Xen虚拟机Credit调度算法的实时性能分析

张天宇 关 楠 邓庆绪
(东北大学信息科学与工程学院 沈阳 110819)

摘 要 为了降低开销以及增加灵活性,通过虚拟化技术将多个系统运行在一个通用计算平台上已成为复杂实时嵌入式系统的趋势。Xen是近年来应用最广泛的虚拟化技术,对其默认使用的Credit调度算法进行实时性能分析,使得能够直接对运行在Xen上的实时系统进行可调度性测试,并且可以通过形式化的资源界限函数对Credit的实时性进行直观的评估。首先分析了Credit调度算法的基本实现,提出并且证明了一种配置VCPU参数的方法使得Credit的实时性得到提升,在此基础上,通过证明得到了Credit算法的基本性质,并得出其在最坏情况下为VCPU分配的资源函数曲线。

关键词 虚拟化,Xen,调度算法,实时系统,资源界限函数
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.026

Analysis of Real-time Performance of Algorithm Credit in Xen Virtual Machine

ZHANG Tian-yu GUAN Nan DENG Qing-xu
(School of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract The development of complex real-time embedded systems has become a trend in recent years. To reduce cost and enhance flexibility, multiple systems are sharing common computing platforms via virtualization technology. We studied the real-time performance of algorithm Credit in Xen which is the most popular virtual machine monitor. Firstly we analyzed the basic implementation of Credit which is the default scheduling algorithm in Xen virtual machine. Then we proposed and proved an effective method that can configure VCPU parameters to improve the real-time performance of Credit for promotion. On this basis, we finally obtained the resources function curve SBF allocated for VCPU in the worst case by showing and getting the basic properties of Credit scheduling algorithm.

Keywords Virtualization, Xen, Scheduling algorithms, Real-time systems, Supply bound function

1 引言

近年来,复杂嵌入式实时系统正在从多个相互隔离的物理主机向通用的计算平台上迁移。通用计算平台不仅能够降低成本和体积,而且能够增加资源分配的灵活性。这种迁移主要是通过虚拟化技术将不同功能的系统映射到不同的虚拟机中实现。然而虚拟化在越来越多地被应用到复杂实时系统的同时,也为实时系统的可调度性分析带来了新的挑战:通用平台需要同时满足所有虚拟机中实时子系统的可调度性,因此需要对每个虚拟机的实时性能进行评估。在此背景下,本文对目前应用最广泛的虚拟化技术Xen进行实时性能的分析。Xen[1]是一个开源的虚拟机项目,其性能接近单机操作系统的性能。在Xen系统中,存在一个轻量级的软件层虚拟机管理器(VMM,也称hypervisor),位于操作系统与硬件之间,向运行在它之上的虚拟机提供、分配并且管理虚拟硬件资源,保证虚拟机之间的相互隔离,实现在一台单独的计算机上运行多个子操作系统。Xen的调度框架如图1所示。多任务是Xen的关键特性,从外部看,有多个专用的操作系统"同时"运行在一台物理机器上。因此,Xen需要在一段固定的时间内将各个CPU时间片合理、高效地分配给在其上运行的各个操作系统,从而达到多任务的目的。这个分配CPU时间片的算法就是通常说的调度算法。

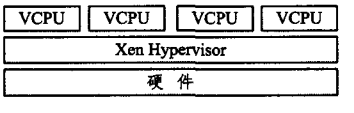

图1 Xen调度框架

在虚拟机中,虚拟处理器(Virtual CPU,VCPU)调度算法是指由虚拟机根据一定的策略决定当前哪一个VCPU在物理处理器上执行的算法。VCPU调度算法的好坏对处理器虚拟化性能影响较大,所以好的调度算法能够很大程度上提高Xen的实时性能,VCPU调度是近年来虚拟化技术研究的一个重点。Xen中两个最常用的调度算法是Credit和SEDF[2],其中系统默认使用的是Credit算法,而Credit最大的特点是其注重维护系统中所有VCPU对资源使用的公平

到稿日期:2015-02-14 返修日期:2015-04-06 本文受国家自然科学基金(61472072),国家973预研计划项目(2014CB360509),国家科技支撑计划项目(2012BAF13B08)资助。
张天宇(1988—),男,博士生,主要研究方向为嵌入式实时系统,E-mail:ztylll@126.com;关 楠(1981—),男,博士,副教授,主要研究方向为嵌入式实时系统、多核调度;邓庆绪(1970—),男,博士,教授,主要研究方向为嵌入式实时系统、无线传感器网络等。

