面向普适计算的资源隔离管理机制

张琛昱1 陈渝1 张乾宇2

1)清华大学计算机系，北京， 100084

2)北京航空航天大学软件学院，北京 100191

摘 要 近年来，智能手机、便携电子设备的进化浪潮带动了普适计算的蓬勃发展。以社交媒体为典型代表的移动端应用如春笋般涌出，导致用户数急剧上升，应用后台的服务器规模也成倍增长，出现了现代的数据中心。数据中心为了确保用户的请求能够迅速得到响应，往往需要预留充足的计算资源，防止突发的密集请求波峰导致资源饱和。资源饱和将使得用户获得响应延后，带来极差的用户体验，也降低了应用商的商业价值。预留大量空闲算力会造成计算资源浪费，使供应成本升高。而同时部署其他计算任务又会引入新的挑战——共享资源冲突带来的性能下降。为了解决这一实际问题，本文从资源管理角度，设计了一套共享资源隔离管理系统，试图解决这一问题。

**关键词** 系统管理；共享资源隔离；利用率提升；任务混部；反馈调节

中图法分类号 \*\*\*\*　　　DOI号 \*投稿时不提供DOI号\* 分类号

Title \*（中英文题目一致）字体为4号Times New Roman,加粗\* Title

NAME Name-Name1) NAME Name2) NAME Name-Name3) \*字体为5号Times new Roman\*Name

1)(Department of \*\*\*\*, University, City ZipCode, China) \*字体为6号Times new Roman\* Depart.Correspond

2)(Department of \*\*\*\*, University, City ZipCode)\*中国不写国家名\*

3)(Department of \*\*\*\*, University, City ZipCode, country)\*外国写国家名\*

**Abstract** \*Abstract (**500英文单词，内容包含中文摘要的内容**). 字体为Times new Roman,字号5号\* Abstract

**Key words** \*key word（**中文关键字与英文关键字对应且一致**，**不要用英文缩写**）; key word; key word; key word\* \*字体为5号Times new Roman \* Key words

**1 引言**

在过去的近10年中，各种智能计算设备层出不穷。从智能手机到智能平板，从电子书再到智能可穿戴设备，各种计算设备不断推陈出新。无论是在硬件性能加强、操作系统优化层面，还是交互方式改进、用户体验提升等方面，移动智能终端都处于一种蓬勃发展与进化的状态之中。这使普适计算这种计算形式悄无声息地逐渐融入到人们的日常生活中来。

与此对应的软件生态，也处在稳健的发展步态之中。其中，典型的软件包含社交媒体应用、音频流媒体应用、在线搜索等。这些应用的用户对于自己请求的响应速度有着极高的要求。若社交媒体评论、视频加载不能够得到及时响应，其用户体验会直线下降。而对于企业本身而言，如果不能及时响应用户请求，也会为自己带来经济上的损失。以Google提供的搜索服务为例：据统计，当搜索服务页面的加载时间从0.4s增加至0.9s时，Google的搜索流量及广告商收入将会下降20%[1]。其中关联，可见一斑。

数据中心为了保持低延迟响应，服务器的平均空闲率很高，造成大量的资源浪费。如果同时部署其他服务，又会在物理机的多处共享资源上产生冲突，导致服务延迟增长。

本文基于当下现有的硬件机制与操作系统层面的软件机制，设计了一套共享资源的隔离管理系统，能够有效提升服务器的资源利用率，从而降低数据中心构建成本，减少资源浪费。

**2 问题与挑战**

**2.1 提升资源利用率**

各种普适计算设备中，存在着庞大的应用软件。而在这些应用背后，无一例外，都存在着相当规模的服务器集群，以应对巨大用户数并发所带来的请求消息洪峰。这些服务器集群组成了现代的大规模数据中心。

构建与维护一个大规模数据中心的成本是高昂的，其中服务器方面的购置费占了相当大的比例。据统计，在现代数据中心中这一比例可达50%-70%[2]。随着电子工艺的发展，同面积芯片可承载的算力和核数越来越多，这一定程度上降低了算力的平均成本。然而随着摩尔定律到达瓶颈期，我们需要其他方法来降低这一花费。提高有限资源的平均利用率是必然的选择。

