CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 36 卷第 1 期 2014 年 1 月
Vol. 36, No. 1, Jan. 2014

文章编号:1007-130X(2014)01-0057-06

应用协同的进程组内存管理支撑技术*

陈鲍孜,吴庆波,谭郁松

(国防科学技术大学计算机学院,湖南 长沙 410073)

摘　要:云计算进行资源聚合的一种重要方式是将不同用户、不同特征的应用聚合起来进行混合部署、同时运行。相比之下,用户态应用的垃圾回收器对服务个体的内存管理针对性更好,而操作系统对整体内存资源分配能力更强。现有内核的机制仅能保证服务在全局内存或进程组内存使用达到上限时被动地进行垃圾回收。结合 Linux 内核中的进程控制组机制以及 eventfd 事件通知机制,设计实现了一个简单高效的应用协同分组内存管理的内核支撑机制。通过在内核中增加应用协同的内存管理机制,进一步增加了系统对应用自主管理内存的支撑能力。实验表明,新的机制没有给原有的操作系统带来明显的性能影响。

关键词:内存管理;Linux 内核;云计算

中图分类号:TP316　　**文献标志码**:A

doi:10.3969/j.issn.1007-130X.2014.01.010

Supporting mechanism for application-assisted memory management of processes group

CHEN Bao-zi, WU Qing-bo, TAN Yu-song

(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: An important way to aggregate resources is to consolidating applications of different users with different features. While user space application garbage collector aims at memory management of specific service, operating system knows better about how to allocate memory resource through the entire system. However, current Linux kernel mechanism can only notify application to do garbage collection when the memory usage reaches the upper limit of either global system. The paper designs and implements the supporting mechanism for collaborative memory management in Linux Kernel by Cgroup and Eventfd subsystem, which can further increase the system's ability to support application's own memory management policy. The experiments show that the new mechanism does not bring significant performance regression to the original operating system.

Key words: memory management; Linux kernel; cloud computing

1　引言

云计算是继 20 世纪 80 年代大型机到客户端-服务器的大转变后的又一次巨变,它基于互联网的计算方式,将共享的软硬件资源和信息按需提供给计算机和其他设备。与传统高性能计算应用长时间满负荷的系统负载相比,云计算的系统负载会随着事件而波动,服务器利用率通常在 10%～50%。

云计算一方面对服务资源进行聚合并统一调配,另一方面又对应用程序透明并表现出一定程度的虚拟化。进行资源聚合的一种重要方式是将不同用户、不同特征的应用聚合起来进行混合部署、同时运行。如何更有效地提高云计算中大规模数

* 收稿日期:2012-07-02;修回日期:2012-11-27
基金项目:核高基重大专项资助项目(2012zx01040001)
通信地址:410073 湖南省长沙市国防科学技术大学计算机学院
Address: College of Computer, National University of Defense Technology, Changsha 410073, Hunan, P. R. China

据中心的服务器资源利用率,降低基础架构运营开销,始终是云计算基础架构所重点考虑的问题。

云计算环境下,应用程序混合部署与同时运行要求操作系统能够有效地进行资源管理。有效的资源管理应该包括五个方面的内容:资源限制、资源隔离、优先级、资源统计与资源控制。其中,最关键的因素是资源限制,它包括了以下两个方面的内容:

(1)限制应用对资源的消费额度;

(2)当资源消费接近额度时,系统采取的相应操作。

研究表明,将内存回收工作部分提交到应用程序的层面来完成会带来两大优势:首先,应用程序进行垃圾回收比系统级的页面回收所需要的代价相对较小;其次,应用程序对所维护缓存页的冷热程度更加清楚,回收释放时更加具有针对性。对于内存资源的管理,操作系统与应用协同进行管理的模式能够有效地提升系统的效率。

2 相关研究

为了降低运营成本,云计算平台通常需要在保证服务质量 QoS(Quality of Service)的前提下尽可能地进行资源聚合。常用的资源聚合方式之一是使用虚拟化技术将不同的服务整合到同一个物理节点中,并通过内存超售(Overcommit)技术进一步提高物理资源的使用效率。

