虚拟集群上面向功耗的形式化的VM调度策略

张鲁飞 陈左宁
(数学工程与先进计算国家重点实验室 无锡 214125)

摘 要 针对虚拟化集群上日益严重的功耗问题,在定义集群、物理机、操作点、能耗、任务集、虚拟机等概念的基础上,提出了形式化的虚拟机(Virtual Machine,VM)调度策略,并有针对性地利用动态电压和频率调整(Dynamic Voltage and Frequency Scaling,DVFS)技术对普通算法进行了扩展改进,提出一个功耗敏感的VM调度算法。首先,利用FFD(First-Fit Decreasing)算法解决虚拟机的初始布局问题,然后尽可能地将虚拟机部署在低电压的物理机上,此外加入性能感知策略,尽量将物理机的电压调节至刚好满足虚拟机性能需求,以避免"奢侈"能耗的浪费,同时增加了虚拟机调度失败时灵活调整电压的规则,以减少错误先验知识对虚拟机后续部署的误导。在理想模型中的仿真实验表明:与现有部署算法相比,本算法具有更好的节能效应,以及相对不大的性能损失。

关键词 虚拟化,DVFS,功耗,调度
中图法分类号 TP315 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.007

Power-efficient Formal Scheduling Policy of VMs in Virtualized Clusters

ZHANG Lu-fei CHEN Zuo-ning
(State Key Laboratory of Mathematical Engineering and Advanced Computing,Wuxi 214125,China)

Abstract Aiming at the power problem of virtualized cluster,based on the defining of cluster,physical machine,operating point,energy,jobs,virtual machine,the formal scheduling policy of Virtual Machines (VMs) was proposed to research the deployment scheme. And the common algorithm was extended and modified with Dynamic voltage and frequency scaling (DVFS) technology for energy-awareness. Using the FFD (First-Fit Decreasing) algorithm,the virtual machines were arranged in frequency sequence,then deployed on a physical machine with lowest voltage. Using performance apperceive policy,the voltage physical machines were set just enough for the total requirement of virtual machines to reduce "luxury" energy respectively. And using the voltage update rules after no virtual machines deployed,the misdirection of the inaccurate heuristic information was avoided. The modified algorithm was researched with experiments in an ideal model. Finally the performance and energy of the modified algorithm were compared with the old. The experimental results show that the modified algorithm meets the need of the jobs,and saves a great deal of energy.

Keywords Virtualization,DVFS,Power,Scheduling

1 引言

在目前的数据中心设计中,功耗已经成为关键要素,它直接影响虚拟机的部署和运行。虚拟化技术是数据中心常用的技术,它有很好的故障和性能隔离作用,可以提高系统的可管理性;另外,利用虚拟化技术在资源整合和动态迁移上的优势,可以减少基础设施的开销。在一个物理机(Physical Machine,PM)上部署多个不同的虚拟机(Virtual Machine,VM)可以提高整体的利用率和效率,然而各个虚拟机在负载特征上的不同会造成功耗和性能的不同,可能会造成功耗的热点,降低整体的性能功耗比。目前的高性能处理器普遍支持动态电压和频率调整技术,它可以通过降低处理器时钟电压和频率有效地减少处理器功耗,已经成为集群降耗的重要手段之一。

本文拟将DVFS技术和虚拟机调度相结合,提出一个功耗敏感的虚拟机调度算法。

2 相关工作

在虚拟化以及绿色计算的相关技术领域,国内外学者已经开展了大量的工作,涉及到任务调度技术、低功耗控制技术等方面[1]。

体系结构方面已有一些功耗管理规范[2]。例如,Intel、Microsoft等软/硬件厂商提出了高级功耗管理(Advanced Power Management,APM)、高级配置与功耗接口(Advanced Configuration and Power Interface,ACPI)等功耗管理规范,利用动态可变电压(Dynamic Voltage Scaling,DVS)和动态可变频率(Dynamic Frequency Scaling,DFS),在满足瞬时性能的前提下,使有效能量供给率最大化。

到稿日期:2013-06-15 返修日期:2013-07-29 本文受国家"863"高技术研究发展计划基金项目(2013AA01A213)资助。
张鲁飞(1986—),男,博士生,助理工程师,主要研究方向为操作系统、云计算,E-mail:zhanglf04@126.com;陈左宁(1957—),女,中国工程院院士,主要研究方向为操作系统、系统结构。