**2.2 混部引入的性能下降**

据若干研究显示，很多数据中心中的服务器平均利用率十分低下，在10%到50%之间[3]。一个重要的原因是大部分服务器都在运行延迟敏感型服务（Latency-critical service，简称LC服务，如社交媒体评论、搜索引擎、在线邮箱、导航等）。这些服务具有一个特点：负载不确定性。虽然在宏观层面上，负载具有一定的昼夜规律，但仍会因为某些不可测原因产生剧烈的波动，导致流量尖峰的出现（如微博热搜）。为了应对这些流量洪峰，数据中心必须确保足够的闲置计算资源。因而，服务器的低利用率是可以理解的。

想提高资源利用率，一个较直接的想法是在LC服务负载较低时，部署另一类计算应用。这一类应用不要求有极低的延迟，但是将耗费较大的计算资源，我们称之为批计算任务（Best-effort batch，简称BE任务，如机器学习模型训练、3D图形渲染等）。

但将这两类任务进行混合部署，将会带来一个严重的问题：共享资源部件上的冲突。这常常体现在末级缓存、内存、IO通道以及网络带宽处。例如，统一物理机的不同物理核常共享一块末级缓存。当任务混部时，常会由于一类任务对缓存的读写，导致另一类任务的缓存缺失，进而产生内存访问行为。由存储的层次性，我们不难预测其响应速度将会明显下降。冲突激烈时甚至还会造成内存带宽拥挤导致的进一步延迟。

LC服务常用尾延迟（tail latency）作为QoS（Quality of Service）的评价指标，即使一点小小的干扰也会令其产生猛烈的增长。因此在传统的数据中心中，任务调度往往避免将这两类任务进行混合部署，间接导致服务器大量的计算资源闲置。

要想通过混部的方式提高资源利用率，做好共享资源部件上不同任务的隔离工作就十分重要。

**3 相关工作**

**3.1 Borg[4]**

针对大规模集群的任务调度及部署，Google开发了名为Borg的分布式容器管理系统，用来承载其内部长时间运行的实际生产服务（可理解为LC服务）和批处理任务（BE任务）。

Borg主要解决大规模集群上的任务调度和资源分配问题。它通过基于优先级、配额的调度决策，使得有限分布式集群资源得到合理分配的同时，保证LC服务的质量。同时，它提供对任务状态的追踪机制和集群间的容错机制，是一个典型的分布式、高可用任务分配、调度框架。

此外，早期的Borg通过检查任务的实时资源使用情况来进行较为粗暴的资源隔离。经过改进后，Borg的任务均运行在基于cgroups的容器中，实现了资源隔离。

**3.2 PARD[5]**

中科院计算所在2015年提出了缩写为PARD（Programmable Architecture for Resourcing-on-Demand）的标签化硬件架构。该架构借鉴SDN（Software-Defined Network）的构建思路，在硬件层面上，对每一个通讯请求赋予一个容器级别（或物理核级别）的标签，再通过中央控制平面（Control Plane）中的策略，在缓存等共享部件上针对不同容器（物理核）的数据请求进行差异化、层次化的处理，从而优先保证LC服务的响应速度，同时减弱共享部件上的干扰。但该系统目前尚无生产级别上可用的硬件支持。

**3.3 一些现有的隔离机制**

之前已存在很多对于末级缓存的隔离机制，包括基于替换策略的隔离、以路为单位的隔离、以及一些细粒度隔离机制[6]。但大部分隔离机制不存在可用于实际生产的应用环境。

对于内存带宽、IO通道带宽的隔离，目前尚无成熟的硬件或软件隔离解决方案。

对于网络带宽隔离，在硬件层面上，很多网卡实现了带宽控制功能及优先级机制。但遗憾的是，这些功能并未统一地暴露给设备驱动。

**4 本文主要成果**

前述相关工作中，Borg是一个综合性的分布式容器任务调度系统，具有庞大、通用的特性，但针对资源隔离仅采用了cgroups的简单措施。对于中等规模及以下的集群，并没有很大的应用价值，而且必须以容器作为应用前提，缺少广泛的应用价值。PARD作为一种新颖的硬件架构，但缺少现行可用于生产的实际产品。但它的标签化差异思想，为我们进行共享资源隔离提供了思路。

