多核多线程结构线程调度策略研究*)

王 晶[1] 樊晓桠[1] 张盛兵[1] 王 海[2]

(西北工业大学计算机学院 西安 710072)[1] (同济大学机械学院热能与环境研究所 上海 200092)[2]

摘 要 片上多核多线程(CMT)结构兼具了片上多处理(CMP)和同时多线程(SMT)结构的优势,支持片上所有处于执行状态的线程每周期并行执行,导致核内与核间硬件资源共享和争用问题。该文在阐述CMT结构的资源共享特征并简要介绍SMT线程调度发展状况的基础上,主要围绕以减少资源争用为目标的线程调度策略和资源划分机制等热点,分析其研究现状,论述已有策略在处理这些问题上的优缺点,并探讨了可能的研究发展方向。

关键词 同时多线程,片上多处理,片上多核多线程,线程调度,资源划分

A Survey of CMT Processor Thread Schedule Policies

WANG Jing[1] FAN Xiao-Ya[1] ZHANG Sheng-Bing[1] WANG Hai[2]

(School of Computer, Northwestern Polytechnical University, Xi'an 710072)[1]

(Heat Power and Environment Institute, College of Mechanical Engineering, Tongji University, Shanghai 200092)[2]

Abstract CMT is provided with the advantages of both SMT and CMP architectures. All executing state threads on chip can execute simultaneously each cycle, which results in resources share and conflict issues. First, resources share characteristic of CMT architecture is introduced, then, the prevailing SMT thread schedule policies. And then, we primarily analyze current threads co-schedule policies and key resources partition mechanisms of CMT architecture. After analyzing the advantages and disadvantages of these techniques, we conclude that, heuristic metrics for CMT thread co-schedule should reflect application characteristics to keep stable and efficient, and resources partition co-operated thread schedule policy may achieve satisfying performance.

Keywords SMT, CMP, CMT, Thread schedule, Resource partition

1 引言

微体系结构的发展已经迈入线程级并行(TLP, Tread Level parallel)的时代[1]。同时多线程(SMT, Simultaneous Multi-Thread)结构和片上多处理(CMP, Chip Multi-Processor)结构[2]都可以在每周期并行执行多个线程,提高了处理器的吞吐率。如何解决线程间共享资源的争用问题成为线程调度的关键。

2005年SUN公司[3]提出了将这两种体系结构结合的片上多核多线程(CMT, CMP of Multi-threading)结构。它兼具两者的优点,使单个处理器的性能得到了显著的提高。在CMT结构中,处理器有多个执行核,核内多线程间资源共享和核间资源共享并存。但面对应用程序的多样性,线程调度问题更趋复杂化,需要进一步研究如何提高整个系统性能的策略和机制[3]。

本文第2、3节分别介绍了几种典型的多线程、多线程多核处理器的结构和多线程结构的调度策略;在第4、5节详细讨论CMT结构的线程调度策略,以及Cache冲突模型和划分策略;最后,对未来研究的热点进行展望。

2 几种典型处理器结构

多线程结构中线程间共享资源可分为三类:前端、执行引擎和存储层次。其中前端包含一级Icache、取指、重命名和分派队列等;执行引擎包含各功能单元;存储层次包括一级Dcache、二级Cache等。SMT[4]结构如图1(a)所示,硬件保存多个线程的执行现场,每个周期可以不同的线程或线程组取指,这些线程共享系统资源。图1(b)是划分型多线程结构(P-MT, Partitioned-MT),FQ, IIQ, FIQ, LSQ都进行了线程间划分。大量文献都在这种结构的基础上展开讨论。图1(c)(d)所示的CMT结构(单核可以是MT或P-MT),核间共享L2cache。IBM的Power5[6]、Intel的Montecito[7]、Sun的Niagara[8]及UltraSPARC[9]都属于CMT结构。

