approximate the fluid schedule as much as possible. Figure 4 details the k^{th} T-L plane.

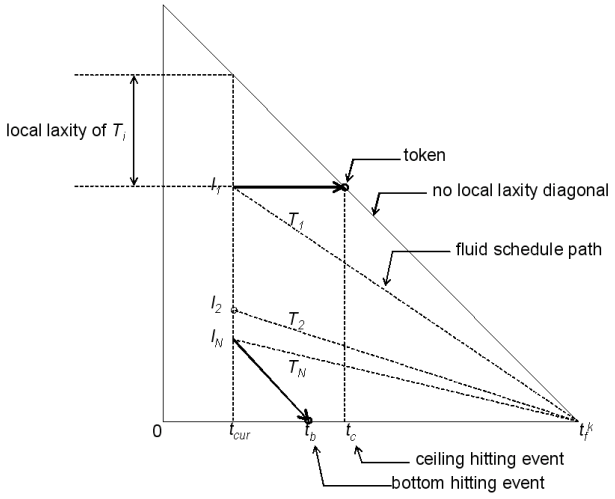

Figure 4: k^{th} T-L Plane

The status of each task is represented as a *token* in the T-L plane. The token's location describes the current time as a value on the horizontal axis and the task's remaining execution time as a value on the vertical axis. The remaining execution time of a task here means one that must be consumed until the time t_f^k, and not the task's deadline. Hence, we call it, *local* remaining execution time.

As scheduling decisions are made over time, each task's token moves in the T-L plane. Although ideal paths of tokens exist as dotted lines in Figure 4, the tokens are only allowed to move in two directions. When the task is selected and executed, the token moves diagonally down, as T_N moves. Otherwise, it moves horizontally, as T_1 moves. If M processors are considered, at most M tokens can diagonally move together. The scheduling objective in the k^{th} T-L plane is to make all tokens arrive at the rightmost vertex of the T-L plane—i.e., t_f^k with zero local remaining execution time. We call this successful arrival, *locally feasible*. If all tokens are made locally feasible at each T-L plane, they are possible to be scheduled throughout every consecutive T-L planes over time, approximating all tasks' ideal paths.

For convenience, we define the *local laxity* of a task T_i as $t_f^k - t_{cur} - l_i$, where t_{cur} is the current time. The oblique side of the T-L plane has an important meaning: when a token hits that side, it implies that the task does not have any local laxity. Thus, if it is not selected immediately, then it cannot satisfy the scheduling objective of local feasibility. We call the oblique side of the T-L plane *no local laxity diagonal* (or NLLD). All tokens are supposed to stay in between the horizontal line and the local laxity diagonal.

We observe that there are two time instants when the scheduling decision has to be made again in the T-L plane. One instant is when the local remaining execution time of a task is completely consumed, and it would be better for the system to run another task instead. When this occurs, the token hits the horizontal line, as T_N does in Figure 4. We call it the *bottom hitting event* (or event B). The other instant is when the local laxity of a task becomes zero so that the task must be selected immediately. When this occurs, the token hits the NLLD, as T_1 does in Figure 4. We call it the *ceiling hitting event* (or event C). To distinguish these events from traditional scheduling events such as task releases and task departures, we call events B and C *sub-events*.

To provide local feasibility, M of the *largest local remaining execution time* tasks are selected first (or LLREF) for every sub-event. We call this, the LLREF scheduling policy. Note that the task having zero local remaining execution time (the token lying on the bottom line in the T-L plane) is not allowed to be selected, which makes our scheduling policy non work-conserving. The tokens for these tasks are called *inactive*, and the others with more than zero local remaining execution time are called *active*. At time t_f^k, the time instant for the event of the next task release, the next T-L plane TL^{k+1} starts and LLREF remains valid. Thus, the LLREF scheduling policy is consistently applied for every event.

3 Algorithm Properties

A fundamental property of the LLREF scheduling algorithm is its scheduling optimality—i.e., the algorithm can meet all task deadlines when the total utilization demand does not exceed the demand capacity of the processors in the system. In this section, we establish this by proving that LLREF guarantees local feasibility in the T-L plane.

