DOI:10.16356/j.1005-2615.2017.06.019

面向多用户环境的弹性云缓存系统研究与实现

陆 悠[1,2] 杜鹏程[1] 吴 帅[2] 吴一娜[2]

(1.苏州科技大学电子与信息工程学院,苏州, 215000;2.东南大学计算机科学与工程学院,南京, 211189)

摘要:由于在改善网络带宽性能、缓解堵塞、提升用户体验等方面的优势,缓存技术在网络应用中被广泛使用,然而当前越来越多的网络应用迁移至云平台,原来针对单一业务和主机环境的缓存管理机制在面对云计算下多用户、多业务混杂环境时暴露出诸如不具备多业务类型的普适性、管理策略与云平台的资源配置脱节、未考虑多用户环境下缓存数据的复杂性、重复性等不足,最终影响缓存效果和资源利用效率,因此研究一种面向多用户环境的弹性云缓存系统,首先分析多用户、多业务的云环境对缓存管理的需求特点,提出基于双层映射的缓存存取模型,然后设计缓存系统框架与功能模块,并研究相应的逻辑、物理资源的配置算法及其他功能的实现方案,最后在校园网环境下实现缓存系统并对其进行验证,分析结果表明本文提出的缓存系统能够有效满足用户需求同时提高资源利用率。

关键词:缓存管理;云计算;双层映射;资源配置

中图分类号:TP311 **文献标志码**:A **文章编号**:1005-2615(2017)06-0883-09

Research and Implementation of Elastic Cloud-Base Buffer Management System for Multi-user Environment

LU You[1,2], *DU Pengcheng*[1], *WU Shuai*[2], *WU Yina*[2]

(1. School of Electronically and Information Engineering, Suzhou University of Science and Technology, Suzhou, 215000, China; 2. School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China)

Abstract: Because of various advantages such as improving the performance of bandwidth, relieving traffic congestion and enhancing user experiences, buffer management systems are widely used in network applications. However, now more and more applications have migrated to the cloud platform, so traditionally buffer management mechanisms expose many shortcomings. These mechanisms cannot deal with different applications of the platform, and their strategies separate from the source management and ignore the requirements of multi-user environment. This paper proposes a novel elastic cloud-base buffer management system for multi-user environment. Firstly, the requirements of multi-application and multi-user clouding environment are analyzed. Second, a two-layer-mapping-based buffer model is proposed to the requirements, and the frame structure function modules are designed. Then the allocation algorithms for logical resources and physical resources are analyzed. Finally, the buffer management system is realized and tested. The experimental results indicate that the mechanism has advantages in ensuring the users′ requirement and improving the resource utilization.

Key words: buffer management; cloud computing; two-layer mapping; resource allocation

---

基金项目:江苏省自然科学基金(BK20151202)资助项目;住建部科研基金(2015-K6-012,2015-K8-035)资助项目。

收稿日期:2016-09-27;**修订日期**:2017-03-20

通信作者:陆悠,男,博士研究生,讲师,E-mail:luyou@seu.edu.cn。

引用格式:陆悠,杜鹏程,吴帅,等.面向多用户环境的弹性云缓存系统研究与实现[J].南京航空航天大学学报,2017,49(6):883-891. LU You, DU Pengcheng, WU Shuai, et al. Research and implementation of elastic cloud-base buffer management system for multi-user environment[J]. Journal of Nanjing University of Aeronautics & Astronautics, 2017,49(6):883-891.

随着网络技术的发展,当前一种新的计算模式云计算[1-3]已越来越深入人们的日常生活。随着大量的应用逐渐迁移至云平台,云数据管理尤其是云缓存管理逐渐引起人们注意,由于改善网络带宽性能、缓解网络堵塞、提升用户体验等优势,缓存系统一直是解决 Internet 共享问题的关键技术之一。然而传统缓存系统大多是针对单一业务并基于主机及其集群构建的,而在云环境下,如何管理大量用户针对其小型、不稳定数据访问,实现云环境下的缓存系统并为云平台下的多种业务提供统一的服务,已成为如今众多网络应用研究人员研究的热门领域[4-10]。

针对大量数据的缓存管理本质上是一个数据共享存储管理问题,在云计算环境下,现有的缓存管理机制大多针对部署至云计算环境的特定业务展开研究,例如针对数据库,现有工作包括能够向不同类型基于数据库的应用提供不同的管理策略,实现自调节的缓存管理机制[5];从缓存资源利用率角度出发提出的缓存优化管理策略[6];针对具有明确 QoS 需求的不同类型请求的缓存分配机制[7]等。也有工作针对 Web 应用展开,研究切入点大多针对缓存的管理、预取的相关算法方面,例如在传统 LFU(Least frequently used)、LRU (Least recently used)、GDS(Greedy dual size)、GDSF (Greedy dual size frequency)等算法基础上,引入机器学习等方法,并根据 Web 应用特点设计的新型缓存算法[8-10]等。除了针对特定应用的缓存管理机制以外,也有研究人员从云平台的底层存储角度出发展开研究,研究成果主要有块级别的缓存存储系统或模块,例如亚马逊公司的 AWS,加州大学的 Eucalyptus,由 NASA 和 Rackspace 合作研发 OpenStack ,以及印第安纳大学的分布式块级别存储系统 VBS(Virtual block store)等[11]。