性而忽略了实时性，这直接导致了 Credit 在处理实时任务时存在缺陷，这也是本文所要研究并解决的重点。对于 SEDF 算法，由于其不支持全局负载平衡，在 Xen 最新版本中已经计划逐步将其淘汰[3]，故不做相关研究。本文主要的贡献是对 Credit 算法的实时性能进行分析：首先通过源码（Xen4.1.4 版本）分析其基本实现，提出并且证明了一种有效配置 VCPU 参数的方法使得 Credit 的实时性能得到提升，在此基础上，通过证明得到 Credit 调度算法的基本性质，并最终得出其在最坏情况下为 VCPU 分配的资源函数曲线。

2 相关工作

近年来，学者们在理论研究领域对 Xen 做了大量研究。Govindan 等人[4] 为了提高 Xen 的 I/O 性能，尤其是缩短 VMM 在处理网络通信任务时的响应时间，对 SEDF 算法进行了修改，增加了负责 I/O 通信的虚拟 CPU 的优先级。美国乔治亚理工学院的 Min Lee 等人[5] 以网络电话为例详细说明 Xen 中软实时任务的负载变化，并且为 Credit 算法设计了一种支持 SMP 负载平衡的软实时调度器，以此在 Xen 中加入了对处理软实时任务的支持。

在对 Xen 的实时性研究方面，近年来由 Lee 等人提出并设计的 RT-Xen[6,7] 受到了研究者们广泛的关注。RT-Xen 是 Xen 的第一个实时虚拟调度框架，拉近了实时调度理论和 Xen 之间的距离。RT-Xen 提供了一个开源的平台来发展和评估实时调度，目前还处于分析和模拟的阶段。他们还提出一种基于 Xen 虚拟平台的集中调度框架和平台结构来支持集成实时系统的调度，并使用周期资源模型作为组建接口的集中调度理论。华中科技大学的金海提出“一种虚拟 CPU 调度算法”[8]，该算法针对多处理机架构研究了虚拟化环境中对虚拟机的监测和 CPU 调度问题，能够获得较高的物理 CPU 利用率，并且可以监测到更精确的可调度状态。上海交通大学的顾振宇等人[9] 也对 Credit 算法进行了优化：在全虚拟化环境中运行反馈进程，该进程会向 hypervisor 反馈 Guest 操作系统的负载状态，从而优化 Xen 对物理 CPU 资源的分配情况。

但是，以上研究都没有对 Xen 调度算法的实时性能给出形式化的评估。而对于实时系统，必须在运行之前对其做可调度性测试，以确保在系统实际运行过程中不会出现任务错失截止期的情况。无论使用哪种可调度性判定方法，都需要对处理器的处理能力进行形式化定义，即资源界限函数。本文即在此背景下，研究 Credit 算法的资源界限函数。

3 Credit 调度算法

Credit 是一种按比例公平共享的非抢占式算法，其允许所有正在执行的任务使用完自己的时间片，当正在执行的任务放弃 CPU 时，CPU 才会调用其他任务执行。Credit 调度算法支持 SMP 负载平衡（当某个 CPU 处于空闲状态时，它将会从其他的 CPU 上寻找可以运行的 VCPU）。这种方法能够很好地保证在系统中当有可以运行的 VCPU 时不会出现空闲的 CPU，有效地提高了系统的资源利用率。因此，Xen 调度器默认使用 Credit 调度算法，但由于其公平性原则，Credit 在处理实时性任务时的性能不高。

3.1 Credit 基本实现

Credit 调度算法为每一个 VCPU 设置二元组（*weight*，*cap*），*weight* 表示 VCPU 可以占用 CPU 时间片的比例；*cap* 决定一个 VCPU 每次占用 CPU 时间的上限值。默认 $cap=0$，表示可以占用任意大小 CPU 时间，当 $cap=50(100)$ 时，表示最多可占用半个（一个）CPU 时间。Credit 算法为每个 VCPU 设置一个信用度值，VCPU 每次获得处理器执行会消耗其信用度值，之后由一个结算线程重新计算各个 VCPU 的信用度值，在运行过程中，调度器根据信用度的大小选择合适的 VCPU 进行调度。Credit 默认每次触发调度的时间片大小为 30ms，即每个 VCPU 在被其他 VCPU 抢占前都可以获得 30ms 的执行时间。每过 30ms，所有可运行的 VCPU 的信用度值就会被重新计算。