在虚拟环境中，硬件和操作系统之间插入了虚拟层，物理资源通常被多个虚拟机共享。传统的功耗管理系统大多处于操作系统层，并不适用于虚拟环境。例如，在一个多虚拟机的物理机上，只有多数虚拟机都减少对内存、硬盘的访问时，才能对内存、硬盘进行低功耗设置，而一个虚拟机中的操作系统不能获取所在物理机上其它虚拟机的功耗信息，难以利用硬件提供的功耗管理机制。

国内外学者提出了一些新的针对虚拟环境的任务调度模型和算法。例如 Magnet、ClientVisor、VirtualPower 等系统针对虚拟机的调度，提出了功耗敏感的请求分配(Power-Aware Request Distribution)和功耗敏感的域分配(Power-Aware Domain Distribution)的两层次控制策略；在虚拟机迁移方面，利用虚拟化带来的进程迁移的机会，可以将大量低负载的进程集中到相对少量的节点上，彻底关闭部分闲置的节点，从而进一步地节约功耗[3]。

3 虚拟机调度问题的理想建模

虚拟集群中虚拟机调度的应用场景是：一个集群上有若干物理机，每个物理机都有不同的电压和频率，也会得出相应的功耗；每一个任务，会生成若干有着性能需求的虚拟机；根据需求特点和集群上功耗分布情况来调度虚拟机。

在形式化 VM 调度策略之前，首先需要对集群、物理机、操作点、能耗、任务集、虚拟机、虚拟机调度等概念进行建模，如表 1 所列。

表 1 虚拟机调度问题的理想建模

对象	描述	形式化
集群	一个集群是若干空间分布的物理机的集合	$C=\{PM_1, PM_2, \cdots, PM_N\}$
物理机	物理机执行时会有不同的电压和频率	$PM_n=\langle PM_n.V, PM_n.F\rangle$；其中 V 和 F 的关系需要满足操作点要求
操作点	一个操作点是处理器在一个频率下的合适电压；频率和电压均是离散分布的	$OP=\{OP_1, OP_2, \cdots, OP_K\}$；$OP_k$ 从低电压至高电压排列；$OP_k=\langle OP_k.V, OP_k.F\rangle$；其中一个物理机的电压 $PM_n.V$ 一定等于某一个操作点 OP_k 的推荐电压 $OP_k.V$，但是此物理机的频率 $PM_n.F$ 需要小于操作点的推荐频率 $OP_k.F$
能耗	能耗是功耗的时间累积，功耗分为两部分：静态功耗和动态功耗，静态功耗一般占劣势；动态功耗与频率成正比，与电压的平方成正比	$E=aV^2Ft$
任务集	一个任务集是若干时间分布的虚拟机的集合	$JOB=\{VM_1, VM_2, \cdots, VM_M\}$
虚拟机	一个虚拟机有期望的运行频率、期望的开始执行时间和执行结束时间	$VM_m=\langle VM_m.F, PM_n.t_s, PM_n.t_t\rangle$
虚拟机调度	调度过程就是把一个虚拟机分到某一个物理 CPU 上	JOB->C

由于是理想建模，因此对一些因素做了近似处理，例如，近似 1：集群中的所有物理机均是同构的；近似 2：同一个 CPU 上的不同 PM 也可以有不同的电压；近似 3：同一个 CPU 上的不同 PM 的电压和频率调整不会相互影响；近似 4：计算功耗时忽略静态功耗；近似 5：虚拟机的生成是由应用驱动的，也就是固定时间分布；近似 6：一个虚拟机有期望的运行频率和对应的执行时间。

4 形式化的虚拟机调度策略

根据理想模型中的约定，一个虚拟机有期望的运行频率和对应的执行时间，而且虚拟机的生成是由应用驱动的，也就是遵从固定时间分布的。因此虚拟机调度问题可以看作是一个输入固定的虚拟机的布局问题。本节将这个布局问题形式化，约束条件就是任务集能否在指定时间内完成[4]。