3.1 Critical Moment

Figure 5 shows an example of token flow in the T-L plane. All tokens flow from left to right over time. LLREF selects M tokens from N active tokens and they flow diagonally down. The others which are not selected, on the other hand, take horizontal paths. When the event C or B happens, denoted by t_j where $0 < j < f$, LLREF is invoked to make a scheduling decision.

We define the *local utilization* $r_{i,j}$ for a task T_i at time t_j to be $\frac{l_{i,j}}{t_f - t_j}$, which describes how much processor capacity

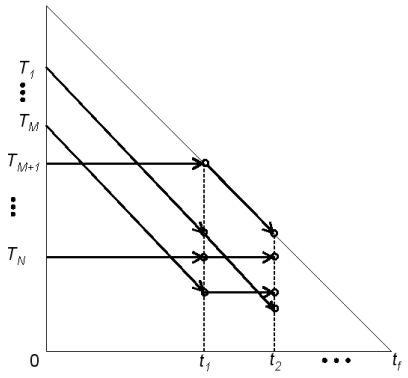

Figure 5: Example of Token Flow

needs to be utilized for executing T_i within the remaining time until t_f. Here, $l_{i,j}$ is the local remaining execution time of task T_i at time t_j. When k is cancelled, it implicitly means the current k^{th} T-L plane.

Theorem 1 (Initial Local Utilization Value in T-L plane)**.** *Let all tokens arrive at the rightmost vertex in the $(k-1)^{th}$ T-L plane. Then, the initial local utilization value $r_{i,0} = u_i$ for all task T_i in the k^{th} T-L plane.*

Proof. If all tokens arrive at t_f^{k-1} with $l_i = 0$, then they can restart in the next T-L plane (the k^{th} T-L plane) from the positions where their ideal fluid schedule lines start. The slope of the fluid schedule path of task T_i is u_i. Thus, $r_{i,0} = l_{i,0}/t_f = u_i$. □

Well-controlled tokens, both departing and arriving points of which are the same as those of their fluid schedule lines in the T-L plane (even though their actual paths in the T-L plane are different from their fluid schedule paths), imply that all tokens are locally feasible. Note that we assume $u_i \leq 1$ and $\sum u_i \leq M$.

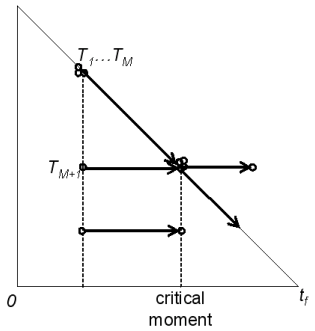

Figure 6: Critical Moment

We define *critical moment* to describe the sufficient and necessary condition that tokens are not locally feasible. (Local infeasibility of the tokens implies that all tokens do not simultaneously arrive at the rightmost vertex of the T-L plane.) Critical moment is the first sub-event time when more than M tokens simultaneously hit the NLLD. Figure 6 shows this. Right after the critical moment, only M tokens from those on the NLLD are selected. The non-selected ones move out of the triangle, and as a result, they will not arrive at the right vertex of the T-L plane. Note that only horizontal and diagonal moves are permitted for tokens in the T-L plane.

Theorem 2 (Critical Moment)**.** *At least one critical moment occurs if and only if tokens are not locally feasible in the T-L plane.*

Proof. We prove both the necessary and sufficient conditions.

Case 1. Assume that a critical moment occurs. Then, non-selected tokens move out of the T-L plane. Since only -1 and 0 are allowed for the slope of the token paths, it is impossible that the tokens out of the T-L plane reach the rightmost vertex of the T-L plane.

Case 2. We assume that when tokens are not locally feasible, no critical moment occurs. If there is no critical moment, then the number of tokens on the NLLD never exceeds M. Thus, all tokens on the diagonal can be selected by LLREF up to time t_f. This contradicts our assumption that tokens are not locally feasible. □

We define *total local utilization* at the j^{th} sub-event, S_j, as $\sum_i^N r_{i,j}$.