Credit 将所有 VCPU 分为两个队列：UNDER 和 OVER，每次触发调度都从 UNDER 中选择队首的 VCPU 调度。初始时，所有 VCPU 信用度值都为 0，并全部顺序加入 UNDER 中，挑选 UNDER 队首的 VCPU 使其执行一个时间片即 30ms，每当 VCPU 被调度时，该 VCPU 的信用度值减少 300，其他 VCPU 的信用度值不变。每 30ms 为所有 VCPU 加上其对应 *weight* 比值的信用度值，检查当前执行的 VCPU 的信用度值，若为负，则将其插入 OVER 队尾。之后检查 OVER 中是否存在信用度值为正的 VCPU，若存在，则将其重新插入到 UNDER 队尾。如果某一个 VCPU 的信用度值超过了总的信用度值（即 300），那么将其信用度值减半，并且之后不再为其增加信用度值，直到该 VCPU 下次被调度执行。当为某一个 VCPU 的信用度值减半之后，会把减少的这部分信用度值加到总的信用度值中，在下一次为 VCPU 增加信用度值时按相应的 *weight* 比例重新分配。之后再挑选 UNDER 队首的 VCPU 执行 30ms。

3.2 Credit 算法实例

假设系统中存在 v_a，v_b，v_c 3 个 VCPU，信用度值总和为 300，v_a，v_b，v_c 的 *weight* 值比例 $W_a:W_b:W_c=1:3:6$，即每次获得的信用度值分别为 $C_a=30$，$C_b=90$，$C_c=180$。初始状态 v_a，v_b，v_c 的信用度值都为 0。Credit 调度 3 个 VCPU 的过程如图 2 所示，其中 v_b 执行结束后，v_c 信用度值 360 超过上限值（300），逐步减半变为 180，并将损失的 180 按 1:3 的比例重新分配给 v_a 和 v_b（−195，15，180）。

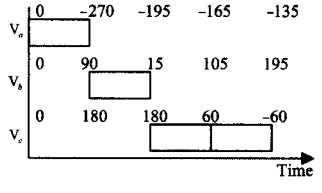

图 2 Credit 算法示例

4 Credit 实时性分析

Credit 最大的特点就是注重保持所有 VCPU 对处理器资源共享的公平性，如果初始状态下某个或某些 VCPU 的 *weight* 比值较大，系统运行过程中，这些 VCPU 的信用度值可能会超过上限值 300，此时该 VCPU 的信用度值被减半，而 Credit 会把这部分“损失”的信用度按比例分配给其他 VCPU。换言之，*weight* 比值较大的 VCPU 在系统运行中会一直被其他 *weight* 比值较小的 VCPU 瓜分处理器资源。但是从实时性的角度考虑，这种限制信用度值上限的方法对某

些处理器需求较高的实时任务来说并不"公平"。因此,本文首先找到一种为所有 VCPU 设置 *weight* 比值的约束,用以保证不会出现某些 VCPU 的信用度值在运行过程中超过上限。

定理 1 Credit 算法执行过程中不出现某些 VCPU 的信用度值超过上限的充分条件:对于任意一个 VCPU,满足 $W_i/T\cdot(n-1)\leqslant 1$,其中 W_i 表示第 i 个 VCPU v_i 对应的 *weight* 比值,T 表示 VCPU 的执行周期,n 为系统中 VCPU 的个数。

证明:首先证明在执行过程中若不出现某些 VCPU 信用度值超过上限的情况下,Credit 算法存在的两个基本性质。

性质 1 Credit 执行过程中存在一个周期 T 等于所有 VCPU 的 *weight* 比值的约分之和,此时称 T 为 VCPU 的一个执行周期,即当系统中所有 VCPU 的信用度值均为 0 时,经过 T 次执行后(任意一个 VCPU 执行 30ms 称为一次执行),所有 VCPU 的信用度值均再次变为 0。

证明:使用 $v_1,v_2,\cdots,v_n$ 分别表示系统中 n 个 VCPU,令所有 VCPU 的 *weight* 比值为 $W_1:W_2:\cdots:W_n$,且 $W_1+W_2+\cdots+W_n=T$,则每次为 v_i 增加的信用度值 $C_i=300*W_i/T$。

(1)首先证明在所有 VCPU 经过 T 次执行后,任意一个 VCPU v_i 一共执行的次数为 W_i 次。假设 v_i 实际执行的次数为 n_i,考虑以下两种情况。