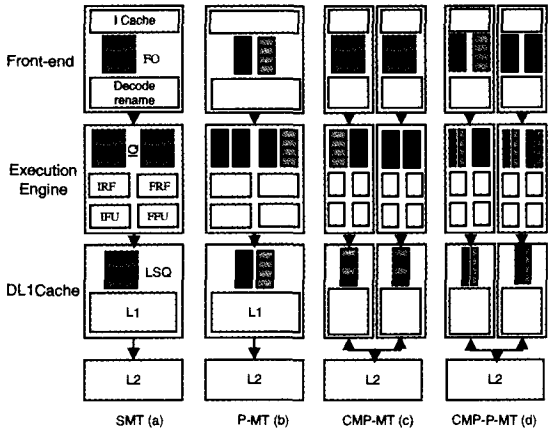

图1 SMT、P-MT、CMP-MT. CMP-P-MT体系结构图

*)本文得到国家自然科学基金(60573143)和新世纪优秀人才支持计划资助。**王 晶** 博士生,研究方向:计算机系统结构。**樊晓桠** 教授,研究方向:计算机系统结构。**张盛兵** 教授,研究方向:计算机系统结构。**王 海** 讲师,研究方向:热能工程,生产过程自动化。

3 MT结构线程调度

作为一种新的体系结构，CMT的线程调度策略研究是建立在SMT结构相关研究基础上的，下面首先介绍SMT结构的线程调度策略研究。

3.1 SMT线程调度策略

SMT中多线程共享核内资源，首先指令队列(IQ)是SMT以及CMT结构的关键共享资源之一。Tullsen[4]提出的SMT结构的基本取指策略及各种侧重于减少IQ阻塞的改进型策略可以部分解决IQ阻塞问题。通过对线程的合理调度可进一步减小资源的争用。

1)共生调度

Snavely[14]等首次将共生概念(两个或者多个线程有效的同时执行)引入SMT线程调度策略中，并讨论了双线程的共生调度策略(SOS, Sample-Optimize-Symbiosis)，其性能比随机调度方案提高了17%。

Tullsen[13]定义了共生的基本特征：多样性(执行指令流中指令类型的多样性)、平衡(调度间隔之间资源利用率差异不大)和低争用(各种资源使用过程中争用情况不严重)。这三个特征是相互关联的。它们难以具体测度，不能作为共生调度的依据。文中进一步定义了10个共生表象，将这10个表象信息作为具体的调度依据。在该调度策略中，一个OS调度时间片被分为重复的抽样－稳定(Iterative Sample-Stable)过程。抽样阶段中各种待选的调度方案选用某种共生表象并收集其相关参数，用于预测好的共生组合，在稳定阶段运行同时执行组合中的线程。抽样阶段中线程可正常执行，可认为抽样阶段额外开销为零。因为每个表象信息只能反映某种资源的争用状况，同时线程执行特性却是不断变化的，因此即使采用所有共生表象投票(score)的方式，也难以取得稳定的效果。

文[15]基于上述思想，提出了考虑线程优先级的SMT任务调度策略，对于一组随机任务组合，可提高系统吞吐率达40%，可将响应时间降低33%。

2)线程敏感性的调度

线程敏感调度利用线程自身执行过程中的执行信息作为反馈进行启发调度。Parekh[13]提出的基于IPC的线程敏感调度策略，分别计算每线程的整数和浮点流水线的IPC，调度中将浮点IPC计数最高的线程和整数IPC计数最高的线程调度在一起。仿真发现其性能相对轮转调度算法(RR, Round Robin)提高了7%～15%。敏感调度策略还可以分别面向L1Dcache、L2cache、DTLB的缺失率和平均内存访问次数等信息进行启发式调度。这类策略只考察线程对某种资源的敏感性，性能有时比RR算法还差。

分析认为，这类策略的效率和三个因素相关：第一，作为启发的资源是否是瓶颈资源；第二，是否可以有效缓解敏感资源的争用；第三，缓解敏感资源争用的同时是否牺牲了其它资源的利用率。其中，基于IPC的线程敏感调度策略不是针对特定资源的，它综合考虑了各方面资源争用情况，因此效果较好，至于IPC对某种资源的敏感程度却很难度量，因此当其中一种或者多种资源争用情况比较严重时，其性能也会降低。

3.2 性能评价指标

在单线程情况下，IPC是性能评价的基本指标。对多线程，需同时考虑多个线程。注意到∑IPC(并行执行线程的IPC之和)或者算术平均(Arithmetic Mean)的评价指标容易受高IPC线程性能的影响。而线程IPC的均方差(Harmonic Mean)指标则偏向于反映低IPC线程的影响。因此，需要一个公平有效的评价指标反映每个线程在调度后受影响的程度。目前常用的评价指标有：