Corollary 3 (Total Local Utilization at Critical Moment)**.** *At the critical moment which is the j^{th} sub-event, $S_j > M$.*

Proof. The local remaining execution time $l_{i,j}$ for the tasks on the NLLD at the critical moment of the j^{th} sub-event is $t_f - t_j$, because the T-L plane is an isosceles triangle. Therefore, $S_j = \sum_{i=1}^{N} r_{i,j} = \sum_{i=1}^{M} \frac{t_f - t_j}{t_f - t_j} + \sum_{i=M+1}^{N} \frac{l_i}{t_f - t_j} > M$. □

From the task perspective, the critical moment is the time when more than M tasks have no local laxity. Thus, the scheduler cannot make them locally feasible with M processors.

3.2 Event C

Event C happens when a non-selected token hits the NLLD. Note that selected tokens never hit the NLLD. Event C indicates that the task has no local laxity and hence, should be selected immediately. Figure 7 illustrates this, where event C happens at time t_c when the token T_{M+1} hits the NLLD.

Note that this is under the basic assumption that there are more than M tasks, i.e., $N > M$. This implicit assumption

holds in Section 3.2 and 3.3. We will later show the case when $N \leq M$.

For convenience, we give lower index i to the token with higher local utilization, i.e., $r_{i,j} \geq r_{i+1,j}$ where $1 \leq i < N$ and $\forall j$, as shown Figure 7. Thus, LLREF select tasks from T_1 to T_M and their tokens move diagonally.

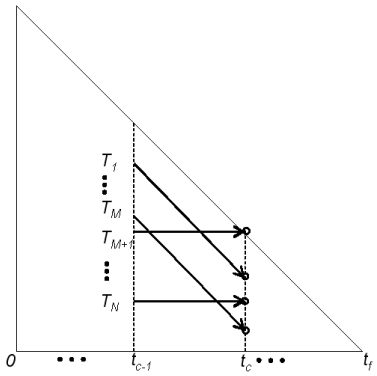

Figure 7: Event C

Lemma 4 (Condition for Event C)**.** *When* $1 - r_{M+1,c-1} \leq r_{M,c-1}$*, the event C occurs at time* t_c*, where* $r_{i,c-1} \geq r_{i+1,c-1}$*,* $1 \leq i < N$*.*

Proof. If the sub-event at time t_c is event C, then token T_{M+1} must hit the NLLD earlier than when token T_M hit the bottom of the T-L plane. The time when T_{M+1} hits the NLLD is $t_{c-1} + (t_f - t_{c-1} - l_{M+1,c-1})$. On the contrary, the time when the token T_M hits the bottom of the T-L plane is $t_{c-1} + l_{M,c-1}$.

$$t_{c-1} + (t_f - t_{c-1} - l_{M+1,c-1}) < t_{c-1} + l_{M,c-1}$$

$$1 - \frac{l_{M+1,c-1}}{t_f - t_{c-1}} < \frac{l_{M,c-1}}{t_f - t_{c-1}}.$$

Thus, $1 - r_{M+1} \leq r_M$ at t_{c-1}. □

Corollary 5 (Necessary Condition for Event C)**.** *Event C occurs at time* t_c *only if* $S_{c-1} > M(1 - r_{M+1,c-1})$*, where* $r_{i,c-1} \geq r_{i+1,c-1}$*,* $1 \leq i < N$*.*

Proof.

$$S_{c-1} = \sum_{i=1}^{M} r_{i,c-1} + \sum_{i=M+1}^{N} r_{i,c-1} > \sum_{i=1}^{M} r_{i,c-1} \geq M \cdot r_{M,c-1}.$$

Based on Lemma 4, $M \cdot r_{M,c-1} \geq M \cdot (1 - r_{M+1,c-1})$. □

Theorem 6 (Total Local Utilization for Event C)**.** *When event C occurs at* t_c *and* $S_{c-1} \leq M$*, then* $S_c \leq M$ *for* $\forall c$ *where* $0 < c \leq f$*, and* $r_{i,c-1} \geq r_{i+1,c-1}$*,* $1 \leq i < N$*.*