①若 $n_i>W_i$:经过 T 次执行后,v_i 一共可以获得的信用度值总数为 $C_i*T=(300*W_i/T)*T=300*W_i$,实际执行 n_i 次需要消耗的信用度值为 $300*n_i$,即经过 T 次执行后,v_i 的信用度值为 $300*(W_i-n_i)$。假设 v_i 的最后一次执行发生在第 t 次,该次执行结束后 v_i 的信用度值记为 C_t,则 C_t 一定满足 $C_t\leqslant 300*(W_i-n_i)$,因为 t 次执行之后 v_i 不再执行,即不再消耗信用度值,信用度值一直在增加,那么 v_i 在第 t 次执行前的信用度值为 C_t-C_i+300(执行一次消耗 300,补充 C_i)。由 $n_i>W_i$ 且 $C_t\leqslant 300*(W_i-n_i)$,得到 $C_t-C_i+300<0$,说明 v_i 在第 t 次执行之前的信用度值为负,此时 v_i 应在 OVER 队列中,一定不可能执行,矛盾,因此 n_i 一定不大于 W_i。

②若 $n_i<W_i$:因为 $n_1+n_2+\cdots+n_n=W_1+W_2+\cdots+W_n=T$,则一定至少存在一个 VCPU 的实际执行次数 $n_j>W_j$,同①矛盾。因此 n_i 一定不小于 W_i。

综上,v_i 实际执行次数 $n_i=W_i$,即在所有 VCPU 经过 T 次执行后,v_i 一共执行了 W_i 次。

(2)证明所有 VCPU 信用度值均为 0 时,经过 T 次执行,所有 VCPU 信用度值再次变为 0。因为所有 VCPU 经过 T 次执行后,v_i 一共执行了 W_i 次,所以 v_i 总共消耗的信用度值为 $300*W_i$,而 v_i 总共获得的信用度值为 $C_i*T=(300*W_i/T)*T=300*W_i$,因此,所有 VCPU 经过 T 次执行后 v_i 的信用度值最终仍为 $300*W_i-C_i*T=0$,性质 1 得证。

性质 2 若 Credit 执行过程中不出现某些 VCPU 信用度值超过上限的情况,那么在每一次调度决定发生之前,系统中所有 VCPU 的信用度值总和为 0。

证明:(归纳法)当第一次调度发生之前(初始状态下),系统中所有 VCPU 的初始信用度值都为 0,所以总和仍为 0。假设当第 k 次调度决定发生之前,所有 VCPU 的信用度值的总和为 0,当下一次即第 $k+1$ 调度之前,有且仅有一个 VCPU 执行过,并将相应的信用度值减少 300,之后将其消耗的 300 信用度值按所有 VCPU 的 *weight* 比值分配给每一个 VCPU,因此所有 VCPU 的信用度总和不变,仍为 0,证毕。

接下来继续对定理 1 进行证明。Credit 算法执行过程中每次为 v_i 增加的信用度值 $C_i=300*W_i/T$,显然,当 v_i 位于 UNDER 队列尾部时,在其下次被调度到时能够累计获得的信用度值最多。那么假设在某一个时刻 t,v_i 位于 UNDER 队列的尾部。

①假设在 t 时刻,v_i 的信用度值为 0,那么此时 UNDER 队列中排在 v_i 之前的 VCPU 最多有 $n-1$ 个。任一时刻刚刚执行完且信用度值仍为正的 VCPU 都会被插入 UNDER 队尾,下次执行一定在 v_i 之后,因此,这 $n-1$ 个 VCPU 最多能够在 v_i 之前执行 $n-1$ 次,之后当 v_i 再次被调度到时,其信用度值为 $0+C_i*(n-1)$。

②假设在 t 时刻,v_i 的信用度不为 0,若小于 0,v_i 一定位于 OVER 队列中,与假设矛盾,因此 v_i 的信用度值一定大于 0,记为 c。假设此时 UNDER 队列中排在 v_i 之前的 VCPU 的个数为 m 个,同上,这 m 个 VCPU 最多能够在 v_i 之前执行 m 次,因此 v_i 下次被调度时其信用度值为 $c+C_i*m$。