资源的高度整合使得不同应用之间对资源产生竞争。但是,如果采用经典操作系统的内存管理方式,容易造成不同应用之间的性能抖动。为了保证应用程序的 QoS,人们通常使用系统级虚拟化技术(如 Xen、KVM 等)对单独应用程序所消费的内存资源进行一定程度的隔离,通过气球驱动(Balloon Driver)[1]等内存管理技术,使得内存资源可以弹性地根据需求进行动态分配。Schwidefsky M 等人[2]在 Linux 系统上实现了一种协作式的内存管理方法,宿主操作系统与客户操作系统通过交换共享内存页面的使用与驻留集大小的信息,降低了换页的概率,提高了系统整体性能。然而,在某些高度聚合云计算应用场景中,由于单节点内的应用实例不断增加,系统级虚拟化自身的开销占总体资源消耗的比重越来越大,因此人们开始寻找更轻量级的解决方案。

同时,为了进一步降低应用实例之间竞争内存资源时带来的性能抖动,许多研究者提出了操作系统与应用程序相互协同的内存管理策略。

Iyer S 等人[3]指出,在多数情况下,操作系统内核对应用程序实际的内存资源使用情况并非十分精确,以至于用户态应用程序实际上并不总是能够最优化地自动对内存进行申请与释放。Yang T 等人[4]设计实现了 CRAMM 系统,该系统由操作系统内核与改进后的用户态 JVM 虚拟机两个重要的部分构成。CRAMM 的操作系统内核负责收集、统计系统内存资源使用情况并反馈到用户态的 JVM 虚拟机中。用户态 JVM 虚拟机根据内核反馈的信息进行有针对性的垃圾回收工作,使得应用程序的堆维持在合适大小,提高系统整体吞吐效率。Hines M R 等人[5]设计实现了 Ginkgo,将内存使用的信息与应用程序性能进行关联建模,通过在 JVM 虚拟机实例之间合理地对内存资源进行重新部署与分配,以较小的性能回退代价节约了大约 27%的内存消耗,有效的提升了系统整体聚合能力。

本文在总结前人研究的基础之上,结合 Linux 内核中成熟的进程控制组机制以及 eventfd 事件通知机制,设计实现了一个简单高效的应用协同分组内存管理的内核支撑机制。通过该机制,可进一步提升云计算基础架构中内存资源利用率。

3 基于进程组的内存管理

在 Linux 内核发展的历史上,人们做了许多关于进程资源分组管理的工作。这些工作可以划分为两个方面的内容:一类为资源的监控,另一类为利用名字空间进行隔离。从另一个角度看,这两方面的工作实际上也是紧密相关的,它们都试图防止进程不受限制地消耗所有的系统资源,从而获得更高的系统利用率与更好的 QoS。

早期 UNIX 操作系统关于资源限制与管理的手段有限,主要是通过 XSI 扩展中所规定的 getrlimit()与 setrlimit()系统调用对内核中 struct rlimit 结构体所描述的资源进行的配置。但是,rlimit 机制主要是针对单个进程的资源,无法对进程组进行约束。

随着内核级容器虚拟化技术的出现,人们开始关注对进程组资源进行统一管理的机制与策略。在 Linux 内核 2.2 版本中,由 Cox A 与 Savochkin A 共同开发了 User Beancounter 机制,用于弥补 setlimit()限制与管理进程组的不足。在此基础上,Emelianov P 等人[6]改良了 Beancounter 的机

制,并将其作为 Linux 内核级容器虚拟化 OpenVZ 项目的核心技术应用到实际的工程领域。在 OpenVZ 的实现中,Beancounter 代表了进程组消费资源的统计,它由一个 ID 号以及一组资源限制参数组成。而进程分组则依赖于 Linux 内核的名字空间。OpenVZ 通过将进程统一划分到某个名字空间来实现容器间的隔离,同时完成对容器内进程集合的内存资源限制。

然而,OpenVZ 作为 Linux 内核级容器虚拟化技术的代表,更加关注不同 VE(Virtual Environment)之间的隔离性以及对包括 vma 与 kmem 在内的各种内存资源的精确审计。由于不同 VE 之间需要用名字空间隔离开来,OpenVZ 中进程组实际上以不同的 VE 为组织单位。而一个 VE 是一个完整的操作系统用户空间,包含了必要的运行时环境及守护进程。因此,作为通用的基于进程组的内存管理技术,Beancounter 显得不够灵活。

为了使 Linux 内核具备通用的进程组容器机制,Google 公司的 Menage P B[7] 设计实现了控制组(Cgroup)。基于 Linux 内核中的控制组机制,人们可以更方便地实现关于进程内存资源的分组管理。