然而纵观这些缓存系统方面的研究成果,可以发现其仍然存在一些问题:

(1)针对单一业务。现有缓存系统管理策略的出发点都是特定业务(数据库检索、读写或者 Web 页面的预读取、替换等)的应用场景,而在云计算平台上部署的业务往往是多类型混杂的,将这些单一的缓存管理机制扩展至云计算平台向所有业务提供服务势必存在适用性、扩展性和伸缩性等方面的问题。

(2)未考虑多用户环境的需求。由于云计算的虚拟化特点,存在不同用户、不同应用的用户数据大量重复的特点,现有缓存机制过于针对单一应用,其背景缺乏云平台下多业务、多用户的特点,因此不能充分考虑多用户环境特点以实现针对性的缓存管理策略,从而提高资源效率。

(3)应用层面的缓存管理与底层资源配置脱节。现有的针对特定业务的缓存管理机制大多基于主机集群,未考虑云计算平台底层资源虚拟化特点,而已有从云平台底层角度出发的缓存管理机制更多注重资源本身的调整、扩展性,缺乏与应用层面缓存策略的配合,在用户申请、释放资源非常频繁的云计算环境下,这种脱节现象往往导致缓存管理机制效率低下且无法尽可能的满足用户需求。

针对以上不足,本文提出一种面向多用户环境的弹性云缓存系统,其主要思路为:在充分分析当前云计算平台和多用户环境特性的基础上,首先对现有缓存系统所对应的业务环境和数据读写进行抽象,提出基于双层映射的缓存存取模型,从而为多用户环境下不同用户和应用环境下提高资源以及数据存取的效率提供基础;然后在该模型基础上设计缓存系统框架和功能模块划分,实现用户接口、应用层面的缓存读写和底层资源配置的解耦合,增强缓存系统的灵活性和可扩展性;接着分析现有的云计算平台技术及工具,研究缓存系统的实现方案,包括设计实现面向逻辑资源和物理机资源的配置算法,以及透明代理的实现路线等;最后在依托苏州科技学院云计算平台,实现缓存系统方案并进行实验分析,结果表明,相对传统缓存管理系统,本文设计的缓存系统能够有效地应对用户复杂的缓存需求同时提高资源利用效率。

1 问题分析与建模

1.1 多用户环境下的缓存系统的特性及现有工作分析

在云计算环境下,缓存系统的典型应用场景包括:页面缓存(用来缓存 Web 内容,如 HTML、CSS 和图片等);应用对象缓存(作为 ORM 等应用框架的二级缓存对外提供服务,可以减轻数据库的负载压力及加速应用访问等); 状态缓存(包括 Session 会话状态及应用横向扩展时的状态数据等)等,在考虑多用户环境下对这些场景中的缓存使用过程进行抽象,可以给出缓存系统处理对象以及系统环境和适用条件等因素的讨论。

定义 1 缓存信息结构:缓存系统所处理的缓存信息本质上可视为一种 key-value 二元结构:用户存入缓存系统的信息可统一为关键字-值模式(例如 URL-对应页面数据,JNDI 名称-数据库连接对象等),而用户从缓存系统中提取信息则通过关键字来查找,这种内容上的二元结构是设计云缓存系统时首先需要注意的特点,由于不同用户存入

的信息在 value 部分可能是重复的（即 key 不同，但 value 相同，例如不同的 JNDI 名称指向相应对象等），显然从物理资源层面来看，这种重复的数据只需保留一个实例以节约资源。

接下来给出缓存系统的环境特点以及适用条件：云计算环境下的缓存系统存在多用户访问的重复性特点，许多文献都指出，由于存在着大量用户，在应用环境下会出现对大量小型数据有着不稳定、重复的数据访问[12]。许多研究都指出，如果能将不同应用环境、多个用户的重复数据被合并为一个缓存单位，通过“单实例多租赁”的方式为不同用户提供相应的缓存服务，则能够通过减少资源成本获取规模经济效益[13]。因此，本文所讨论的缓存系统适用于面向较大规模的用户、频繁进行较多数据读写和查询操作的场景，例如分布式的 Web 应用、数据库操作等。

由此，本文主要针对缓存信息结构、缓存系统环境特性以及适用条件，研究相应的面向多用户环境的缓存系统，由于缓存系统本质上是一种数据存储方面的共享。而目前云计算环境下的数据共享模式主要包括共享主机的独立数据库架构，共享数据库－独立数据架构和共享数据库－共享数据架构 3 种由低到高的共享级别。考虑到共享数据库－共享数据架构能够支撑的用户数据最多，资源消耗成本最小，十分适合 SaaS 模式下利用长尾效应来获取规模经济效益的设计理念，因此本文采用基于共享数据库、共享数据架构构建缓存系统的思路，但面对多用户环境特点，现有基于共享数据库与数据架构已有工作[8-11]还有以下不足：