·带权加速比 $WS(t)$[10]：$\sum_{i=1}^{n}(NewIPC_i/OldIPC_i)$。其中，$NewIPC_i$ 表示在某种调度策略下，线程 i 和其它线程同时执行时获取的IPC；$OldIPC_i$ 表示线程 i 单独运行时的IPC。可认为它是一个评价多线程系统中混合任务执行的公平指标。文[10,13,17]中均采用了 $WS(t)$。

·几何平均：文[16]等采用该指标，它有利于描述协同调度中线程IPC降低的程度。

4 CMT结构线程协同调度策略

CMT结构中存在大量的资源争用，研究主要集中在两方面：第一，对线程进行协同调度减少资源争用。第二，如何进行硬件资源有效划分减少争用以及对资源管理以协助调度。

线程协同调度侧重于在OS分配的时间片内，对可运行线程组，进行合理的划分，同一个划分内的线程间争用较少，将它们调度到同一个核来减少线程间资源争用，从而提高系统吞吐率。目前，CMT协同调度策略主要分为三类：

4.1 基于挖掘的策略

该类策略根据抽样结果指导调度行为。优点是不需要事先了解负载的特性，缺点是挖掘空间随着线程和核个数的增长而快速增长，适用于核、线程组合数量有限的情况。

1)CMT共生调度策略

CMT的共生调度[17]是该类策略的典型代表，它是SMT共生的进一步扩展。其主要思想和实现方式基本同文[13]，挖掘指标的选择上可以加入功耗因素的影响。CMT的共生调度，不一定是一种负载平衡的调度。仿真发现，在综合考虑功耗、能量指标的情况下，非平衡式共生调度相对平衡式调度在减少功耗方面具有一定的优势。

2)Prefer Last调度策略

Prefer Last[17]策略在抽样过程中根据当前调度决策进行合理变化后产成新的抽样集合。该策略在保持当前调度决策优势的基础上，加入线程最新运行状况的观察结果。

4.2 基于启发的策略

1)基于某种资源的线程敏感性策略

此类策略[12]通过硬件计数器得到某种敏感资源的共享情况，通过协同调度减轻该资源的争用。主要有基于数据Cache争用的调度、基于寄存器利用率的调度、基于寄存器争用的调度等。此类算法实现较复杂，在减轻某种资源争用后往往会加剧其它资源的争用程度。因此对不同的线程组合可能会有较大的性能差异。

2)基于IPC的调度策略

主要思想是：根据IPC高低划分选定的线程组，将IPC高的线程分在一起，由同一个核执行。主要思想和具体实现方法和基于IPC的SMT线程调度类似。IPC高的线程往往表示其长延迟操作较少[18]，即资源释放快，不易堵塞流水线和队列资源。如果将高IPC线程和低IPC线程调度到一个核时，执行速度不匹配，IPC低者持有资源不放往往会堵塞高者，因此这种将IPC高的线程进行协同调度的策略取得了较好的效果。但是，IPC本身较多变，前一周期IPC高的线程不能代表下个周期也高，这成为基于IPC的调度有时不能获取

好性能的一个原因。

3)基于 RIR 的调度策略

文[12]针对双核、双线程的 CMT 结构(类似 Intel 的超线程 Xeon 处理器[19]),进行了分析后认为:线程执行的不同阶段,IPC 是不断变化的,不同的线程组合,IPC 受影响的程度不同,对同一组待划分线程,IPC 值并不稳定。也就是执行后获取的 IPC 值不能代表线程本身的特性,该文提出采用 RIR (Ready to In-Flight Ratio)作为启发依据。RIR 高表示线程在获取资源后可以快速执行,争用引起的资源的短少对性能的影响很大,线程对资源的敏感度较高,反之亦然。实质上,IPC 反映的是线程受到的影响有多大,RIR 反映的是线程容易受资源短少影响的程度。

4)Electron 策略

Electron 策略[17]计算每个核的 EDF(Energy Delay Product),在运行中定期将 EDF 最低的核上的一个线程转移到 EDF 最高的核上,经过若干次这样的操作,就可能会出现空闲的核。这时,彻底释放该处理器核,从而达到降低功耗的效果。

