软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(1):50−66 [doi: 10.3724/SP.J.1001.2013.04276]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

云计算环境下分布式缓存技术的现状与挑战*

秦秀磊[1,2,3], 张文博[1], 魏 峻[1,2], 王 伟[1], 钟 华[1], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100190)

[2](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[3](中国科学院 研究生院,北京 100049)

通讯作者: 秦秀磊, E-mail: qinxiulei08@otcaix.iscas.ac.cn

摘 要: 作为云平台提升应用性能的一种重要手段,分布式缓存技术近年来受到了工业界和学术界的广泛关注.从云计算与分布式缓存技术的结合入手,分析介绍了分布式缓存的特性、典型应用场景、发展阶段、相关标准规范以及推动缓存技术发展的若干关键要素.为系统地了解分布式缓存技术的现状和不足,建立了一个云环境下分布式缓存技术的分析框架——DctAF.该框架从分析云计算的特点和缓存技术的边界出发,涵盖 6 个分析维度.基于DctAF 框架,对当前缓存技术进行总结和分析,并对典型系统进行比较.在此基础上,深入阐述了云环境下分布式缓存系统面临的挑战;围绕上述挑战,分析和比较了已有的研究工作.

关键词: 分析框架;云计算;分布式缓存

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 秦秀磊,张文博,魏峻,王伟,钟华,黄涛.云计算环境下分布式缓存技术的现状与挑战.软件学报,2013,24(1):50−66. http://www.jos.org.cn/1000-9825/4276.htm

英文引用格式: Qin XL, Zhang WB, Wei J, Wang W, Zhong H, Huang T. Progress and challenges of distributed caching techniques in cloud computing. Ruanjian Xuebao/Journal of Software, 2013,24(1):50−66 (in Chinese). http://www.jos.org.cn/1000-9825/4276.htm

Progress and Challenges of Distributed Caching Techniques in Cloud Computing

QIN Xiu-Lei[1,2,3], ZHANG Wen-Bo[1], WEI Jun[1,2], WANG Wei[1], ZHONG Hua[1], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

Corresponding author: QIN Xiu-Lei, E-mail: qinxiulei08@otcaix.iscas.ac.cn

Abstract: As an important application of acceleration in the cloud, the distributed caching technology has received considerable attention in industry and academia. This paper starts with a discussion on the combination of cloud computing and distributed caching technology, giving an analysis of its characteristics, typical application scenarios, stages of development, standards, and several key elements, which have promoted its development. In order to systematically know the state of art progress and weak points of the distributed caching technology, the paper builds a multi-dimensional framework, DctAF. This framework is constituted of 6 dimensions through analyzing the characteristics of cloud computing and boundary of the caching techniques. Based on DctAF, current techniques have been analyzed and summarized; comparisons among several influential products have also been made. Finally, the paper describes and highlights the several challenges that the cache system faces and examines the current research through in-depth analysis and comparison.

* 基金项目: 国家重点基础研究发展计划(973)(2009CB320704); 国家自然科学基金(61173003, 61100068); 国家高技术研究发展计划(863)(2012AA011204); 国家科技支撑计划(2011BAH15B03)

收稿时间: 2011-11-01; 修改时间: 2012-02-15; 定稿时间: 2012-06-25; jos 在线出版时间: 2012-07-27

CNKI 网络优先出版: 2012-07-27 10:59, http://www.cnki.net/kcms/detail/11.2560.TP.20120727.1059.002.html

Key words: analysis framework; cloud computing; distributed cache

云计算描述了一种新的基于互联网的 IT 服务增值、使用和交付模式[1],是数据共享与服务共享计算模式的结合体[2].云计算环境下,为了应对海量数据与用户请求带来的挑战,解决传统数据库面临的大规模数据访问瓶颈问题,分布式缓存技术得以引入,为用户提供高性能、高可用、可伸缩的数据缓存服务.分布式缓存将数据分布到多个缓存服务节点,在内存中管理数据,对外提供统一的访问接口,基于冗余备份机制实现高可用支持,又被称为内存数据网格(in-memory data grid).分布式缓存拉近了对象与应用间的距离,是云平台提升应用性能的一种重要手段.Searchsoa 认为,对于数据密集型的 Web 应用,如果失去分布式缓存这一关键技术的支撑,云的潜能将是十分有限的[3].Forrester 2010 年技术报告[4]中提出了弹性缓存平台(elastic caching platform)的概念,强调弹性缓存的关键特性是动态扩展性与高可用性.动态扩展性表达了缓存平台可提供透明的服务扩展的能力,在线为云应用增加或者减少缓存资源,以适应负载的动态变化.高可用性则表达了缓存平台可以容忍节点失效,任意单一节点失效均不会导致数据丢失或服务终止.图 1 给出了一个典型的分布式缓存系统.

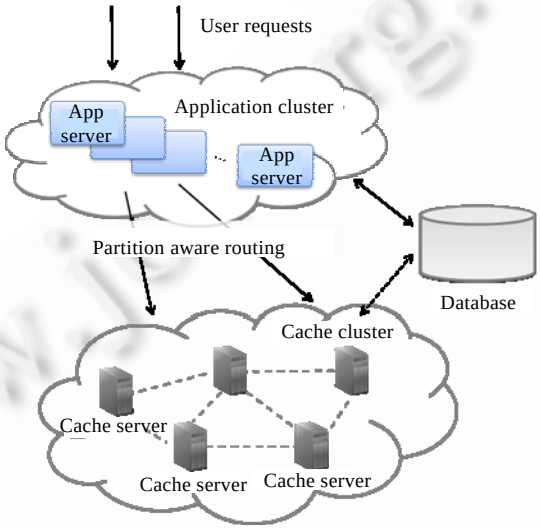


Fig.1 Distributed cache system

图 1 分布式缓存系统

现有的分布式缓存产品[4]主要包括 Oracle Coherence,IBM WebSphere eXtreme Scale(WXS),GigaSpaces eXtreme Application Platform(XAP),Terracotta Server Array,Ncache,VMware GemFire,Red Hat Infinispan 等.2009 年,Oracle 将 Coherence 产品集成到其企业私有云解决方案中[5].2010 年,云虚拟化解决方案提供商 VMware 宣布收购 GemStone,进入分布式缓存市场.同年 7 月,微软的弹性缓存服务产品 AppFabric Caching 发布,并被整合到其云平台 Windows Azure 中.2011 年 8 月,亚马逊推出了其弹性云缓存服务 Amazon ElastiCache 的 Beta 版[6].此外,IBM WebSphere eXtreme Scale 与 Couchbase 均已部署在 Amazon 的弹性计算云 EC2 中,可为用户提供个性化、可定制的缓存服务.

本文第 1 节介绍分布式缓存,包括分布式缓存的特性、典型应用场景、发展历程、相关的标准规范以及与 NoSQL 和极限事务处理间的关系.第 2 节阐述云环境下分布式缓存技术的分析框架 DctAF.第 3 节简要介绍几个典型的分布式缓存产品;同时,基于 DctAF 框架,对典型系统进行分析与比较.第 4 节阐述云计算环境下分布式缓存面临的新挑战.第 5 节对本文工作加以总结,并给出分析与展望.

1 分布式缓存概述

1.1 分布式缓存的特性

分布式缓存具有如下特性:

1) 高性能:当传统数据库面临大规模数据访问时,磁盘 I/O 往往成为性能瓶颈,从而导致过高的响应延迟.分布式缓存将高速内存作为数据对象的存储介质,数据以 key/value 形式存储,理想情况下可以获得 DRAM 级的读写性能;
2) 动态扩展性:支持弹性扩展,通过动态增加或减少节点应对变化的数据访问负载,提供可预测的性能与扩展性;同时,最大限度地提高资源利用率;
3) 高可用性:可用性包含数据可用性与服务可用性两方面.基于冗余机制实现高可用性,无单点失效(single point of failure),支持故障的自动发现,透明地实施故障切换,不会因服务器故障而导致缓存服务中断或数据丢失.动态扩展时自动均衡数据分区,同时保障缓存服务持续可用;
4) 易用性:提供单一的数据与管理视图;API 接口简单,且与拓扑结构无关;动态扩展或失效恢复时无需人工配置;自动选取备份节点;多数缓存系统提供了图形化的管理控制台,便于统一维护;
5) 分布式代码执行(distributed code execution):将任务代码转移到各数据节点并行执行,客户端聚合返回结果,从而有效避免了缓存数据的移动与传输.最新的 Java 数据网格规范 JSR-347[7]中加入了分布式代码执行与 Map/reduce 的 API 支持,各主流分布式缓存产品,如 IBM WebSphere eXtreme Scale, VMware GemFire,GigaSpaces XAP 和 Red Hat Infinispan 等也都支持这一新的编程模型.

