Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections

抽象的

As mobile computing is getting popular; there is an increasing interest in techniques that can minimize energy consumption and prolong the battery life on mobile devices. Processor voltage scheduling is an effective way to reduce energy dissipation by reducing the processor speed. In this paper; we study voltage scheduling for real-time periodic tasks with non-preemptible sections. Three schemes are proposed to address this problem. The static speed algorithm derives a static feasible speed based on the Stack Resource Policy (SRP). As worst-case blocking does not always occur; the novel dual speed algorithm switches the processor speed to a lower value whenever possible. The dynamic reclaiming algorithm deploys a reservation-based approach to reclaim unused run time for redistribution. It effectively decreases the processor idle time and further reduces the processor speed. The feasibility conditions are given and proved. Simulation results show that the two dynamic algorithms can reduce processor energy consumption' by up to 80 percent over the static speed scheme.

隨著移動計算越來越流行；人們對使移動設備上的能耗降至最低並延長電池壽命的技術的興趣日益濃厚。處理器電壓調度是通過降低處理器速度來減少能耗的有效方法。在本文中;我們研究具有不可搶占部分的實時周期性任務的電壓調度。提出了三種方案來解決這個問題。靜態速度算法根據堆棧資源策略（SRP）得出靜態可行速度。由於最壞情況下的阻塞並不總是發生；新穎的雙速算法在可能的情況下將處理器速度切換到較低的值。動態回收算法部署了一種基於預留的方法來回收未使用的運行時間以進行重新分配。它有效地減少了處理器的空閒時間，並進一步降低了處理器速度。給出了可行性條件並進行了證明。仿真結果表明，與靜態速度方案相比，這兩種動態算法可以將處理器能耗降低多達80％。

1引言

For convenience and ease of access, more and more personal computing and communication devices are becoming portable and mobile. These include laptop computers, pocket PCs and PDAs. Most of them are powered by batteries with limited power capacity. Some recent portable systems are equipped with powerful processors that were available on desktop computers only a few years ago. The performance boost comes at the cost of higher energy consumption. On the other hand, as battery technology has not been keeping up with the speed of the processors, the limited battery power has become a major concern. How to conserve power and prolong battery life is of critical impor tance and has received much attention recently [1,2,3]

為了方便和易於訪問，越來越多的個人計算和通信設備正在變得便攜和移動。 這些包括筆記本電腦，掌上電腦和PDA。 它們中的大多數由功率容量有限的電池供電。 最近的一些便攜式系統配備了功能強大的處理器，而這些處理器僅在幾年前才可在台式計算機上使用。 性能提升是以更高的能耗為代價的。 另一方面，由於電池技術不能跟上處理器的速度，因此有限的電池功率已成為主要問題。 如何節約能源和延長電池壽命至關重要，並且最近受到了很多關注[1,2,3]。

Work exists on evaluating the power consumption of portable computing systems [4, 5]. It was found that the display (including backlight), processor, hard disk and wireless LAN card attribute to most of the power consumption [4, 5l. In particular, the processor may consume up to 25% of the system power for laptop computers [4]. This percentage is even higher for PDAs since hard disks are not used. If the power I consumed by the processor is reduced, the power consumption of the whole system is also substantially reduced.

存在評估便攜式計算系統功耗的工作[4、5]。 發現顯示器（包括背光燈），處理器，硬盤和無線局域網卡是大多數功耗的原因[4，5l。 特別是，處理器可能會消耗多達25％的筆記本電腦系統功率[4]。 由於不使用硬盤，因此PDA的百分比甚至更高。 如果減少了處理器消耗的功率I，則整個系統的功耗也將大大降低。

As the processor may not be fully utilized all the time, the variation in system load can be exploited to reduce power dissipation. The processor can be turned off or made to operate at lower a speed when the system has no or little work to do. Some processors (e.g., the Crusoe processor of Transmeta [6] and the Intel StrongARM processor [7]) allow the processor voltage to be dynamically adjusted. This is called dynamic voltage scaling (DVS). The relationship among power consumption rate (P), supply voltage (Vs) and clock frequency (f) can be described by the following formula [8]:

由於處理器可能不會一直被充分利用，因此可以利用系統負載的變化來減少功耗。 當系統沒有工作要做或幾乎沒有工作要做時，可以關閉處理器或使其以較低的速度運行。 某些處理器（例如Transmeta的Crusoe處理器[6]和Intel StrongARM處理器[7]）允許動態調整處理器電壓。 這稱為動態電壓縮放（DVS）。 功耗率（P），電源電壓（Vs）和時鐘頻率（f）之間的關係可以通過以下公式[8]描述：



where C is the switched capacitance. Furthermore, as the operating frequency f (and therefore the processor speed) is approximately proportional to the supply voltage [9], the power consumption rate is roughly proportional to the cube of the supply voltage. It follows that if we decrease the processor speed by lowering the supply voltage, the power consumption will be reduced. Voltage scaling is particularly useful for real-time applications because a longer computation time is acceptable as long as the deadlines are not violated. With voltage scaling, the job scheduler must make two types of decisions: which job to run and at what speed to run it. In this paper, we shall refer to job scheduling with voltage scaling simply as "voltage scheduling" for short.

其中C是開關電容。 此外，由於工作頻率f（因此處理器速度）大約與電源電壓[9]成正比，因此功耗率大致與電源電壓的立方成正比。 因此，如果我們通過降低電源電壓來降低處理器速度，則功耗將會降低。 電壓縮放對於實時應用特別有用，因為只要不違反最後期限，可以接受更長的計算時間。 對於電壓縮放，作業調度程序必須做出兩種類型的決策：要運行哪個作業以及以什麼速度運行它。 在本文中，我們將帶電壓縮放的作業調度簡稱為“電壓調度”。

Much work has been done on voltage scheduling with the objective of minimizing processor cncrgy consumption[I,2,3,9, 10, 11, 12]. Most of them assume the tasks are preemptible. However, in reality, tasks may have regions that are not preemptible. We call these regions blocking sections. An example is when the task is holding some critical resources or is in the middle of an atomic transaction. Assuming fully preemptible tasks in this case may cause deadline misses or result in incorrect computation.

為了使處理器的功耗最小化，已經在電壓調度方面做了很多工作[I，2,3,9,10,11,12]。 他們中的大多數人認為任務是可搶占的。 但是，實際上，任務可能具有不可搶占的區域。 我們稱這些區域為阻塞區域。 例如，當任務持有一些關鍵資源或處於原子事務中間時。 假設在這種情況下完全可搶占的任務可能會導致截止期限丟失或導致計算不正確

In this work, we consider voltage scheduling of real-time periodic tasks with blocking sectiuns. We show how to calculate a static speed by which all admitted tasks can be feasibly scheduled with minimal energy. A dynamic dual speed algorithm is also proposed to run the processor at an even lower speed in some intervals. Furthermore, as thc actual execution time usually differs from the declared worstcase execution time (WCET), a reservation-based scheme which dynamically reclaims processor time for redistribution is proposed to further reduce energy consumption.

在這項工作中，我們考慮了帶有閉塞的實時周期性任務的電壓調度。 我們展示瞭如何計算靜態速度，通過該速度可以以最少的精力可行地安排所有已接受的任務。 還提出了動態雙速算法，以在某些時間間隔內以更低的速度運行處理器。 此外，由於實際執行時間通常與聲明的最壞情況執行時間（WCET）不同，因此提出了一種基於預留的方案，該方案可動態回收處理器時間以進行重新分配，以進一步降低能耗。

本文的其餘部分安排如下：第2節介紹了系統模型。第3節介紹了靜態速度算法的可行性條件和公式。 第4節介紹了兩種動態電壓調度算法。 性能評估結果在第5節中介紹。第6節回顧了有關電壓調度的相關研究。 最後，第7節總結了本文。

2系統模型

2.1任務模型

Real-time periodic tasks are considered in this paper. A periodic task is a sequence of jobs released at constant intervals (called the period). We denote the set of tasks by T. Each task T; E T is characterized by four parameters:

本文考慮了實時的周期性任務。 定期任務是按固定間隔（稱為周期）釋放的一系列作業。 我們用T表示任務集。 E T的特徵在於四個參數：

A.i: time the task is first released. • D;: relative deadline of the task. • Pi: period of the task. • Ei: worst-case execution time (WCET) of any job in the task.