Proof. We let $t_c - t_{c-1} = t_f - t_{c-1} - l_{M+1,c-1}$ as δ. Total local remaining execution time at t_{c-1} is $\sum_{i=1}^{N} l_{i,c-1} = (t_f - t_{c-1})S_{c-1}$ and it decreases by $M \times \delta$ at t_c as M tokens move diagonally. Therefore,

$$(t_f - t_c)S_c = (t_f - t_{c-1})S_{c-1} - \delta M.$$

Since $l_{M+1,c-1} = t_f - t_c$,

$$l_{M+1,c-1} \times S_c = (t_f - t_{c-1})S_{c-1} - (t_f - t_{c-1} - l_{M+1,c-1})M.$$

Thus,

$$S_c = \frac{1}{r_{M+1,c-1}} S_{c-1} + (1 - \frac{1}{r_{M+1,c-1}})M. \qquad (1)$$

Equation 1 is a linear equation as shown in Figure 8.

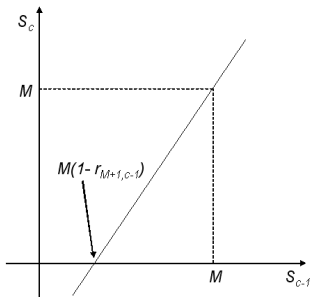

Figure 8: Linear Equation for Event C

According to Corollary 5, when event C occurs, S_{c-1} is more than $M \cdot (1 - r_{M+1,c-1})$. Since we also assume $S_{c-1} \leq M$, $S_c \leq M$. □

3.3 Event B

Event B happens when a selected token hits the bottom side of the T-L plane. Note that non-selected tokens never hit the bottom. Event B indicates that the task has no local remaining execution time so that it would be better to give the processor time to another task for execution.

Event B is illustrated in Figure 9, where it happens at time t_b when the token of T_M hits the bottom. As we do for the analysis of event C, we give lower index i to the token with higher local utilization, i.e., $r_{i,j} \geq r_{i+1,j}$ where $1 \leq i < N$.

Lemma 7 (Condition for Event B)**.** *When* $1 - r_{M+1,b-1} \geq r_{M,b-1}$*, event B occurs at time* t_b*, where* $r_{i,b-1} \geq r_{i+1,b-1}$*,* $1 \leq i < N$*.*

Proof. If the sub-event at time t_b is event B, then token T_M must hit the bottom earlier than when token T_{M+1} hits the NLLD. The time when T_M hits the bottom and the time

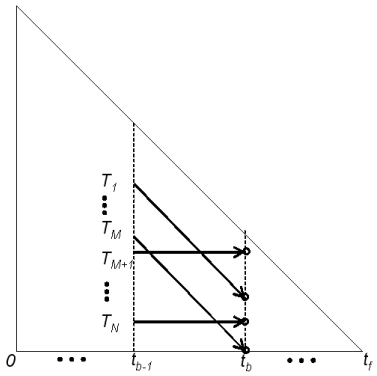

Figure 9: Event B

when T_{M+1} hits the NLLD are respectively, $t_{b-1} + l_{M,b-1}$ and $t_{b-1} + (t_f - t_{b-1} - l_{M+1,b-1})$.

$$t_{b-1} + l_{M,b-1} < t_{b-1} + (t_f - t_{b-1} - l_{M+1,b-1}).$$

$$\frac{l_{M,b-1}}{t_f - t_{b-1}} < 1 - \frac{l_{M+1,b-1}}{t_f - t_{b-1}}.$$

Thus, $r_M \leq 1 - r_{M+1}$ at t_{b-1}. □

Corollary 8 (Necessary Condition for Event B)**.** *Event B occurs at time t_b only if $S_{b-1} > M \cdot r_{M,b-1}$, where $r_{i,b-1} \geq r_{i+1,b-1}$, $1 \leq i < N$.*

Proof.

$$S_{b-1} = \sum_{i=1}^{M} r_{i,b-1} + \sum_{i=M+1}^{N} r_{i,b-1} > \sum_{i=1}^{M} r_{i,b-1} \geq M \cdot r_{M,b-1}.$$