对于情况②,v_i 一定是从 OVER 队列中增加了一次信用度值 C_i 后被插入到 UNDER 队列中的,所以很显然 c 一定满足 $c<C_i$;另一方面,因为此时 $c>0$,即 v_i 的信用度值大于 0,由性质 2,任意时刻所有 VCPU 的信用度值之和等于 0,推出此时一定至少存在一个 VCPU 的信用度值小于 0,即 OVER 队列中至少存在一个 VCPU,那么此时 UNDER 队列中除了 v_i 最多还有 $n-2$ 个 VCPU,即 $m\leqslant n-2$。综上,$0+C_i*(n-1)\geqslant c+C_i*m$。所以为了保证不出现信用度值超过上限的情况,只需要保证 $0+C_i*(n-1)\leqslant 300$,即 $W_i/T*(n-1)\leqslant 1$,定理 1 得证。

在实时调度研究领域,对于一个给定的实时任务集,判断其可调度性的传统分析手段,即使用需求界限函数(Demand Bound Function,DBF[10])以及资源界限函数(Supply Bound Function,SBF[11])的方法。简单来讲,*DBF* 表示在固定大小的时间区间内,任务集在最坏情况下需要的处理器资源大小。相应地,*SBF* 表示在固定大小的时间区间内,处理器能够保证最少提供的处理器资源大小。显然,如果对于某个给定的任务集,其对处理器需求的上限都不超过处理器提供处理能力的下限,即 $DBF\leqslant SBF$,那么该任务集一定是可调度的。因此,为了提高可调度性,好的调度算法需要尽可能地降低任务集的 *DBF*,使其满足可调度性判定的条件;同样地,"好的"处理器则需要尽可能地提升其 *SBF* 来保证任务集的可调度。换言之,衡量一个处理器实时性能的高低,最直接且有效的方式即是得到其资源界限函数 *SBF*。

定义 1(资源界限函数 *SBF*) 对于一个给定的处理器,使用 $S(a,b)$ 表示其在时间区间 $[a,b)$ 内提供的处理器资源的大小,则其 $sbf(l)=\{\min(S(a,b))\mid b-a=l\}$。接下来证明在定理 1 的基础上 Credit 算法的资源界限函数 $sbf^*(l)$。

根据 Xen 的体系结构,系统中所有的实时任务全部运行在不同的 VCPU 中,那么对于其中某个 VCPU v_i 下的任务而言,其获得的处理器资源大小是由该 VCPU 所获得实际处理器资源多少决定的。换言之,本文所要研究的 Credit 下的 *SBF* 即是每个 VCPU 在 Credit 算法执行过程中所获得的最坏情况下的实际处理器资源。

定理 2 任意一个 VCPU v_i 的两次连续执行之间最多间隔 $T-W_i+1$ 次其他 VCPU 的执行,其中 W_i 表示 v_i 对应的

weight 比值，T 为执行周期。

证明：在 Credit 算法执行过程中的某个执行周期 T_p 内，假设 v_i 在 T_p 内的最后一次执行结束之后，其他 VCPU 还需要执行 m 次。由于 Credit 每次调度都是选择位于 UNDER 队列队首的 VCPU，并且每个执行周期结束后所有 VCPU 的信用度值均为 0，因此每个周期内最后一个执行的 VCPU，其执行结束后信用度值变为 0，不会被插入到 OVER 队列，而是仍然位于 UNDER 队列队首，即下个执行周期开始时，仍然是该 VCPU 被第一个调度。因此，在周期 T_p 内的这 m 次执行中，只有最后一个执行的 VCPU，记为 v_{last}，如图 3 所示，在下个周期 T_p+1 开始时，在 UNDER 队列中的位置位于 v_i 之前，而其他 VCPU 由于在 T_p 内比 v_i 更晚进入 OVER 队列，因此在 T_p+1 内重新回到 UNDER 队列时也一定排在 v_i 之后。在下个周期 T_p+1 中，v_{last} 只会在周期开始时执行一次就会进入 OVER 队列，并且当 v_{last} 重新回到 UNDER 队列时也一定排在 v_i 之后（即 v_{last} 再一次执行之前 v_i 一定已经执行过了），那么在下个周期 T_p+1 开始时的 UNDER 队列中，排在 v_i 之前的 VCPU 在 v_i 的首次执行前最多连续执行 $T-W_i-m+1$ 次（即在周期 T_p 内，$T-W_i-m$ 次的执行全部为单位大小的 VCPU（即每个 VCPU 都只执行一次），这样能保证在 T_p+1 内连续执行次数最多）。综上，v_i 跨越连续周期 T_p 和 T_p+1 的两次执行之间最多间隔 $m+(T-W_i-m+1)=T-W_i+1$ 次。