1.2 典型应用场景

分布式缓存的典型应用场景可分为以下几类:

1) 页面缓存.用来缓存 Web 页面的内容片段,包括 HTML、CSS 和图片等,多应用于社交网站等;
2) 应用对象缓存.缓存系统作为 ORM 框架的二级缓存对外提供服务,目的是减轻数据库的负载压力,加速应用访问;
3) 状态缓存.缓存包括 Session 会话状态及应用横向扩展时的状态数据等,这类数据一般是难以恢复的,对可用性要求较高,多应用于高可用集群;
4) 并行处理.通常涉及大量中间计算结果需要共享;
5) 事件处理.分布式缓存提供了针对事件流的连续查询(continuous query)处理技术,满足实时性需求;
6) 极限事务处理.分布式缓存为事务型应用提供高吞吐率、低延时的解决方案,支持高并发事务请求处理,多应用于铁路、金融服务和电信等领域.

1.3 分布式缓存的发展

分布式缓存经历了多个发展阶段,由最初的本地缓存到弹性缓存平台直至弹性应用平台[8],目标是朝着构建更好的分布式系统方向发展(如图 2 所示).

1) 本地缓存:数据存储在应用代码所在内存空间.优点是可以提供快速的数据访问;缺点是数据无法分布式共享,无容错处理.典型的,如 Cache4j;
2) 分布式缓存系统:数据在固定数目的集群节点间分布存储.优点是缓存容量可扩展(静态扩展);缺点是扩展过程中需要大量配置,无容错机制.典型的,如 Memcached;
3) 弹性缓存平台:数据在集群节点间分布存储,基于冗余机制实现高可用性.优点是可动态扩展,具有容错能力;缺点是复制备份会对系统性能造成一定影响.典型的,如 Windows Appfabric Caching;
4) 弹性应用平台:弹性应用平台代表了云环境下分布式缓存系统未来的发展方向.简单地讲,弹性应用平台是弹性缓存与代码执行的组合体,将业务逻辑代码转移到数据所在节点执行,可以极大地降低数据传输开销,提升系统性能.典型的,如 GigaSpaces XAP.

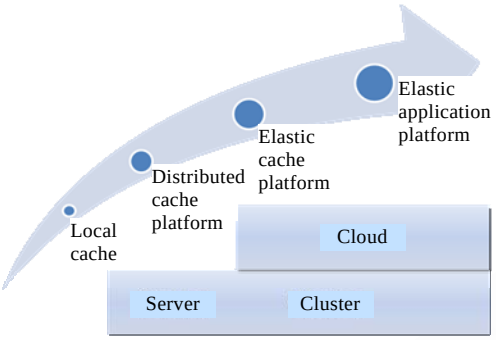


Fig.2 Development of distributed cache

图 2 分布式缓存的发展

1.4 分布式缓存与NoSQL

NoSQL 又称为 Not Only Sql,主要是指非关系型、分布式、支持水平扩展的数据库设计模式.NoSQL 放弃了传统关系型数据库严格的事务一致性和范式约束,采用弱一致性模型[9].相对于 NoSQL 系统,传统数据库难以满足云环境下应用数据的存储需求,具体体现在以下 3 个方面:

1) 根据 CAP 理论,一致性(consistency)、可用性(availability)和分区容错(partition tolerance)这 3 个要素最多同时满足两个,不可能三者兼顾[10,11].对云平台中部署的大量 Web 应用而言,数据可用性与分区容错的优先级通常更高,所以一般会选择适当放松一致性约束.传统数据库的事务一致性需求制约了其横向伸缩与高可用技术的实现;
2) 传统数据库难以适应新的数据存储访问模式.Web 2.0 站点以及云平台中存在大量半结构化数据,如用户 Session 数据、时间敏感的事务型数据、计算密集型任务数据等,这些状态数据更适合以 Key/Value 形式存储,不需要 RDBMS 提供的复杂的查询与管理功能;
3) NoSQL 提供低延时的读写速度,支持水平扩展,这些特性对拥有海量数据访问请求的云平台而言是至关重要的.传统关系型数据无法提供同样的性能,而内存数据库容量有限且不具备扩展能力.

分布式缓存作为 NoSQL 的一种重要实现形式,可为云平台提供高可用的状态存储与可伸缩的应用加速服务,与其他 NoSQL 系统间并无清晰的界限.云平台中应用访问与系统故障均具有不可预知性,为了更好地应对这些挑战,应用软件在架构时通常采用无状态设计,大量状态信息不再由组件、容器或平台来管理,而是直接交付给后端的分布式缓存服务或 NoSQL 系统.

1.5 分布式缓存与极限事务处理

随着云计算与 Web 2.0 的进一步发展,许多企业或组织时常会面对空前的需求:百万级的并发用户访问、每秒数以千计的并发事务处理、灵活的弹性与可伸缩性、低延时及 7×24×365 可用性等.传统事务型应用面临极限规模的并发事务处理,出现了极限事务处理型应用,典型的有铁路售票系统.Wikipedia 认为,极限事务处理是每秒多于 500 次事务或高于 10 000 次并发访问的事务处理[12].Gartner 将极限事务处理(extreme transaction processing,简称 XTP)定义为一种为事务型应用的开发、部署、管理和维护提供支持的应用模式,特点是对性能、可扩展性、可用性、可管理性等方面的极限需求[13].Gartner 在其报告中预测指出,极限事务处理型应用的规模将由 2005 年的 10%提升至 2010 年的 20%[14],极限事务处理技术是未来 5 年~10 年的热点技术[13].

极限事务处理的引入,无疑给传统 Web 三层架构带来了新的挑战.即,如何在廉价的、标准化的硬件和软件平台之上,对大容量、业务关键型的事务处理应用提供良好的支撑.分布式缓存作为一种关键的 XTP 技术,可为事务型应用提供高吞吐率、低延时的技术解决方案.其延迟写(write-behind)机制可提供更短的响应时间,同时极大地降低数据库的事务处理负载,分阶段事件驱动架构(staged event-driven architecture)可以支持大规模、高并发的事务处理请求.此外,分布式缓存在内存中管理事务并提供数据的一致性保障,采用数据复制技术实现高可

用性,具有较优的扩展性与性能组合.

1.6 相关的标准规范

JSR-107[15]与 JSR-347[7]是分布式缓存的两项标准规范.JSR-107 是由 Java 标准制订组织 JCP(Java Community Process)维护的一项 Java 对象缓存服务(object caching service for Java,简称 OCS4J)规范.该规范定义了 Java 缓存对象的类型和属性,同时给出了缓存 API 和操作语义说明,包括对象的创建、共享访问、失效和一致性维护等,最初由 Oracle 提交,并有望成为云应用平台 Java EE 7(JSR 342)[16]规范的一部分.

JSR-347 是由 Red Hat 提交 JCP 的一项 Java 数据网格规范,它提供了对数据网格访问、存储和数据管理的 API 说明,并强调了扩展性及数据的持久性存储.作为 JSR-107 规范的超集,JSR-347 复用了其大部分接口规范,同时针对 JSR-107 规范的局限性,添加了许多新特性(如数据分区、复制和事务等)和新的 API 支持,如异步非阻塞 API、数据分组与定位 API、分布式代码执行与 map/reduce API 等.

2 分布式缓存技术分析框架 DctAF

用户需求、数据特征和应用场景的多样性,决定了分布式缓存技术间的差异.建立全面客观的分布式缓存技术综合评价体系,对于更加深入、细致地分析和理解现有缓存技术的特点、不足以及明确云环境下分布式缓存面临的挑战,具有重要的指导意义.本文在深入分析云计算的特点[17]和缓存技术边界的基础上,建立了云环境下分布式缓存技术的分析框架(distributed cache techniques analysis framework,简称 DctAF).该框架从 6 个维度对现有缓存技术进行总结、比较与分析,这些维度分别对应了云计算所关注的 6 个方面的内容,同时也对应了研究领域中需要重点解决的若干方面问题.涵盖的 6 个分析维度如下所示:

1) 弹性资源供给(elasticity);
2) 可用性与可靠性(availability and reliability);
3) 敏捷性与自适应性(agility and adaptability);
4) 多承租(multi-tenancy);
5) 数据管理(data management);
6) 数据安全与隐私防护(security, privacy and compliance).

本文对分析框架的上述方面进一步细化,针对每个维度建立若干分析侧面(如后文图 3 所示).这些分析侧面用于桥接云计算的功能需求、特点与分布式缓存相关技术,并在二者之间建立映射.一般而言,缓存技术的差异性主要体现在适用场景、性能与扩展性、网络通信与计算开销等诸多方面,DctAF 框架对各类缓存技术进行了比较和分析.DctAF 框架的建立,对缓存技术的选择和开发具有重要的指导意义,对分析和评价现有分布式缓存产品同样具有重要的意义.

