doi:10.3969/j.issn.1000-436x.2014.z2.010

SaaS 多租户数据放置的优化调整策略研究

李晓娜[1,2]，李庆忠[2]，孔兰菊[2]，闫中敏[2]
（1. 青岛大学 软件技术学院，山东 青岛 266071；2. 山东大学 计算机科学与技术学院，山东 济南 250101）

摘 要：采用副本技术的 SaaS 多租户数据为满足租户访问及服务提供商管理的需求，在云中多节点必须合理放置。针对多租户数据特征和节点负载情况，对负载超重节点或超轻节点，通过调整数据副本数目和位置进行放置策略维护及优化，在满足所有租户 SLA 需求的同时最小化服务提供商的成本代价，实现服务质量与管理成本 2 个目标的均衡。并通过实验与随机放置策略和贪婪放置策略进行比较，证明该策略的可行性和有效性。

关键词：SaaS；多租户数据库；SLA；优化调整；副本放置
中图分类号：TP311 **文献标识码**：A **文章编号**：1000-436X(2014)Z2-0063-09

Research on the optimization adjustment strategy for the SaaS multi-tenant data placement

LI Xiao-na[1,2], LI Qing-zhong[2], KONG Lan-ju[2], YAN Zhong-min[2]
(1. Department of Software Technology, Qingdao University, Qingdao 266071, China;
2. Department of Computer Science and Technology, Shandong University, Jinan 250101, China)

Abstract: The multi-tenant data stored in cloud using replica technology must be reasonably placed, meet the requirements for data access by tenants and management by service providers. According to the characteristics of multi-tenant data and the load of the nodes, for the outweight nodes and the ultralight nodes, through adjusting the number and position of the replicas to maintenance and optimization the placement strategy so that meet the SLA requirements at the same time minimize the overall cost, realize the balance between the quality of service and the cost of management. Experimental results through comparison with random placement strategy and greedy placement policy demonstrate the feasibility and effectiveness of the proposed strategy.

Key words: SaaS; multi-tenant database; SLA; optimization adjustment; replica placement

1 引言

SaaS（software as a service）[1]是云计算环境下的一种新型的商业模式。SaaS 模式下，服务提供商采用单实例多租赁（single instance multi-tenancy）的软件交付方式，支持多个租户按需定制 SaaS 应用。租户无需关注应用及数据存储等实现细节，通过与服务提供商签订服务水平协议（SLA, service level agreements）[2]，对其租赁的服务提供性能保障。SaaS 模式下的多租户数据，存储在服务提供商的共享数据库中，基于运营成本考虑，SaaS 服务提供商都选择共享数据库独立模式或共享数据库共享模式[3]2 种方案实现存储。如何对多租户数据进行有效管理，成为普遍关注的一个现实问题。

收稿日期：2014-10-21
基金项目：国家自然科学基金资助项目（61303085,61303005）；国家科技支撑计划基金资助项目（2012BAH54F01）；山东省自然科学基金资助项目（ZR2013FQ014）；山东省科技发展计划基金资助项目（2012GGX10134）；山东省自主创新基金资助项目（2013CXC30201）
Foundation Items：The National Natural Science Foundation of China (61303085,61303005); The National Key Technologies R&D Program (2012BAH54F01); The Natural Science Foundation of Shandong Province (ZR2013FQ014); Science and Technology Development Plan Project of Shandong Province (2012GGX10134); Shandong Province Independent Innovation Major Special Project (2013CXC30201)

为满足云环境扩展性、可用性、可靠性[4]等特点，云中数据管理要求采用副本技术将数据放置在不同节点。放置策略的好坏，会直接影响整个系统的负载均衡、读写效率、一致性维护等系统性能[5]。针对 SaaS 多租户数据的放置，会进一步关系到服务提供商的管理维护成本和租户的服务体验。因此，SaaS 模式下多租户数据放置策略的研究变的异常重要。

本文以共享数据库共享模式作为多租户数据存储，针对 SaaS 模式下多租户数据放置问题进行建模并确定成本代价模型。当出现负载超重或负载超轻节点时，充分考虑数据特征和负载情况，提出一种满足所有租户 SLA 需求和优化成本代价的数据放置调整策略和算法，实现 SaaS 服务提供商用尽可能少的成本代价为所有租户提供满足要求的服务这 2 个目标之间的平衡。