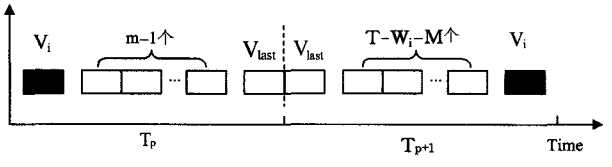

图 3　定理 2 证明示意

定理 2 证明了对于任意 VCPU 在最坏情况下的两次执行之间的时间间隔大小。接下来具体分析对于系统初始状态下设置不同的 VCPU 的 *weight* 比值其相应的资源界限函数的形式化定义。

定理 3　根据系统初始对 VCPU 配置的不同，若所有 VCPU 的 *weight* 比值设为 $W_1:W_2:\cdots:W_n=1:1:\cdots:1$，则任意一个 VCPU v_i 的资源界限函数为：

$$sbf_i(t)=\left[\frac{t \bmod T^2}{T+1}\right]+\left[\frac{t}{T^2}\right]\cdot T \tag{1}$$

若 $W_1:W_2:\cdots:W_n\neq 1:1:\cdots:1$，$v_i$ 的资源界限函数为 $sbf_i{}^*$，具体形式化定义见证明。

证明：使用 $v_1,v_2,\cdots,v_n$ 表示 n 个 VCPU，令所有 VCPU 的 *weight* 比值为 $W_1:W_2:\cdots:W_n$，并且根据性质 1，$W_1+W_2+\cdots+W_n=T$。

(1)首先证明对于某个 v_i，将其他所有 VCPU 任意合并或拆分（如 $W_a:W_b:W_c:W_d=1:2:3:4$，v_b 和 v_c 合并后变为 $W_a:W_{bc}:W_d=1:5:4$），对 v_i 所获得的处理器资源量无影响，即 v_i 在每个周期 T 内的执行次数不变。

证明：将除 v_i 之外的 VCPU 任意合并或拆分，v_i 的 *weight* 比值不变，而在性质 1 中已经证明过，对于 *weight* 比值为 W_i 的 VCPU v_i，其在每个周期 T 内的执行次数为 W_i，因此 v_i 在每个周期 T 内的执行次数不变。

(2)求 v_i 在不同 *weight* 比值配置下的资源界限函数 sbf_i。

证明：①若所有 VCPU 的 weight 比值相同，即 $W_1:W_2:\cdots:W_n=1:1:\cdots:1$。此时假设 v_i 处在初始 UNDER 队列的第 i 位，由于所有的 VCPU 的 *weight* 比值都是单位大小的，每个 VCPU 在每个周期 T 内都只执行一次，则在第一个执行周期 T 内，所有 VCPU 都按照初始 UNDER 队列中的顺序依次执行，因此 v_i 将在第 i 次得到执行。因为每个周期 T 后所有 VCPU 的信用度值都变为 0，因此第一个执行周期 T 后，初始 UNDER 队列中的最后一个 VCPU（记为 v_n）变为 UNDER 队列的队首，即在下一个执行周期中第一个执行，其他所有 VCPU 都顺序依次向后推一位，因此在第二个周期 T 内 v_i 将在第 $i+1$ 次得到执行，之后依次类推。以 a,b,c,d 4 个 VCPU，*weight* 比值 $W_a:W_b:W_c:W_d=1:1:\cdots:1$ 见例，执行效果如图 4 所示。这种情况下，对于任意一个 VCPU v_i，其 sbf_i 的形式化定义为式(2)。sbf_i 曲线如图 5 所示。

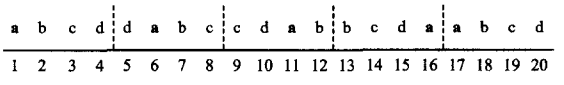

图 4　执行效果示例

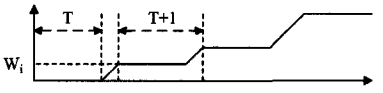

图 5　sbf 曲线

②若 $W_1:W_2:\cdots:W_n\neq 1:1:\cdots:1$。当 v_i 的 *weight* 比值 $W_i\neq 1$ 时，假设初始 UNDER 队列中 VCPU 的顺序为 $v_1,v_2,\cdots,v_i,\cdots,v_n$，根据 v_i 所处在队列中位置的不同，v_i 的执行状态可以分为以下两种情况。