3.1 Linux 内核控制组机制

Linux 内核控制组的设计目标是为资源管理提供一个统一的框架,包括整合已有的 cpuset 等子系统,并为未来开发新的资源管理子系统提供基本的接口。

按照 Linux 内核中的实现,控制组的设计如图 1 所示,包含以下三个重要的概念:

(1)Cgroup,即控制组 Control group 的简称,它是具体控制进程行为的分组。Linux 内核控制组机制中与进程分组相关的资源控制均以其为单位实现。

(2)Hierarchy 为一个具有树状结构的 Cgroup 集合,系统中的每一个进程都会对应到某个 Hierarchy 中的某个 Cgroup。

(3)Subsystem 为具体某一类资源控制器的子系统,例如 Memory Subsystem 为分组内存资源管理的一类控制器。其作为 Cgroup 中可以动态添加/删除的模块,在 Cgroup 框架下提供多种行为的控制策略。Subsystem 必须通过"附属"(attach)操作关联到具体的 Hierarchy 上才能产生作用。

这三个概念与各个任务之间的对应关系如图 2 所示,包括以下几个方面:

(1)每次在系统中创建新的 Hierarchy 时,该系统中的所有任务都是其根节点的初始成员。

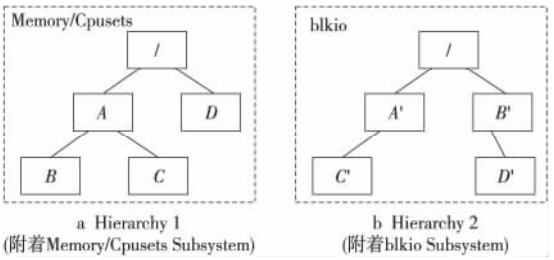

Figure 1 Concepts of Cgroup, Hierarchy, Subsystem

图 1 Cgroup、Hierarchy、Subsystem 概念示意图

(2)一个 Subsystem 只能被附属到一个 Hierarchy 之上。

(3)一个 Hierarchy可以附属多个 Subsystem。

(4)一个任务可以是多个 Cgroup 的成员,但必须属于不同的 Hierarchy。

(5)当系统中的任务创建子任务时,该子任务将自动成为其父任务所在的 Cgroup 成员。系统管理员可以根据需求将该子任务迁移到不同的 Cgroup 中,但在初始创建时它总是继承其父进程所在的控制组。

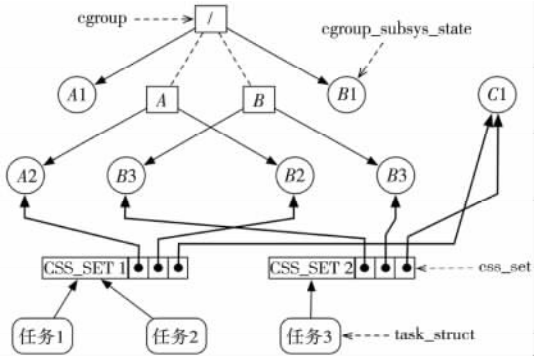

Figure 2 Relationship of Cgroup components and task

图 2 Cgroup 各个概念与任务之间的关系示意图

3.2 内存控制组

内存控制组允许系统以进程组为粒度限制管理用户态应用程序内存的消费。相比 Beancounter 机制,内存控制组主要关心用户空间页面。对于隔离性要求不高的应用场景,内存控制组能够提供更好的灵活性和相对较低的开销。

对于每一个内存控制组,系统允许管理员设置该组进程的内存使用硬上限、软上限、swappiness 等参数,并提供组内 OOM(Out of Memory)等行为控制。Linux 内核在原有基础上的重新设计与实现,以更好地完成内存控制组所带来的新机制。下文将从内核的设计与实现两个角度分别阐述新机制带来的挑战。

在设计层面,主要面临的是如何处理共享页面

和线程组共享地址空间的问题。为了简化逻辑,减少统计计数给系统所带来的性能回退,内存控制组在实现时将多个进程组共享的页面计数归属到第一个访问该页面的进程组中。同时,为了兼顾公平性,当持有共享页面的进程组释放该页面时,该页面的计数将迁移到其他某个持有该页面的进程组中。由于 Linux 内核中线程是作为一种轻量级的进程来处理,因此实际上会存在属于同一进程的不同线程处于不同的 Cgroup 分组当中。而这些线程实际上是共享了相同的地址空间,因此其中相关的所有页面的计数在严格意义上应该被不同的分组所共享。但是,内核为了简化实现逻辑,将内存的使用计数仅归属于线程组主线程所在的 Cgroup 分组中。