•A.i：首次釋放任務的時間。

•D ；：任務的相對期限。

•Pi：任務期限。

•Ei：任務中任何作業的最壞情況執行時間（WCET）。

In this paper, we assume the relative deadline of a task is equal to its period. Each job in a task can be considered as a processing request. It is associated with an absolute deadline by which the job should be completed. We say a job meets the deadline if the job is completed at or beforc before its deadline, and it misses the deadline otherwise. In this paper, we assume hard real-time tasks, i.e., there should be no deadline miss. A task Ti = (Ji,l, h2, Ji,3 ,"" Ji,n ) consists of n jobs, where job Ji,j is characterized by its releasc timc r i,j, the execution time e"j (-S Ei) and the absolute deadline di ,j' The execution time is defined as the time required to process the job at the processor's maximum speed. Furthermore, jobs are preemptible except when they are running in their blocking sections. A job can have zero, one or more blocking sections, and Gi denotes the length of the longest blocking scction of any job in task T,2• Thc positions of the blocking sections are randomly distributed within a job except that they are non-overlapping.

在本文中，我們假設任務的相對期限等於任務的期限。任務中的每個作業都可以視為一個處理請求。它與完成任務的絕對截止日期相關。我們說，如果某個工作在截止日期之前完成，或者在該截止日期之前完成，那麼該工作會在截止日期之前完成，否則它將錯過該截止日期。在本文中，我們假設要執行艱鉅的實時任務，即不應錯過最後期限。任務Ti =（Ji，l，h2，Ji，3，“” Ji，n）包含n個作業，其中作業Ji，j的特徵在於其釋放時間ri，j，執行時間e“ j（-S Ei）和絕對截止時間di，j'執行時間定義為以處理器的最大速度處理作業所需的236時間，此外，作業是搶占性的，除非它們在其阻塞區域中運行。該作業可以為零，一個或多個阻塞部分，Gi表示任務T，2中任何作業的最長阻塞段的長度。2。阻塞部分的位置隨機分佈在一個作業中，但它們是不重疊的。

2.2處理器型號

The processor is capable of dynamic voltage scaling and its speed is proportional to the supply voltage. The maximum and minimum possible supply voltages are denoted as v'nax and V.nin, respectively, while the corresponding proccssor speeds are Smax and Smin, respectively. The processor voltage can be adjusted at discrete steps within the range. Throughout this paper, we assume the processor's maximum speed is I and all other speeds are normalized with respect to the maximum speed. As observed in [2], the voltage transition delay is very short. We therefore assume the voltage transition cost is negligible and the voltage can be adjusted at any time (whether inside or outside a blocking section). We also assume the processor power follows formula (l), which in our case can be simplified to P = K . Vs 3 where K is a constant.

處理器具有動態電壓縮放功能，其速度與電源電壓成正比。 最大和最小可能的電源電壓分別表示為v'nax和V.nin，而相應的處理器速度分別為Smax和Smin。 可以在該範圍內以不連續的步長調整處理器電壓。 在整個本文中，我們假定處理器的最大速度為I，並且所有其他速度均相對於最大速度進行了歸一化處理。 如在[2]中觀察到的，電壓轉換延遲非常短。 因此，我們認為電壓轉換成本可以忽略不計，並且可以隨時調整電壓（無論是在閉塞部分的內部還是外部）。 我們還假定處理器功率遵循公式（l），在我們的情況下可以簡化為P = K。 VS 3，其中K為常數。

3靜態阻塞感知電壓調度

In this section we show how to find a static voltage/speed setting to minimize energy consumption in a system consisting of periodic tasks with blocking sections. In the static scheme, the processor voltage is changed only when a new task arrives or when an existing task terminates.

在本節中，我們展示如何找到一個靜態電壓/速度設置，以最大程度地減少由具有阻塞部分的周期性任務組成的系統中的能耗。 在靜態方案中，僅當新任務到達或現有任務終止時才更改處理器電壓。

The Stack Resource Policy (SRP) was proposed by Baker to schedule tasks with shared resources [13]. The core idea is that a job is allowed to preempt a lower priority job only if all the resources it needs are available. The feasibility conditiun uf the SRP was also derived and is listed in Theorem 1:

堆棧資源策略（SRP）由貝克提出，用於計劃具有共享資源的任務[13]。 核心思想是，僅當作業所需的所有資源均可用時，才允許該作業搶占優先級較低的作業。 SRP的可行性條件也已得出，並在定理1中列出：

Theorem 1 [13] Suppose n periodic tasks are sorted by their periods. They are schedulable by the earliest deadline first (EDF) algorithm with the SRP if

定理1 [13]假設n個週期性任務按其周期排序。 如果有SRP，可以通過最早的截止日期優先（EDF）算法對它們進行調度



where Bi is the maximum length that a job in Ti can be blocked.

其中Bi是可以阻止Ti中的作業的最大長度。

In voltage scheduling, if the SRP is used with EDF [14], the processor speed can be reduced according to Theorem 2. Note that we have replaced D with P in the formula since they have the same value in our task model.

在電壓調度中，如果SRP與EDF [14]一起使用，則可以根據定理2降低處理器速度。請注意，由於在任務模型中它們具有相同的值，因此我們在公式中將D替換為P。

Theorem 2 Suppose n periodic tasks are sorted by their periods. They can be feasibly scheduled by EDF with the SRP at processor speed H(O < H ::; 1) if

定理2假設n個週期性任務按其周期排序。 如果滿足以下條件，則可以由EDF使用SRP以處理器速度H（O <H≦1）進行可行的調度。

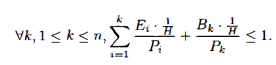


where Bi is the maximum length that a job in Ti can be blocked.

其中Bi是可以阻止Ti中的作業的最大長度。

Proof: Note that scheduling a task set T at processor speed H is equivalent to scheduling a task set T\* at the maximum processor speed where the execution times and resource holding times of T\* are 1/ H times the corresponding values in T. Hence, the above inequality can be rewritten as

證明：請注意，以處理器速度H調度任務集T等效於以最大處理器速度調度任務集T \*，其中T \*的執行時間和資源保持時間為T中相應值的1 / H倍。 ，上述不等式可以改寫為



Moreover, the maximum blocking time a job in Tt may encounter would also be scaled up by 1/ H times. According to Theorem 1, T\* is schedulable by the SRP at full processor speed (=1), so the original task set T is schedulable at speed H.

此外，Tt中的作業可能遇到的最大阻塞時間也將增加1 / H倍。 根據定理1，T \*可通過SRP在全處理器速度（= 1）下進行調度，因此原始任務集T可在速度H下進行調度。

Our specific task model can be regarded as a special case of the SRP, where only one resource is shared and all tasks need to request the resource to complete. If a job does not actually need the resource, it is still required to request the resource for zero time unit at the start of its execution. Such a job corresponds to a job without a blocking section in our task model. The feasible static processor speed can be calculated according to Corollary 1.

我們特定的任務模型可以視為SRP的一種特殊情況，其中僅共享一個資源，並且所有任務都需要請求資源才能完成。 如果作業實際上不需要資源，則仍需要在執行開始時以零時間單位請求資源。 這樣的作業對應於我們的任務模型中沒有阻塞部分的作業。 可以根據推論1計算可行的靜態處理器速度。

Corollary 1 Suppose n periodic tasks are sorted by their periods. They can be feasibly scheduled by EDF at processor speed H(O < H:::: 1) if

推論1假設n個週期性任務按其時間段排序。 可以通過EDF以處理器速度H（O <H ≦1）i合理地調度它們



where 1 < j≦n.

Proof: Note that in EDF scheduling, a job can only be blocked by jobs in tasks with larger periods, so max{ Gj IPk < Pj} is the maximum time that any job in Tk can be blocked.

證明：請注意，在EDF調度中，作業只能被周期較大的任務中的作業阻塞，因此max {Gj IPk <Pj}是Tk中任何作業可以被阻塞的最長時間。

The static processor speed needs to be re-computed only when a task completes or when a new task arrives. The objective is to find a minimum H such that the inequalities are satisfied. If the re-calculated H exceeds 1, the newly arrived task is not admitted to the system and the original H value is restored. Otherwise the system will run at speed H until the value is changed again. For discrete processor speed levels, the lowest speed level that is greater than or equal to H is used.

僅當任務完成或新任務到達時，才需要重新計算靜態處理器速度。 目的是找到滿足不等式的最小值H。 如果重新計算的H超過1，則新到達的任務將不被允許進入系統，並且原始H值將被恢復。 否則，系統將以速度H運行，直到再次更改該值。 對於離散處理器速度級別，使用大於或等於H的最低速度級別。

4動態阻塞感知電壓調度

By applying a feasible static speed as in Section 3, the task set is guaranteed to be schcdulable. However, there may exist idle intervals which could be exploited to further reduce the processor speed. The idle intervals come from two sources. First, the feasibility test is based on the WCETs and the maximum blocking lengths of the tasks. The actual execution times and blocking sections of individual jobs are usually shorter. Second and more importantly, given the feasibility condition, the total processor utilization of the admitted tasks is usually lower than the normalized static feasible speed H. This difference becomes substantial as the length of blocking sections increases. In this case, the processor is still under-utilized even if all jobs use their WCETs. However, if a single static speed is used, it is not possible to further reduce it below H without incurring deadline misses. In this section, we propose new algorithms that dynamically slow down the processor at strategic intervals that will preserve the feasibility of the task set.