4.1 VM 生命周期

可以创建 4 个 VM 队列来描述整个调度流程，分别是初始化队列(initialized_queue)、调度队列(schedule_queue)、执行队列(running_queue)、结束队列(over_queue)。在一个 VM 的生命周期中，当初始化时，VM 被放入初始化队列；到达 VM 的期望启动时间时，VM 被放入调度队列；当 VM 根据调度算法调度成功时，被放入执行队列；当 VM 执行完毕时，被放入结束队列，如图 1 所示。

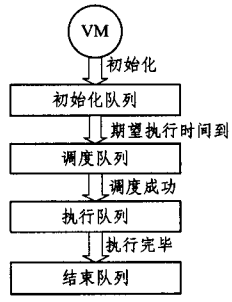

图 1 VM 生命周期

4.2 虚拟机调度阶段

可将虚拟机调度过程分为初始化阶段、调度阶段和结束阶段。在初始化阶段，所有虚拟机加入初始化队列；在结束阶段，所有 VM 已执行完毕，所有 PM 可以关掉；在调度阶段，每一轮都需要做如下操作，首先要处理执行完毕的 VM，将其关闭，放入结束队列中；然后将期望执行时间已到的 VM 加入调度队列；最后用调度算法判断调度队列中的 VM 是否可以调度，若是，将其加入执行队列，如图 2 所示。

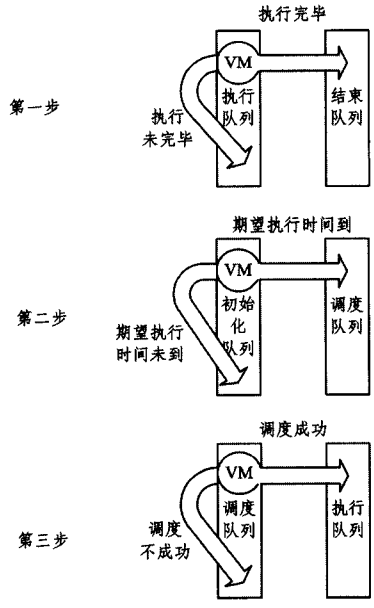

图 2 虚拟机调度阶段

4.3 虚拟机调度过程的形式化描述

以上的虚拟机调度过程可以形式化，如表 2 所列。

表 2　虚拟机调度过程的形式化描述

1. 初始化：
2. 调度队列 $\xi=\Phi$
3. 时间 t=0
4. 开始调度：
5. 时间前行 t=t+1
6. 判断是否所有 VM 都没有执行完毕
7. 若是，跳至 10
8. 若否，$\{VM_x \cdots VM_y\}$ 已执行完毕
9. 对于 ALL $VM_m \in \{VM_x \cdots VM_y\}$，它在 PM_n 上执行完毕，将其关闭，放入结束队列
10. 若有新的 VM 参加调度，更新调度队列 $\xi=\xi \cup \{VM_x \cdots VM_y\}$
11. 判断调度队列是否为空
12. 若是，跳至 16
13. 若否，从 ξ 中取出 VM_j，取过一遍之后跳至 16
14. 按照调度算法判断 VM_j 是否可以调度成功
15. 若是，将 VM_j 部署在 PM_n 上，将其加入执行队列
16. 判断是否全部调度完毕
17. 若是，跳至 19
18. 若否，跳至 4
19. 结束调度：
20. 将所有 PM 关掉

下面的策略将利用这个形式化的描述，在其中嵌入利用 DVFS 机制的功耗敏感的虚拟机调度算法。

5　利用 DVFS 机制的功耗敏感的虚拟机调度算法

在上一节提出的虚拟机布局问题中，若将任务集完成时间作为优化目标，就是一个装箱问题(Bin Packing)，严格求出其最优解是一个 NP 难问题[5]。

而若将功耗也考虑在内，那么虚拟机布局优化将更为复杂。但是利用动态电压和频率调整技术可以将约束条件简化为各物理机电压最低。在理想建模中，动态功耗与频率成正比，与电压的平方成正比，物理机执行时会有不同的电压和频率，而且近似地认为同一个 CPU 上的不同 PM 的电压和频率调整不会相互影响，功耗只计算动态功耗部分。因此问题简化为，在满足虚拟机需要的速度的前提下，尽可能地将虚拟机部署在低电压的物理机上。