（1）存储接口不一致，现有方案大多针对特定应用（如 Web 应用[8-9]、分布式存储环境[10-11]等），目前面对多种应用类型提供较为通用和一致的存储接口和功能的工作较少。

（2）在数据存取模型方面，目前的方案较少针对多用户环境下“单实例多租赁”模式进行特别设计。

（3）存储方案在资源利用方面更偏重于容量的可扩展性，并不关注资源本身的利用效率，没有针对用户数量自身的频繁增减情况进行优化。

针对现有工作的不足，本文在分析云计算环境下缓存系统的特性基础上，提出构建一种具备通用一致接口、基于“单实例多租赁”模式并充分考虑资源效率的的弹性云缓存系统，从而为云计算环境下的各类应用提供高效的支撑。

1.2　基于双层映射的云缓存存取模型设计

为实现“单实例多租赁”的存取模式，考虑到缓存信息的 key-value 二元结构，本文提出一种使用双层映射的存取模型。该模型主要基于哈希表技术，然后引入映射机制来打破用户信息的二元结构，基于双层映射的缓存存取模型如图 1 所示。

在图 1 中，缓存系统使用两层逻辑空间（用户逻辑空间、全局逻辑空间）和一层物理空间来实现对用户信息的存储。系统在用户逻辑空间中为每个用户配置缓存表，用于保存每个用户缓存的信息 key 值和对应全局逻辑空间地址哈希值，并记录用户 ID、存储空间大小等初始化信息，在全局逻辑空间中则保存所有用户缓存信息对应的物理存储地址，而物理空间则用于存放用户缓存的实际信息。在具备双层映射的读取模型基础上，接下来给出缓存系统管理的具体机制与操作过程。

（1）写操作：用户通过统一的用户接口（透明代理）向该缓存空间存入数据时，该信息被格式化为 key-value 形式，缓存系统对 value 数据进行哈希计算得到唯一的、该信息的全局逻辑空间地址哈希值（这方面已有很多映射算法可用[14]）。然后在用户缓存表中写入键及全局逻辑空间地址，然后在全局逻辑空间对应地址处写入该信息实际的物理存储地址。当不同用户、不同应用环境下存储同样的数据（即值相同），即使 key 互不相同，value 对应的全局逻辑空间地址必然相同，对应物理存储地址也必然相同，从而实现相同的 value 只需保留一个实例，避免重复存储对资源的浪费。

（2）读操作：用户通过统一的用户接口（透明代理）向缓存空间读取信息时则根据 key 进行获取。

（3）更新操作：用户通过统一的用户接口（透明代理）向缓存空间更新信息时，设更新的数据可格式化为 key-value(new) 形式，而原有数据则为 key-value(old) 形式，因此更新过程中，系统查看 value(old) 地址是否有其他用户的 key 值所指向，如果没有其他用户指向，那么可以直接在 value(old) 的物理地址上对数据进行更新，使其值为 value(new)，如果有其他用户指向，那么将 key-value(new) 视为一次新的写操作同时，保留 value(old) 的物理内容以及其上的用户指向不变。

（4）删除操作：用户通过统一的用户接口（透明代理）向缓存空间删除信息时，设删除的数据为 key-value(old) 形式，在删除过程中，系统查看 value(old) 地址是否有其他用户的 key 值所指向，如果没有其他用户指向，那么可以直接删除在 key-value(old)，否则只需删除提出请求的用户的 key 即可。

（5）除此以外，缓存系统还使用写操作互斥锁机制，以进一步确保缓存系统中数据的一致性。

管理机制除了上述各项操作以外，还需要存取模型中还有相应模块负责用户存储空间使用情况的监控、逻辑存储位置与物理存储位置的映射与维护，以及全局上对多个用户重复存储到相同逻辑位置数据的引用维护等功能。

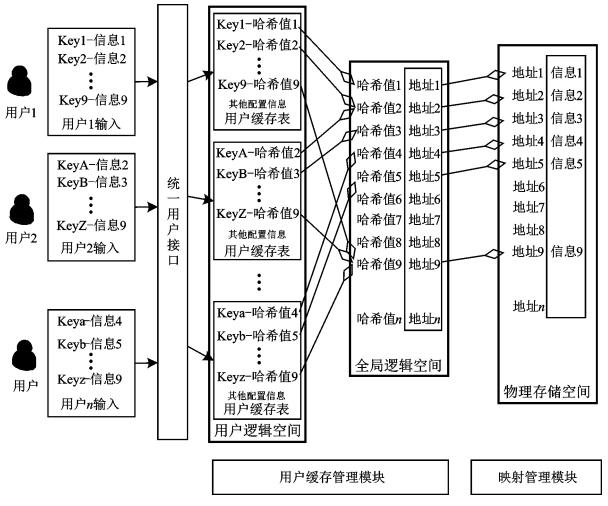

图 1 基于双层映射的缓存存取模型

Fig. 1 Catching model based on two-layer-mapping

由此可见，使用基于双层映射的缓存存取模型之后，用户实际使用的物理缓存空间可能是与其他用户“共用”的，从而实现“单实例多租赁”模式，充分提高资源效率，而各项具体操作则能够保证数据的一致性、完整性等方面的要求。