通過像第3節中那樣應用可行的靜態速度，可以保證任務集是可調度的。但是，可能存在空閒間隔，可以利用這些空閒間隔來進一步降低處理器速度。空閒間隔來自兩個來源。首先，可行性測試基於WCET和任務的最大阻塞長度。單個作業的實際執行時間和阻塞部分通常較短。其次，更重要的是，在給定可行性條件的情況下，所允許任務的總處理器利用率通常低於歸一化靜態可行速度H。隨著阻塞部分長度的增加，此差異變得很大。在這種情況下，即使所有作業都使用其WCET，處理器仍未得到充分利用。但是，如果使用單個靜態速度，則不可能將其進一步降低到H以下，而不會導致截止期限丟失。在本節中，我們提出了新的算法，該算法以戰略間隔動態降低處理器速度，從而保留了任務集的可行性。

4.1雙速切換算法

If the tasks are fully preemptible, they can be feasibly scheduled by EDF with a minimum static speed L [I J such that

如果任務是完全可搶占的，則可以由EDF以最小靜態速度L [I ]



Comparing with (2), it is easy to see L ::; H. We refer to L as the utilization speed, or simply the "low speed". In contrast, the static speed H (or "high speed") calculated in Section 3 ensures a job will not miss its deadline even if worst-case blocking occurs. We propose a dual speed algorithm that allows the processor to operate at speed L whenever possible. In the dual speed algorithm, if a job blocks a higher priority job, then during the lifetime of the low priority job (Le., until its deadline), the processor must run at speed H. In all other situations the processor may run at the low speed L. The Dual Speed (DS) algorithm is formally presented in Figure 1. If the system is executing in a high speed interval (i.e., the system is running at speed H), then End\_H in the algorithm indicates the time point at the end of that interval. Otherwise End\_H = -1. If there is no work to do, the processor will enter the idle state. Note that with discrete speed levels, the SeLSpeed function in the algorithm sets the processor speed to the lowest speed level that is greatcr than or cqual to thc speed spccified in the parameter.

與（2）相比，很容易看到L ::; H。我們將L稱為利用率，或簡稱為“低速”。相反，在第3節中計算的靜態速度H（或“高速”）可確保即使發生最壞情況的阻塞，作業也不會錯過其最後期限。我們提出了一種雙速算法，該算法允許處理器盡可能以速度L運行。在雙速算法中，如果作業阻塞了較高優先級的作業，則在低優先級作業的生命週期內（例如，直到其截止日期），處理器必須以速度H運行。在所有其他情況下，處理器都可以以速度H運行。雙速（DS）算法正式如圖1所示。如果系統以高速間隔執行（即係統以速度H運行），則算法中的End\_H表示時間點在該間隔結束時。否則，End\_H = -1。如果沒有任何工作要做，則處理器將進入空閒狀態。請注意，對於離散速度級別，算法中的SeLSpeed函數將處理器速度設置為最低速度級別，該最低速度級別大於或等於參數中指定的速度。

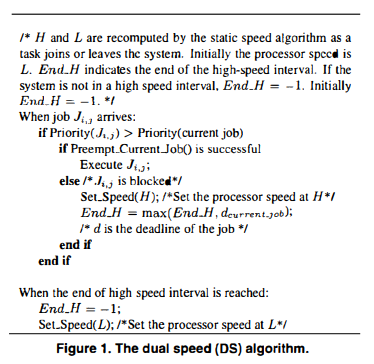


圖1.雙速（DS）算法

In the dual speed scheme, the high speed and the low speed need to be fe-calculated only when a task joins or leaves the system. By maintaining two task lists respectively sorted by the tasks' periods and maximum blocking lengths, the calculation of Hand L can be done in O(n) time where n is the number of tasks. After H and L have been calculated, both procedures in Figure I only take 0(1) time. As Hand L are not frequently changed, the overhead of the algorithm is minimal.

在雙速方案中，僅當任務加入或離開系統時才需要對高速和低速進行計算。 通過維護兩個分別按任務週期和最大阻塞長度排序的任務列表，可以在O（n）時間內完成Hand L的計算，其中n是任務數。 計算完H和L之後，圖I中的兩個過程僅花費0（1）時間。 由於手L不會頻繁更改，因此該算法的開銷很小。

An example is givcn in Figure 2 to illustrate the difference between the static speed algorithm and the dual speed algorithm. The up and down arrows denote the arrival times and deadlines of the jobs, respectively. The white boxes indicate job execution intervals, while the shaded boxes indicate blocking sections. The two jobs in Tl both need 1 time unit to finish at the high speed H while the job in T2 needs 4 timc units. Suppose the low speed L is half the value of the high speed. Under the static speed algorithm, the processor runs at speed H throughout interval [0,6] (Figure 2a), while under the dual speed algorithm, the processor runs at speed L before time point 4, at which blocking occurs (Figure 2b). Based on (1), the dual speed algorithm would save 25 percent of energy compared to the case of static speed.

圖2中的givcn是一個示例，用於說明靜態速度算法和雙重速度算法之間的差異。 上下箭頭分別表示作業的到達時間和截止日期。 白色框表示作業執行間隔，而陰影框表示阻塞部分。 T1中的兩個作業都需要1個時間單位才能完成高速H，而T2中的兩個作業需要4個時間單位。 假設低速L是高速值的一半。 在靜態速度算法下，處理器在整個時間間隔[0,6]中以速度H運行（圖2a），而在雙速度算法下，處理器以時間L在發生阻塞的時間點4之前運行（圖2b） 。 基於（1），與靜態速度相比，雙速算法將節省25％的能量

Although the dual speed algorithm reduces the processor speed, the feasibility of the task set is still maintained. The following theorem guarantees that if a task set is schedulable by the static speed algorithm, it is also schedulable by the dual-speed algorithm.

儘管雙速算法降低了處理器速度，但任務集的可行性仍然得以維持。 以下定理保證，如果任務集可以通過靜態速度算法進行調度，則它也可以通過雙速算法進行調度。

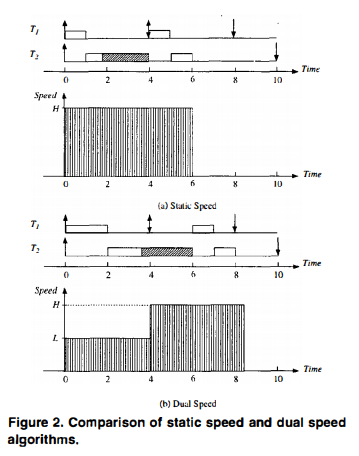
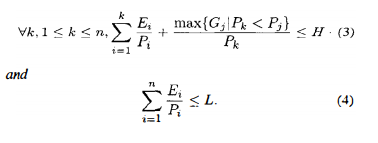


圖2.靜態速度和雙重速度算法的比較

Theorem 3 Suppose n periodic tasks with blocking sections are sorted by their periods. They c all be feasibly scheduled by the dual speed EDF algorithm with high speed H and low speed L if

定理3假設n個具有阻塞部分的周期性任務按其周期排序。 它們可以通過具有高速H和低速L的雙速EDF算法可行地調度



Proof: We prove the theorem by contradiction.

證明：我們通過矛盾證明了該定理。

Suppose the claim is false and t is the earliest time that a job misses its deadline. We find another time t' before t which is the latest time point such that no active job arrived before t' has a deadline at or before t. If i' does not exist, we let t' = O. Constructed in this way, the processor is never idle during (t', t] and only two categories of jobs can execute in the interval. The jobs in the first category, denoted by M, are released after t' and have deadlines at or before t. The second category, if exists, consists of a single job, denoted by Jk, which has a deadline after t and is executing in its blocking section at time t'.

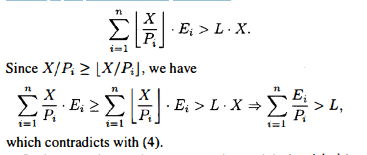
假設索賠是錯誤的，並且t是工作錯過其截止日期的最早時間。 我們在t之前找到另一個時間t'，這是最近的時間點，因此在t'之前沒有截止時間在t或之前沒有活動的作業。 如果i'不存在，則讓t'=O。以這種方式構造，處理器在（t'，t]期間永遠不會處於空閒狀態，並且在該時間間隔內只能執行兩類作業。 用M表示，在t'之後釋放，並且截止時間為t或之前。第二類（如果存在）由單個作業組成，用Jk表示，該任務的截止時間在t之後，並且在時間t的阻塞部分中執行。 '

We consider two cases: only jobs in M are executed during (t', t], and both Jk and jobs in M are executed during the interval.