不同于上述各工作中或过于庞大冗余的调度体系，或过于缺乏可用性的隔离架构，本文的具体贡献如下：

1. 对各共享资源部件进行了干扰试验，验证了LC服务与BE任务混部所带来的QoS下降效应，进行简要分析。
2. 重点聚焦共享资源上的隔离机制，设计了一套资源管理调节系统，提高了系统资源利用率。
3. 采用时下较新的软硬件技术，在各个共享部件上实现资源管理，具有较强实用性。

**5 混部干扰分析**

**5.1 Benchmark**

要度量干扰所带来的负面效应，我们需要一个统一的评判标准。Tailbench[7]是MIT的科研人员制作的一套专门针对延迟敏感型，也即LC服务的尾延迟测量基准工具。我们可以用它测量指定任务在不同环境下的尾延迟。

Tailbench涵盖了8个不同领域的LC测试服务，我们挑选其中的两项服务进行测试。

第一项是xapian[8]。xapian是一个开源的C++信息检索库，提供了包括分词、索引在内的诸多功能。Tailbench基于xapian构建了一项多线程的server端搜索服务，负责从client处接受query进行搜索查询，并返回搜索结果给client。在这个过程中，client会统计相关的尾延迟信息。

第二项是silo[9]。Silo是MIT科研人员设计的一种内存中的快速事务性数据库。Tailbench基于silo同样构建了一项在线存储服务。同样由server端和client端构成。Server端接受存储query，完成存储过程，返回存储结果给client。由client记录尾延迟。

**5.2 实验方法**

5.2.1 总体思路

对于一项LC服务，我们能够确定若干档不同的负载。

1. 首先，在不同的负载下，我们先运行该服务，进行无干扰的尾延迟测量，取得基准数据。
2. 随后，我们同时运行LC服务与一项作为干扰项的BE任务，并测量干扰后的尾延迟数据。
3. 最后，在不同的测试环境下，我们计算干扰后的尾延迟与无干扰时的比值，作为影响程度的度量值。

每一项干扰，都尽可能的局限在某单一共享资源之上。如果涉及多项共享资源，则尽量把多余的共享资源可能带来的影响降至最低。

5.2.2 Stream-LLC

首先，我们进行LLC（Last Level Cache，末级缓存）上的干扰实验。干扰方式是通过名为Stream-LLC的干扰程序进行的。该程序会在死循环中不断遍历访问自己开辟的一个指定大小数组，来产生频繁的读写请求，从而达到缓存访问的目的。其中，干扰程序利用多线程来提高干扰效率，同时通过定距访问（如每64个int访问一次）防止缓存预取带来的低效问题。

我们设定开辟的数组大小分别为整个缓存的25%、50%、100%，将干扰程度分为small、medium、big三档。

通过频繁访问cache，将LC任务在cache中的数据替换掉，使其额外进行内存访问，即对其造成了干扰。

5.2.3 Stream-DRAM

进行DRAM（也即内存带宽）上的干扰方式与Stream-LLC类似，我们只需要将开辟数组调整至远大于cache大小的规模，即可因频繁产生cache缺失所导致的内存访问。

这里造成的干扰主要在于进行内存访问时，与LC服务产生了内存带宽上的竞争。

5.2.4 Hyperthread

Hyperthread作为Intel芯片中的常用技术，通过复用流水线中的空闲部件，允许将一个物理核拆分成两个逻辑核使用。

我们的目的在于利用这项已有的技术，测试仅依靠Hyperthread是否能将干扰带来的额外尾延迟控制在一个合理的范围内。如果它是可控的，那么我们可以直接使用Hyperthread进行混部。

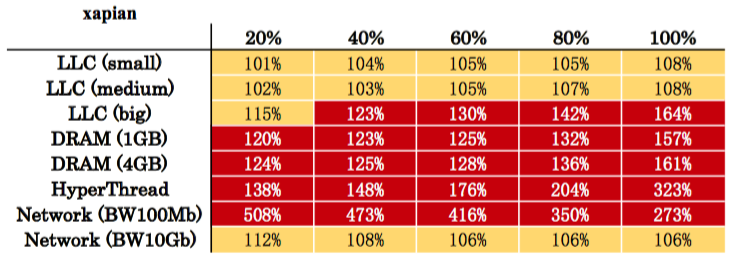
5.2.5 Network

网络带来的干扰体现在出和入两个方向上。这里由于接收不同来源数据包带来的干扰不是主观可控的，我们只考虑出方向的干扰。

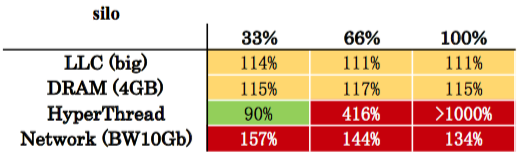
在发送数据包时，由于发送队列的拥塞，LC服务的数据包不能及时地发送出去，就造成了响应速度降低。