2.1 弹性资源供给

本节主要从弹性供给维度综合分析分布式缓存相关技术.弹性供给作为云计算的重要特征,是指在满足用户服务水平协议(SLA)的前提下,依据外界环境的变化动态调整缓存资源分配[17],实现资源的优化利用和按需供给.如图 3 所示,根据资源容量变化与否将该维度进一步细化为动态扩展和负载均衡两个分析侧面,这两个侧面分别从横向和纵向维度实现弹性资源供给.而数据迁移则是弹性供给的支撑技术.

动态扩展在线增加或删除节点以实现缓存容量的调整,同时保障服务持续可用.容量规划(capacity planning)为弹性供给提供支持,其目标是确定初始和调整后的缓存容量和节点数.由于数据访问的规模与模式具有时变性,导致某些缓存节点成为瓶颈并制约整个系统的吞吐量.负载均衡对系统性能短板的消除及服务质量保障等具有重要作用.负载均衡的目标是使缓存数据与访问负载在节点间尽可能地均匀分布,包含客户端均衡与服务端均衡两种实现方法.

数据迁移是实现节点动态扩展与负载均衡的关键技术.目前,多数分布式缓存支持在线数据迁移,但迁移时缺乏对热点数据的均衡处理.数据迁移包括制定迁移计划(migration plan)、路由信息同步、用户请求转发及数

据一致性管理等核心内容.需要指出的是,数据迁移过程中伴随的大量状态同步会给系统性能带来一定影响,因此,如何有效降低迁移开销是需要着力解决的问题.

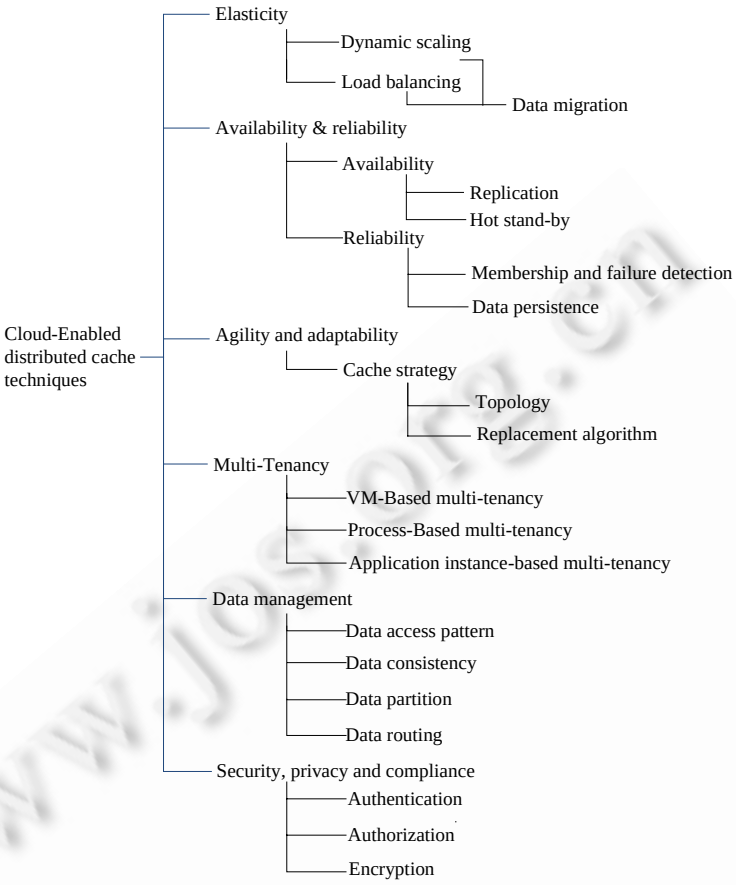


Fig.3 Distributed caching technologies analysis framework—DctAF

图 3 分布式缓存技术分析框架——DctAF

2.2 可用性与可靠性

该维度的主要分析目标是分布式缓存的可用性与可靠性保障技术.为避免软、硬件故障导致的缓存服务中断,消除系统中的单点失效(single point of failure),分布式缓存通常采用冗余机制实现高可用,相关技术包括复制与热备.复制技术主要针对缓存数据的可用性,粒度较细;而热备则主要针对关键节点的可用性,粒度较粗.可用性同时强调冗余机制不能引入较大的性能开销.可靠性描述了缓存在规定的任务时间内无故障运行的可能性,相关技术包括成员关系维护与失效检测、数据持久化,前者主要从缓存节点的可靠性角度出发,而后者则主要从数据可靠性角度出发.

复制技术中,每个节点的缓存数据以同步或异步复制方式备份至其他缓存节点,备份数一般是用户可配置的.当出现节点失效时,系统可以自动实施故障切换,将用户请求透明地迁移至备份节点,同时将缓存数据在节点间重新分布.分布式缓存中可能会存在某些关键服务节点(如配置服务器等),这些节点对缓存集群的运行起着至关重要的作用.为保障其可用性,通常采用热备技术为关键服务节点配置若干备份服务器,当主服务器失效时,备份服务器自动接管其服务.

成员关系维护与失效检测是分布式缓存重要的可靠性保障技术,支持协议包括 Gossip[18],Jgroups[19]和 HeartBeat 等.Gossip 协议中,每个节点周期性地随机选取节点进行消息通信,收到消息的节点重复该过程,直至

消息扩散到所有节点.该协议在大型、异构、动态变化的网络环境中具有良好的鲁棒性,其不足是占用内存资源较多,收敛时间长,启动延迟大.Jgroups 提供了灵活可定制的协议栈,其通信协议包括 UDP(IP multicast)、TCP 和 JMS.Jgroups 支持消息重传、大消息拆分和同序消息接收等,用户可灵活配置协议栈以满足个性化需求.HeartBeat 协议采用心跳消息表征节点的健康状态,如果某一节点的心跳消息超过一定周期未被收到,则该节点会被标记为失效节点.

数据持久化技术可以有效防止缓存服务器重启或整个集群失效而导致的数据丢失(主要针对关键业务数据或状态数据).缓存对象被定期同步至磁盘中,服务器重启后,磁盘中的数据可以重新被激活并被使用[20].持久化技术的另一个应用场景是应对节点内存空间不足.传统分布式缓存一般会利用替换算法直接将对象替换出内存,造成数据丢失.为了保障应用状态存储的可靠性,以 Couchbase 为代表的部分缓存产品提供了异步持久化功能,将超出节点内存容量的数据保存至磁盘[21].

2.3 敏捷性与自适应性

云计算具有开放、动态和多变等特点,这些特点决定了分布式缓存需要具有敏捷性、主动性和自适应性.敏捷性与自适应性描述了系统依照内、外部环境的变化(如请求数据量、读写比例等)及时做出响应和调整的能力,该维度与弹性资源供给维度有着紧密的关联.深入分析不同缓存策略的特点和适用场景,对自适应机制的实施具有重要意义.根据策略目标的不同,可将缓存策略进一步细化为若干个维度,如拓扑结构、缓存替换算法和数据一致性策略等,本节主要选取前两个维度进行分析.

从拓扑结构角度划分,现有缓存策略主要有 3 种[22,23]:全复制(replicated)策略、分区(partitioned)策略和近区策略(near 策略,也称 client 策略[24]或 local 策略[25]).全复制策略中,数据写入和更新复制到所有缓存节点,最终,每个节点都有一份完整的数据拷贝(如图 4 所示).该策略的优点是本地读取数据、响应延迟低、提供最高级的可用性支持,缺点是内存占用率高、通信开销大.分区策略中,每个节点负责缓存唯一的 1/*N* 份数据,所有数据更新操作均在单跳内完成,有效避免了全复制策略引入的高通信开销问题.其不足在于数据访问包含远程操作,访问速度不及全复制策略(如图 5 所示).近区策略在分区策略的基础上增加了一个容量较小的前端缓存,用来加速热点数据访问.用户请求对象时,可从前端缓存中无延迟加载,加载不到会自动从后端缓存加载.后端缓存容量较大,但访问速度不及前端缓存(如图 6 所示).全复制策略适用于缓存数据量小或读请求较多的场景,分区策略适用于缓存数据量大、写操作频繁的场景.近区策略适用于大量热点数据访问或读较多的场景.当数据更新操作频繁时,缓存间同步开销较大,不建议使用该策略[22].

缓存替换策略用来决策当缓存空间占满,需要额外的空间时,系统如何选择要替换的数据项.传统缓存替换算法主要包括 LRU,LFU,MRU,SIZE 和 Random 等.这些算法采用的替换标准包括访问时间、访问频率和数据项大小等.LRU 实现机制简单,适用于高局部性的数据访问模式,但应对顺序访问或随机访问效果不佳;LFU 则与 LRU 相反,适用于顺序或随机访问模式;MRU 将最近访问的数据项替换出缓存,适用于循环访问模式,但应对高局部性的数据访问效果较差.