1.3 相关定义与说明

为了更好地描述弹性云缓存系统的设计与实现，本文进一步给出以下系统相关的一些定义说明。

定义 2 客户端 Client：是用户使用缓存系统接口，部署于用户端，负责用户与缓存系统的通信交互，包括用户 ID 维护、安全配置等内容；代理层面向用户提供缓存的读写功能。

定义 3 代理 Proxy：是云平台中的代理进程，每个 Proxy 负责相应的客户端，是用户接入缓存系统的一个统一接口，Proxy 实现对用户逻辑空间和全局逻辑空间的各项操作，根据用户缓存信息的请求在全局上、逻辑层面进行管理决策并存取。

定义 4 Docker：是部署于实际的物理机之上的逻辑资源点，所有 Docker 以分布式方式承载存取模型中的用户逻辑空间、全局逻辑空间和物理存储空间，能够将代理层在逻辑层面的管理决策映射至物理层，并负责物理资源的管理、配置等。

2 弹性云缓存系统设计

根据多用户环境下缓存存取问题的特性分析，可以制定自适应弹性云缓存的系统架构设计，主要思路是对系统的功能进行分层，从而实现用户接口、应用层面的缓存读写和底层资源配置的解耦合。本文采用 3 层架构来实现缓存系统，具体内容如图 2 所示。

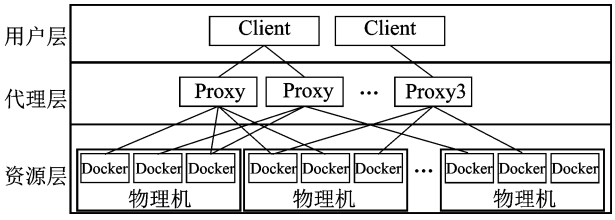

图 2 系统架构设计

Fig. 2 Architecture design of catching system

缓存系统从用户到实际物理资源可以在逻辑上分为 3 个层次，用户层由各 Client 构成，部署于云计算平台，代理层则由多个 Proxy 代理进程构成，每个 Proxy 负责相应的客户端，资源层则由多个物理机上分布式部署的 Docker 组成。

接下来给出缓存系统的功能模块图，主要内容如图 3 所示，各功能模块介绍如下。

(1) Client 模块：提供用户使用缓存系统的操作接口。

(2) 用户管理模块：负责对用户的统一管理，包括 ID 维护、身份认证、秘钥管理等功能。

(3) 缓存系统功能入口：用户调用系统功能的操作接口，包括对缓存空间的请求与释放、数据读写等。

(4) 用户缓存管理模块：负责对 docker 的管理，主要是缓存存取模型中用户逻辑空间和全局逻辑空间的管理，包括用户缓存表的配置、哈希值运算、多用户引用管理等。

(5) 映射管理模块：负责全局逻辑空间到物理存储空间的映射关系的配置、管理和计算等相关功能。

(6) 资源管理模块：负责对物理设备的管理，以及物理存储空间的数据读写、配置和管理等功能。

(7) 设备访问接口：向涉及物理设备的数据读写、设备配置等方面的策略实施提供具体设备的操作接口。

3 系统实现相关算法

在云缓存系统的具体实现过程中，如何对资源进行配置(包括逻辑、物理资源)以及缓存存取的实现最为关键，而资源配置方面，根据缓存系统的框架以及缓存存取模型的设计思路，其本身还分为逻辑资源配置和物理资源的配置两个层次，本文就这 3 方面分别展开说明。