下面提出的功耗敏感的虚拟机调度算法将试图解决这个类装箱问题。

5.1　算法描述

在理想建模中，假设一个虚拟机有期望的运行频率和对应的执行时间。若以任务集的完成时间为优化目标，则在每一轮调度中，需要尽量地将调度队列中所有 VM 放入 PM 中执行。由于每个 VM 有期望的运行频率，但是 PM 的频率上限是确定的，因此在每一轮调度中，期望用最小数量的 PM 容纳调度队列中的所有 VM。这是一个典型的装箱问题。

按照形式化的虚拟机调度策略中的描述，每一轮更新后的调度队列为 $\xi=\{VM_1, VM_2, \cdots, VM_J\}$，它们的频率就是装箱问题中物品的负荷；而集群 $C=\{PM_1, PM_2, \cdots, PM_N\}$ 中每一个 PM 可容纳的频率就是装箱问题中箱子的负荷。

首先，采用装箱问题中常用的 First-Fit Decreasing(FFD)算法解决虚拟机的初始布局问题，这种算法并不能取得最优解，但是它迅速而有效[6]。FFD 算法是 FF (First Fit) 算法的改进。FF 算法的特点是：1)按物品给定的顺序装箱；2)每个物品总是放在能容纳它的具有最小标号的箱子中。FFD 算法的改进之处在于先将物品按长度从大到小排序，然后用 FF 算法对物品装箱。该算法的计算复杂性为 $O(n\log^n)$。可以证明 FFD 给出的解和最优解 OPT 之间的关系为：$FFD \leqslant \frac{11}{9}OPT+\frac{6}{9}$。因此形式化调度策略中的第 10 步改为表 3 所列。

表 3　算法改进 1

旧	10. 若有新的 VM 参加调度，更新调度队列 $\xi=\xi \cup \{VM_x \cdots VM_y\}$
新	10. 若有新的 VM 参加调度，更新调度队列 $\xi=\xi \cup \{VM_x \cdots VM_y\}$，将调度队列按 VM. F 降序排列，$\xi=\{VM_1, VM_2, \cdots, VM_J\}$，$J \leqslant M$

其次，在满足虚拟机需要的速度的前提下，尽可能地将虚拟机部署在低电压的物理机上，以降低功耗，因此每次按照 FF 算法对物品装箱时，PM 的编号按照电压从小到大重新排序。形式化调度策略中的第 14 步改为表 4 所列。

表 4　算法改进 2

旧	14. 按照调度算法判断 VM_j 是否可以调度成功
新	14. 按照 PM. V 升高的顺序依次判断是否存在 PM_n. F 可以支持 VM_j. F

然后，DVFS 技术允许物理机的电压独立变化，一个物理机的电压 $PM_n.V$ 一定等于某一个操作点 OP_k 的推荐电压 $OP_k.V$，但是此物理机的频率 $PM_n.F$ 需要小于操作点的推荐频率 $OP_k.F$。因此，在调度阶段的第一步之后，可以将一些执行完毕的 VM 关闭，之后重新调整集群 $C=\{PM_1, PM_2, \cdots, PM_N\}$ 中每一个 PM 的电压。形式化调度策略中的第 9 步之后需要增加表 5 所列步骤。

表 5　算法改进 3

旧	无
新	9、10. 将 $PCPU_n$. V 尽可能地降低至操作点 OP_k 的电压 OP_k. V，使得 OP_k. F 可以支持所有在其上运行的 VCPU. F 之和，即 $PCPU_n.V=OP_k.V$

最后，只降低物理机的频率会导致可容纳的频率不足以支持 VM 的频率需要，因此需要在 VM 调度失败时提供 PM. V灵活调整的机制。形式化调度策略中的第 15 步之后需要增加表 6 所列步骤。

表 6　算法改进 4

旧	无
新	15、16(1). 若否，找到支持频率范围最大的 $PCPU_i$，判断 $PCPU_i$. F 是否不足以支持 $VCPU_j$. F 15、16(2). 若是，跳至 13 15、16(3). 若否，将 $PCPU_i$. V 升高至尽可能低电压的操作点 OP_k，使得 OP_k. F 可以支持 $VCPU_j$. F，即 $PCPU_j.V=OP_k.V$，并将 $VCPU_j$ 部署在 $PCPU_n$ 上，即 $PCPU_n.F=PCPU_n.F+VCPU_j.F$，更新调度队列 $\xi=\xi-VCPU_j$