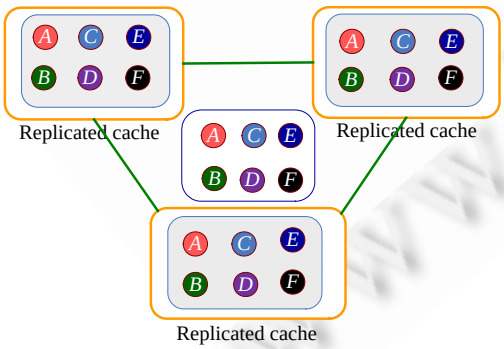


Fig.4 Replicated strategy

图 4 全复制策略

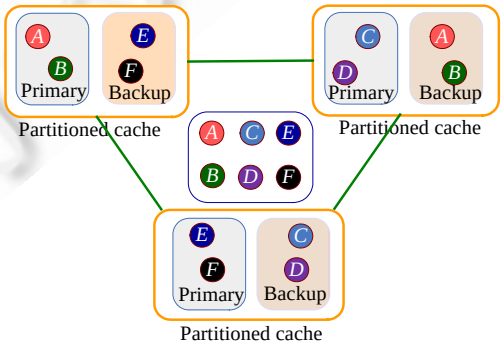


Fig.5 Partitioned strategy

图 5 分区策略

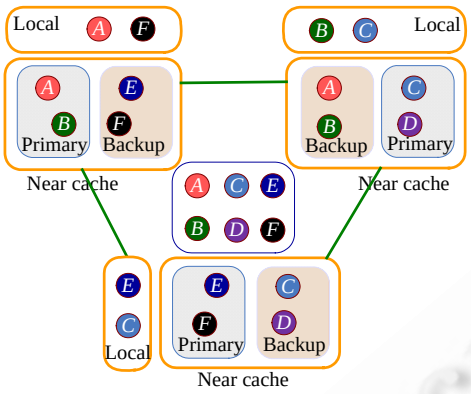


Fig.6 Near strategy

图 6 近区策略

2.4 多承租

多承租(multitenancy)分析维度反映的是分布式缓存对多承租的支持与实现.多承租是云计算的关键特性[26],其基本思想是,宿主在某一服务层上的一个或多个服务实例为来自不同组织(租户)的并发用户请求提供服务,实现资源的共享使用,目的是提高资源利用率,降低均摊在每个租户上的基础设施与管理成本.根据共享性与隔离性方面存在的差异,本文将分布式缓存支持的多租户模式归纳为 3 种:VM-租户模式、进程-租户模式和应用实例-租户模式.这 3 种多租户模式共享程度依次升高,低共享程度的多租户模式隔离性好,易于管理和维护,但其缺点是系统开销大;高共享程度的多租户模式扩展性好,但隔离性有限,维护难度增加.

在 VM-租户模式中,服务器硬件被划分为若干 VM 实例,每个 VM 实例运行一个缓存节点,每个缓存节点承载一个租户应用的部分或全部数据,典型的,如 Amazon ElastiCache.在进程-租户模式中,每个 VM 实例运行多个缓存节点,每个缓存节点为一个租户应用服务,典型的,如 Gigaspaces[27].在应用实例-租户模式中,每个 VM 实例运行一个缓存节点,该节点承载了多个租户应用的缓存数据.Couchbase 基于相互隔离的数据桶(bucket),允许租户配置不同的端口采用授权机制访问各自的缓存数据[28].相比前两种租户模式,应用实例-租户模式采用线程级的资源管理机制,共享程度高,但难以精确隔离和有效调度底层物理资源,租户间性能隔离与安全隔离面临更多挑战.

2.5 数据管理

数据管理是云计算的核心问题之一.本节主要从数据管理维度对现有缓存技术进行总结分析.如图 3 所示,根据管理功能的差异,可将该维度进一步细化为数据访问模式、数据一致性、数据分区和数据路由这 4 个分析侧面.数据访问模式描述了缓存与底层数据源间的交互模式,其评价指标主要包括性能、扩展性和可用性.一致性管理主要从数据复制的角度进行考虑,分析和评价的内容是不同复制技术及其对性能与一致性的权衡.数据分区与数据路由分别用来解决数据的分布和定位问题,二者之间有着紧密的联系.

2.5.1 数据访问模式

1) 预留缓存(cache-aside):该应用负责从数据库读取或写入数据,缓存系统不直接与数据库交互.该模式的优点在于开发者拥有对数据访问的完全控制,缓存机制的实现与应用解耦;缺点在于数据网格的特性难以使用[30−32];
2) 通读(read-through):当该应用请求的对象未存在于缓存时,缓存自动地从数据库中加载.该模式的优点在于缓存自动处理并发数据访问,可有效降低数据库访问压力,同时简化了应用程序代码;缺点在于请求对象仅能以 key 而非 SQL 查询的形式从数据库中加载,性能不及预刷新(refresh-ahead)[29−32];
3) 预刷新(refresh-ahead):缓存根据配置好的预刷新时间,在对象过期前自动、异步地从数据源中加载数

据.与通读模式相比,该模式延长了热点数据的生命周期,有效降低了响应时间;但其获益的多少取决于对象访问预测的准确度.准确度越低,对系统吞吐率影响越大(大量不必要的请求到达数据库)[29,30];

4) 通写(write-through):缓存负责将对象更新同步至数据库中,该模式的优点在于数据库发生错误时,更新请求可回滚;其缺点在于写性能及扩展性受数据库制约[29−31];

5) 延迟写(write-behind):对象更新请求被放入执行队列中,超过指定的时间间隔后异步写入数据库.该模式的优点主要有:

(1) 提升了应用性能,用户不再需要等待数据写入数据库中;

(2) 极大地降低了数据库的访问压力,对同一对象的多次更新操作可以合并.此外,对不同对象的操作也可以组合为一个单一的数据库事务,使得开销较大的写操作次数得以减少;

(3) 应用的执行不受数据库失效的影响;

(4) 提供线性的扩展性(linear scalability).

其缺点在于:

(1) 未对执行队列中的对象更新请求提供持久化支持;

(2) 整个集群失效时会产生数据丢失[29,30,32].

2.5.2 数据一致性

复制技术的主要目标是提高可用性,同时可通过创建数据副本均衡节点负载、提升系统性能.从复制模式角度可将复制技术分为主从复制(primary-backup replication)和多主复制(multiple-master replication).主从复制中,读写操作均在主副本执行,对主副本的更新被转发至从副本;多主复制则允许任何副本执行读写操作,每个副本将更新转发至其他副本.复制技术从时间维度又可分为同步复制和异步复制.同步复制中,客户端需要等待复制操作全部完成才能得到响应,性能开销大,但可以保障即时一致性;异步复制不必等待复制操作完成即可返回结果,性能和扩展性较优,可以保障数据的最终一致性,但却牺牲了即时一致性,这意味着应用可能会在某一时刻读取到不一致数据.

Quorum 协议是一种应用于 Quorum 系统的数据副本控制协议,该协议存在 3 个关键参数:N,R 和 W,N 表示复制因子,R 和 W 分别表示成功执行一次读、写操作所需要的最少节点数.满足 $R+W>N$ 可以保障数据访问的强一致性.用户可以对 R,W 值进行设置,并合理权衡性能、可用性与数据一致性需求[33].较低的 R,W 值意味着低响应延迟,但同时引入了数据不一致性的风险(无法保证所有副本均写入成功);较高的 R,W 值意味着较强的数据一致性,但会降低系统性能.因此,需要兼顾多方面需求,合理设置 R,W 值.

2.5.3 数据分区

数据分区(data partitioning)负责将数据均匀存储至缓存服务器节点,目前最流行的数据分区算法是一致性哈希(consistent hashing)及其改进算法.一致性哈希[34,35]的基本思想是,将哈希函数的输出定义为一个哈希环,缓存数据项的键值被映射到哈希环上的某个位置,沿顺时针方向找到的第 1 个服务器节点即是存放该数据项的目标节点.一致性哈希算法很大限度上降低了节点变化对数据分区造成的影响,然而,该算法在数据均衡分布及应对非均匀访问方面仍存在较大的不足.Amazon 于 2007 年提出了一种改进的一致性哈希[33],该算法旨在提升数据的均匀分布,并增加对异构节点的支持.其基本思想是,将整个哈希空间分为若干等大小的 Q 份数据分区(也称为虚拟节点,$Q>>N$,N 为缓存节点数),每个缓存节点依据其处理能力分配不同数量的数据分区.客户端采用二次哈希映射定位数据项.具体来讲:首先,*key* 值经哈希函数映射至环上的某个位置记为 *token*;然后,*token* 值再次被哈希映射为某一分区标识.得到分区标识后,客户端从分区服务器映射表中查询得到存放该数据分区的缓存节点.