a)第一个周期 T 的最后一次执行是 v_i。在这种情况下，v_i 位于第 2 个周期 UNDER 队列的队首，即 $v_i,v_1,v_2,\cdots,v_n$，因为 $W_i<T$，即每次为 v_i 增加的信用度值 $C_i=300*W_i/T<300$，队首的 v_i 执行一次被减去 300 信用度值后增加的信用度值不够使其信用度值大于 0，所以 v_i 只能执行一次就被加入到 OVER 中，之后 UNDER 中剩余的 $v_1,v_2,\cdots,v_n$ 依次执行，最后执行 W_i-1 次 v_i（v_i 在每个 T 内一共执行 W_i 次）。由于在该周期 T 的最后一次执行仍然是 v_i，因此之后的每一个周期都重复该周期的执行过程，如图 6 所示。此时 v_i 所获得的处理器资源可看成时分复用 TDMA（Time-Division Multiplexing Access）[12]，则周期为 T，时间片大小为 W_i 的 TDMA 所提供的 sbf 为：

$$sbf_i^{TDMA}(t)=\max(t \bmod T-T+W_i,0)+\left\lfloor\frac{t}{T}\right\rfloor\cdot W_i \tag{2}$$

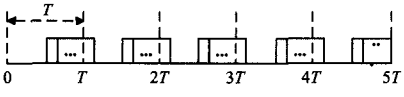

图 6　TDMA 模式示意

b)第一个周期 T 的最后一次执行不是 v_i。在这种情况下，v_i 一定是从 T 中的某个位置开始连续执行了 W_i 次。若不是，假设 v_i 从某个位置开始第一次执行，并且没有连续执行完 W_i 次，那么根据情况 1 中的分析，v_i 被加入 OVER 队列之后直到其他所有 VCPU 执行完后才能执行，则周期 T 的最后一次执行一定是 v_i。假设 v_i 在周期 T 内开始执行的起始位置为 k，那么由于该周期内最后一次执行的 VCPU 会在下个周期第 1 个被调度，因此 v_i 在下个周期开始执行的起始位置向后推移 1 位变为 $k+1$，然后连续执行 W_i 次，以此类推，

直到某个周期的最后一次执行为 v_i。之后的执行将会重复情况 1，即 TDMA 模式执行，如图 7 所示，此时 VCPU v_i 的 sbf 为：

当 $t \leqslant (N-1)*(T+1)$ 时，

$$sbf_i^*(t) = \max\{t \bmod (T+1) - (T+1) + W_i, 0\} + \lfloor t/(T+1) \rfloor \cdot W_i \quad (3)$$

当 $(N-1)*(T+1) < t \leqslant N*T$ 时，

$$sbf_i^*(t) = (N-1) \cdot W_i + \max\{t \bmod (T+1) - T - N + W_i + 1, 0\} \quad (4)$$

当 $t > N*T$ 时，

$$sbf_i^*(t) = N \cdot W_i + sbf_i^{TDMA}(t - N \cdot t) \quad (5)$$

其中，$N = T - k - W_i + 1$，表示 v_i 最多经过 N 个周期后其获得的处理器资源变为 TDMA 模式如图 7 所示；$k = \lceil (T - W_i) * (W_i - 1) / W_i \rceil$ 表示为了保证第一个周期 T 的最后一次执行不是 v_i，那么在初始 UNDER 队列中 v_i 可以出现的最早位置是 k（即若 v_i 出现在第 k 次之前，则第一个周期的最后一次执行一定不是 v_i，若 v_i 出现在第 k 次之后，则 v_i 的 sbf 一定优于 sbf^*，因为 v_i 能够更快地变为 TDMA 模式），sbf^* 曲线如图 8 所示。定理 3 证毕。

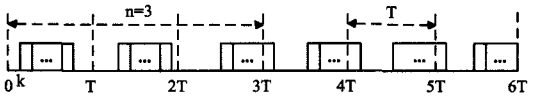

图 7　$N=3$，表示经过 3 个周期变为 TDMA 模式

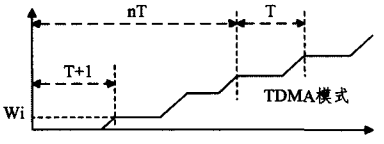

图 8　sbf^* 曲线

结束语　本文主要研究了 Xen 虚拟机 Credit 调度算法的实时性能，使得能够直接对运行在 Xen 上的实时系统进行可调度性分析，并且可以通过形式化的资源界限函数对算法的实时性进行直观的评估。通过对 Xen 源码（Xen4. 1. 4）分析 Credit 的算法原理，提出并且证明了一种配置 VCPU 参数的方法，即找到使信用度值不超过上限 300 的条件，使得 Credit 的实时性得到提升。在此基础上，通过证明得到 Credit 两个基本性质：周期性以及任意时刻所有 VCPU 信用度值总和为 0，并最终得出其在最坏情况下为 VCPU 分配的资源函数曲线。在之后的工作中，将会根据本文的实时性分析结果，对 Credit 算法进行改进，并将其实现在 Xen4. 1. 4 版本中。