在实现层面的主要问题是如何高效处理任务在分组间迁移时的相关页面统计计数。为了提高系统的性能,内核在处理相关逻辑时进行了简化。当任务进行迁移时,内核仅将任务本身迁移到新分组当中,而之前的统计计数仍保留在旧分组。当原先计数所代表的内存页面被释放时,内核再在原分组内减去对应的计数。在迁移之后产生的新计数,内核会自动统计到新的分组记录中。

同时,为了尽量减少对内核数据结构 struct page 的改动并最大程度地降低开销,内存控制组在实现时将原先全局 LRU 链表划分为每进程组的 LRU 链表,各个组之间的 LRU 链表通过所在 Hierarchy 的树状结构关联起来。因此,当内核进行全局内存回收时,将从单一遍历全局 LRU 链表转变为从 Cgroup 根节点开始遍历各组局部 LRU 链表,进行页面交换与页面回收。这种设计虽然带来了一部分额外的开销,但简化了对原有页面回收实现的修改。

3.3 内存控制组与 Beancounter 对比测试

为了能够更直观地了解内存控制组在实际应用中的性能,本文将其与 Beancounter 机制进行了对比测试。由于 Beancounter 是 OpenVZ 的核心组成机制之一,因此需要通过 OpenVZ 的虚拟机性能反映。为此,本文选取了 LXC(Linux Container)内核级虚拟化项目作为内存控制组的参照环境。这两个项目均基于 Linux 内核,采用相同名字空间机制隔离不同的 VE;并且,LXC 与 OpenVZ 可以使用相同的用户态环境(即 VE 的根文件系统)。不同之处在于,LXC 项目使用控制组机制进行资源限制,而 OpenVZ 使用 Beancounter 机制。

对比测试的实验平台为 Intel 双路 Xeon E5620(16 核)、24 GB(6×4 GB)内存,采用麒麟 Linux 3.2 服务器操作系统(内核版本 2.6.32),使用 OpenVZ 官方提供的 VE 根文件系统,共 16 个 VE,每个 VE 分配 1.5 GB 物理内存配额,测试用例为 Unixbench,结果如图 3 所示。

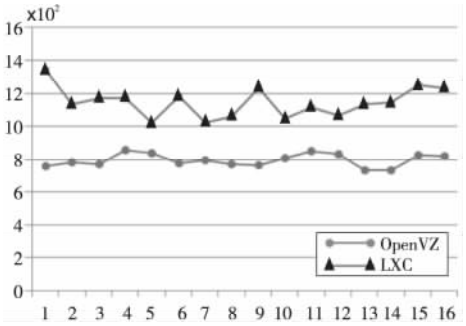

Figure 3 Performance comparison of openVZ and LXC

图 3 OpenVZ 与 LXC 性能对比测试结果

从 Unixbench 的结果观察,使用内存控制组的 LXC 平均性能要高于使用 Beancounter 的 OpenVZ,但其 VE 间性能波动相对较大,隔离性相比较而言不如 OpenVZ。当以提升系统整体吞吐量为主要目标时,内存控制组相对而言更加合适。

4 应用协同的内存管理支撑机制

用程序混合部署与同时运行要求系统软件能够有效地进行资源管理。目前,Linux 内核使用内存控制组时,主要是通过限制组内内存分配的硬上限(Limits)与软上限(Soft Limit)来进行管理。当组内内存申请失败时,在组内私有的 LRU 链表上触动组内回收,如果回收失败,将会触发组内 OOM。当应用程序充分竞争造成全局内存紧张时,系统首先试图将各组使用内存量回收到软上限,然后再由 kswapd 内核守护线程进行全局页面回收。

研究表明,使用用户态协同的内存垃圾回收机制,能进一步提升系统整体性能。其主要原因有两个方面:首先,系统换页的代价少则只需要进行一次 IO,多则需要进行两次,而应用程序进行垃圾回收的代价通常更小;其次,软上限应该设定的值在实际应用中没有客观计算标准,而应用程序对所维护的缓存页的冷热程度更加清楚,回收释放时更加具有针对性,降低了抖动发生的概率。