这里我们使用iPerf[10]，一个测试网络带宽的benchmark。它会尽可能地挤占出口带宽，发送数据，从而达到干扰效果。

**5.3 实验结果**



**矢量图！！！**



**5.4 结果分析**

1. LLC上的干扰需要分情况讨论。可以观察到，当进行small和medium档的干扰时，xapian所受影响不那么明显。当进行big档的干扰时，其尾延迟都体现出稍明显的增加。Xapian随着负载的增大，访问cache更加频繁，所受影响变得更加明显。

我们可以想象，对于xapian，在代码循环中有一块常用的工作指令集。例如，循环中负责搜索、索引、打分、排序等操作，并返回结果。当LLC受到big规模的干扰时，常用工作指令集产生的缺失将异乎寻常地频繁，所带来的影响也更明显。

1. DRAM上的干扰体现为LLC（big）干扰的增强版。/\*比较简略\*/
2. 通过对Hyperthread环境下的干扰测试，我们得出结论：仅利用Hyperthread进行“伪”资源隔离是不可行的。因此我们在实际应用中，必须保证物理核的独占性。
3. 通过在不同带宽环境下的网络干扰实验，我们观察到了网络方面的干扰也是较为明显的。Xapian在不同带宽环境下，受到的干扰程度也不同，带宽越小，干扰越明显。
4. 在相同条件下对不同的LC服务进行相同的干扰实验，其干扰程度也不尽相同。如对于LLC（big）干扰，xapian体现得更为明显。而对于网络干扰，则是silo更为敏感一些。这说明，不同类型的LC服务之间有着不同的特性。如果要进行隔离与调整，显然我们不能采用静态策略，而应该采用一种动态调整的隔离机制。

**6 系统设计的技术基础**

**6.1 cgroups**

cgroups，控制群组（Control Groups），是Linux内核的一项功能，用来分配与隔离一组进程（或线程）所占用的资源，如CPU、内存、I/O等。该功能最初由Google的两名工程师创造，后被合并至Linux内核中。

cgroups以一个虚拟文件系统（VFS，Virtual File System）的形式存在，其下分为若干子系统，如cpuset子系统、memory子系统等。通过有层次地挂载任务组，可实现逻辑复杂的资源管理。Docker也正是通过使用这一技术实现了对物理资源的隔离控制。

**6.2 Intel® RDT**

Intel® RDT，英特尔®资源调配技术（Intel® Resource Director Technology），是英特尔®公司针对旗下Xeon®系列CPU研发的一种共享资源调控机制。英特尔®编写了对应的C函数库PQoS来支持相应的资源调配功能，根据内核版本的不同，该函数库通过Linux perf（Performance Event，Linux kernel 2.6.31后内建的系统效能分析工具）、resctrl（Recource Control，Linux kernel 4.10后内建的供RDT使用的接口）等不同接口来实现相应的功能。这里我们主要借助其CAT与MBM两种功能。

6.2.1 CAT

CAT，高速缓存分配技术（Cache Allocation Technology）。英特尔®在芯片层面增加了相应的指令，用于将末级缓存分配成不同的配额，分配给不同的服务类（CLOS，Class of Service）；而在软件层面，实现了物理核到指定CLOS的映射绑定操作，以及不同的CLOS对末级缓存的分配操作。

6.2.2 MBM

MBM，内存带宽监控技术（Memory Bandwidth Monitoring）。内存带宽实际中很难直接测量，PQoS库通过估计末级缓存的缺失率、利用率等参数来间接测量内存带宽。我们这里以物理核为基本单位来测量内存带宽占用情况。

6.3 eBPF

eBPF(extended Berkeley Packet Filter)是由linux提供的一套内核跟踪工具，使得用户可以安全、高效的监控内核的运行状态。ebpf程序运行在内核提供的一个简单的虚拟机上，并可以与用户态程序交换数据。本文使用eBPF来实现对LC和BE任务网络流量的实时监控。