□

Theorem 9 (Total Local Utilization for Event B)**.** *When event B occurs at time t_b and $S_{b-1} \leq M$, then $S_b \leq M$, where $r_{i,b-1} \geq r_{i+1,b-1}$, $1 \leq i < N$.*

Proof. $t_b - t_{b-1}$ is $l_{M,b-1}$. The total local remaining execution time at t_{b-1} is $\sum_{i=1}^{N} l_{i,b-1} = (t_f - t_{b-1})S_{b-1}$, and this decreases by $M \cdot l_{M,b-1}$ at t_b as M tokens move diagonally. Therefore:

$$(t_f - t_b)S_b = (t_f - t_{b-1})S_{b-1} - M \cdot l_{M,b-1}.$$

Since $t_f - t_b = t_f - t_{b-1} - l_{M,b-1}$,

$$(t_f - t_{b-1} - l_{M,b-1})S_b = (t_f - t_{b-1})S_{b-1} - M \cdot l_{M,b-1}.$$

Thus,

$$S_c = \frac{1}{1 - r_{M,b-1}} S_{c-1} + (1 - \frac{r_{M,b-1}}{1 - r_{M,b-1}})M. \qquad (2)$$

Equation 2 is a linear equation as shown in Figure 10.

According to Corollary 8, when event B occurs, S_{b-1} is more than $M \cdot r_{M,b-1}$. Since we also assume $S_{b-1} \leq M$, $S_b \leq M$. □

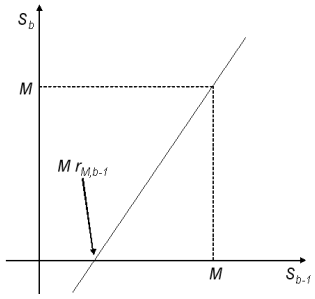

Figure 10: Linear Equation for Event B

3.4 Optimality

We now establish LLREF's scheduling optimality by proving its local feasibility in the T-L plane based on our previous results.

In Section 3.3 and 3.2, we suppose that $N > M$. When less than or equal to M tokens only exist, they are always locally feasible by LLREF in the T-L plane.

Theorem 10 (Local Feasibility with Small Number of Tokens)**.** *When $N \leq M$, tokens are always locally feasible by LLREF.*

Proof. We assume that if $N \leq M$, then tokens are not locally feasible by LLREF. If it is not locally feasible, then there should exist at least one critical moment in the T-L plane by Theorem 2. Critical moment implies at least one non-selected token, which contradicts our assumption since all tokens are selectable. □

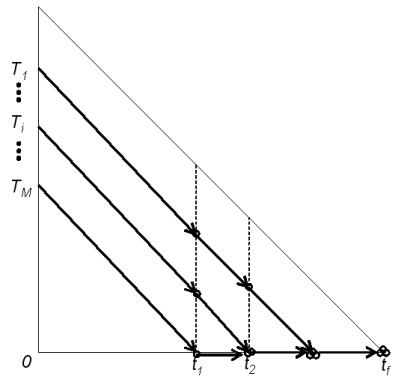

Figure 11: Token Flow when $N \leq M$

Theorem 10 is illustrated in Figure 11. When the number of tasks is less than the number of processors, LLREF can select all tasks and execute them until their local remaining execution times become zero.

We also observe that at every event B, the number of active tokens is decreasing. In addition, the number of events

B in this case is at most N, since it cannot exceed the number of tokens. Another observation is that event C never happens when $N \leq M$ since all tokens are selectable and move diagonally.

Now, we discuss the local feasibility when $N > M$.

Theorem 11 (Local Feasibility with Large Number of Tokens)**.** *When $N > M$, tokens are locally feasible by LLREF if $S_0 < M$.*

Proof. We prove this by induction, based on Theorems 6 and 9. Those basically show that if $S_{j-1} \leq M$, then $S_j \leq M$, where j is the moment of sub-events. Since we assume that $S_0 < M$, S_j for all j never exceeds M at any sub-event including C and B events. When S_j is less than M for all j, there should be no critical moment in the T-L plane, according to the contraposition of Corollary 3. By Theorem 2, no critical moment implies that they are locally feasible. □

When $N(> M)$ number of tokens are given in the T-L plane and their S_0 is less than M, event C and B occur without any critical moment according to Theorem 11. Whenever event B happens, the number of inactive tokens decreases until there are M remaining tokens. Then, according to Theorem 10, all tokens are selectable so that they arrive at the rightmost vertex of the T-L plane with consecutive event B's.