我們考慮兩種情況：在（t'，t]期間僅執行M中的作業，並且在該間隔期間同時執行Jk和M中的作業。

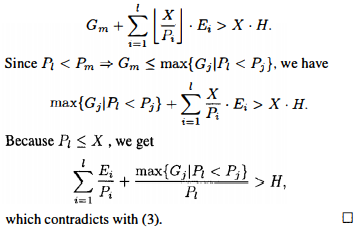
Let X = t - t'. In the first case, because all tasks are periodic, the processor demand generated by the jobs in M is bounded by ��=l lX/P,J . E;. On the other hand, as the processor is never idle and its speed is greater than or equal to L in the entire interval, the processor demand the processor can handle is at least L . X. Since a job misses its deadline at time t by assumption, the total processor demand in the interval must exceed what the processor can handle in the same interval. Therefore

令X = t-t'。 在第一種情況下，因為所有任務都是周期性的，所以M中的作業所生成的處理器需求以��= 11 X / P，J為邊界。 E;。 另一方面，由於處理器從不處於空閒狀態，並且在整個時間間隔內其速度都大於或等於L，因此處理器可以處理的處理器需求至少為L。 X。由於假設作業在時間t錯過了最後期限，因此間隔中的處理器總需求必須超過處理器在相同間隔中可以處理的需求。 所以



In the second case, the processor demand during (tf, t] is larger than that in the first case due to the execution of Jk. However, since Jk will no longer be executed once it leaves its blocking section, the total time Jk can execute during (tf, t] cannot exceed its longest blocking section. Suppose J k belongs to task T m. Then the time it can execute is bounded by Gm. Therefore the total processor demand is bounded by G m + ��= 1 l X / Pd . Ei, where PI is the longest period that is smaller than or equal to X. As the deadline of Jk is later than t, the processor must have been operating at speed H throughout the whole interval (tf, t], therefore the amount of work processed is X . H. If thcre is a deadline miss at time t, the processor demand must be greater than the work processed, i.e.,

在第二種情況下，由於執行Jk，在（tf，t]期間的處理器需求比第一種情況要大，但是，由於Jk一旦離開其阻塞部分將不再執行，因此總時間Jk可以 在（tf，t]內執行不能超過其最長的阻塞部分。假設J k屬於任務T m，那麼它的執行時間以Gm為界，因此處理器總需求以G m +��= 1 l為界 X / Pd。Ei，其中PI是小於或等於X的最長周期。由於Jk的最後期限晚於t，因此在整個時間間隔（tf，t]中，處理器必須一直以速度H運行， 因此，如果在時間t錯過了最後期限，則處理器的需求必須大於已處理的工作，即



The basic dual speed algorithm can be extended to further reduce energy consumption. This is achieved by shortening the lengths of high speed intervals. In the extension, a high speed interval can be terminated at once if one of the following conditions occurs: i) Ajob whose deadline is later than or equal to End\_H starts execution, ii) The processor becomes idle. Because these two situations can never occur in the middle of the interval (tf, t] as shown in the proof of Theorem 3, the feasibility of the task set is not impaired.

基本的雙速算法可以擴展以進一步降低能耗。 這是通過縮短高速間隔的長度來實現的。 在擴展中，如果滿足以下條件之一，則可以立即終止高速間隔：i）截止日期晚於或等於End\_H的作業開始執行； ii）處理器變為空閒。 如定理3的證明所示，由於這兩種情況永遠不會出現在間隔（tf，t]的中間，因此不會削弱任務集的可行性。

4.2動態回收算法

In the dual speed algorithm, the processor speed is always higher than or equal to the utilization speed, which suggests room for further speed reduction. Moreover, the utilization speed is calculated based on the tasks' WCETs, but the actual processing demand is often lower. When a job completes early, the processor would have some idle time. If this portion of time can be redistributed to the other pending jobs, the processor speed can be further reduced. In this section, we present a reservation-based scheme that dynamically collects the residue time from early completions. As the algorithm is an extension to the dual speed algorithm, we call this algorithm the Dual Speed Dynamic Reclaiming (DSDR) algorithm. Aydin et al. [I] proposed a similar approach for a fully preemptible environment, but their algorithm is not applicable when blocking sections are present.

在雙速算法中，處理器速度始終高於或等於利用率，這表明有進一步降低速度的空間。 此外，利用速度是根據任務的WCET計算的，但是實際的處理需求通常較低。 當作業提早完成時，處理器將有一些空閒時間。 如果可以將這部分時間重新分配給其他掛起的作業，則可以進一步降低處理器速度。 在本節中，我們提出一種基於預留的方案，該方案可動態收集早期完成的殘差時間。 由於該算法是雙速算法的擴展，因此我們將此算法稱為雙速動態回收（DSDR）算法。 艾登（Aydin）等人。 [I]針對完全可搶占的環境提出了一種類似的方法，但是當存在阻塞部分時，它們的算法不適用。

First, we need to define the run time of a job and distinguish it from the job's execution time. The run time (denoted by R) can be viewed as a budget assigned to a job. It specifies the wall clock time3 that can be used to process a job and is consumed as the job executes. The run time of a job has a deadline which is set equal to the job's deadline for reclaiming purpose. The execution time (denoted by E) describes the time needed to complete the job under the maximum processor speed. Given the run time and the execution time of ajob, the speed at which the processor should operate is S = E . Smax/ R, where Smax is the maximum processor speed.

首先，我們需要定義作業的運行時間，並將其與作業的執行時間區分開。 運行時間（用R表示）可以看作是分配給作業的預算。 它指定可用於處理作業並在作業執行時消耗的掛鐘時間3。 作業的運行時間的截止時間設置為等於作業出於回收目的的截止時間。 執行時間（用E表示）描述了在最大處理器速度下完成作業所需的時間。 給定作業的運行時間和執行時間，處理器應以的速度為S = E。 Smax / R，其中Smax是最大處理器速度。

We first introduce a reservation-based extension to the dual speed algorithm in which run time is reserved for each job. In order to maintain the same feasibility conditions as in the dual speed algorithm, we need to allocate each job the same amount of run time as it will get under the dual speed algorithm. We take the processor speed determined by the dual speed algorithm (H or L) as the base speed for a job and allocate run time to each job so it can complete its WCET at the base speed before the run time is depleted. When a job arrives, it is assigned an initial run time assuming speed L, which is the optimal allocation if no task is blocked. Note that the processor is never idle if all jobs execute at their WCETs and at speed L. However, since the base speed cannot be determined until the first time the job is selected for execution, the actual run time of the job may be adjusted at that time. As the job executes, it consumes run time and it must complete before its run time is depleted.

我們首先為雙速算法引入基於保留的擴展，其中為每個作業保留運行時間。為了保持與雙速算法相同的可行性條件，我們需要為每個作業分配與雙速算法相同的運行時間。我們將由雙重速度算法（H或L）確定的處理器速度作為作業的基本速度，並為每個作業分配運行時間，以便它可以在運行時間耗盡之前以基本速度完成其WCET。作業到達時，將為其分配初始運行時間（假設速度L），如果沒有阻止任何任務，這是最佳分配。請注意，如果所有作業均以其WCET和速度L執行，則處理器永遠不會處於空閒狀態。但是，由於在選擇首次執行該作業之前無法確定基本速度，因此可以將作業的實際運行時間調整為那時。作業執行時，它消耗了運行時間，並且必須在運行時間耗盡之前完成。

The system maintains a Free Run Time list called the FRT-list to collect the run time not consumed. Similar to the CASH queue in [IS], each item in the FRT-list contains two values: the amount of run time and its deadline. The list is sorted in increasing order of the deadlines. The free run time comes from two sources. The first source is the residue run time when a job completes. The deadline of the reclaimed run time is set equal to the deadline of the completed job. A job may also contribute run time to the FRT-list if it starts to execute (for the first time) in a high speed interval. In this case, because the job's base speed is H, its actual run time assigned could be less than its initial run time allocated when it arrived. The difference in run time is taken from the job and is inserted to the FRT-list. The deadline of this free run time is set to the end of the high speed interval. In this way, DSDR effectively reclaims unused run time for redistribution, which in turn reduces the processor idle time and leads to decreased processor speed.