**6.4 TC**

TC，linux流量控制（Linux Traffic Control)是由Linux内核提供的一种通过控制网络设备流量来保证Qos(Quality of Service)的机制。

TC提供了多种控制流量的策略，本文主要使用HTB（Hierarchical Token Bucket）来实现对LC和BE流量的分别控制。

**7 系统设计**

**7.1 整体结构**

本文设计的共享资源隔离系统，除了完成资源隔离的功能外，还附带了一个简单的任务调度器，可完成基础的BE任务调度功能。整体而言，系统分为4大组成部分：顶层控制器TopController、核/缓存/内存带宽控制器CoreMemoryController、网络带宽控制器NetworkController以及BE任务调度器Tap。四个组成部分各自具有一个主循环，系统运行后，四个部分各起一个线程运行。

**图！！！！**

7.1.1 顶层控制器

顶层控制器实现的功能如算法1所示。每隔10秒，顶层控制器拉取LC服务的尾延迟数据，计算负载情况和响应差。如果当前的尾延迟大到不可接受，立刻禁止BE任务运行，将全部资源分配给LC任务供其运行。这种情况常见于突发的请求洪峰，此时需要进入一段时间的冷却状态，禁止顶层控制器再次进行调度。接着我们判断负载情况：负载值过高，禁止BE任务运行确保稳定；负载值安全，则可以混部BE任务。当负载值处于两者之间时，如果尾延迟有逼近最大值的倾向，则暂停BE任务的资源增长。必要时，可适当减少其占用资源，防止LC服务不稳定。

算法中的具体数值均为经验参数，可根据实际情况进行调整。

算法**1** TopController’s Algorithm

# *max\_latency：可以接受的最大尾延迟*

# *cur\_tail\_latency() 当前LC服务的实时尾延迟*

# *cur\_load()* *当前LC服务的负载：0% ~ 100%*

*counter* *= 0;*

WHILE (True) {

*sleep(10);*

*latency = cur\_tail\_latency();*

*load = cur\_load();*

*slack = (max\_latency - latency) / max\_latency;*

IF(*slack < 0*) {

*counter++;*

IF (*counter > 1*) {

*disable\_BE();*

*cooling\_down();*

}

} ELSE IF (*load > 85%*) {

*counter++;*

IF (*counter > 1*)

*disable\_BE();*

} ELSE IF (*load < 70%*) {

*counter = 0;*

*enable\_BE();*

} ELSE IF (*slack < 40%*) {

*counter = 0;*

*pause\_BE\_grow();*

IF (*slack < 20%*)

*weaken\_BE\_a\_little();*

}

}

7.1.2 核/缓存/内存带宽控制器

我们将这三者的控制集成在一个控制器中，原因是这三种资源在任务调度方面具有较强的耦合性。我们要完成的任务是，在物理核、缓存、内存带宽分配这三个维度上，寻找到一个最优点（全局或局部），使得LC服务的尾延迟不超过最大值，同时使得资源利用率尽量地高。根据Google的数据研究，若将多数LC服务的最大负载能力视为z，将其占用的物理核数、缓存大小分别视为x与y，则z是关于x与y两个变量的凸函数[11]。这意味着，我们可以采取类似于梯度下降的方法，来找到这个二维平面上的一组最优（至少是较优）的取值，从而使得LC服务在占用尽可能少的资源的同时，计算效率最高。同时，这也使BE任务得到了尽可能多的共享计算资源。

我们通过现有手段可以实现缓存和物理核的分配，但内存带宽的控制尚未有直接控制的方法。由于内存带宽占用与物理核数有一定的相关性，所以我们这里通过控制LC服务与BE任务的物理核分配情况，来间接阻止内存带宽达到饱和。

该控制器的算法如算法2所示。该控制器第一要避免的是内存带宽饱和，我们通过控制BE任务得到的物理核数来间接实现此目标。通过MBM技术，我们可以测量当前的总内存带宽，若超出我们为其设定的经验上限，则砍去多余的BE物理核。

接下来，我们分别在物理核、缓存两个维度上进行“类梯度下降”，从而找到最优点。在缓存方面，我们采取较为保守的策略，每次为BE任务增加一个单位的末级缓存空间，测量其slack和内存带宽占用的变化情况，如果引起了负面效应，则取消此次分配，转移到物理核维度进行优化。在物理核方面，我们采取相对较激进的策略（因为在其他控制器处可以降低物理核数），我们每次根据当前物理核的分配情况及内存带宽，估算再为BE任务增加一个物理核后的带宽。如果该值没有超出范围且slack较为充裕，我们就可以为BE任务增加一个物理核。当然，也要事先检查LC服务在减少一个物理核后，计算资源是否充足。