Recall that we consider periodically arriving tasks, and the scheduling objective is to complete all tasks before their deadlines. With continuous T-L planes, if the total utilization of tasks $\sum_{i=1}^{N} u_i$ is less than M, then tokens are locally feasible in the first T-L plane based on Theorems 10 and 11. The initial S_0 for the second consecutive T-L plane is less than M by Theorem 1 and inductively, LLREF guarantees the local feasibility for every T-L plane, which makes all tasks satisfy their deadlines.

3.5 Algorithm Overhead

One of the main concerns against global scheduling algorithms (e.g., LLREF, global EDF) is their overhead caused by frequent scheduler invocations. In [14], Srinivasan *et al.* identify three specific overheads:

1. *Scheduling overhead*, which accounts for the time spent by the scheduling algorithm including that for constructing schedules and ready-queue operations;
2. *Context-switching overhead*, which accounts for the time spent in storing the preempted task's context and loading the selected task's context; and
3. *Cache-related preemption delay*, which accounts for the time incurred in recovering from cache misses that a task may suffer when it resumes after a preemption.

Note that when a scheduler is invoked, the context-switching overhead and cache-related preemption delay may not happen. Srinivasan *et al.* also show that the number of task preemptions can be bounded by observing that when a task is scheduled (selected) consecutively for execution, it can be allowed to continue its execution on the same processor. This reduces the number of context-switches and possibility of cache misses. They bound the number of task preemptions under Pfair, illustrating how much a task's execution time inflates due to the aforementioned overhead. They show that, for Pfair, the overhead depends on the time quantum size.

In contrast to Pfair, LLREF is free from time quanta. However, it is clear that LLREF yields more frequent scheduler invocations than global EDF. Note that we use the number of scheduler invocations as a metric for overhead measurement, since it is the scheduler invocation that contributes to all three of the overheads previously discussed. We now derive an upper bound for the scheduler invocations under LLREF.

Theorem 12 (Upper-bound on Number of Sub-events in T-L plane)**.** *When tokens are locally feasible in the T-L plane, the number of events in the plane is bounded within $N + 1$.*

Proof. We consider two possible cases, when a token T_i hits the NLLD, and it hits the bottom. After T_i hits the NLLD, it should move along the diagonal to the rightmost vertex of the T-L plane, because we assume that they are locally feasible. In this case, T_i raises one sub-event, event C. Note that its final event B at the rightmost vertex occurs together with the next releasing event of another task (i.e., beginning of the new T-L plane). If T_i hits the bottom, then the token becomes inactive and will arrive at the right vertex after a while. In this case, it raises one sub-event, event B. Therefore, each T_i can cause one sub-event in the T-L plane. Thus, N number of tokens can cause $N + 1$ number of events, which includes N sub-events and a task release at the rightmost vertex. □

Theorem 13 (Upper-bound of LLREF Scheduler Invocation over Time)**.** *When tasks can be feasibly scheduled by LLREF, the upper-bound on the number of scheduler invocations from time t_s to t_e is:*

$$(N+1) \cdot \left(1 + \sum_{i=1}^{N} \left\lceil \frac{t_e - t_s}{p_i} \right\rceil \right),$$

where p_i is the period of T_i.

Proof. Each T-L plane is constructed between two consecutive events of task release, as shown in Section 2.2. The number of task releases during the time between t_s and t_e is $\sum_{i=1}^{N} \lceil \frac{t_e - t_s}{p_i} \rceil$. Thus, there can be at most $1 + \sum_{i=1}^{N} \lceil \frac{t_e - t_s}{p_i} \rceil$

number of T-L planes between t_s and t_e. Since at most $N+1$ events can occur in every T-L plane (by Theorem 12), the upper-bound on the scheduler invocations between t_s and t_e is $(N+1) \cdot (1 + \sum_{i=1}^{N} \lceil \frac{t_e - t_s}{p_i} \rceil)$. □

Theorem 13 shows that the number of scheduler invocations of LLREF is primarily dependent on N and each p_i — i.e., more number of tasks or more frequent releases of tasks results in increased overhead under LLREF.