3.1 逻辑资源配置算法

首先介绍逻辑资源层面上的 Docker 配置算

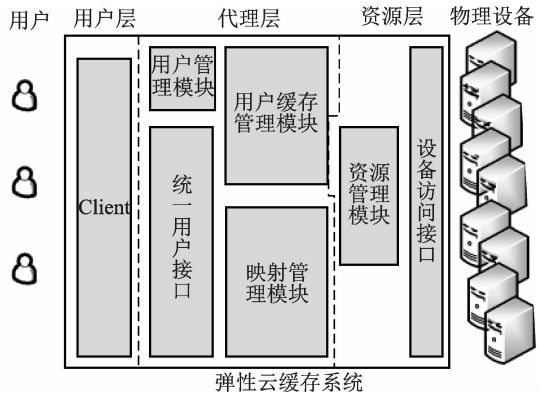

图 3　缓存系统功能模块图

Fig. 3　Function module diagram of catching system

法，配置思路为：缓存系统根据用户的需求（即缓存大小）申请，以 Docker 为单位向其分配符合需求的逻辑资源，并在其上开辟一定单位用于设置用户逻辑空间与用户缓存表，供用户在实际工作中独立使用，除此以外，则采取将所有用户剩余空间集群的方式，共同组成全局逻辑空间，向所有用户提供服务。为了后续物理资源映射的方便，本文对 Docker 的存储空间按大小进行分类，设 Docker 具有 A 类（本文设 $A=4$），其配置为 $A\{s_1,s_2,\cdots,s_A\}$，其中 $s_1,s_2,\cdots,s_A$ 为从小到大排列的存储空间大小值（本文设配置为 $A\{100\ \text{MB},200\ \text{MB},400\ \text{MB},800\ \text{MB}\}$），然后为了确保物理机的负载均衡，设负载界限值 U（本文设 $U=1$ GB），每个界限值 U 范围内的缓存请求都将建立至少 2 份 Docker 以确保使用 2 台物理机分担承载，即缓存请求 Ask 时，将请求分为 2 份分别建立 Docker，当缓存请求 Ask 时则将请求分为 4 份分别建立 Docker，以此类推；最后对请求划分之后构建 Docker 时则使用背包算法，构建最少数量的 Docker，每个 Docker 的空间大小单位 $\text{Docker}_n A\{s_1,s_2,\cdots,s_A\}$，且使得 Docker 大于等于请求值。根据以上思路，给出 Docker 配置算法如下。

算法 1　Docker 配置算法

输入　用户缓存请求 Ask，Docker 存储配置参数 $A\{s_1,s_2,\cdots,s_A\}$，负载界限值 U

输出　$\text{Docker}_{\text{list}}=\{\text{Docker}_1,\ \text{Docker}_2,\cdots,\ \text{Docker}_n\}$

判断 Ask 属于，(0，U)，(0，U)，…的区间，确定将 Ask 分为 X 份

对每一份 Ask/X

使用背包算法，求得 Docker_{X1}，Docker_{X2}，…满足以下条件：

(a) $(\text{Docker}_{X1},\text{Docker}_{X2},\cdots)=\text{Ask}/X$

(b) Docker 数量最小

将 Docker_{X1}，Docker_{X2}，…，加入 $\text{Docker}_{\text{list}}$

若 $\text{Docker}_{\text{list}}$ 不为空，返回 $\text{Docker}_{\text{list}}$

用户初次提交缓存请求之后，还会有增加缓存、删除缓存的请求，对这两种请求的处理方法如下。

(1) 增加缓存：根据用户提交的增加缓存请求，同样采用算法 1 处理，修改用户逻辑空间和缓存配置表之后，然后将得到的新 Docker 与已知 Docker 相结合，组成新的缓存集群。

(2) 删除缓存：根据用户提交的删除缓存请求，在现有 Docker 集群中找出该用户存储空间使用率最低的 Docker 对其中用户的逻辑内容进行删除，同时分析信息存储的引用情况进行用户逻辑空间和缓存配置表的修改，进而在确定仅当只有当前用户引用该信息的前提下进一步进行物理存储空间的操作，从而确保命中率不会大幅下降。（注：提交的删除大小为现有集群各 Docker 结点最大内存的线性组合）。

算法分析：该算法仍是基于贪婪法进行设计，其时间复杂度为 $O(n^2)$，但其仅由用户提出调用，考虑到用户在正常使用时较少频繁调整其缓存大小，因此该开销在可以接受范围之内。

3.2　物理机资源配置算法

算法的主要思路是：根据 Docker 集群配置信息，首先对各个 Docker 容量进行排序；然后按物理机剩余内存从小到达进行排序；按顺序将 Docker 列表中的成员与物理机列表中的成员进行顺序匹配，若 Docker 缓存配置大小小于等于物理机剩余容量，则将 Docker 配置到该物理机中。若均物理机队列中所有成员都不满足 Docker 需求，则在新的物理机中生成 Docker，并将新物理机加入物理机队列中。确保每个缓存集群中的 Docker 均分布在不同的物理机中，根据这一策略，即使有 Docker 节点失效，仍能确保其他 Docker 节点继续工作，而不影响集群的使用情况。同时保证物理机中的剩余内存容量尽可能的少。根据以上思路，给出 Docker 配置算法如下：

算法 2　物理资源配置算法

输入　$\text{Docker}_{\text{list}}=\{\text{Docker}_1,\ \text{Docker}_2,\cdots,\ \text{Docker}_n\}$，已部署 Docker 的物理机列表 $C_{\text{list}}=\{C_1,C_2,C_3,\cdots\}$，未部署 Docker 的物理机列表 $CN_{\text{list}}=\{Cn_1,Cn_2,\cdots\}$

输出　分配方案 $\text{Result}=\{(\text{Docker}_1,c_1),(\text{Docker}_2,c_2),\cdots\}$

对 $\text{Docker}_{\text{list}}$ 按容量从小到大排序，得结果 $\text{Docker}_{\text{list}'}=\{\text{Docker}_{1'},\ \text{Docker}_{2'},\cdots,\ \text{Dock-}$

$er_{n'}\}$

对 $C_{list}=\{C_1,C_2,C_3,\cdots\}$ 按剩余内存空间从小到大排序，得结果 $C_{list'}=\{C_{1'},C_{2'},C_{3'},\cdots\}$