4.3 基于指令类型的调度策略

文[11]中分析了 CMT 体系结构的长延时容忍优势,认为:如果可以隐藏所有的长延迟操作,m 线程现场 n 核的 CMT 可获得的加速比应该和 m * n 核 CMP 相同,相比起来需要的硬件资源少,可实现性好。该文讨论的系统中,每个核为 RISC 结构,FU 是瓶颈资源。基本思想是:用指令的混合方式作为启发依据,对于长延迟操作指令为主的线程,功能单元得不到有效利用,将此类线程和计算为主、延迟较少的线程协同调度。此外,可认为长延迟操作多的线程将很多运行时间浪费在内存和长延时操作上,CPI 较高,可采用 CPI 作为启发依据,将高 CPI 线程和 CPI 低者协同调度。实验表明,这种方式可获得将近 2 倍的加速比。

5 Cache 争用模型和划分策略

不论 SMT、CMP 还是 CMT 的协同调度研究,内存层次始终是调度策略必须重点考虑的因素。大量文献进行了 Cache 监控管理协同调度研究[20,21,23,25]和 Cache 争用模型的研究[16,22~25]。

文[16]介绍了一种基于硬件计数器的动态内存层次监控机制。这些计数器表征了当 Cache 大小增加时命中率的提高量,从而获取 Cache 大小和命中率之间的关系。在采用 LRU 替换策略的情况下,可以精确估计出每线程性能增益和 Cache 大小之间的关系,进行贪婪式(最大化系统吞吐率) Cache 划分或面向减少 Cache 争用的线程调度。仿真结果表明当 Cache 大小为 1MB~2MB 时,该策略将 L2cache 缺失率减少 14%以上。

文[24]从 OS 的角度分析了 L2cache 划分的重要性,提出了首要考虑公平性的 CMP 系统 L2Cache 划分方法。对共享 L2cache 的 CMP 系统仿真发现,该策略相对于不进行划分时公平性平均提高了 4 倍,吞吐率平均提高了 15%。同时发现提高公平性的同时,吞吐率也得到了提高。

文[22]中针对 CMP 系统的 L2cache 共享问题,提出了三个 Cache 性能模型:FOA(Frequency of Acess),SDC(Stack Distance Competition),Prob(Inductive Probability)。通过预测共享 Cache 争用情况协助线程调度。文[25]中也提出了一个多线程的数据局部性冲突模型来估计协同调度中的线程的 L2cache 缺失状况,该方法需要考察所有的线程组合。

结束语 要想寻求一个好的调度方法,需要综合考虑应用程序多样性以及体系结构两个方面的特征。在当前的应用环境下,不同的应用程序具有不同的资源需求。异构型硬件和适应性硬件可以和应用程序多样性相适应。随着进程或者线程个数的增加,以及处理器的个数的增加,需考察的调度决策数呈指数增长,基于挖掘的策略很难适应这种增长。基于启发的策略不存在这方面的问题,但如何寻找一个有效的启发指标是需要进一步解决的问题。因此一个好的调度算法应该和具体的体系结构以及进程本身相关。

对 CMT 系统来说,根据线程特征对所有资源进行核间划分可以减少核间资源争用。先对核间 L2cache 进行划分,缓解核间资源争用情况,再对核内线程进行协同调度减少核内资源竞争的策略,是我们下一步的研究目标。它作为一种软硬件相互适应的解决 CMT 系统资源共享问题的策略,可能会取得令人满意的性能。

参 考 文 献

1 Raasch, Reinhardt S. The Impact of Resource Partitioning on SMT Processors. In: Proceedings of the 12th International Conference of Parallel Architectures and Compilation Techniques, September 2003. 15~26

2 Chaudhry S, Caprioli P, Yip S, et al. Sun Microsystems: High-Performance Throughtput Computing. IEEE Computer Society, 2005. 8~15

3 Spracklen L, Santosh G. Abraham Scalable Systems Group: Chip Multithreading: Opportunities and Challenges. HPCA, 2005. 248~252

4 Tullsen D, Eggers S, Emer J, et al. choice: Instruction fetch and issue on an implement able simultaneous multithreading processor, ISCA96, May 1996. 191~202