系統維​​護一個稱為FRT-list的空閒運行時間列表，以收集未消耗的運行時間。與[IS]中的CASH隊列類似，FRT列表中的每個項目都包含兩個值：運行時間量和截止時間。該列表按截止日期的升序排序。自由運行時間有兩個來源。第一個來源是作業完成時的殘渣運行時間。回收運行時間的截止時間設置為等於已完成作業的截止時間。如果作業以高速間隔開始（首次）執行，則作業也可能將運行時間貢獻給FRT列表。在這種情況下，由於作業的基本速度為H，因此分配的實際運行時間可能小於到達時分配的初始運行時間。從作業中獲取運行時間的差異，並將其插入到FRT列表中。該自由運行時間的最後期限設置為高速間隔的結尾。這樣，DSDR有效地回收了未使用的運行時間以進行重新分配，從而減少了處理器的空閒時間並導致處理器速度降低。

When a job is scheduled to run, it is eligible to use its own run time as well as the run time in the FRT-list with deadline earlier than or equal to the job's deadline. With the additional free run time, the job can be processed at a lower speed. If the run time in an item of the FRT-list is depleted, the item is removed.

計劃運行作業時，它有資格使用其自己的運行時間以及FRT列表中的運行時間，且其截止時間早於或等於該任務的截止時間。 使用額外的自由運行時間，可以以較低的速度處理作業。 如果FRT列表中某個項目的運行時間已用完，則該項目將被刪除。

Before we formally present the algorithm, we first introduce the following notations used in the algorithm:

在正式提出算法之前，我們首先介紹該算法中使用的以下符號：

• Ji: the current job of task Ti. (Since at any time each task can only have one job present, no ambiguity is introduced. Correspondingly, d, indicates the deadline of job J;.)

•Ji：任務Ti的當前工作。 （由於每個任務在任何時候都只能有一個任務，因此不會引起歧義。相應地，d表示任務J的截止日期；）

• Ri'(t): the available run time of job Ji at time t.

•Ri'（t）：作業Ji在時間t的可用運行時間。

• R[(t): the run time in the FRT-list that can be used by job J, at time t.

•R [（t）：在時間t，作業J可以使用的FRT列表中的運行時間。

• E[(t): the worst-case residue execution time of job Ji under the maximum speed Smax at time t.

•E [（t）：在時間t的最大速度Smax下，作業Ji的最壞情況下的殘差執行時間。

The core of the DSDR algorithm is given in Figure 3. H, Land End\_H in the figure have the same meanings as in the dual speed algorithm. Note that the processor speed is calculated according to the usable run time R; (t) + R'; (t) and the worst-case residue execution time E[(t) except when the current job is blocking another job. In the latter case, the processor speed is always set at H to reduce the blocking time. To complete the DSDR algorithm, the following rules are used to update the run time and the worst-case residue execution time of a job, and the run time in the FRT-list:

DSDR算法的核心在圖3中給出。圖中的H，Land End\_H與雙速算法具有相同的含義。 注意，處理器速度是根據可用的運行時間R計算的。 （t）+ R'； （t）和最壞情況的殘差執行時間E [（t），除非當前作業阻止了另一個作業。 在後一種情況下，處理器速度始終設置為H，以減少阻塞時間。 為了完成DSDR算法，使用以下規則來更新作業的運行時間和最壞情況的殘差執行時間，以及FRT列表中的運行時間：

1. As job Ji executes, it consumes run time at the same speed as wall clock time starting from the front of the FRT-list if R[(t) > O. Otherwise, R�(t) is used. E[ (t) is reduced by the processor speed per unit time.

當作業Ji執行時，如果可以使用的FRT列表中的運行時間 > 0，它將消耗與FRT列表開頭的掛鐘時間相同速度的運行時間。否則，將使用Ji可用運行時間。利用HSpeed剩下的時間最大效益的減少處理器速度。

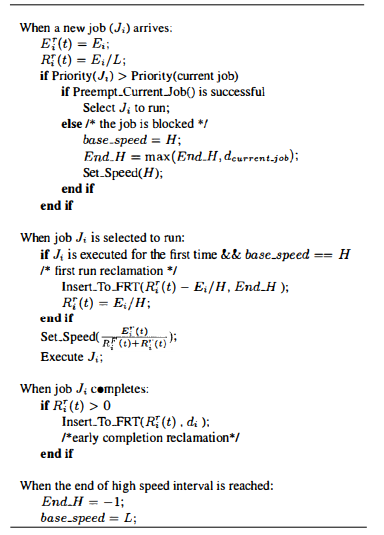
1. 在執行作 業Ji時，如果R [（t）> O，它將消耗與FRT列表開頭的掛鐘時間相同的運行時間。否則，將使用R.（t）。 E [（t）通過單位時間的處理器速度降低。

2. When the processor is idle, if R[(t) > 0, the run time in the FRT-list is reduced at the same speed as wall clock time.

1. 當處理器空閒時，如果R [（t）> 0，則FRT列表中的運行時間以與掛鐘時間相同的速度減少。

3. If a new task arrives at time t, then the run time of all jobs is set to ET (t) j H', where H' is the newly calculated high speed due to the task arrival. The FRT-list is cleared.

1. 如果新任務在時間t到達，則將所有作業的運行時間設置為ET（t）j H'，其中H'是由於任務到達而新計算的高速。 FRT列表被清除。



Note that these rules do not need to be carried out at every time unit. Instead, Rule 1 is applied only when the current job (Ji) completes, blocks another job, or is preempted, and Rule 2 is used only if a new job arrives when the processor is idle.

請注意，不必在每個時間單位都執行這些規則。 相反，僅噹噹前作業（Ji）完成，阻止另一個作業或被搶占時才應用規則1，並且僅當處理器空閒時新任務到達時才使用規則2。

Based on the above discussions, the following lemmas can be proved.

根據以上討論，可以證明以下引理。

Lemma 1 When a task set is scheduled by DSDR, no job will deplete its run time before it completes.

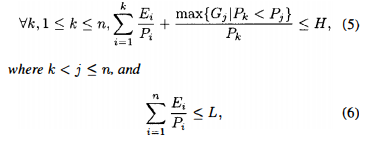
引言1當DSDR安排了任務集時，沒有作業會在完成之前耗盡其運行時間。

The proof should be clear from the DSDR algorithm (Figure 3) and the three updating rules above. Note that a job may use the run time in the FRT-list before consuming its own allocated run time.

從DSDR算法（圖3）和上面的三個更新規則應該清楚地證明這一點。 請注意，作業在使用自己分配的運行時間之前可能會使用FRT列表中的運行時間。

Lemma 2 Suppose n periodic tasks with blocking sections are sorted by their periods. If

引理2假設n個帶有阻塞部分的周期性任務按其周期排序。 如果



then when DSDR is used with high speed H and low speed L, the run time of a job is always depleted at or before the job's deadline. Furthermore, the blocks of run time in the FRT-list are also depleted before their deadlines

那麼，當DSDR與高速H和低速L一起使用時，作業的運行時間總是在作業的最後期限或之前耗盡。 此外，FRT列表中的運行時間塊在其截止日期之前也已耗盡

Proof: We prove the lemma by contradiction.

證明：我們通過矛盾證明引理

Suppose the claim is not true. Let t be the earliest instant that the run time of a job or in the FRT-list is not depleted at its deadline. We choose another time h before t, which is the latest time point such that no pending jobs arrived before tl has a deadline at or before t, and the FRT-list does not contain any item whose deadline is at or before t. If such a time point does not exist, let t1 = O. Constructed in this way, the processor never stops consuming run time throughout the interval (iI, il. Furthermore, the run time consumed during the interval is either generated after i1 and has a deadline before t (denoted as run time A), or has a deadline after t (denoted as run time B).

假設聲明不正確。 令t為作業或FRT列表中的運行時間在截止日期之前沒有耗盡的最早時刻。 我們選擇t之前的另一個時間h，這是最近的時間點，這樣，在tl之前的最後期限為t或之前，沒有任何待處理的作業到達，並且FRT列表中沒有任何截止日期為t或之前的項目。 如果不存在這樣的時間點，則令t1 =O。以這種方式構造，處理器將永不停止消耗整個時間間隔內的運行時間（iI，il。此外，間隔期間消耗的運行時間要么在i1之後生成，要么 t之前的期限（表示為運行時間A），或者t之後的期限（表示為運行時間B）。

We consider two cases. In the first case, only the run time in A is consumed. Note that run time is only generated on job releases. Let Y = i - t1. the total amount of run time in A is bounded by :Z=�1lY/P,J . EdL. As there is still run time left at time t, the amount of run time in A must be greater than the run time consumed in the interval, which is Y. Therefore which contradicts with (6).