2 相关工作

随着云计算的发展，如何将数据对象高效合理地放置到云存储系统中，满足云存储节点数量大、系统伸缩性强、数据可靠性要求高等方面的需求成为一个非常有挑战性的课题。

Hadoop 分布式文件系统（HDFS, Hadoop distributed file system）[6]采用一种称为机架感知（rack awareness）的策略来进行数据副本放置，但它主要针对分析型应用的大规模数据集。Amazon[7]的云存储系统使用了 Dynamo 架构，利用改进的一致性散列算法，将数据放置在虚拟节点，并在后继节点进行副本存储。上述几种云数据管理所采用的副本放置机制，主要针对分析型数据，不适合处理 SaaS 应用中事务型工作负载，不结合事务访问特点进行放置，会导致 SaaS 多租户数据产生大量更新代价并且增加分布式事务成本。

在传统的分布式计算和网格计算领域，对副本放置问题已经进行了广泛深入的研究。文献[8]为用户选择最优副本提供了 QoS 参数的合理依据。文献[9]针对副本放置产生的存储代价，提出了考虑数据读和写的副本放置算法，该算法基于一种每个节点都可以读取其他任何节点上的副本的树模型，但在实现过程中并没有考虑网络的通信代价和其他限制，比如节点负载是否均衡、QoS（服务质量）是否满足等，而对于云中多租户数据副本放置，这些都是需要考虑的因素。

文献[10]证明了在一般网络图上进行副本放置问题是 NP(non-deterministic polynomial)完全问题，并通过将图简化成树模型，在考虑基于请求延迟限制的基础上，提出一种基于动态规划的副本放置算法。文献[11]提出了 p-median、p-center 和多目标等几种基于启发式方法的副本放置算法，在副本放置中考虑网络延迟和用户请求，并将其结果和利用整数线性规划得出的最优解进行了比较。但以上几种算法都仅仅考虑将独立的某个数据的 p 个副本如何以最优方式进行放置，而多租户数据是将 M 个租户数据共同存储，独立数据的最优放置并不能带来全局共享数据的最优放置。

为确保 SaaS 多租户数据在云中的可用性，文献[12]证明了副本数量的阈值。副本数量过多，会产生副本开销，包括相应的存储代价和一致性维护代价，副本数量过少，会影响数据可用性。基于此，本文在研究过程中，将数据副本的数目设置在一定取值范围$[R_{\min}, R_{\max}]$。

综上所述，SaaS 模式下，租户租赁定制 SaaS 应用，应用部署在云中，承载 SaaS 应用的云无法直接了解租户的需求。而现有的云中副本管理技术[13]，虽然直接面对租户，却无法应用到 SaaS 环境。其他领域的副本放置技术也缺乏对多租户数据的支持。因此，本文将结合 SaaS 多租户数据共享存储、数据划分等特点，针对节点不同的计算能力和负载能力，通过进一步优化调整数据放置中的副本数目和位置，在满足租户 SLA 需求的同时，使服务提供商维护和管理数据的成本代价尽量最小化。

3 问题描述与建模

SaaS 多租户数据副本在云中多节点放置问题如图 1 所示。

前期工作[14]提出在模式不变的情况下，把在不同事务中多次被一起访问的多租户数据（定义为具有较高相关性）聚集在一起，作为数据划分粒度 group，减少分布式事务代价。本文将延续，以 group 作为数据放置单位。图 1 中相关符号及深层定义如表 1 所示。

租户的访问请求，首先经过 Master 处理，获得各数据副本的位置列表等控制信息，然后经通信链路直接从相应数据节点访问数据。通信链路实现数据读写过程中应用节点与数据节点的交互，读写的