2.5.4 数据路由

在客户端驱动的数据路由中,客户端基于哈希算法将用户请求直接定位至目标服务器节点.该方法的优点在于响应延迟低;数据路由操作在单跳内完成,可有效降低网络开销;节点间相互独立,有利于扩展机制的实现.但其执行效率很大程度上取决于客户端集群视图的新鲜度,新鲜度越高,则执行效率越高.在服务端驱动的数据

路由中,用户请求到达任一缓存节点,该节点负责将请求逐跳转发至目标节点.该方法的优点在于客户端与缓存服务端的耦合度较低,缺点在于多跳转发引入了较高的网络开销[36],同时增大了缓存服务节点的处理压力.

2.6 数据安全与隐私防护

云计算的开放性以及租户行为的不确定性,要求缓存服务提供商能够提供完善的用户管理与数据安全机制,有效识别和防止应用的恶意行为,保护租户数据免受网络攻击,同时提供可信的访问控制与数据隐私防护,相关技术包括身份认证、授权和加密等.

- 身份认证(authentication)[37−39]:支持认证口令的加密和解密,认证方式包括缓存服务端与客户端间的认证及缓存服务器间的认证;
- 授权(authorization)[37,39]:对不同用户提供不同层次的访问权限,以确保缓存资源受控、合法地被使用.授权范围包括基本操作,如缓存数据的读取、修改和删除等,也包含某些管理操作,如缓存服务节点的启停、数据复制等;
- 加密(encryption)[39]:缓存客户端与服务端的数据通信采用加密处理,确保数据在传输过程中未被非法篡改.

3 典型系统

3.1 Oracle Coherence

Oracle Coherence 是 Oracle 提供的一款内存数据网格解决方案,为大容量应用及关键任务提供支持,广泛应用于金融服务、电信、旅游和物流等行业领域,同时也是 Oracle 私有云方案的重要组成部分[5].Coherence 采用事件驱动的体系结构,提供一种去中心化、对等的数据管理技术,支持节点的动态伸缩与数据迁移,提供容错与服务可用性保障,支持极限事务处理(XTP),提供 100%的事务完整性、数据可靠性及零延迟的极限性能.在数据管理方面,提供多版本并发控制,支持并行查询与缓存索引.最新的 3.7 版本引入了弹性数据、动态负载均衡和自动代理发现功能,基于内存和磁盘存储网格数据,智能地均衡客户端连接数,同时支持与硬件负载均衡器 F5 的集成.

3.2 Memcached

Memcached 是目前最为著名的开源分布式缓存产品,被 YouTube,Reddit,Zynga,Facebook,Orange 和 Twitter 等在内的多家世界知名企业所使用.云平台 Google App Engine 和 Amazon Web Services 同样提供了使用 Memcached 服务的 API[40].Memcached 基于 Client/Server 架构,服务端基于 C 语言实现,同时提供了多种语言的客户端 API.Memcached 的优势在于高性能与轻量级,具有如下特点:

- 基于二进制协议(binary protocol)实现客户端服务器端通信,协议简单,可节省 CPU 资源;采用基于 libevent 的事件处理机制,便于在 Linux,BSD,Solaris 等操作系统上实现高性能;
- 使用 Slab Allocation 机制分配、管理内存,可有效解决内存碎片问题;
- 服务器节点间互不通信,最大限度地降低了通信开销[41].

3.3 Terracotta

Terracotta 是一个基于 JVM 的集群解决方案,提供包括 Http Session 复制、POJO 集群在内的多种功能.Terracotta 采用细粒度对象管理机制[42],基于代码注入(binary code injection)方法截获集群节点对数据的修改请求,增量传输对象中变化的属性,有效降低了计算开销与通信开销.用户可在不改变现有系统代码的前提下实现由单机 Java 应用向集群化应用的无缝迁移,简化了应用开发的复杂度.Terracotta 使用集群锁(clustered lock)用以保障数据的一致性,其 BigMemory 机制支持进程内的非堆(off-heap)存储,有效解决了 JVM 垃圾回收产生的性能瓶颈问题.

3.4 典型系统比较

表 1 基于 DctAF 框架对典型系统进行了比较和分析,比较范围包括弹性资源供给、可用性与可靠性、敏捷性与自适应、多承租、数据管理、数据安全与隐私防护这 6 个维度.表中的缩写字母对应表下方的注释,“√”表示支持,“*–”表示部分支持,“×”表示不支持.

Table 1 Comparison of typical cache systems based on DctAF

表 1 基于 DctAF 的典型系统比较

System	Elasticity		Availability and reliability				Agility and adaptability	
	Dynamic scaling	Load balancing	Replication	Hot stand-by	Failure detection	Disk persistence	Topology	Replacement algorithm
Oracle coherence	√	C	P/M/S/A	×	TCMP	√	R/P/N	LRU, LFU, Hybrid
Microsoft appFabric caching	√	×	P/M/S	×	H	×	R/P/N	LRU LRU-threshold
Memcached	×	×	×	×	×	×	P	LRU
Terracotta & ehcache	√	C	P/M/S/A	√	J	√	R/P	LRU, FIFO LFU
IBM WXS	√	×	P/M/S/A	×	H	×	R/P	LRU, LFU
GigaSpaces XAP	√	C/S	P/M/S/A	×	T	×	R/P/N	LRU
Redhat infinispan	√	C	P/M/S/A	×	J	√	R/P	LRU, FIFO LIRS
Couchbase	√	C	P/M/S/A	×	H	√	P	LRU

Load balancing: C—Client-Based load balancing, S—Server-Based load balancing.
Replication: P—Primary-Backup replication, M—Multiple-Master replication, S—Synchronous replication, A—Asynchronous replication.
Topology: R—Replicated strategy, P—Partitioned strategy, N—Near strategy.
Failure detection: J—Jgroups, H—Heartbeat, TCMP—Tangosol cluster management protocol, T—Timeout-Based failure detection.

System	Multi-Tenancy	Data management				Security privacy and compliance
		Data access pattern	Data consistency	Data partition	Data routing	
Oracle coherence	×	C/RT/RA/WT/WB	S/E	P	S–	AE/AO/E
Microsoft appFabric caching	V	C/RT/RA/WT/WB	S/E	R	C	AE/E
Memcached	×	C	×	C	C	×
Terracotta & ehcache	×	C/CS/RT/WT/WB	S/E	N	C	AE
IBM WXS	×	C/RT/WT/WB	S/E	N	C	AE/AO
GigaSpaces XAP	P	C/RT/WT/WB	S/E	N	C	AE/AO/E
Redhat infinispan	A	C/RA/WT/WB	S/E	C	C	AE
Couchbase	A	C	S/E	MC	C	AE

Multi-Tenancy: V—VM-Based multi-tenancy, P—Process-Based multi-tenancy, A—Application instance-based multi-tenancy.
Data access pattern: C—Cache-Aside, CS—Cache-As-Aor, RT—Read-Through, RA—Refresh-Ahead, WT—Write-Through, WB—Write-Behind.
Data partition: N—Naïve hashing, C—Consistent hashing, MC—Modified consistent hashing, R—Region-Based hashing, P—Private hashing.
Data consistency: S—Strong consistency, E—Eventual consistency.
Data routing: C—Client-Driven routing, S—Server-Driven routing.
Security privacy and compliance: AE—Authentication, AO—Authorization, E—Encryption.

从第 2 节分析内容和表 1 可以看出,目前,主流分布式缓存技术与产品呈现出如下特点:

1) 在弹性资源供给方面,大部分商业软件都支持在线扩展并自动均衡数据分布,具备良好的可伸缩性,部分软件(如 Gigaspaces XAP,Microsoft AppFabric Caching)同时提供了容量规划工具与测试基准的支持;负载均衡的实现以客户端均衡为主,易于实施;数据迁移技术主要以容量均衡为目标,缺乏对热点数据的处理;
2) 在可用性与可靠性方面,复制技术已成为主流,热备则更多地针对特定的缓存架构;关键数据的可靠性保障技术已由内存延伸至磁盘,对失效检测协议的支持以 HeartBeat 和 Jgroups 为主;
3) 在敏捷性与自适应性方面,大部分商业软件都支持多种拓扑结构策略和替换算法,但仅停留在静态

配置层面,缺乏运行时动态自配置、自优化能力;

4) 在多承租方面,这一云计算的关键特性得到了许多新兴商业与开源软件(如 Gigaspaces XAP, Couchbase 等)的支持,成熟度较高的缓存产品对该维度的支持则相对滞后.从呈现的特点来看,支持的多租户模式较为多样化,主要针对 IaaS 层和 PaaS 层;技术实现以资源隔离为主,缺乏对性能隔离的进一步支持;

5) 在数据管理方面,大部分产品都能够支持包括预留缓存、通读、通写和延迟写在内的多种数据访问模式,还提供了对多种一致性水平的支持.在数据分区方面,哈希算法占据主流,支持的哈希算法以简单余数算法和一致性哈希为主.客户端驱动的数据路由方法具有响应延迟低、网络开销小等优点,是目前主流的数据路由方式;

6) 在数据安全与隐私保护方面,可信的数据访问控制对缓存服务提供商而言尤为重要,目前对于身份认证技术的支持较为广泛.