我們考慮兩種情況。 在第一種情況下，僅消耗A中的運行時間。 請注意，運行時間僅在作業版本上生成。 令Y = i-t1。 A中的總運行時間為：Z =�1lY/ P，J。 EdL。 由於在時間t仍有剩餘運行時間，因此A中的運行時間量必須大於該間隔中消耗的運行時間，即Y。因此，與（6）相矛盾。



In the second case, the run time in both A and B is consumed in the interval. We choose a third time t2 which is the latest time point before t such that the deadline of the run time being consumed at t2 is after i. Since run time in B is consumed in the interval, i2 must exist and t1 < i2 < i. The scenario is illustrated in Figure 4.

在第二種情況下，在該時間間隔中消耗了A和B中的運行時間。 我們選擇第三時間t2，它是t之前的最新時間點，以使t2消耗的運行時間的截止日期在i之後。 由於B中的運行時間是在該時間間隔內消耗的，因此i2必須存在並且t1 <i2 <i。 該場景如圖4所示。

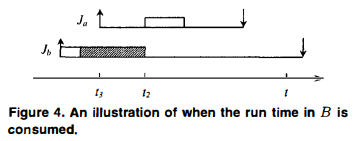
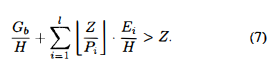


圖4.消耗B中的運行時間的示意圖。

Before time t2, the executing job Jb must be blocking at least one job whose deadline is at or before i. Otherwise Jb would have been preempted before t2 or the assumption that tl < tz would be violated. Furthermore, there is no run time in the FRT-list at i2 whose deadline is at or before t, otherwise run time in B would not have been used at ti me t2. Let J a be the first job that is blocked by Jb• We use i3(tl ::::: t3) to denote the arrival time of Ja. In the interval (i3, i2J, the processor does not consume the run time generated during (t3, iJ, but in the interval (t2, t] it only consumes the run time generated in the period (t3, i]. Suppose Z = t -i3. Since the base speed throughout (i3, i] is H, thc amount of run time generated after t3 which has a deadline at or before i is bounded by :Z=;=I l Z / PiJ . Ed H. In the above expression, I1 is the maximum period that is smaller than or equal to Z. Moreover, as the processor is operating at speed H throughout (t3, i2j, the total amount of run time that can be consumed during (t3, t] is therefore bounded by Gb/H + :Z=;=1 lZ/P,j . EdH, where Gb is the length of the longest blocking section in Jb• As there is still residue run time at time t whose deadline is at t and the processor keeps consuming run time throughout the period, we have

在時間t2之前，正在執行的作業Jb必須阻止至少一個截止日期為i或之前的作業。否則，Jb將在t2之前被搶占，否則將違反tl <tz的假設。此外，在FRT列表中的i2上沒有截止時間在t或t之前的運行時間，否則B的運行時間就不會在t2時刻使用。令J a為被Jb阻止的第一個作業。我們使用i3（tl ::::: t3）表示Ja的到達時間。在間隔（i3，i2J中，處理器不消耗（t3，iJ）期間生成的運行時間，但是在間隔（t2，t]中，處理器僅消耗在周期（t3，i]中生成的運行時間。 = t -i3。由於整個（i3，i]的基本速度為H，因此t3之後產生的運行時間的總和為i或i之前的最後期限為：Z =; = I l Z / PiJ。Ed H在上述表達式中，I1是小於或等於Z的最大周期。此外，由於處理器在整個（t3，i2j）都以速度H運行，因此在（t3，i因此，t]的界線為Gb / H +：Z =; = 1 lZ / P，j。EdH，其中Gb是Jb中最長的阻塞區域的長度。在t時刻，處理器在整個期間一直消耗著運行時間，我們有



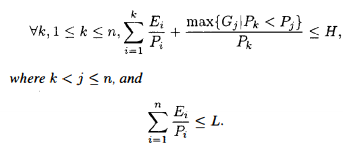
因為Z≧PI和Z / Pi≧l Z / Pi J'（7）可以重寫為



與（5）相矛盾。

Theorem 4 Suppose n periodic tasks with blocking sections are sorted by their periods. They can be feasibly scheduled by DSDR with high speed H and low speed L if

定理4假設n個具有阻塞部分的周期性任務按其周期排序。 可以通過DSDR以高速度H和低速度L合理地安排它們，如果



Proof: As a job's deadline is the same as the deadline of its run time, the claim directly follows from Lemma I and Lemma 2.

證明：由於工作的截止日期與工作時間的截止日期相同，因此索取直接來自引理I和引理2。

Similar to the extension of the dual speed algorithm, the high speed intervals in DSDR can also be shortened. If the deadline of the run time being consumed is later than or equal to the end of the current high speed interval, the high speed interval is tcrminated at oncc. Note that this cxtension does not affect the proof of Lemma 2, so the conclusions in Lemma 2 and Theorem 4 are still valid.

與雙速算法的擴展類似，DSDR中的高速間隔也可以縮短。 如果消耗的運行時間的最後期限晚於或等於當前高速間隔的結束，則將高速間隔定為oncc。 請注意，這種擴展不會影響引理2的證明，因此引理2和定理4中的結論仍然有效。

5績效評估

Simulation experiments were carried out to evaluate the effectiveness of the proposcd algorithms in saving energy. In this section, we first describe the assumptions used and the characteristics of the task sets. The simulation results are then presented and analyzed.

通過仿真實驗評估了擬議算法在節能方面的有效性。 在本節中，我們首先描述所使用的假設和任務集的特徵。 然後給出仿真結果並進行分析。

5.1仿真設置

The event-driven simulator imitates the processing of periodic tasks on a single processor that is capable of voltage/speed scheduling. We assume a maximum processor speed (=1) and a minimum processor speed (=0.1). Speed levels between the two bounds are discrete and spaced by 0.1. The supply voltage is proportional to the processor speed and the power consumption follows formula (l). We also assume the processor does not consume power when it is in the idle state. The simulation experiments consisted of three phases: task generation, admission control, job scheduling and execution.

事件驅動模擬器可在具有電壓/速度調度功能的單個處理器上模擬週期性任務的處理。 我們假設最大處理器速度（= 1）和最小處理器速度（= 0.1）。 兩個界限之間的速度水平是離散的，並且間隔為0.1。 電源電壓與處理器速度成正比，功耗遵循公式（l）。 我們還假設處理器處於空閒狀態時不消耗功率。 仿真實驗包括三個階段：任務生成，准入控制，作業調度和執行。

To simulate the mixed workload in real systems, we generated tasks whose periods belonged to one of three ranges: long period (l000~5000ms), middle period (100~ 1000ms) and short period (20~lOOms). The WCETs of the tasks in the three categories were (1~1000ms), (1~100ms) and (1~20ms), respectively. The tasks were uniformly distributed in thcse categorics. Within each category, the tasks' periods and WCETs were randomly selected from the corresponding ranges. After the task set was generated, the WCETs of the tasks were scaled such that the total processor utilization would not exceed the desired value, which was specified by the utilization factor. The blocking factor specified the maximum percentagc the blocking section could occupy in a job's execution time, so the maximum length of any blocking section of a task was WCET x blocking\_factor.

為了模擬實際系統中的混合工作負載，我們生成了任務，任務的周期屬於以下三個範圍之一：長周期（l000“，5000ms），中周期（100”，1000ms）和短週期（20“，100ms）。WCET 這三個類別中的任務分別為（l'“ I Odoms，（1'” 100ms）和（1？20ms）。這些任務按這些類別均勻地分佈。在每個類別中，任務的周期和WCET 從相應的範圍中隨機選擇。生成任務集後，對任務的WCET進行縮放，以使總處理器利用率不超過期望值，該值由利用率係數指定。阻塞係數指定最大百分比agc 阻塞部分可能會佔用作業的執行時間，因此任務的任何阻塞部分的最大長度為WCET xblocking\_factor。

All tasks generated must undergo an admission control procedure in the order of their released time. We uscd the formula in Section 3 to calculate a static processor speed H. If the required speed was smaller than the maximum processor speed, the new task was admitted; otherwise the task was dropped without processing.

所生成的所有任務必須按照其釋放時間的順序進行准入控製程序。 我們使用第3節中的公式來計算靜態處理器速度H。如果所需速度小於最大處理器速度，則接受新任務；否則，開始執行新任務。 否則，任務將被丟棄而不進行處理。

Finally, the admitted tasks periodically gcnerated jobs. The slack factor specified the difference between the actual job execution time and the task's WCET. Specifically, the execution times of the jobs in a task were uniformly distributed between the task's WCET and (I-slack\_factor) x WCET. Each job could at most have one blocking section, and we used blocking\_prob to represcnt the probability that a job would be assigned a blocking section. The position of the blocking section was randomly chosen. Released jobs were stored in the job queue and processed in the EDF order. The voltage scheduling algorithms presented in this paper were used to determine the processor speed for the current executing job

最後，已接受的任務會定期增加工作量。 鬆弛因子指定了實際作業執行時間與任務的WCET之差。 具體來說，任務中作業的執行時間在任務的WCET和（I-slack\_factor）x WCET之間均勻分配。 每個作業最多可以有一個阻塞部分，我們使用blocking\_prob來表示將為作業分配一個阻塞部分的可能性。 阻塞部分的位置是隨機選擇的。 已釋放的作業存儲在作業隊列中，並按EDF順序進行處理。 本文介紹的電壓調度算法用於確定當前執行作業的處理器速度。

5.2實驗結果

Extensive simulations were conducted with the three proposed algorithms4 In each experiment, we generated a task set of 30 tasks. All tasks were released at time O. The experiment was carried out for 100,00Oms and the energy consumption was recorded. In order to improve accuracy, we performed simulations on 10 distinct task sets for each set of parameters and took the average. The static speed algorithm was used as the baseline and the power consumptions of the other two algorithms were normalized with the baseline.