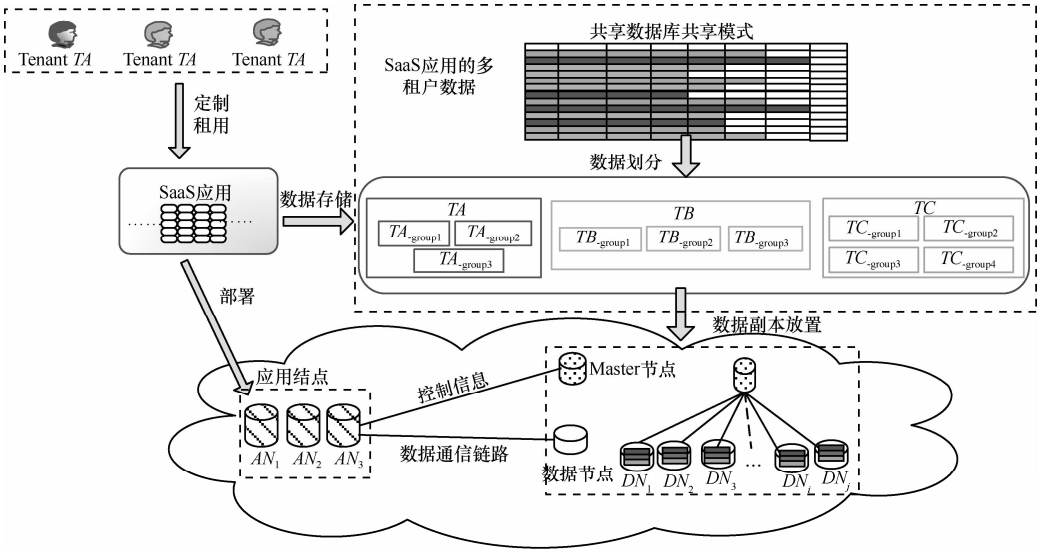

图 1　SaaS 多租户数据放置

表 1　　**符号定义**

符号	说明
$T=\{TA, TB, TC\cdots\}$	定制租用 SaaS 应用的租户集合
$TX=\{TX_{\text{-group1}}, TX_{\text{-group2}},\cdots,TX_{\text{-groupy}}\cdots\}$	采用共享数据库共享模式存储的多租户数据，经过数据划分，租户 TX 形成的数据集合，group 数目因租户而异
$RTX_{\text{-groupy}}$	数据 $TX_{\text{-groupy}}$ 的副本数目，满足 $R_{\min}\leqslant RTX_{\text{-groupy}}\leqslant R_{\max}$，$R_{\min}$ 是确保云中数据可用性设置的最小副本数，$R_{\max}$ 是为减少副本开销设置的最大副本值
Master 节点	Master 节点，存储控制元数据信息
$DN=\{DN_1, DN_2\cdots DN_i, \cdots\}$	数据节点集合，是数据存储和读写的载体。
$DNi=\{ C_{DN_i},\overline{S}_{DN_i},\overline{\lambda}_{DN_i} \}$	分别表示数据节点 DN_i 的计算能力、DN_i 存储空间的上限值、DN_i 在负载均衡状态下的理想负载
$AN=\{AN_1, AN_2,\cdots, AN_i,\cdots\}$	应用节点集合，其上部署 SaaS 应用，租户可从多个节点定制租赁应用，但应用不会跨界点部署
$R^i_{X,y}=1$	对数据 $TX_{\text{-groupy}}$ 的访问请求自应用节点 AN_i 发出
$M^i_{X,y}=1$	表示数据 $TX_{\text{-groupy}}$ 的一个副本存储在节点 DN_i 上

数据在通信链路上传输。

SaaS 模式下，租户 SLA 需求以及服务提供商的利益成本有不同的计算模型[15]。下面给出本文的具体描述和定义。

定义 1　(租户 SLA 需求)　本文选择响应时间作为衡量满足租户 SLA 需求的标准。若系统中所有数据节点处于负载均衡状态，则任何租户的访问请求都能在 SLA 规定的服务响应时间内完成，本文称为满足租户 SLA 需求。

定义 2　(负载标识)　一段时间内，系统负载记为 $\lambda=[\lambda_{TA}, \lambda_{TB},\cdots, \lambda_{TM}]$；任一租户 TX 的访问负载记为 $\lambda_{TX}=\lambda^w_{TX}+\lambda^r_{TX}$ 由读负载和写负载 2 部分构成；数据 $TX_{\text{-groupy}}$ 的负载记为 $\lambda_{TX_{\text{-groupy}}}=\lambda^w_{TX_{\text{-groupy}}}+\lambda^r_{TX_{\text{-groupy}}}$，其存储于不同节点的任一副本的负载记为

$$\lambda'_{TX-\text{groupy}}=\lambda'^w_{TX-\text{groupy}}+\lambda'^r_{TX-\text{groupy}}$$