4.1 Linux 内核 eventfd 通知机制

可应用于传统 UNIX 进程间的通信机制包括管道、套接字、信号等等。一般情况下,管道机制常

被编程者作为通知机制来异步唤醒 select(或等价的 poll/epoll)调用。

eventfd 是 Linux 内核中一个新的高效线程间事件通知机制,一方面它比传统的管道少用一个文件描述符;另一方面,eventfd 的缓冲区管理相对简单,全部缓冲仅有 8 字节。利用该通知机制,编程者只需要通过 eventfd 系统调用获得关于事件的文件描述符,然后使用经典的 IO 函数监听从该文件描述符传递过来的通知事件。

4.2 进程组内存临界通知机制的设计与实现

实现应用感知的内存管理,在操作系统内核一级关键是需要具备两方面的能力。一方面,资源聚合要求在单一节点上聚合多个应用实例,并且内核能为应用实例之间提供一定程度的资源隔离与 QoS。通过内存控制组的支持,资源隔离与 QoS 能够在一定程度上得到保障。另一方面,为了能够进一步高效利用系统资源环境,需要操作系统具备将进程组内存压力通知相关用户态应用的能力。同时,系统应该提供灵活的机制供应用程序将对内存资源的具体需求反馈给操作系统,以便系统能够做出更精确的资源控制。

为了达到以上目标,本文完成了以下两个方面的工作。首先,将 eventfd 的机制与 Linux 内核控制组相结合,通过向 Cgroup 虚拟文件系统中指定的控制文件进行配置将两种机制关联起来,实现针对各个控制组的事件通知机制的框架。然后,在前者的基础上,为每个内存控制组增加内存使用阈值数组。分组内的应用根据自身的特点将预期的阈值通过 Cgroup 文件系统写到该数组中。当进程组所消耗的内存数量超过阈值时,内核将通过 eventfd 机制通知相关应用进行用户态的内存垃圾回收。详细设计的示意图如图 4 所示。

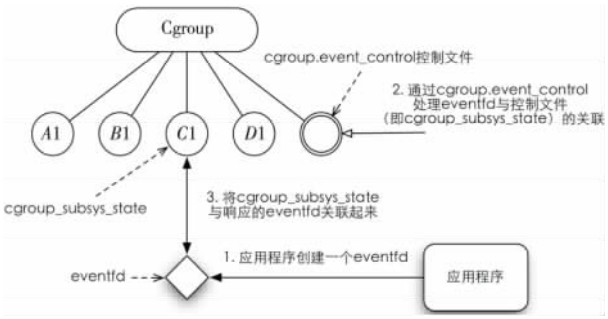

Figure 4 Processes group notification mechanism

图 4 进程组通知机制示意图

通过 Cgroup 虚拟文件系统进行关联,避免了增加新的系统调用所带来的复杂度。应用程序利用传统的编程接口申请创建并获得 eventfd,并将该 eventfd 描述符、对应的 Cgroup 控制文件(即 cgroup_subsys_state 在 Cgroup 虚拟文件系统中关联的文件描述符)以及语义相关的参数同时写入指定的 Cgroup 控制文件中。内核处理该文件的写入操作时,启动其对应的 Cgroup 中相关事件监听与处理逻辑,完成 Cgroup 与 eventfd 的关联。

内存控制组在分组进行 charge/uncharge、组内页面进行迁移时进行阈值检测,如果超过阈值则通过先前注册的 eventfd 向应用程序发送通知。应用程序在接到通知后即可进行相应的处理逻辑。在一个内存控制组中,本文设置了多个阈值的槽位。应用程序可以根据需要在同一个控制组中对多个阈值进行监听。为了能够进一步加快处理速度,在向控制组注册通知的时候,内核对存储阈值的数组进行重新排序,并设置当前阈值指针为不大于当前内存使用量的最大槽位。

这种设计尽可能地利用了 Linux 的已有成熟机制,降低了引进新机制后对系统稳定性带来的风险,实现了一种简单可靠的内存组临界通知机制。

4.3 测试与评估

本文的主要工作基于麒麟 Linux 3.2 服务器操作系统稳定版的 2.6.32 内核,主要目标是对新机制所带来的潜在性能回退进行评估。评估的实验平台为开启 Virtio 机制的 KVM 虚拟化客户端操作系统,虚拟机配置为六个 CPU 核心,2 GB 物理内存。实验结果如图 5 和图 6 所示。