4 云计算环境下分布式缓存面临的新挑战

基于 DctAF 框架对当前缓存技术和典型系统的比较分析,本文认为,云计算的多承租、弹性资源供给、敏捷性与自适应性等特点和需求,给缓存系统的运行、维护和管理带来了新的困难与挑战.现有分布式缓存技术与云计算实际需求间仍存在一定差距,主要体现在缓存性能隔离、缓存数据迁移和缓存策略自适应这 3 个方面.

4.1 挑战1:缓存服务的性能隔离

多租户场景下,尤其是对应用实例-租户模式而言,所有租户共享单一缓存服务实例,引入的资源竞争会导致部分租户的服务质量(QoS)需求无法得到满足,即所谓的性能干扰(performance interference)问题,文献[43]称其为资源劫持.例如:当缓存服务采用传统的缓存替换策略,如 LRU,LFU 时,易导致请求内容宽泛、访问频率高的租户大量占用共享内存并获得额外的收益;而请求内容集中、访问频率低的租户则由于缓存内容被替换出缓存,命中率下降,而使服务质量和利益受到损害.此外,不同租户对网络 I/O 资源的使用,同样会出现上述问题.因此,如何保障各租户应用间的性能隔离,成为多承租面临的关键问题之一.由于每个租户的 QoS 需求不尽相同,这就需要在最大限度地提高系统资源利用率的基础上,确保所有租户的 QoS 需求同时得到满足.面向多承租的缓存服务性能隔离机制正逐渐成为当前研究的热点和难点.

IBM 海法实验室的 Chockler 等人[44]提出了一种改进的全局缓存替换方法.在替换策略的设计上,他们给出两种方案:第 1 种是通过为租户动态加权的方式确保访问频率低的租户内容不会很快被替换出内存;第 2 种方案是根据租户当前占用内存空间是否少于目标空间,动态选取该租户或其他租户数据替换出缓存.该方法存在的不足在于性能隔离过程收敛速度慢,难以适应环境的动态变化.比较多的研究工作采用动态资源分配的方法,其基本思想是直接逻辑划分缓存资源,利用资源隔离来实现性能隔离,该方法隔离效果较好.宾夕法尼亚州立大学的 Patrick 等人[45−47]近两年发表了一系列这方面的论文.卡内基梅隆大学的 Wachs 等人[48]采用基于 Trace 的方法建立分析模型(analytic model),根据该模型完成逻辑资源划分.弗吉尼亚大学的 Lu 等人[49]提出了一种基于反馈控制的方法,将基于 QoS 的缓存服务差分问题规约为闭环控制问题,采用命中率指标刻画租户 QoS 需求,基于 Z 转换(Z-transform)方法将命中率与 QoS 需求间的差值与缓存资源建立映射关系,确定资源调整值.该方法采用单一的资源映射模型,未考虑不同租户具有不同的访问模式,因而存在一定的局限性.

4.2 挑战2:虚拟化环境下数据迁移的优化

数据迁移是实现节点动态扩展与弹性负载均衡的关键技术.对部署在 Xen 虚拟化环境中的缓存系统而言,VM 间性能干扰[50−52]会对数据迁移产生无法忽略的影响.具体来讲,迁移数据的发送与接收异步完成,并与 VM 调度机制相关.而 Xen 现有的调度算法——Credit 算法在 I/O 调度处理方面存在一定缺陷,主要表现为:调度时仅关注每个 domain 的状态,而不考虑其剩余的 credit 值及收发包的数量;一味追求资源分配的公平性而忽略了每个 domain I/O 处理的实时性需求.当部署的 VM 与缓存节点数增加时,I/O 请求队列的长度也随之增加,导致

大量的数据迁移操作被阻塞.因此,需要准确刻画 VM 干扰因素对迁移产生的影响.如何在重均衡数据的同时降低迁移开销,是缓存系统面临的另一项挑战.迁移开销从横向、纵向两个维度可分解为迁移时间与性能衰减度.如果迁移可用的带宽资源过多,虽然迁移时间短,但会引入高时延抖动;如果可用的带宽资源过少,虽然时延抖动小,但迁移时间过长,系统性能长期处于次优化状态.因此,制定迁移计划时需要在迁移时间与性能衰减度之间,进行权衡,最小化迁移开销.

弗吉尼亚大学 Lu 等人[53]的研究工作主要针对存储服务数据迁移过程中的 QoS 保障问题.他们提出了一种基于反馈控制的方法,周期性地求解满足 QoS 约束下的最优迁移速率.文献[54]同样采用控制理论解决数据迁移中的开销优化问题.基于多元回归法构建迁移时间与性能衰减度的预测函数,将二者线性加权得到迁移开销模型,最终实现以最小化开销为目标的迁移速率控制.线性开销模型的不足主要有两点:一是权值难以确定,大多基于主观经验人工设定,准确性和客观性无法得到保障;二是迁移时间与性能衰减度分别位于开销的水平维度和垂直维度,线性加权模型无法准确刻画二者间的关系.康涅狄格大学的学者[55]针对异构存储系统在满足不同传输能力约束前提下如何降低数据迁移时间的问题展开了研究.他们将这一问题规约为图着色(multi-edge coloring problem)问题,考虑如何优化数据迁移调度,以使得迁移消耗的时间片最少.美国东北大学 Kunkle 等人[56]的研究工作在制定迁移计划的同时考虑了迁移开销的优化.迁移算法迭代地选取开销最小的分区迁移方案.该工作的不足主要有两点:一是未考虑迁移时间的影响;二是定义的开销模型与迁移数据量和负载相关,而未包含对开销具有很大影响的迁移带宽元素,难以准确刻画迁移开销.加州大学圣巴巴拉分校 Das 等人[57]的工作主要针对多承租场景下数据库集群的数据迁移问题,他们提出了一种轻量级的、基于迭代复制的数据迁移方法.

4.3 挑战3:缓存策略的自适应与自管理

各缓存策略模型(包括替换策略、一致性策略等)的设计往往基于不同场景,不存在某种策略在所有场景下均表现最优的情况.例如,全复制策略适用于小规模集群、缓存数据量小或读请求较多的场景,而分区策略则适用于集群规模和缓存数据量较大、写操作频繁的场景.LRU 适用于高局部性的数据访问模式,而 LFU 则适用于顺序或随机访问模式.云平台中部署了大量的 Web 应用,内部部署环境变化(平台系统升级、VM 迁移等)和外部负载变化(负载量波动、访问模式切换)频繁.为进一步优化服务性能,增强缓存系统的柔性与自适应性,有必要为分布式缓存提供灵活的自适应支持机制.这其中面临着两项关键挑战:一是自适应方法、规则或模型如何建立,这些规则或模型应具备演化的能力以适应环境的变化;二是如何有效控制自适应开销.

阿姆斯特丹自由大学的 Pierre 等人[58]使用跟踪驱动模拟(trace-driven simulation)方法为每个 Web 文件选取最优一致性策略.文献[59]中,Sivasubramanian 等人针对决策过程中模拟器需要模拟所有一致性策略而引入的开销较大的问题,提出了一种启发式算法.该算法可以有效降低空间规模,并优化性能,但以牺牲一定的准确率为代价.印度理工学院的 Deolasee 与 IBM 印度实验室的 Bhide 等人[60]提出一种基于规则的方法,策略选取规则由带宽、数据变化率以及用户一致性需求要素组成.IBM 阿尔马登研究中心[61]提出了一种基于自适应规则的缓存替换算法.文献[62]根据统计结果及识别规则(detection rule)对当前访问模式进行分类,根据分类结果采用相应的最优替换策略.基于规则的方法的优点在于实施简单,规则元素往往由关键性能参数组成,决策效率高;缺点在于规则往往比较固定,难以适应环境与需求的动态变化,且规则的制定大多由人工完成,易引入主观因素的影响.加州大学圣克鲁兹分校的学者[63]提出了一种基于机器学习的缓存策略自适应方法.他们将最优策略选择规约为多专家决策(multiple experts)问题.文献[23]采用 Ripper 算法离线训练得到分类模型,基于该模型决策最优缓存策略.里斯本技术大学的 Couceiro 等人[64]采用离线学习和在线学习两种方法为每个事务选取最优复制策略,实验结果表明,在线学习效果明显优于离线学习.