5 Tullsen D, Eggers S, Levy H. Simultaneous multithreading: Maximizing on-chip parallelism. ISCA, June 1995. 392~403

6 Merritt R. IBM Weaves Multithreading into Power5. February 2003

7 McNairy C, et al. A Dual-Core, Dual-Thread Itanium Processor. IEEE Micro, March 2005, 25(2): 10~20

8 Kongetira P, Aingaran K, Olukotun K. Niagara: A 32-Way Multithreaded Sparc Processor. IEEE Micro, 2005, 25(2): 21~29

9 www. sun. com

10 Snavely A, et al. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor. In: Proceedings of ASPLOS IX, November 2000. 234~244

11 Fedorova A, Small C, Nussbaum D, et al. Chip Multithreading Systems Need a New Operating System Scheduler. In: 11th ACM SIGOPS European Workshop, Leuven, Belgium, September 2004. 31~35

12 El-Moursy A, Garg R, Albonesi D H, et al. Compatible Phase Co-Scheduling on a CMP of Multi-Threaded Processors. In: 20th International Parallel and Distributed Processing Symposium, April 2006. 10~22

13 Parekh S, Eggers S, Levy H. Thread-Sensitive Scheduling for SMT Processors: [Technical report]. University of Washington, 2000

14 Snavely A, Mitchell N, Carter L, et al. Explorations in Symbiosis on two Multithreaded Architectures. In: Workshop on Multithreaded Execution And Compilation (MTEAC), January 1999. 568~572

15 Hily S, Seznec A. Contention on 2nd level Cache may, limit the effectiveness of simultaneous multithreading: [Technical Report PI-1086]. IRISA, 1997

16 Edward Suh G, Devadas S, Rudolph L. A New Cache Monitoring Scheme for. Memory-Aware Scheduling and Partitioning, HPCA-8: 1, February 2002

(下转第 289 页)

ISM.DLL ISAPI 应用程序，IIS 会重定向所有针对.htr 资源的请求到 ISM.DLL，ISM.DLL 打开这个文件并执行之。但是在这样做之前，ISM.DLL 会截断送给自己的缓冲区内容，去掉.htr 后缀以及结尾的空格，于是我们想访问的文件内容被返回。显然，漏洞的本质在于 IIS 不严格按照接口实现，去掉了.htr 后缀以及结尾的空格。

3.2.2 竞争条件处理错误

竞争条件是在多任务的执行环境下，多个实体对同一个资源进行操作，造成执行结果逻辑上错误的情况。竞争条件所产生的问题是漏洞空窗期(window of vulnerability)的出现。漏洞空窗期是一个能够让攻击者获得暂时的权限，去执行非法行为的短暂时段。举例来说，发生在读写上的竞争条件被称为 time-of-check-time-of-use，简称为 TOCTOU。与 TOCTOU 类似的漏洞是普遍出现的，比如一个进程首先检查一个对象是否存在，或者检查对象的内容，接着读写这个对象，这里就包含一个假定，在从检查到实际读写这个过程中对象不可能被改写，但是这个假定不总是成立的，许多符号连接漏洞都出在这里。

推而广之，安全检测和处理需求必须同步，如果先有安全性检测，再处理需求，可能需求就不一样了，后面的实现必须按照接口的要求实现。如果程序按照这种思路来编写，可能会导致一些微妙的逻辑错误。一个典型的例子是：IIS 在加载可执行 CGI 程序时，会进行两次解码。第一次解码是对 CGI 文件名进行 http 解码，然后判断此文件名是否为可执行文件，例如检查后缀名是否为".exe"或".com"等等。在文件名检查通过之后，IIS 会再进行第二次解码。正常情况下，应该只对该 CGI 的参数进行解码，然而，IIS 错误地将已经解码过的 CGI 文件名和 CGI 参数一起进行解码。这样，CGI 文件名就被错误地解码了两次。通过精心构造 CGI 文件名，攻击者可以绕过 IIS 对文件名所作的安全检查，例如对"../"或"./"的检查，在某些条件下，攻击者可以执行任意系统命令。

3.2.3 应用系统与操作系统对表达形式的理解不同

应用系统和操作系统对于很多问题的理解是不尽相同的，举个例子来说，在 Windows 操作系统中一个文件存在多种表示方法，除了普通的格式以外，还有 86 格式和流格式表示，系统对于这些表示方法的权限设置应该是完全一致的。如果应用程序忽略其中某种文件表示方法，很可能导致安全漏洞，IIS4.0 之前版本的::$DATA 泄漏源码漏洞就是这样产生的：