上述算法面向功耗，利用 DVFS 机制对装箱问题中常用的 FFD 算法进行改进，称为理想(ideal)算法。

5.2　模拟结果与分析

在模拟环节中，需要建立任务和集群两个模块，随机地设定虚拟机的期望执行频率和执行时间，然后记录每个操作点集合中虚拟机的数量和总的功耗，并与普通的算法结果比较。

5.2.1　模拟流程

本文选择了 Intel Pentium M 1.4 GHz 的处理器来进行模拟，它共有 5 个操作点，最低频率为 0.6GHz，最高频率为 1.4GHz。

在集群模块中，我们固定了 PM 的数量，为 10。对于每

个 PM，在理想算法中，首先被置为最小操作点。

在任务模块中，VM 的数量 JOB_AMOUNT 从 10 到 1000 不等。对于每个 VM，我们随机地给予它一个 20 以内的期望启动时间、一个 5 以内的期望执行时间，以及一个 0.1GHz～1.0GHz 的期望执行频率。

我们设置了 4 个 VM 队列。当初始化时，VM 被放入初始化队列；当到达 VM 的期望启动时间时，VM 被放入调度队列；VM 根据调度算法调度成功时，被放入执行队列；VM 执行完毕时，被放入结束队列。4 个队列中 VM 数量之和总是等于 JOB_AMOUNT，当 over_queue 中 VM 的数量等于 JOB_AMOUNT 时，任务结束。

5.2.2 VM 的操作点分布

以上就是模拟程序的流程。每个实例运行 100 遍，来消除任务模块中的随机性。在 JOB_AMOUNT 从 10 增加到 1000 的过程中，每个 PM 上同时请求运行的 VM 增多，调度的难度增大。很多的 PM 需要提升至更高的操作点，以提供更大的频率范围给更多的虚拟机调度。但是即使调度队列上有几十个 VM，调度算法通过 VM 执行完毕时及时地降低 PM 电压，还是获得了 50%以上的低电压操作点，如图 3 所示。

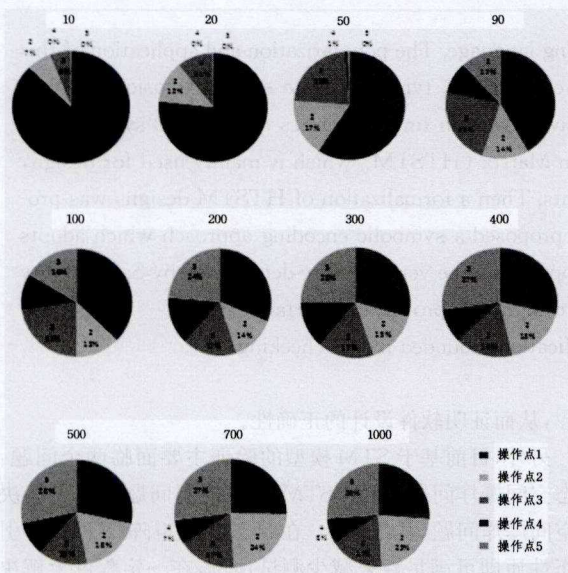

图 3 虚拟机在操作点上的分布变化

5.2.3 理想算法和普通算法的比较

在理想算法中，当 VM 执行完毕时，会及时地降低 PM 电压；当处理调度队列时，会在满足虚拟机需要的速度的前提下，尽可能地将虚拟机部署在低电压的物理机上。为了证明这个算法在执行速度和功耗上均有良好的表现，我们设计了两种普通算法。

一种是 max 算法，初始化时，它将所有 PM 均设置为最大电压和最大频率，且在调度过程中不改变。这相当于在 BIOS 中禁用了处理器的 DVFS 功能。另外，在处理调度队列时，我们仅仅是随机地选出 PM。