依次取出 $Docker_{list'}$ 的各个成员 $Docker_n$

依次取出 $C_{list'}$ 的各成员，取第一个能满足 $Docker_n$ 的 C_m，将 $(C_m,Docker_n)$ 加入 result

更新 C_m 的剩余容量，重新排序 $C_{list'}$

若找不到能满足 $Docker_n$ 的 C_m，取 $CN_{list}=\{Cn_1,Cn_2,\cdots\}$ 得第一个成员 $C_{m'}$，

将 $(C_{m'},Docker_n)$ 加入 Result，

更新 $C_{m'}$ 剩余容量，将其加入 $C_{list'}$，重新排序 $C_{list'}$

返回 result

该算法仍是基于贪婪法进行设计，其时间复杂度为 $O(n^2)$，但由于资源配置仅当初始化以及物理内存耗尽时才调用，因此其开销在可以接受范围之内。

3.3 缓存存取算法

算法的主要思路是：根据用户的操作请求，如果是存入，则将其格式化为 key-value 形式，然后使用文献[14]中的哈希算法对 value 进行哈希，求得全局逻辑空间的存储地址，将 key-地址存入用户缓存表，查看全局逻辑空间地址是否已有物理实例，如有则不做处理，否则将 value 生成存储实例存入物理机，并在全局逻辑空间地址上写入物理实例存放地址。如果是读取信息请求，则根据 key 值查询用户缓存表和全局逻辑空间，找到 value 的物理实例地址并返回值。根据以上思路，给出缓存存取算法如下。

算法 3 缓存存取算法

输入：用户请求(key-value)

输出：存取结果

判断请求类型

如果是写

对 value 进行 hash 计算，得 $L_{address}=$ Hash(value)，将 key-$L_{address}$ 存入用户缓存表

查询全局逻辑空间 $L_{address}$ 地址，如果已有数据，goto(10)

生成 value 实例，存入物理机地址 $P_{address}$，将 $P_{address}$ 写入全局逻辑空间 $L_{address}$ 处，goto (13)

如果是更新

查看原 value 引用情况，若仅有当前用户的引用，则将新 value 实例存入物理机地址 $P_{address}$

若有其他用户引用，则将用户请求改为写请求，goto(2)

如果是读取

从用户缓存表中查询 key，得其 $L_{address}$，从全局逻辑空间查询 $L_{address}$，得物理机地址 $P_{address}$

从物理机地址 $P_{address}$ 处读取 value，goto (12)

返回 value

返回 success

由于缓存的存储主要通过 hash 计算双层映射的关系，因此无论是缓存的读、写还是更新，其时间和空间复杂程度都为 $O(n)$，因此算法的效率可以得到保证。

4 实验与分析

为了验证缓存系统设计方案的可行性、合理性，本文依托苏州科技学院的云计算集群，对缓存系统进行实现和验证，该集群配置为：刀片机 64 台，包括 512 核 CPU，1 024 GB 内存，400 余 TB 的存储量，在该集群中实现缓存系统，系统界面如图 4 所示。

为了验证缓存系统的性能，本文设计如下实验环境，首先设计两种业务类型(Web 应用与数据库)混杂的应用背景，其中 Web 应用基于 Apache Tomcat 的集群，数据库应用则基于 MySql 开发部署，两种业务分别设置一定数量的用户；然后使用云集群中的其他核心运行客户端，模拟用户基于泊松分布随机同时访问 Web 网站或数据库的页面或信息，其中 Web 应用大致是访问 1 000～5 000 个大小为 10～100 KB 页面文件，数据库应用则是访问 500～1 000 个大小为 500 KB～1 MB 的数据；接下来实现缓存系统，设定系统根据用户的到达和离开自动进行缓存的配置，每位用户分配 100 MB 缓存，限定缓存系统总容量为 25 GB；最后通过比较不同用户规模下的资源利用率、承载缓存系统的设备负载以及用户的读取速度，其中为了防止网络带宽影响读取速度实验结果，本文使用了云计算平台内的光纤网络，以确保所有用户的网络速度。

本文选择的对照方法则不采用本文缓存系统，其中对照方法 1 主要使用 Tomcat 所配套的缓存管理策略，所有用户共享全部内存，根据用户访问页面请求到达的先后各自分配；对照方法 2 则使用分布式存储数据库的缓存机制，针对每个用户的数据需求，分配其请求大小的缓存，实验中按算术平

图 4　缓存系统操作界面示意图

Fig. 4　Diagram of catching system's operating interface

均原则进行分配；对照方法 3 则使用文献[8]中基于机器学习的缓存分配方法，根据用户过往的使用历史来进行缓存的预分配；对照方法 4 则使用文献[10]中方法，从负载均衡角度出发对用户缓存进行分配和调整。验证目标则包括应用所在虚拟机的缓存利用率、相应设备的负载以及缓存的命中率。