5 总结与展望

云计算的引入,有力地推动了 IT 领域的深刻变革,同时也给分布式缓存技术的发展带来了难得的机遇.作为云平台提升应用性能的一种重要手段,分布式缓存技术近年来受到了工业界和学术界的广泛关注.本文力图全

面总结和分析云环境下分布式缓存技术的现状与挑战,为从事该领域的研究者及工作人员提供有益的参考.本文首先分析、介绍了分布式缓存的特性、典型应用场景、发展阶段、相关标准规范以及推动缓存技术发展的若干关键要素.在此基础上,建立了一个云环境下分布式缓存技术的分析框架——DctAF,该框架从分析云计算的特点和缓存技术的边界出发,涵盖 6 个分析维度.DctAF 框架的建立,对缓存技术的选择和开发具有重要的指导意义.本文基于 DctAF 框架对当前缓存技术进行总结与分析,并对典型系统进行了比较.进一步阐述了云环境下分布式缓存所面临的挑战;围绕上述挑战,分析和比较了已有的研究工作.

NoSQL 技术、Web 应用的规模及使用模式的发展,为分布式缓存及相关领域的研究带来了新的契机.综合 DctAF 框架的分析和近些年对云计算与分布式缓存的关注和研究,本文认为,未来几年,该领域的研究趋势和研究重点包含以下几个方面:

(1) 编程框架.当传统关系型数据库作为数据的存储载体时,通常由 ORM(object-relational mapping)框架完成对象模型到关系模型的映射,同时将 JPA(Java persistence API)查询转换为对应的 SQL 语言;当分布式缓存或数据网格作为主要存储载体时,需要一种新的机制与框架来完成对象模型到 Key/Value 模型的映射.Red Hat 于 2011 年提出一种新的对象网格映射框架 OGM(object/grid mapper).目前,该框架已支持基本的 CRUD 操作,但在复杂查询方面存在较大不足,且映射机制引入大量数据冗余.文献[65]在 Join 查询支持方面进行了有益的探索;

(2) 数据一致性.事务型应用作为企业应用的重要组成部分,强调事务保障与数据一致性;其他类型应用则更多地强调低延时、扩展性与可用性.现有的一致性协议在性能、一致性水平和通信开销等方面存在差异,适用于不同的需求场景.因此,需要根据应用特点和自身业务需求在 CAP 三者之间有效权衡.文献[66]建立了不同并发控制协议的一致性开销模型,基于开销分析和制定的规则对每个数据请求自适应地选取不同的一致性水平.此外,一致性协议的优化同样是需要继续深入研究的问题;

(3) 测试基准.不同的分布式缓存软件满足不同类型的用户需求(包括性能、事务性、可用性和扩展性等),测试基准对用户进行产品的选择至关重要.现阶段针对测试基准的研究仍比较薄弱,TPC-W[67]测试基准侧重于对传统事务型操作的性能分析与评价,Yahoo! Cloud Serving Benchmark(YCSB)[68]主要用来评价 NoSQL 系统的性能与扩展性.因此,有必要深入研究不同类型应用(包括数据密集型与事务密集型等)的特点和负载模式.

致谢 感谢魏峻研究员和王伟博士在本文写作过程中提出的宝贵意见,还要感谢参与课题工作的硕士生朱鑫、罗嵘两位同学及项目组博士生赵鑫同学,感谢他们对本文的材料收集与整理所做的工作.最后,我们特别感谢匿名审稿人富有建设性的宝贵意见和建议.

References:

[1] Cloud computing. Wikipedia. 2007. http://en.wikipedia.org/wiki/Cloud_computing

[2] Chen K, Zheng WM. Cloud computing: System instances and current research. Ruanjian Xuebao/Journal of Software, 2009,20(5): 1337−1348 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3493.htm [doi: 10.3724/SP.J.1001.2009.03493]

[3] Earls A. Distributed data grids: Foundation for future cloud computing? 2010. http://searchsoa.techtarget.com/news/1518647/Data-Grids-Foundation-for-future-cloud-computing

[4] Gualtieri M, Rymer JR. The forrester wave: Elastic caching platforms. Q2, 2010. ftp://ftp.software.ibm.com/software/solutions/soa/pdfs/wave_elastic_caching_platforms_q2_2010.pdf

[5] Platform-as-a-Service private cloud with oracle fusion middleware. Oracle White Paper, 2009. http://www.oracle.com/us/technologies/cloud/036500.pdf

[6] Amazon ElastiCache. 2011. http://aws.amazon.com/elasticache/

[7] JSR-347. 2011. http://jcp.org/en/jsr/detail?id=347

[8] Gualtieri M. Elastic caching platforms balance performance, scalability and fault tolerance. 2010. http://blogs.forrester.com/mike_gualtieri/10-03-18-elastic_caching_platforms_balance_performance_scalability_and_fault_toleranc

[9] NoSQL. Wikipedia. 2009. http://en.wikipedia.org/wiki/NoSQL

[10] Brewer EA. Towards robust distributed systems. In: Proc. of the 19th Annual ACM Symp. on Principles of Distributed Computing (PODC 2000). 2000. [doi: 10.1145/343477.343502]

[11] Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. ACM SIGACT News, 2002,33(2). [doi: 10.1145/564585.564601]

[12] Extreme transaction processing. Wikipedia. 2012. http://en.wikipedia.org/wiki/Extreme_Transaction_Processing

[13] Fenn J. Hype cycle for emerging technologies. Gartner Report, 2010. http://www.planetlarg.net/my-scripts/docs-to-read/gartner/hype_cycle_for_emerging_tech_2010.pdf

[14] Gold E. Extreme transaction processing. 2011. http://natishalom.typepad.com/nati_shaloms_blog/files/gigaspaces-dream-machine.pdf

[15] JSR-107. 2001. http://jcp.org/en/jsr/detail?id=107

[16] JSR-342. 2011. http://jcp.org/en/jsr/detail?id=342

[17] Schubert L, Jeffery K, N-Lutz B. The future of cloud computing: Opportunities for European cloud computing beyond 2010. 2010. http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[18] Gossip protocol. Wikipedia. 2007. http://en.wikipedia.org/wiki/Gossip_protocol

[19] Jroups. 2002. http://www.jgroups.org/overview.html

[20] Terracotta server arrays architecture. 2009. http://64.95.112.233/documentation/terracotta-server-array/server-arrays

[21] Ingenthron M. Growing data sets beyond memory. 2012. http://www.couchbase.org/wiki/display/membase/Growing+Data+Sets+Beyond+Memory

[22] Qin X, Zhang W, Wang W, Wei J, Zhong H, Huang T. A comparative evaluation of cache strategies for elastic caching platforms. In: Proc. of the 11th Int'l Conf. on Quality Software (QSIC 2011). 2011. 166−175. [doi: 10.1109/QSIC.2011.14]

[23] Qin X, Zhang W, Wang W, Wei J, Zhong H, Huang T. Online cache strategy reconfiguration for elastic caching platform: A machine learning approach. In: Proc. of the 35th Annual IEEE Int'l Computer Software and Applications Conf. (COMPSAC 2011). 2011. 523−534. [doi: 10.1109/COMPSAC.2011.73]

[24] NCache: Caching topologies. 2008. http://www.alachisoft.com/ncache/caching-topology.html

[25] Nori AK. Distributed caching platforms. In: Proc. of the 36th Int'l Conf. on Very Large Data Bases (VLDB 2010). Singapore: VLDB Endowment Inc., 2010. 1645−1646.

[26] Multitenancy. Wikipedia. 2008. http://en.wikipedia.org/wiki/Multitenancy

[27] Making dynamic scaling simple. GigaSpaces XAP. 2010. http://www.gigaspaces.com/xap7-1

[28] Multitenancy with couchbase. 2012. http://www.couchbase.com/docs/couchbase-manual-1.8/couchbase-introduction-architecture-buckets.html

[29] Caching data sources. Oracle. 2011. http://download.oracle.com/docs/cd/E24290_01/coh.371/e22837/cache_rtwtwbra.htm#CFHEDIGA

[30] Peralta P. Successfully scaling Java applications in spring. Oracle Corp. 2007. http://www.nejug.org/events/download?f=41

[31] Khan I. Using read-through & write-through in distributed cache. 2008. http://www.alachisoft.com/resources/articles/readthru-writethru-writebehind.html

[32] Programming model. Windows Server AppFabric Caching. 2011. http://msdn.microsoft.com/en-us/library/hh334298.aspx

[33] Hastorun D, Jampani M, Kakulapati G, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazon's highly available key-value store. In: Proc. of the ACM Symp. on Operating Systems Principles (SOSP 2007). 2007. 205−220. [doi: 10.1145/1323293.1294281]

[34] Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In: Proc. of the 29th Annual ACM Symp. on theory of Computing (STOC'97). 1997. 654−663. [doi: 10.1145/258533.258660]

[35] Karger D, Sherman A, Berkheimer A, Bogstad B, Dhanidina R, Iwamoto K, Kim B, Matkins L, Yerushalmi Y. Web caching with consistent hashing. Computer Networks, 1999,31(11):1203−1213. [doi: 10.1016/S1389-1286(99)00055-9]

[36] Pfaffhauser F. Scaling a cloud storage system autonomously [MS. Thesis]. Zuerich: ETH Zuerich, 2010.