参 考 文 献

[1] Pratt I. Xen2. 0 and the Art of Virtualization[M]. OLS, O3, Ottawa, 2004

[2] Chisnall D. The Definitive Guide to the Xen Hypervisor[M]. Pearson Education, 2007

[3] Credit Scheduler [OL]. http://wiki. xen. org/wiki/Credit_Scheduler

[4] Govindan S, Nath A R, Das A, et al. Xen and Communication-aware CPU Scheduling for Consolidated Xen-based Hosting Platforms[C]// VEE'07. New York: ACM, 2007: 126-136

[5] Lee M, Krishnakumar A S, et al. Supporting Soft Real-Time Tasks in the Xen Hypervisor[C]// VEE'10. Pittsburgh, PA, USA, 2010: 324-336

[6] Xi S, Wilson J, Lu C, et al. Rt-xen: Towards Real-time Hypervisor Scheduling in Xen[C]// EMSOFT. New York, NY, USA: ACM, 2011: 39-48

[7] Lee J, Xi S, Chen S, et al. Realizing Compositional Scheduling through Virtualization[C]// RTAS. Washington, USA: IEEE Computer Society, 2012: 13-22

[8] Jin H, Zhong A, Wu S, et al. Virtual Machine VCPU Scheduling in the Multi-core Environment: Issues and Challenges[J]. Journal of Computer Research & Development, 2011, 48(7): 1216-1224

[9] Gu Zhen-yu, Zhang Shen-sheng, Li Xiao-yong. Optimization of Credit Scheduling Algorithm in Xen[J]. Micro Computer Application, 2009, 25(2): 1-3

[10] Baruah S, Mok A, Rosier L. Preemptively Scheduling Hard-real-time Sporadic Tasks on One Processor[C]// RTSS. 1990: 182-190

[11] Mok A, Feng X, Chen D. Resource Partition for Real-time Systems[C]// RTAS. 2001: 75-84

[12] Marimuthu S P, Chakraborty S. A framework for compositional and hierarchical real-time scheduling[C]// RTCSA. IEEE, 2006: 91-96

(上接第 94 页)

自身定位。地图上方是两个下拉框，用来方便用户选择前往的地点和用户的出发点，文本框用来表示为用户提供的一条便捷的路径。图 4 展示室内环境的应用，运行界面与室外导航相似，不同之处在于多了老师姓名搜索框和在路口的实时导航。

结束语　本文将 RFID 技术应用到室内定位，并结合 GIS 技术实现了一个校园导航系统，在 RFID 技术的应用方面取得了一定的进展。当然，系统在数据更新和维护方面应该有更为详细的处理方法，使得系统在空间数据有变化的情况下能自适应地调整，以保证系统的准确性和长期使用价值。

参 考 文 献

[1] Liu Yun-hao, Yang Zheng, Wang Xiao-ping, et al. Location, Localization, and Localizability[J]. Journal of Computer Science and Technology, 2010, 25(2): 274-297

[2] Xie Lei, Sheng Bo, Tan C C, et al. Efficient Tag Identification in Mobile RFID Systems[C]// INFOCOM. 2010

[3] Want R. An Introduction to RFID Technology[J]. IEEE Pervasive Comput, 2006, 5(1): 25-33

[4] Chiesa M, Genz R, Heubler F, et al. RFID[M]. March 2002

[5] Maguire Y, Pappu R. An optimal q-algorithm for the iso 18000-6c rfid protocol[J]. IEEE Transactions on Automation Science and Engineering, 2009, 6(1): 16-24

[6] Zheng Yuan-qing, Li Mo. P-MTI: Physical-layer Missing Tag Identification via Compressive Sensing School of Computer Engineering[C]// 2013 Proceeding IEEE on INFORCOM. 2013: 917-925

[7] Zhen B, Kobayashi M, Shimuzu M. Framed aloha for multiple rfid objects identification[J]. IEICE Transactions on Communications, 2005, E80-B(3): 991-999

[8] Johnson D B. A Note on Dijkstra's Shortest Path Algorithm [J]. Journal of the ACM, 1973, 20(3): 385-388