实验 1：验证缓存资源的利用率，在设置 50，100 和 300 不同用户数量的情况下，运行实验过程 10 min 后计算平均值，考察缓存系统和所使用的内存使用率，其结果如图 5 所示。从实验 1 的结果中可以看出，本文缓存系统充分利用了“单实例多租赁”模式，能有效提高资源的利用效率，尤其是随着用户数量的增多，甚至在每个用户分配的逻辑容量综合已经超过物理内存容量的情况下，其利用率仍可保持在 92%左右，而对照方法中，常见的 Web 服务器和数据库应用(对照方法 1、2)的性能较差，尤其随着用户的增多，其缓存已经不敷使用，对照方法 3 虽然能够根据用户行为历史对缓存分配进行优化，但对不同用户的相同访问请求并无针对性措施，因此利用率随着用户增多也有较高增长，而对照方法 4 更多考虑缓存服务器的负载均衡，其利用率也有较大幅度上涨，虽然总体表现比对照方法 1 和 2 为强，但相对本文方法，在用户数量较多的情况下仍出现缓存不足的现象。

实验 2：验证业务服务器的负载，在设置 50，100 和 300 不同用户数量的情况下，运行实验过程 10 min 后计算平均值，考察业务服务器所在虚拟机的 CPU 占用率，其结果如图 6 所示。从实验 2 的结果中可以看出，本文缓存系统在确保用户缓存数据的基础上，同样可以有效降低应用服务器的负载，尤其是随着用户数量的增多，其负载仍可保持在 80%左右，而对照方法 1 和 2 随着用户的增多，

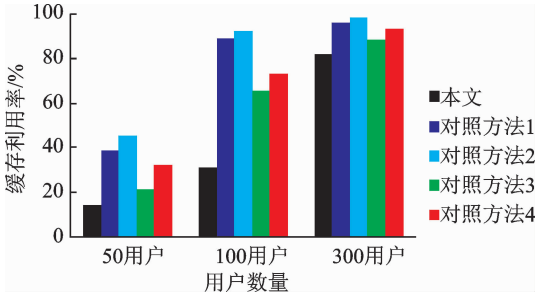

图 5　实验 1 结果示意图

Fig. 5　Result of experiment 1

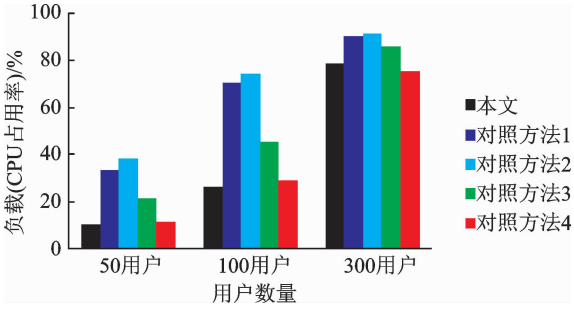

图 6　实验 2 结果示意图

Fig. 6　Result of experiment 2

CPU 的负载已经接近极值。对照方法 3 仅根据用户行为历史对缓存分配进行优化而不考虑不同用户的相同访问请求，因此总体负载也有较高增长，而对照方法 4 主要考虑缓存服务器的负载均衡，其 CPU 利用率有优势，总体表现比对照方法 1、2、3 为强，但由于本文方法充分考虑了用户重复请求的情况，在用户数量较大的时候，对照方法 4 虽然具备一定优势，但本文方法的负载结果与其较为接近。

实验 3：验证用户访问业务的 I/O 速度，在设置 50，100 和 300 不同用户数量的情况下，运行实验过程 10 min 后计算平均值，考察用户读取数据时的 I/O 速度，其结果如图 7 所示。从实验 3 的结

果中可以看出,本文缓存系统支持下,用户的读取信息的 I/O 速度能够得到较好的提高和维持,无论用户数量是否增长,其速度基本能维持在 70 KB/s左右,而对照方法 1、2 随着用户的增多,读取速度会有较大幅度的下降。对照方法 3 根据用户行为历史对缓存分配进行优化,在用户数量较少的情况下读取速率最高,在用户数量增多的情况下,由于其未考虑用户的重复访问,因此读取速率有所下降,而对照方法 4 主要考虑缓存服务器的负载均衡,其读取速率相对较高。

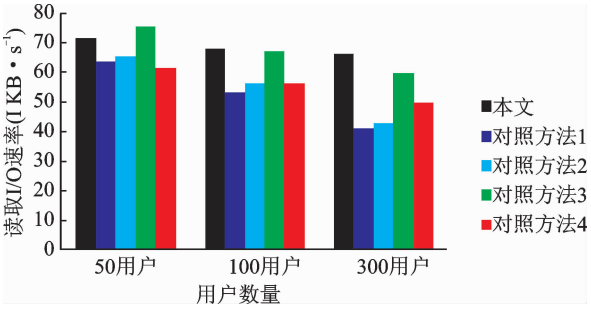

图 7 实验 3 结果示意图

Fig. 7 Result of experiment 3

实验 4:验证用户访问缓存的命中率,在设置 50,100 和 300 不同用户数量的情况下,运行实验过程 10 min 后计算平均值,考察用户读取数据时的缓存命中率,其结果如图 8 所示。从实验 4 的结果中可以看出,本文缓存系统与对照方法相比,由于考虑多用户的重复访问,实际上相当于增加了可用缓存数量,因此命中程度相比对照方法有一定优势,且随用户的增多优势较为明显,基本能保持在 60%以上。对照方法中,方法 3 由于考虑到用户的访问模式,其命中率在用户数量较少时也有优势,但随着用户数量上升其命中率也有下降,其他对照方法则命中率较本文方法为低。