3.5.1 Experimental Evaluation

We conducted simulation-based experimental studies to validate our analytical results on LLREF's overhead. We consider an SMP machine with four processors. We consider four tasks running on the system. Their execution times and periods are given in Table 1. The total utilization is approximately 1.5, which is less than 4, the capacity of processors. Therefore, LLREF can schedule all tasks to meet their deadlines. Note that this task set's α (i.e., $\max^N \{u_i\}$) is 0.818, but it does not affect the performance of LLREF, as opposed to that of global EDF.

Table 1: Task Parameters (4 Task Set)

Tasks	c_i	p_i	u_i
T_1	9	11	0.818
T_2	5	25	0.2
T_3	3	30	0.1
T_4	5	14	0.357

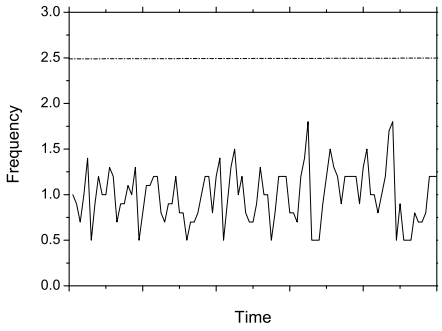

Figure 12: Scheduler Invocation Frequency with 4 Tasks

To evaluate LLREF's overhead in terms of the number of scheduler invocations, we define the scheduler invocation frequency as the number of scheduler invocations during a time interval Δt divided by Δt. We set Δt as 10. According to Theorem 13, the upper-bound on the number of scheduler invocations in this case is $(4+1) \cdot (1 + \lceil 10/11 \rceil + \lceil 10/25 \rceil + \lceil 10/30 \rceil + \lceil 10/14 \rceil) = 25$. Therefore, the upper-bound on the scheduler invocation frequency is 2.5.

In Figure 12, the upper-bound on the scheduler invocation frequency and the measured frequency are shown as a dotted line and a fluctuating line, respectively. We observe that the actual measured frequency respects the upper-bound.

Table 2: Task Parameters (8 Task Set)

Tasks	c_i	p_i	u_i
T_1	3	7	0.429
T_2	1	16	0.063
T_3	5	19	0.263
T_4	4	5	0.8
T_5	2	26	0.077
T_6	15	26	0.577
T_7	20	29	0.69
T_8	14	17	0.824

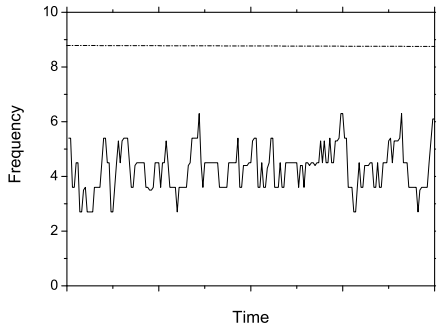

Figure 13: Scheduler Invocation Frequency with 8 Tasks

We now consider eight tasks; the task parameters are given in Table 13. The total utilization for this set is approximately 3.72, which is slightly less than the number of processors. Therefore, LLREF again can schedule all tasks to meet their deadlines. When we set Δt as 10, the upper-bound on the number of scheduler invocations is computed as 88, and the upper-bound on the scheduler invocation frequency is computed as 8.8.

Figure 13 shows the upper-bound on the invocation frequency and the actual frequency for the 8-task set. Consistently with the previous case, the actual frequency never moves beyond the upper-bound. We also observe that the average invocation frequencies of the two cases are approximately 1.0 and 4.0, respectively. As expected (by Theorem 13), the number of tasks proportionally affects LLREF's overhead.

Thus, our experimental results validate our analytical results on LLREF's overhead.