在该控制器中，优化循环不断进行，最终会达到一个较稳定的“收敛”状态。

算法**2** CoreMemoryController’s Algorithm

# *clear\_status() 重置各共享部件状态*

*# grow\_safe() 判断为BE增加一核后，LC服务是否还有充足运算资源*

*state* = GROW\_LLC;

WHILE (True) {

*sleep(5);*

IF (*BE\_disabled*) {

*clear\_status();*

CONTINUE;

} ELSE IF (*BE\_paused*) {

CONTINUE;

}

*total\_bw = measure\_total\_bw();*

IF (*total\_bw > dram\_limit*) {

*overage = total\_bw – dram\_limit;*

*BE\_cores\_decline(overage/BE\_bw\_per\_core());*

}

IF (*state* == GROW\_LLC) {

*old\_slack = cur\_slack();*

*BE\_cache\_grow();*

*slack\_diff = cur\_slack() – old\_slack;*

*bw\_diff = measure\_total\_bw() – total\_bw;*

IF (*bw\_diff < 0* OR

*slack\_diff < -15%* OR

*cur\_slack() < 0*) {

*BE\_cache\_roll\_back();*

*state* = GROW\_CORES;

}

} ELSE IF (*state* == GROW\_CORES) {

*need = LC\_bw() + BE\_bw() + BE\_bw\_per\_core();*

*slack = cur\_slack();*

IF (*need > dram\_limit*)

*state* = GROW\_LLC;

ELSE IF (*slack > 30%* AND *grow\_safe()*) {

*BE\_cores\_incline(1);*

*state* = GROW\_LLC;

}

}

}

7.1.3 网络带宽控制器

网络带宽控制器通过算法3来监控并控制LC服务与BE任务的网络实时带宽使用情况。我们实时监测LC服务的网络带宽占用，并为BE任务限定一个比剩余带宽量要再小10% \* *total\_bw*的量。这样做的原因是为LC服务留下一部分灵活变动空间。一旦网络洪峰到来，LC服务的流量可以继续升高。同时，该控制器也有一定的缓冲时间进行再次调整。如果LC流量超过了总带宽的90%，我们就停止为BE任务分配带宽限额。

算法**3** NetworkController’s Algorithm

# *total\_net\_bw: 设定的最大网络带宽值，一般由实际人工测量得出*

WHILE (True) {

sleep(1);

*LC\_bw = measure\_LC\_net\_bw();*

*BE\_new\_bw = 90% \* total\_net\_bw – LC\_bw;*

*Set\_new\_BE\_net\_bw(BE\_new\_bw);*

}

7.1.4 BE任务调度器

BE任务调度器负责实时监测当前BE任务的管理状态。如果检测到当前未有BE任务运行，且*BE\_enabled*为真，则负责从任务数据库中获取BE任务信息，通过*fork()*与*execvp()*创建子进程并执行，记录其pid便于系统进行管理操作。然后挂起，等待其任务结束，登记任务完成状态。

**7.2 具体实现**

7.2.1 物理核分配

我们维护LC与BE任务各所占用的物理核数。一般而言，如果我们限定系统可使用的总物理核数比实际可用的物理核数少一些，那么有如下关系：

如果我们设定系统可用物理核数等于实际可用物理核数，那么我们要预留一些物理核给操作系统与其他应用程序。映射关系如图x。

**图！！！！**

我们使用cgroups中的cpuset子系统。创建两个策略组：LC与BE。分别设定其可用内存节点为全部节点、可用的物理核情况、物理核独占性。最后将两任务的pid通过VFS写入到设置文件中去，完成物理核的分配。

* + 1. CAT缓存分配

我们维护两类信息。

1. 从物理核到CLOS的类别映射。
2. 各CLOS的CBM（Capacity Bitmasks）。

在CAT技术中，初始情况下所有物理核被默认归至CLOS0中。我们令LC服务和BE任务分别归属于CLOS1与CLOS2。然后逐个设置两个任务对应的物理核所属的CLOS。