[37] IBM Websphere extreme scale. 2004. http://www.redbooks.ibm.com/redbooks/SG247683/wwhelp/wwhimpl/js/html/wwhelp.htm

[38] Security concepts. Gigaspaces XAP. 2009. http://www.gigaspaces.com/wiki/display/XAP7/Security+Concepts

[39] Security model. Windows Server AppFabric Caching. 2011. http://msdn.microsoft.com/en-us/library/ff718179.aspx

[40] Memcached. Wikipedia. 2003. http://en.wikipedia.org/wiki/Memcached

[41] Memcached FAQ. 2011. http://code.google.com/p/memcached/wiki/NewStart

[42] Terracotta DSO documentation. 2011. http://www.terracotta.org/confluence/display/docs/Home

[43] Lin H, Han Y. Performance management for multi-tenant Web applications. Chinese Journal of Computers, 2010,33(10): 1881−1895 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2010.01881]

[44] Chockler G, Laden G, Vigfusson Y. Data caching as a cloud service. In: Proc. of the 4th Int'l Workshop on Large Scale Distributed Systems and Middleware (LADIS 2010). 2010. [doi: 10.1145/1859184.1859190]

[45] Patrick CM, Garg R, Son SW, Kandemir M. Improving I/O performance using soft-QoS based dynamic storage cache partitioning. In: Proc. of the 11th IEEE Int'l Conf. on Cluster (Cluster 2009). 2009. 1−10. [doi: 10.1109/CLUSTR.2009.5289192]

[46] Prabhakar R, Srikantaiah S, Patrick CM, Kandemir M. Dynamic storage cache allocation in multi-server architectures. In: Proc. of the ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage, and Analysis (SC 2009). 2009. 1−12. [doi: 10.1145/1654059.1654068]

[47] Prabhakar R, Srikantaiah S, Kandemir M, Patrick CM, Kandemir M. Adaptive multi-level cache allocation in distributed storage architectures. In: Proc. of the 24th ACM Int'l Conf. of Supercomputing (ICS 2010). 2010. 211−221. [doi: 10.1145/1810085.1810115]

[48] Wachs M, Abd-El-Malek M, Thereska E, Ganger GR. Argon: Performance insulation for shared storage servers. In: Proc. of the 5th USENIX Conf. on File and Storage Technologies (FAST 2007). San Jose: USENIX Association, 2007. 61−76.

[49] Lu Y, Abdelzaher TF, Saxena A. Design, implementation, and evaluation of differentiated caching services. IEEE Trans. on Parallel and Distributed Systems, 2004,15(5):440−452. [doi: 10.1109/TPDS.2004.1278101]

[50] Koh Y, Knauerhase R, Brett P, Bowman M, Wen Z, Pu C. An analysis of performance interference effects in virtual environments. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS 2007). 2007. 200−209. [doi: 10.1109/ISPASS.2007.363750]

[51] Pu X, Liu L, Mei Y, Sivathanu S, Koh Y, Pu C. Understanding performance interference of I/O workload in virtualized cloud environment. In: Proc. of the 3rd IEEE Int'l Conf. on Cloud Computing (Cloud 2010). 2010. 51−58. [doi: 10.1109/CLOUD.2010.65]

[52] Mei Y, Liu L, Pu X, Sivathanu S, Dong X. Performance analysis of network I/O workloads in virtualized data centers. IEEE Trans. on Service Computing, 2011. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5928316 [doi: 10.1109/TSC.2011.36]

[53] Lu C, Alvarez GA, Wilkes J. Aqueduct: Online data migration with performance guarantees. In: Proc. of the USENIX Conf. on File and Storage Technologies (FAST 2002). Monterey: USENIX Association, 2002. 219−230.

[54] Lim HC, Babu S, Chase JS. Automated control for elastic storage. In: Proc. of the 7th Int'l Conf. on Autonomic Computing (ICAC 2010). 2010. 1−10. [doi: 10.1145/1809049.1809051]

[55] Kari C, Kim Y, Russell A. Data migration in heterogeneous storage systems. In: Proc. of the 31st Int'l Conf. on Distributed Computing Systems (ICDCS 2011). 2011. 143−150. [doi: 10.1109/ICDCS.2011.46]

[56] Kunkle D, Schindler J. A load balancing framework for clustered storage systems. In: Proc. of the 15th Int'l Conf. on High Performance Computing (HiPC 2008). 2008. 57−72. [doi: 10.1007/978-3-540-89894-8_9]

[57] Das S, Nishimura S, Agrawal D, Abbadi AE. Live database migration for elasticity in a multitenant database for cloud platforms. Technical Report, CS-2010-09, Santa Barbara: The University of California, 2010.

[58] Pierre G, van Steen M, Tanenbaum AS. Dynamically selecting optimal distribution strategies for Web documents. IEEE Trans. on Computers, 2002,51(6):637−651. [doi: 10.1109/TC.2002.1009149]

[59] Sivasubramanian S, Pierre G, Steen MV. A case for dynamic selection of replication and caching strategies. In: Proc. of the 8th Workshop on Web Caching and Content Distribution. NewYork, 2003. 275−282.

[60] Bhide M, Deolasee P, Katkar A, Panchbudhe A, Ramamritham K, Shenoy P. Adaptive push-pull: Disseminating dynamic Web data. IEEE Trans. on Computers, 2002,51(6):652−668. [doi: 10.1109/TC.2002.1009150]

[61] Megiddo N, Modha DS. ARC: A self-tuning, low overhead replacement cache. In: Proc. of the 2nd USENIX Conf. on File and Storage Technologies (FAST 2003). San Francisco: USENIX Association, 2003. 115−130.

[62] Choi J, Noh SH, Min SL, Ha EY, Cho YK. Design, implementation, and performance evaluation of a detection-based adaptive block replacement scheme. IEEE Trans. on Computers, 2002,51(7):793−800. [doi: 10.1109/TC.2002.1017699]

[63] Ari I, Amer A, Miller EL, Brandt SA, Long DE. Who is more adaptive? ACME: Adaptive caching using multiple experts. In: Proc. of the Workshop on Distributed Data and Structures (WDAS 2002). Paris, 2002. 143−158.

[64] Couceiro M, Romano P, Rodrigues L. PolyCert: Polymorphic self-optimizing replication for in-memory transactional grids. In: Proc. of the ACM/IFIP/USENIX 12th Int'l Middleware Conf. (Middleware 2011). 2011. 309−328. [doi: 10.1007/978-3-642-25821-3_16]

[65] Wei Z, Pierre G, Chi C. Scalable join queries in cloud data stores. In: Proc. of the 12th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing (CCGrid 2012). 2012. [doi: 10.1109/CCGrid.2012.28]

[66] Fetai I, Schuldt H. Cost-Based adaptive concurrency control in the cloud. Technical Report, CS-2012-001, Basel: The University of Basel, 2012.

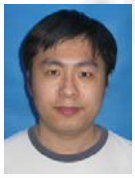
[67] TPC-W benchmark. 2002. http://www.tpc.org/tpcw/

[68] Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In: Proc. of the ACM Symp. on Cloud Computing (SoCC 2010). 2010. 143−154. [doi: 10.1145/1807128.1807152]

附中文参考文献:

[2] 陈康,郑纬民.云计算:系统实例与研究现状.软件学报,2009,20(5):1337−1348. http://www.jos.org.cn/1000-9825/3493.htm [doi: 10.3724/SP.J.1001.2009.03493]

[43] 林海略,韩燕波.多租户应用的性能管理关键问题研究.计算机学报,2010,33(10):1881−1895. [doi: 10.3724/SP.J.1016.2010.01881]



秦秀磊(1982－),男,山东烟台人,博士生,主要研究领域为网络分布式计算,软件工程.
E-mail: qinxiulei08@otcaix.iscas.ac.cn

王伟(1982－),男,博士,副研究员,主要研究领域为网络分布式计算,软件工程.
E-mail: wangwei@otcaix.iscas.ac.cn

张文博(1976－),男,博士,副研究员,CCF 会员,主要研究领域为网络分布式计算,软件工程.
E-mail: zhangwenbo@otcaix.iscas.ac.cn

钟华(1971－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.
E-mail: zhongh@otcaix.iscas.ac.cn

魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.
E-mail: wj@otcaix.iscas.ac.cn

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.
E-mail: tao@otcaix.iscas.ac.cn