另外一种是 normal 算法，它与 max 算法不同之处仅仅在于初始化时，将所有 PM 均设置为中等电压和中等频率。

模拟结果如图 4 和图 5 所示。

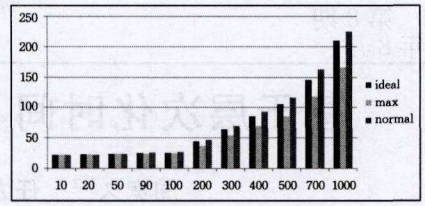

图 4 执行时间的比较

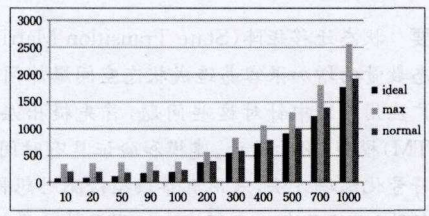

图 5 功耗的比较

从模拟结果可以看出，与 normal 算法相比，理想算法在速度上占有一定优势，这是因为理想算法在负载较重的时候可以将 PM 的电压调高，以调度更多的 VM，而且理想算法的功耗也相对较低。

与 max 算法相比，理想算法在速度上稍显劣势，特别在 VM 比较多的情况下，速度上要慢 20%左右，这是由于理想算法为了节能需要，初始化时将所有 PM 置成最小操作点，使得很多 VM 不能正确调度上去。但是理想算法的节能效果大约有 30%，权衡一下，节能效果还是比速度减缓效应明显。

在 VM 比较少的情况下，理想算法与两种普通算法在速度上几乎没有差别，而且节能效果更明显，分别有大约 70%和 40%的节能效应。

5.3 模拟结果与分析

通过对模拟结果的分析，我们可以得出以下结论：

(1)理想算法可以灵活地设置物理机电压，使任务尽可能地在低电压模式下运行。

(2)在负载较轻的任务模式下，将物理机设置在低电压状态，既对虚拟机速度不造成影响，又可以获得极大的节能效应。

(3)在负载较重的任务模式下，将物理机设置在低电压状态，会对虚拟机速度造成一定影响，但是通过理想算法及时地根据负载变化情况调整物理机的电压，可以将影响降低；另外，理想算法通过 DVFS 机制可以取得很好的节能效应[7]。

结束语 针对虚拟化集群上日益严重的功耗问题，本文提出一个理想化模型来描述集群、物理机、操作点、能耗、任务集、虚拟机等概念，并提出形式化的 VM 调度策略问题。本文利用 DVFS 技术，给出一个简单的理想的功耗敏感的虚拟机调度算法。通过模拟得出，理想算法可以灵活地设置物理机电压，使任务尽可能地在低电压模式下运行，理想算法既适合负载较轻的任务模式也适合负载较重的任务模式。

参 考 文 献

[1] Magklis G, Semeraro G, Albonesi D, et al. Green Maturity Model for Virtualization[OL]. Issue 18-Green Computing, The Architecture Journal, http://www.architecturejournal.net, 2009

(下转第 46 页)

表 2 Z3 求解器验证修正后系统性质结果

Property	Step	Verdict	Time(s)
invalid	50	unsat	2.86
static1	50	unsat	3.35
static2	50	unsat	3.62
static3	50	unsat	4.18
dynamic1	16	sat	0.87
dynamic2	50	unsat	4.79

结束语 本文针对 STM 的不足提出 TSTM 的概念，用以表示包含时间因素的软件系统的设计、建模和验证，将时间因素形式化为模型的时间变量来驱动模型的运转。针对当前软件规模和复杂性的不断提高使得建模过程极易出现状态空间爆炸的问题，本文提出 HTSTM 的概念，将 TSTM 进行层次化建模，从而在一定程度上解决该问题。基于 HTSTM 模型本文给出形式化表示方法，并在此基础上提出一种符号化编码方法，采用有界模型检测的方法，使用 SMT 求解器对模型性质进行验证。从实验结果可以看出，本文提出的方法可以较好地整合 SMT 求解器[14]等现有资源，在可以接受的时间范围内验证模型性质，找出模型中的设计错误或者逻辑错误。

但本文方法仍然有很多改进的空间。我们下一步的工作重点是使用启发式思想或者其他方法对需要验证的性质进行优化，或者对形式化过程进行优化，从而降低模型验证需要的时间和空间方面的开销。

参 考 文 献

[1] Matsumoto M, Anada K, Ueshima D, et al. Model Checking of State Transition Matrix[R]. ITSSV 2005, AIST Technical Report. 2005:2-11