在调整cache大小时，我们通过CBM来设置每个CLOS可使用的cache情况。映射关系如图x。

**图！！！！**

7.2.3 MBM内存带宽测量

在MBM技术中，我们可以获取每个物理核的实时内存带宽占用情况。对于NUMA（Non-Uniform Memory Access）架构的物理机，我们可对本地内存与远端内存分别进行测量。结合LC、BE任务对应的物理核情况，就可测得总带宽，以及LC、BE的各占带宽。

**图！！！**

7.2.4 网络带宽分配

我们主要使用cgroups+TC来实现LC任务和BE任务的网络带宽分配。

首先我们在cgroups的net\_cls子系统中创建两个策略组：LC和BE，然后分别在其中写入网络分类号以及进程pid，这样被控制的进程所发出的数据包就会带上相应的网络分类号。

TC在网络设备上建立一个控制队列，并在队列上建立两个分类，然后根据网络数据包上的分类号将数据包投递到对应的分类中。TC可以对不同的分类设置具体的网络流量限制，这样就可以实现对LC和BE程序网络带宽的实时控制。

7.2.5 网络带宽监控

如上所述，受控制的程序所发出的数据包会带上相应的网络分类号。我们使用eBPF在TC流量的出口监控被发送出的网络数据包，然后根据数据包携带的分类号统计出一定时间内特定类别的进程所发送的数据长度，最后将统计信息传出内核以供用户态程序使用。

**8 初步实验测定**

8.1 实验环境

本次实验环境配置如下表。

表3 实验环境

|  |  |
| --- | --- |
| CPU | Intel® Xeon®  E5-2683 v4 @2.10GHz 16 Cores/32 Threads |
| 内存 | 128GB Samsung DDR4 |
| 末级缓存大小 | 40MB |
| 实验动用物理核数 | 8 |
|  |  |

8.2 实验数据

8.3 简要分析

**9 总结与展望**

**对投稿的基本要求**：

（1）研究性论文主体应包括引言（重点论述研究的科学问题、意义、解决思路、价值、贡献等）、相关工作（为与引言部分独立的一个章节）、主要成果论述、关键实现技术、验证（对比实验或理论证明）、结论（结束语）等内容；系统实现或实验应有关键点的详细论述，以便读者能够重复实现论文所述成果。实验应有具体的实验环境设置、全面细致的数据对比分析。

（2）综述应包括引言、问题与挑战、研究现状分析、未来研究方向、结论等内容。以分析、对比为主，避免堆砌文献或一般性介绍、叙述。

（3）定理证明、公式推导、大篇幅的数学论述、原始数据，放到论文最后的附录中。

**稿件提交时的基本要求：**

（1）本模板中要求的各项内容正确齐全，无遗漏；

（2）语句通顺，无中文、英文语法错误，易于阅读理解，符号使用正确，图、表清晰无误；

（3）在学术、技术上，论文内容正确无误，各项内容确定。

## **1.1** 二级标题 \*字体为5号黑体\*标题**2**

### 1.1.1 三级标题 \*字体为5号宋体\*标题3

\*正文部分, 字体为5号宋体\* 正文文字

文件排版采用MS Word。

**正文文字要求语句通顺，无语法错误，结构合理，条理清楚，不影响审稿人、读者阅读理解全文内容。以下几类问题请作者们特别注意**：

1)文章题目应明确反映文章的思想和方法；文字流畅，表述清楚；

2)中文文字、英文表达无语法错误；

3)公式中无符号、表达式的疏漏，没有同一个符号表示两种意思的情况；

4)数学中使用的符号、函数名用斜体；

5)使用的量符合法定计量单位标准；

6)矢量为黑体，标量为白体；

7)变量或表示变化的量用斜体；

8)图表规范，量、线、序无误，位置正确（图表必须在正文中有所表述后出现，即…如图1所示）(注意纵、横坐标应有坐标名称和刻度值)。

9)列出的参考文献必须在文中按顺序引用，即参考文献顺序与引用顺序一致，各项信息齐全(格式见参考文献部分)；

10）首次出现的缩写需写明全称，首次出现的符号需作出解释。

11）图的图例说明、坐标说明全部用中文或量符号。

12）图应为矢量图。

13）表中表头文字采用中文。

14）公式尺寸：

标准：10.5磅

下标/上标：5.8磅

次下标/上标：4.5磅

符号：16磅

次符号：10.5磅

15）组合单位采用标准格式，如：“pJ/bit/m4”应为 “pJ/(bit·m4)”



图X 图片说明 \*字体为小5号，图片应为黑白图，图中的子图要有子图说明\*

表X 表说明 \*表说明采用黑体\*

|  |  |
| --- | --- |
| \*示例表格\* | \*第一行为表头,表头要有内容\* |
|  |  |
|  |  |
|  |  |
|  |  |

算法**X**. 算法名称.