使用提出的三種算法進行了廣泛的仿真4。在每個實驗中，我們生成了一個包含30個任務的任務集。 所有任務在時間O釋放。實驗進行了10萬毫秒，並記錄了能耗。 為了提高準確性，我們針對每組參數對10個不同的任務集進行了仿真，並取了平均值。 靜態速度算法用作基準，而其他兩種算法的功耗均用基准進行了歸一化。

阻塞參數Blocking Parameters

In the first set of simulations, we evaluated the effect of the blocking sections on power consumption. The blocking parameters blocking\_prob and blocking-fador were varied to produce task sets with different numbers of blocking sections and different maximum blocking lengths. In these experiments, we let all jobs execute at their WCETs. The performance would improve if the actual execution time is less.

在第一組仿真中，我們評估了阻塞部分對功耗的影響。 更改了阻塞參數blocking\_prob和blocking-fador，以生成具有不同數量的阻塞部分和不同最大阻塞長度的任務集。 在這些實驗中，我們讓所有作業都在其WCET上執行。 如果實際執行時間較少，則性能會提高。

We first fixed blocking\_prob to 0.5 and varied blocking\_factor between 0 and 0.3. The range of blocking factor was chosen based on two reasons: i) The blocking sections are usually short in practical applications. ii) As the blocking factor exceeded 0.3, the number of tasks not admitted increased quickly, which affected the accuracy of the results. Figure 5 shows the normalized energy consumption of the dual speed algorithm and the DSDR algorithm under utilization factors 0.4 and 0.6, respectively. As the maximum blocking length increased with blocking factor, the value of high speed H was also increased. The energy consumed by all three algorithms grew with H, but the energy consumption ot' the two dynamic algorithms grew slower because these two algorithms switched to low speed modes in some intervals. As a result, the dual speed algorithm saved up to 70 percent of the energy consumed under the static algorithm. Since DSDR utilized the run time reclaimed in high-speed intervals, it saved more energy. However, the difference is not significant. As observed in the experiments, blocking only occurred rarely and the durations of high speed intervals were very short, which left very little room for DSDR to reclaim. Another observation is that when the blocking factor was smaller than 0.09, thc discrete values of high and low speeds coincided, consequently all three algorithms consumed the same amount of energy.

我們首先將blocking\_prob固定為0.5，然後將blocking\_factor調整為0到0.3之間。選擇阻塞因子的範圍是基於兩個原因：i）阻塞部分在實際應用中通常很短。 ii）當阻塞因子超過0.3時，不被接受的任務數量迅速增加，這影響了結果的準確性。圖5分別顯示了在使用係數0.4和0.6下雙速算法和DSDR算法的歸一化能耗。隨著最大阻塞長度隨阻塞因子的增加而增加，高速H的值也隨之增加。三種算法消耗的能量都隨著H的增加而增加，但是這兩種動態算法的能量消耗卻變慢了，因為這兩種算法在一定間隔內切換到了低速模式。結果，雙速算法可節省多達70％的靜態算法能耗。由於DSDR利用了高速間隔中回收的運行時間，因此節省了更多的能源。但是，差異並不明顯。如實驗中所觀察到的，阻塞很少發生，並且高速間隔的持續時間非常短，這幾乎沒有DSDR回收的空間。另一個觀察結果是，當阻塞因子小於0.09時，高速和低速的離散值重合，因此所有三種算法消耗的能量相同。

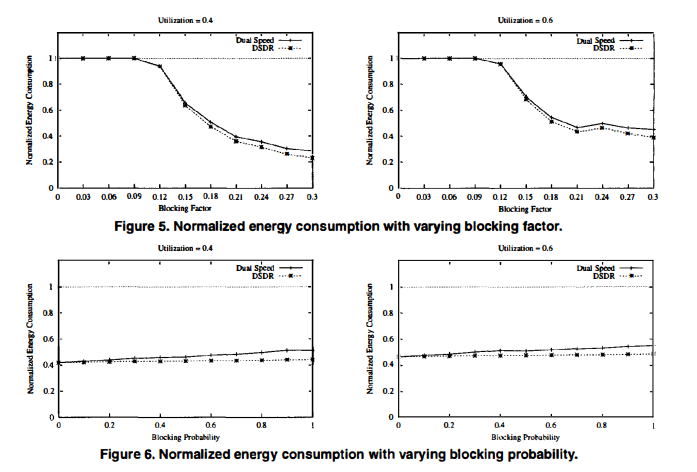


圖5.具有變化的阻塞因子的歸一化能耗。

圖6.具有不同阻塞概率的歸一化能耗。

In the similar way, we have varied blocking\_prob with blocking\_factor fixed at 0.2. As shown in Figure 6, the energy consumed by the dual speed algorithm increased with the blocking probability. However, the observed actual blocking rate was vcry low, so the normalized energy consumption was only increased by less than 10 percent. In DSDR, as a job executing in a high speed interval can use the run time in the FRT-Iist to compensate for its run time loss due to run time reclamation (see Section 4.2), the impact on energy consumption was even less (less than 4 percent).

以類似的方式，我們更改了blocking\_prob，並將blocking\_factor固定為0.2。 如圖6所示，雙速算法消耗的能量隨阻塞概率的增加而增加。 但是，觀察到的實際阻塞率非常低，因此歸一化能耗僅增加了不到10％。 在DSDR中，由於以較高的時間間隔執行的作業可以使用FRT-Iist中的運行時間來補償由於運行時間回收而導致的運行時間損失（請參閱第4.2節），因此對能耗的影響甚至更少（更少） 超過4％）。

鬆弛因子Slack Factor

In this set of simulations, we fixed blocking\_factor and blocking\_prob to 0.2 and 0.5, respectively. Figure 7 shows the simulation results for slack factor between 0 and 0.9. The dual speed algorithm was insensitive to the variation of the slack factor. Again, DSDR far outperformed the dual speed algorithm due to its ability to reclaim unused run time from early completions. For example, at utilization 0.4 and slack factor 0.5, DSDR saved 70 percent while the dual speed algorithm saved only about 50 percent of the power consumed under the static speed algorithm.

在這組模擬中，我們分別將blocking\_factor和blocking\_prob固定為0.2和0.5。 圖7顯示了鬆弛因子在0到0.9之間的仿真結果。 雙速算法對鬆弛因子的變化不敏感。 同樣，由於DSDR能夠從早期完成中回收未使用的運行時間，因此其性能遠勝於雙速算法。 例如，在利用率0.4和鬆弛因子0.5的情況下，DSDR節省了70％，而雙速算法僅節省了靜態速度算法下所消耗功率的50％。

處理器利用率Processor Utilization

Simulations experiments were also performed at different system load levels. Again we fixed blocking\_factor and blocking\_prob at 0.2 and 0.5, respectively. Two sets of experiments were carried out with slack factor set at 0.5 and 0.8, respectively. The utilization factor was varied between 0.1 and 1.0 (see Figure 8). The normalized energy consumptions remained relatively constant when the utilization was low but grew as the utilization exceeded 0.6. It turned out the growth was due to task dropping in the admission control phase. Tasks with large blocking sections or small periods were more likely to be dropped since they would significantly increase H. As a result, H - L began to shrink, which reduced the energy saving of the dynamic algorithms. In all cases, DSDR consistently outperformed the dual speed algorithm.

還在不同的系統負載水平下進行了仿真實驗。 同樣，我們將blocking\_factor和blocking\_prob分別固定為0.2和0.5。 進行了兩組實驗，鬆弛因子分別設置為0.5和0.8。 利用率在0.1到1.0之間變化（見圖8）。 當利用率較低時，歸一化的能源消耗保持相對恆定，但當利用率超過0.6時，標準化能耗就會增加。 事實證明，增長是由於在准入控制階段任務下降。 具有較大阻塞部分或較小周期的任務更有可能被丟棄，因為它們會顯著增加H。結果，HL開始縮小，這降低了動態算法的能耗。 在所有情況下，DSDR始終優於雙速算法。

In summary, both dynamic voltage scheduling algorithms outperformed the static speed algorithm under all parameter settings. Taking advantage of the dynamic reclaiming mechanism, DSDR saved even more energy than the dual speed algorithm, especially when the actual execution times were less than their WCETs.