[2] Watanabe M. Extended Hierarchy State Transition Matrix Design Method -Version 2.0[R]. CATS Technical Report. 1998

[3] Clarke E, Kroening D, Ouaknine J, et al. Completeness and complexity of bounded model checking[C]// Proceedings of the Verification, Model Checking, and Abstract Interpretation. Springer Berlin Heidelberg, 2004:85-96

[4] Clarke E M, Grumberg O, Peled D. Model checking[M]. The MIT press, 1999

[5] Armando A, Mantovani J, Platania L. Bounded model checking of software using SMT solvers instead of SAT solvers[M]// Model Checking Software. Springer Berlin Heidelberg, 2006: 146-162

[6] Kong Wei-qiang, Fukuda A, Watanabe M. An SMT approach to bounded model checking of design in state transition matrix[C]// Proceedings of the Computational Science and Its Applications (ICCSA). 2010:231-238

[7] Kong Wei-qiang, Katahira N, Watanabe, et al. Formal verification of software designs in hierarchical state transition matrix with SMT-based bounded model checking[C]// Proceedings of the Software Engineering Conference (APSEC). 2011:81-88

[8] Latvala T, Biere A, Heljanko K, et al. Simple bounded LTL model checking[C]// Proceedings of the Formal Methods in Computer-Aided Design-2004. 2004:186-200

[9] Jussila T, Dubrovin J, Junttila T, et al. Model checking dynamic and hierarchical UML state machines[C]// Proc. MoDeV2a: Model Development, Validation and Verification. 2006:94-110

[10] Dubrovin J, Junttila T. Symbolic model checking of hierarchical UML state machines[C]// Proceedings of the Application of Concurrency to System Design. 2008:108-117

[11] Biere A, Cimatti A, Clarke EM, et al. Symbolic Model Checking without BDDs[C]// TACAS 1999. 1999:193-207

[12] Latvala T, Biere A, Heljanko K, et al. Simple is better: Efficient bounded model checking for past LTL[C]// Proceedings of the Verification, Model Checking, and Abstract Interpretation. 2005:380-395

[13] Lee E A, Seshia S A. Introduction to embedded systems: A cyber-physical systems approach[M]. Lee & Seshia, 2011

[14] Barrett C, Sebastiani R, Seshia S A, et al. Satisfiability modulo theories[J]. Handbook of Satisfiability, 2009, 185:825-885

[15] Veanes M, Bjørner N, Alexander R. An SMT approach to bounded reachability analysis of model programs[C]// Proceedings of the Formal Techniques for Networked and Distributed Systems-FORTE 2008. Springer Berlin Heidelberg, 2008:53-68

[16] 曾锃，赵建华. 基于程序分析的代码查询技术[J]. 计算机科学，2012，39(2):143-147

[17] 何炎祥，吴伟，陈勇，等. 基于 SMT 求解器的路径敏感程序验证[J]. 软件学报，2012，23(10)

[18] Dubrovin J, Junttila T, Heljanko K. Symbolic step encodings for object based communicating state machines[M]. Springer Berlin Heidelberg, 2008:96-112

[19] Cousot P. Abstract interpretation based formal methods and future challenges[C]// Proceedings of the Informatics. 2001:138-156

[20] De Moura L, Bjørner N. Z3: An efficient SMT solver[M]// Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008:337-340

(上接第 41 页)

[2] Scott M. Dynamic frequency and voltage scaling for a multiple-clockdomain microprocessor[J]. IEEE Micro, 2003, 23(6):62-68

[3] Nathuji R, Schwan K. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems[C]// SOSP'07. Stevenson, Washington, USA, October 2007

[4] Verma A, Ahuja P, Neogi A. Power-aware Dynamic Placement of HPC Applications[C]// Proceedings of the 2008 ACM International Conference on Supercomputing(ICS'08). 2008:175-184

[5] Fallenbeck N, Picht H, Smith M, et al. Xen and the art of cluster scheduling[C]// First International Workshop on Virtualization Technology in Distributed Computing. 2006:4-4

[6] Kim K H, Buyya R, Kim J. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters[C]// CCGRID. 2007:541-548

[7] Ge R, Feng X, Cameron K. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters[C]// Proceedings of the 2005 ACM/IEEE conference on Supercomputing. IEEE Computer Society Washington, DC, USA, 2005