NTFS 文件系统是与微软的 Windows NT 操作系统，NTFS 文件系统被设计为支持包含在一个文件中的多个数据流。这是一种不需要重新构造文件就能给一个文件添加额外属性或信息的机制。NTFS 上的文件本身的内容包含在没有名字的"流"中，具有内部数据类型 ：$DATA ，也就是包含了所有内容的的主数据流。当 IIS 服务器接收到形如：

http://www.hackart.org/sheepxxy.asp::$DATA

这样的请求时，IIS 本身并不理解数据流格式，因此请求被递交给 Windows，如果 Windows 的文件系统是 NTFS，则它将认为该用户是请求浏览 sheepxxy.asp 这个文件的主数据流，便将 sheepxxy.asp 的内容而不是经过 asp.dll 处理过的结果返还给用户，致使发生源代码泄露。类似的可能出现问题的地方还有：

汉字半字节被截断：按照 Windows 的理解，如果发现半个汉字，会把这半个汉字删掉，例如%81 会被去掉，".asp%81"就变成了".asp"，这与接口定义是不一致的，这个接口实现上的错误导致了 IIS 的一个泄漏源代码漏洞。

Windows 自动删除文件末尾的空格或者句点：在 Windows 中，dir linux.\与 dir linux 的效果是一致的，操作系统会自动删除文件末尾的空格或者句点，应用系统应该理解操作系统的这种特性，否则就容易出错。但是类似的鲜为人之的操作系统的特性还很不少，这就导致了很多安全问题。

结论 虽然有很多具体的漏洞分析技术，但是，面对越来越庞大和复杂的软件系统，这些分析技术往往无能为力，迄今为止，这个领域的工作往往还是要依赖于个人的经验。

我们的工作表明，发现安全漏洞的关键，往往在于正确地分析系统的安全需求，用安全需求来指明分析的方向，将具体的分析手段用于这些比较窄的方向，能够大大降低分析的工作量，提高分析效率。

参考文献

1 Larochelle D, Evans D. Statically Detecting Likely Buffer Overflow Vulnerabilities. USENIX Security, 2001

2 Du Wenliang, Mathur A P. Vulnerability Testing of Software System Using Fault Injection: [Coast TR 98-02]. 1998

3 Flake H. Auditing binaries for security vulnerabilities. http://www.blackhat.com/presentations/bh-europe-00/HalvarFlake/HalvarFlake.ppt

(上接第 258 页)

17 DeVuyst M, Kumar R, Dean T. Exploiting Unbalanced Thread Scheduling for Energy and Performance on a CMP of SMT Processors. In: IPDPS-2006, Rhodes Island, Greece, April 2006. 10～18

18 Tullsen D, Brown J. Handling long-latency loads in a simultaneous multithreaded processor. In: 34th International Symposium on Microarchitecture, Dec. 2001. 318～327

19 Intel Corporation. IA-32 Intel Architecture Optimization: Reference Manual. http://www.intel.com/design/pentium4/ manuals, 2004

20 Kihm L J, Connors D A. Implementation of Fine-Grained Cache Monitoring for Improved SMT Scheduling. In: Proceedings of the 22nd IEEE International Conference on Computer Design, October 2004. 326～331

21 Settle A, Kihm J L, Janiszewski A, et al. Architectural Support for Enhanced SMT Job Scheduling. In: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, October 2004. 63～73

22 Chandra D, Guo F, Kim S, et al. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, February 2005. 340～351

23 Edward Suh G, Devadas S, Rudolph L. A New Memory Monitoring Scheme for Memory-Aware Scheduling and partitioning. In: Proceedings of the Eighth International Symposium on HPCA'02, February 2002. 117～118

24 Kim S, Chandra D, Solihin Y. Fair Cache sharing and partitioning in a chip multiprocessor architecture. In: the 13th PACT, 2004. 111～122

25 Fedorova A, Seltzer M, Small C, et al. Performance Of Multithreaded Chip Multiprocessors And Implications For Operating System Design. In: Proceedings of USENIX 2005 Annual Technical Conference, June 2005. 395～398