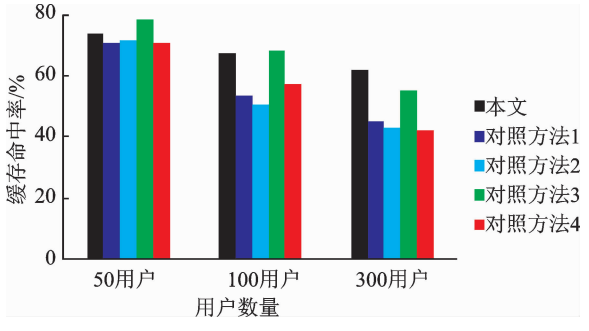

图 8 实验 4 结果示意图

Fig. 8 Result of experiment 4

从上述实验结果可以看出,本文缓存系统在提高资源效率、访问命中率等方面较对照方法有一定优势。

5 结束语

随着云计算技术的飞速发展,越来越多的网络应用已经迁移至云平台,相应的,原来适用于单一业务、单一主机环境的缓存管理机制已经不能适应如今云环境下多用户、多业务混杂的局面,本文针对现有缓存管理方法云平台下存在的在对多业务应用的普适性不足;对多用户环境特定需求的针对性不强;以及上层缓存管理与底层资源配置脱节,导致资源利用效率较差等缺点,提出一种面向多用户的弹性云缓存系统,首先分析多用户、多业务的云环境对缓存管理的需求特点,设计基于双层映射的缓存存取模型,然后研究缓存系统的总体框架与功能模块,在其基础上设计逻辑和物理资源的配置算法以及其他功能模块的实现路线,最后在校园网环境下实现本文的设计方案并对缓存系统进行验证,分析结果表明本文提出的缓存系统能够有效满足用户需求同时提高资源利用率。下一步工作将进一步研究缓存替换策略、调度方法以及增删方面的优化策略等。

参考文献

[1] MASTELIC T, OLEKSIAK A, CLAUSSEN H, et al. Cloud computing: Survey on energy efficiency [J]. ACM Computing Surveys (CSUR), 2015, 47 (2): 33.

[2] CHOU D C. Cloud computing: A value creation model[J]. Computer Standards & Interfaces, 2015, 38: 72-77.

[3] 张洁, 何利文, 黄斐一, 等. 一种应用于云计算环境下的服务发现架构[J]. 南京航空航天大学学报, 2013, 45(4): 556-562.
ZHANG Jie, HE Liwen, HUANG Feiyi, et al. Service discovery architecture applied in cloud computing environments[J]. Journal of Nanjing University of Aeronautics & Astronautics, 2013, 45(4): 556-562.

[4] PSARAS I, CHAI W K, PAVLOU G. In-network cache management and resource allocation for information-centric networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(11): 2920-2931.

[5] WANG J, BALAZINSKA M. Toward elastic memory management for cloud data analytics[C]//Proceedings of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond. [S. l.]: ACM, 2016: 7.

[6] PARKES G, SWASEY J H, UNDERWOOD F M, et al. The effects of catch share management on MSC certification scores[J]. Fisheries Research, 2016,

182:18-27.

[7] STEFANOVICI I, THERESKA E, O'SHEA G, et al. Software-defined caching: Managing caches in multi-tenant data centers[C]//Proceedings of the Sixth ACM Symposium on Cloud Computing. [S. l.] ACM, 2015: 174-181.

[8] ALI W, SHAMSUDDIN S M, ISMAIL A S. Intelligent web proxy caching approaches based on machine learning techniques[J]. Decision Support Systems, 2012, 53(3): 565-579.

[9] ALI W, SHAMSUDDIN S M, ISMAIL A S. Intelligent Naive bayes-based approaches for Web proxy caching[J]. Knowledge-Based Systems, 2012, 31: 162-175.

[10] ZHANG G, LI Y, LIN T. Caching in information centric networking: A survey[J]. Computer Networks, 2013, 57(16): 3128-3141.

[11] VASHIST S, GUPTA A. A review on distributed file system and its applications[J]. International Journal of Advanced Research in Computer Science, 2014, 5(7):235-237.

[12] YANG F, NIE S. The application of improved quantum self-organizing neural network model in web users access mode mining[M]. Communications and Information Processing: Springer Berlin Heidelberg, 2012: 385-393.

[13] JUMAGALIYEV A, Whittle J. Model-driven engineering for multi-tenant SaaS application development [C] //Proceedings of the 3rd Workshop on Cross-Cloud Infrastructures & Platforms. [S. l.]: ACM, 2016: 8.

[14] TAO H, YATING W, BINGYAO C, et al. A dynamic data allocation method with improved load-balancing for cloud storage system[C]//Smart and Sustainable City 2013 (ICSSC 2013), IET International Conference on. [S. l.]: IET, 2013: 220-225.