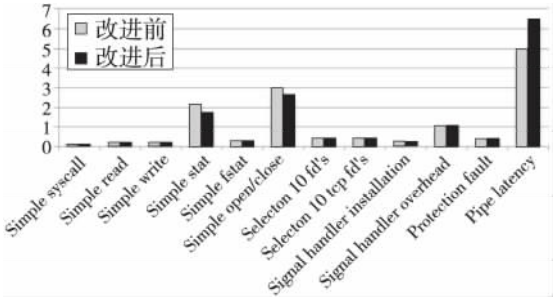

Figure 5 Comparison of latency

图 5 延迟对比

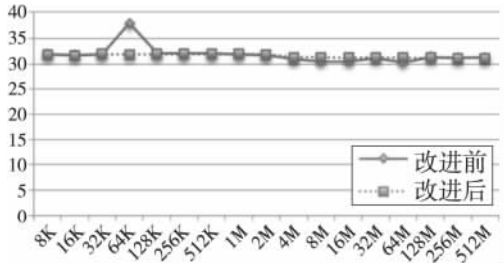

Figure 6 Memory access bandwidth comparison of mmap

图 6 mmap 访存带宽对比

实验结果表明,以 Cgroup 与 eventfd 为基础的进程组内存临界通知机制是资源聚合环境下一种简单实用的内存管理方法,且新机制的引入未对现有的稳定版内核带来明显的性能回退。

5 结束语

在大规模云计算的环境下,如何提高系统资源利用率是学术界与工业界一直关注的重点。本文在前人的研究基础上,设计了一种可以由应用程序主动向内核注册触发内存回收条件的机制,兼顾了应用对自身内存资源了解与系统对全局内存资源规划两方面的优势。通过该机制,系统管理员能很容易地完成进程组范围内存资源限制,并可以通过 eventfd 事件通知相关应用在用户态进行垃圾回收。

下一步的工作,将在这个机制的基础上实现用户态应用自主事件的注册,以及根据内核通知进行自主垃圾回收。

参考文献:

[1] Waldspurger C A. Memory resource management in VMware ESX server [C]// Proc of the 5th Symposium on Operating Systems Design and Implementation, 2002:181-194.

[2] Schwidefsky M, Franke H, Mansell R, et al. Collaborative memory management in hosted Linux environments [C]// Proce of Ottawa Linux Symposium, 2007:313-328.

[3] Iyer S, Navarro J, Druschel P. Application-assisted physical memory management for general-purpose operating systems [R]. Huston, TX 77005, USA: Rice University, 2004.

[4] Yang T, Berger E D, Kaplan S F, et al. CRAMM: Virtual memory support for garbage-collected applications[C]// Proc of the 7th USENIX Symposium on Operating Systems Design and Implementation, 2006:103-116.

[5] Hines M R, Gordon A, Silva M, et al. Application know best: Performance-driven memory overcommit with ginkgo [C]// Proc of the 3rd International Conference on Cloud Computing Technology and Science, 2011:1.

[6] Emelianov P, Lunev D, Korotaev K. Resource management: Beancounters[C]// Proc of Ottawa Linux Symposium, 2007: 285-292.

[7] Menage P B. Adding generic process containers to the Linux kernel[C]// Proc of Ottawa Linux Symposium, 2007:46-57.

作者简介:

陈鲍孜(1986-),男,湖南长沙人,硕士生,研究方向为操作系统。**E-mail**:chen_baozi@163.com

CHEN Bao-zi, born in 1986, MS candidate, his research interest includes operating system.

吴庆波(1969-),男,浙江宁波人,博士,研究员,研究方向为操作系统和虚拟化。**E-mail**:wqb123@263.net

WU Qing-bo, born in 1969, PhD, research fellow, his research interests include operating system, and virtualization.

谭郁松(1976-),男,江西鹰潭人,博士,副研究员,CCF 会员(E200016861M),研究方向为操作系统和虚拟化。**E-mail**:yusong.tan@gmail.com

TAN Yu-song, born in 1976, PhD, associate research fellow, CCF member (E200016861M), his research interests include operating system, and virtualization.