總之，在所有參數設置下，兩種動態電壓調度算法均優於靜態速度算法。 利用動態回收機制，DSDR比雙速算法節省了更多的能量，尤其是當實際執行時間少於其WCET時。

6相關工作

Since the seminal paper by Weiser et al. [12], much work on voltage scheduling has been published. The work in this area can be classified into two categories: intervalbased scheduling [3, 10, 1 2] and profile-based scheduling [ 1, 2, 9, 1 1].

由於Weiser等人的開創性論文。 [12]，關於電壓調度的許多工作已經出版。 該領域的工作可以分為兩類：基於間隔的調度[3、10、1 2]和基於概要文件的調度[1、2、9、1 1]。

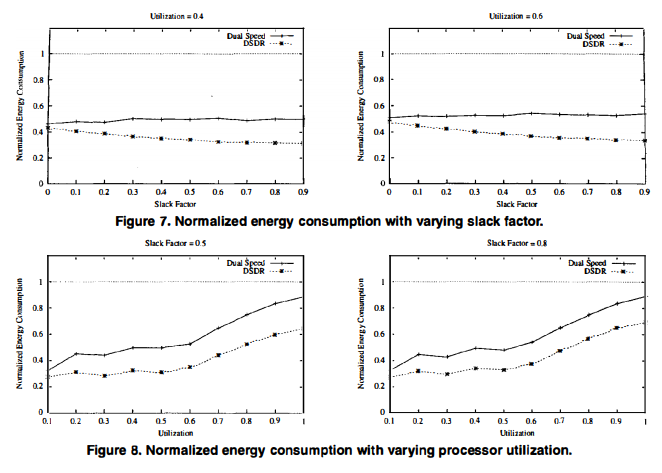


圖7.具有變化鬆弛因子的標準化能耗

圖8.處理器利用率不同時的標準化能耗。

Interval-based scheduling estimates the processor utilization based on observations in the past interval(s). For example, the PAST algorithm records the processor utilization in the previous interval. If the utilization exceeded the upper threshold, the supply voltage is incremented; if the utilization was below the lower threshold, the supply voltage is decremented [12]. The weighted-AVG algorithm [10] makes use of the processor utilization in the previous interval as well as the average utilization in all past intervals. Thus prediction is made based on both short-term and longterm system behaviors. Simulations using real-life traces were also carried out to evaluate the performance of these algorithms. More recently, Lorch et al. [3] proposed to use distributions to estimate the processing requirements of future tasks from the recent behaviors of similar tasks. They suggested using the gamma model for its low complexity and good predicating capability.

基於時間間隔的調度基於過去時間間隔的觀察值來估計處理器利用率。 例如，PAST算法記錄先前間隔中的處理器利用率。 如果利用率超過上限閾值，則電源電壓增加；否則，電源電壓增加。 如果利用率低於下限閾值，則電源電壓會降低[12]。 加權AVG算法[10]利用了先前間隔中的處理器利用率以及所有過去間隔中的平均利用率。 因此，基於短期和長期系統行為進行預測。 還使用現實生活中的痕跡進行了仿真，以評估這些算法的性能。 最近，Lorch等人。 [3]提出使用分佈來根據相似任務的近期行為來估計未來任務的處理需求。 他們建議使用伽瑪模型，因為它具有較低的複雜度和良好的預測能力

Interval-based algorithms assume the workload is more or less stable (or follows some distributions) so that the behaviors of future tasks can be accurately estimated based on past observations. The accuracy in predicting future tasks significantly affects possible energy reduction. This kind of strategies is not suitable for real-time tasks. As time constraints are not considered, the algorithms may improperly lower the processor speed and cause deadline misses, especially when the execution demands vary greatly from job to job.

基於時間間隔的算法假定工作負載或多或少是穩定的（或遵循某些分佈），因此可以根據過去的觀察準確估計未來任務的行為。 預測未來任務的準確性會顯著降低能耗。 這種策略不適用於實時任務。 由於未考慮時間限制，因此算法可能不適當地244降低處理器速度並導致截止期限丟失，尤其是當執行要求因工作而異時尤其如此。

Profile-based scheduling is usually used to schedule realtime tasks. Some knowledge of the released jobs, and to some extent of the future jobs in the current task set is supposed to be known. Yao et al. [11] derived an optimal offline scheduling algorithm for aperiodic real-time tasks. An online approximation algorithm was also presented in the same paper. Hong et al. [9] used a similar planning-based approach to handle non-preemptible tasks. The traditional real-time periodic task model is used in some recent work. Pillai and Shin [2] studied both fixed priority and dynamic priority scheduling and proposed schemes under the ratemonotonic (RM) algorithm and the earliest-deadline-first (EDF) algorithm individually. Their look-ahead algorithm takes the processing requirements of future jobs into consideration and delay processing of these jobs as much as possible. Aydin et at [I] proved there is an optimal static voltage that can feasibly schedule periodic tasks with minimum power consumption. They also proposed a dynamic algorithm which uses the slacks of the jobs that complete early.

基於配置文件的調度通常用於調度實時任務。應當知道在當前任務集中已發布作業的某些知識，以及在某種程度上將來的作業。姚等。 [11]推導了一種針對非週期性實時任務的最優離線調度算法。在同一篇論文中還提出了一種在線近似算法。 Hong等。 [9]使用類似的基於計劃的方法來處理不可搶占的任務。在最近的一些工作中使用了傳統的實時定期任務模型。 Pillai和Shin [2]分別研究了固定優先級和動態優先級調度，並分別提出了基於速率單調（RM）算法和最早截止時間優先（EDF）算法的方案。他們的超前算法考慮了未來作業的處理要求，並儘可能地延遲了這些作業的處理。艾登（Aydin）等人[I]證明了有一個最佳的靜態電壓，可以以最小的功耗可行地安排週期性任務。他們還提出了一種動態算法，該算法利用了較早完成的工作的懈怠。

All of the above work assumed that tasks were either fully preemptible or completely non-preemptibIe. Scheduling tasks with blocking sections usually require a higher processor speed than the speed needed for fully preemptible tasks (i.e., the utilization speed). Our approaches utilize this fact and allow the processor to operate at the utilization speed or lower at strategic intervals without impairing the feasibility of the task set.

以上所有工作均假設任務是完全可搶占的或完全不可搶占的。 與阻塞部分一起調度任務通常需要比完全可搶占任務所需的速度（即，利用速度）更高的處理器速度。 我們的方法利用了這一事實，並允許處理器在策略性間隔內以利用率或更低的利用率運行，而不會損害任務集的可行性。

7結論

In this paper, we have i nvestigated voltage scheduling of real-time periodic tasks with non-preemptible blocking sections. Three voltage scheduling schemes have been proposed to minimize energy consumption while satisfying the tasks' time constraints. The static speed scheme is based on the stack resource policy (SRP) [13] and calculates a minimal feasibly static speed. Instead of always operating at one static speed, the dual speed algorithm lowers the processor speed to the utilization speed in some intervals. We have also presented a reservation-based scheme which reserves run time for each job. A reclaiming mechanism is used to collect the unused run time and redistribute it to jobs that are able to make use of it. As the jobs that receive extra run time are eligible to run for a longer period of time, the processor speed can be further reduced to save energy.

在本文中，我們研究了具有不可搶占阻塞部分的實時周期性任務的電壓調度。 已經提出了三種電壓調度方案，以在滿足任務的時間約束的同時，將能耗降至最低。 靜態速度方案基於堆棧資源策略（SRP）[13]，併計算最小的可行靜態速度。 雙速算法不是始終以一個靜態速度運行，而是在某些時間間隔內將處理器速度降低至利用率。 我們還提出了一種基於預留的方案，該方案為每個作業保留運行時間。 回收機制用於收集未使用的運行時間，並將其重新分配給能夠利用它的作業。 由於獲得額外運行時間的作業有資格運行更長的時間，因此可以進一步降低處理器速度以節省能源。

Feasibility conditions have been derived for the three algorithms and provcd mathematically. Simulation experiments were carried out to evaluate the performance in energy saving. The results show that both dynamic algorithms can significantly reduce energy consumption compared with the static speed algorithm in all scenarios.

推導了這三種算法的可行性條件，並在數學上進行了證明。 進行了仿真實驗，以評估其節能性能。 結果表明，在所有情況下，與靜態速度算法相比，兩種動態算法都可以顯著降低能耗。