4 Conclusions and Future Work

We present an optimal real-time scheduling algorithm for multiprocessors, which is not based on time quanta. The algorithm called LLREF, is designed based on our novel technique of using the T-L plane abstraction for reasoning about multiprocessor scheduling. We show that scheduling for multiprocessors can be viewed as repeatedly occurring T-L planes, and correct scheduling on a single T-L plane leads to the optimal solution for all times. We analytically establish the optimality of our algorithm. We also establish that the algorithm overhead is bounded in terms of the number of scheduler invocations, which is validated by our experimental (simulation) results.

We have shown that LLREF guarantees local feasibility in the T-L plane. This result can be intuitively understood in that, the algorithm first selects tokens which appear to be going out of the T-L plane because they are closer to the NLLD.

However, we perceive that there could be other possibilities. Theorems 6 and 9 are fundamental steps toward establishing the local feasibility in the T-L plane. We observe that the two theorems' proofs are directly associated with the LLREF policy in two ways: one is that they depend on the critical moment, and the other is that the range allowed for S_{j-1} is computed under the assumption of LLREF, where j is c or b. This implies that scheduling policies other than LLREF may also provide local feasibility, as long as they maintain the critical moment concept. (The range allowed for S_{j-1} could be recomputed for each different scheduling event.) The rules of such scheduling policies can include: (1) selecting as many active tokens as possible up to M at every sub-event; and (2) including all tokens on the NLLD for selection at every sub-event.

If such algorithms can be designed, then tradeoffs can be established between them and LLREF — e.g., on algorithm complexity, frequency of event C, etc. Note that even in such cases, the upper-bound of scheduler invocations given in Theorems 12 and 13 hold, because those theorems are not derived under the assumption of LLREF. Designing such algorithms is a topic for further study.

We also regard that each task's scheduling parameters including execution times and periods are sufficiently longer than the system clock tick, so that they can be assumed as floating point numbers rather than integers. This assumption may not be sufficient for the case when task execution time is shorter so that integer execution times are a better assumption. However, our intuition strongly suggests that even when task parameters are considered as integers, it should be possible to extend our algorithm to achieve a certain level of optimality—e.g., the error between each task's deadline and its completion time is bounded within a finite number of clock ticks.

Many other aspects of our task model such as arrival model, time constraints, and execution time model can be relaxed. All these extensions are topics for further study.

5. Acknowledgements

Acknowledgements This work was supported by the U.S. Office of Naval Research under Grant N00014-00-1-0549 and by The MITRE Corporation under Grant 52917.

References

[1] J. Anderson, V. Bud, and U. C. Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *IEEE ECRTS*, pages 199–208, July 2005.

[2] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Algorithmica*, volume 15, page 600, 1996.

[3] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *IEEE ECRTS*, pages 209– 218, 2005.

[4] J. Carpenter, S. Funk, et al. A categorization of real-time multiprocessor scheduling problems and algorithms. In J. Y. Leung, editor, *Handbook on Scheduling Algorithms, Methods, and Models*, page 30.130.19. Chapman Hall/CRC, Boca Raton, Florida, 2004.

[5] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 21st IEEE Real-Time Technology and Applications Symposium*, 2001.

[6] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, December 1989.

[7] U. C. Devi and J. Anderson. Tardiness bounds for global edf scheduling on a multiprocessor. In *IEEE RTSS*, 2005.

[8] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127140, 1978.

[9] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic tasks systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

[10] P. Holman and J. Anderson. Adapting pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, To appear. Available at: `http://www.cs.unc.edu/~anderson/papers.html`.

[11] K. Lin, Y. Wang, T. Chien, and Y. Yeh. Designing multimedia applications on real-time systems with smp architecture. In *IEEE Fourth International Symposium on Multimedia Software Engineering*, page 17, 2002.

[12] QNX. Symmetric multiprocessing. `http://www.qnx.com/products/rtos/smp.html`. Last accessed October 2005.

[13] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. In *Information Processing Letters*, pages 93–98, Nov 2002.

[14] A. Srinivasan, P. Holman, J. Anderson, and S. Baruah. The case for fair multiprocessor scheduling. In *IEEE International Parallel and Distributed Processing Symposium*, page 1143, 2003.