输入：… …

输出：… …

\*《计算机学报》的算法描述字体为小5号宋体, IF 、THEN等伪代码关键词全部用大写字母，变量和函数名称用斜体\*

致 谢 \*致谢内容.\* 致谢

参 考 文 献

[1] 网上的文献[[1]](#footnote-2)(举例：The Cooperative Association for Internet Data Analysis(CAIDA),http://www.caida.org/data 2010,7,18) **\*请采用脚注放于正文出现处，每页的脚注从1开始编序号\***

[2] 中文的参考文献需给出中英文对照。形式如[3]。

[3] Zhou Yong-Bin, Feng Deng-Guo. Design and analysis of cryptographic protocols for RFID. Chinese Journal of Computers, 2006, 29(4): 581-589 (in Chinese)  
(周永彬, 冯登国. RFID安全协议的设计与分析. 计算机学报, 2006, 29(4): 581-589)

[4] 期刊、会议、书籍名称不能用缩写。

[编号] 作者(外国人姓在前，名在后可缩写, 后同). 题目(英文题目第一字母大写，其它均小写)：副标题(如果有). 刊名(全称), 年, 卷(期): 页码 **\*期刊论文格式\***

[编号] 作者. 文章题目(英文题目第1字母大写，其它均小写)：副标题(如果有)//Proceedings of the … (会议名称). 会议召开城市, 会议召开城市所在国家, 年: 页码 **\*会议论文集论文格式\***

[编号] 作者. 文章题目(英文题目第一字母大写, 其它均小写): 副标题(如果有)//编者. 文集标题. 出版地: 出版社, 出版年: 页码 **\*文集格式\***

[编号] 作者. 书名: 副标题(如果有). 版次(初版不写). 出版社地点: 出版社, 出版年 **\*书籍格式\***

[编号] 作者. 文章题目[博士学位论文/硕士学位论文]. 单位名称,单位地点, 年 **\*学位论文格式\***

[编号] 作者. 文章题目(英文题目第一字母大写，其它均小写). 单位地点: 单位, 技术报告: 报告编号, 年 **\*技术报告\***

[编号] 专利拥有人. 专利名称，专利授权国家，专利授权日期

**\*技术专利\***

website optimization andy king

the datacenter as a computer[7] Heracles

[3]众多未定

[4] Borg

[5]Pard

[6]Heracles 68

[7]tailbench

[8]xapian

[9]silo

[10]iperf

[11]heracles

附录X.

\***附录内容**置于此处，字体为小5号宋体。附录内容包括：**详细的定理证明、公式推导、原始数据**等\*

**Author1**, … …\*计算机学报第1作者提供照片电子图片，尺寸为1寸。英文作者介绍内容包括：出生年,学位(或目前学历),职称,主要研究领域（**与中文作者介绍中的研究方向一致**）.\* \*字体为小5号Times New Roman\*

第一作者

照片

(高清照片)

**AuthorX**, … …\*英文作者介绍内容包括：出生年,学位(或目前学历),职称,主要研究领域（**与中文作者介绍中的研究方向一致**）。\* \*字体为小5号Times New Roman\*

**Background**

\*论文背景介绍为**英文**，字体为小5号Times New Roman体\*

论文后面为400单词左右的英文背景介绍。介绍的内容包括：

本文研究的问题属于哪一个领域的什么问题。该类问题目前国际上解决到什么程度。

本文将问题解决到什么程度。

课题所属的项目。

项目的意义。

本研究群体以往在这个方向上的研究成果。

本文的成果是解决大课题中的哪一部分，如果涉及863\973以及其项目、基金、研究计划，注意这些项目的英文名称应书写正确。

1. The Cooperative Association for Internet Data Analysis(CAIDA),http://www.caida.org/data 2010,7,18 [↑](#footnote-ref-2)