当系统所有数据节点 DN_i 的负载值都满足 $(1-\beta)\overline{\lambda}_{DN_i}\leqslant\lambda_{DN_i}\leqslant(1+\beta)\overline{\lambda}_{DN_i}$ 时，称系统负载均衡。β 为系统设置参数。文本假设 $\overline{\lambda}_{DN_i}$ 是 DN_i 根据自身计算能力承受的理想负载，即 $\overline{\lambda}_{DN_i}=\dfrac{C_{DN_i}}{\sum\limits_{DN_i\in DN}C_{DN_i}}\lambda$。

定义 3　(数据节点维护成本 $Cost_p$)　开启一个数据节点，单位时间内产生的维护成本代价假设为 P，系统节点总数为 Z 时，单位时间内数据节点维护成本记为 $Cost_p=ZP$。

本文采用“读一个写全部（read-one-write-all）”[16]模式，将租户的写请求分配给所有数据副本处理，将读请求传播至某个数据副本响应，并且每个数据

副本均匀分担数据读负载，则 $\lambda'^{w}_{TX-\text{group}y}=\lambda^{w}_{TX-\text{group}y}$，$\lambda'^{r}_{TX-\text{group}y}=\dfrac{\lambda^{r}_{TX-\text{group}y}}{R_{TX-\text{group}y}}$。读写过程中，数据传输经过通信链路产生通信代价，令矩阵 $\boldsymbol{AD}[i,j]$表示应用节点 AN_i 与数据节点 DN_j 间的带宽值，值越大传输时间越短，通信代价越小。

定义 4 (数据写过程通信代价) 为维护数据副本之间的一致性，写请求被传播至数据副本所在所有节点进行同步。数据经过通信链路传输将产生相应通信代价。写过程通信代价与数据的写负载及矩阵 $\boldsymbol{AD}$ 的带宽值有关，记为

$$Cost_w=\sum_{TX\in T,TX_{-\text{group}y}\in Data(TX)}[\lambda^{w}_{TX_{-\text{group}y}}(\sum_{M^{i}_{X,y}=1,R^{i}_{X,y}=1}\boldsymbol{AD}[i,j])]$$

定义 5 (数据读过程通信代价) 数据副本所在的某一个节点响应读请求。数据经过通信链路产生的通信代价与数据的读负载及矩阵 $\boldsymbol{AD}$ 的带宽值有关，记为

$$Cost_r=\sum_{TX\in T,TX_{-\text{group}y}\in Data(TX)}[\lambda'^{r}_{TX_{-\text{group}y}}(\sum_{M^{i}_{X,y}=1,R^{i}_{X,y}=1}\boldsymbol{AD}[i,j])]$$

定义 6 (整体成本代价) 本文整体成本代价定义为 SaaS 服务提供商维护所有数据节点的成本，以及数据访问过程中产生的通信代价成本的加权和。即 $COST=aCost_p+bCost_w+cCost_r$，其目标是最小化整体成本代价 $COST$，其中 $a,b,c\in[0,1]$。数据放置过程中，本文假设本着优先选择现存节点的原则，令权值 $a>b,a>c$。

4 SaaS 多租户数据放置的优化调整策略及算法

针对 SaaS 租户和 SaaS 服务提供商不同的目标要求，SaaS 多租户数据放置分为 3 个阶段：初始放置、新数据放置和优化调整，如图 2 所示。本文主要解决优化调整阶段的问题。

4.1 初始放置

初始状态，租户数目少数据量小，租户数据经初始划分，形成最初的 group[14]集合。系统启用 $R_{\min}$ 个数据节点，可实现初始数据的放置，副本位于不同节点。

4.2 新数据放置

系统发展过程中不断产生新数据记为 D^{i}_{new}，新数据放置阶段，决定哪些数据节点存储新数据副本。通过判断新数据 D^{i}_{new} 是否属于现有租户 TX，进一步是否属于 TX 现有的 group，将采用 3 种不同的策略进行放置。使同一租户属于同一 group 的数据放置在一起，放置过程中保持副本之间通信代价最小。此部分内容已在前期工作[17]中详细描述。

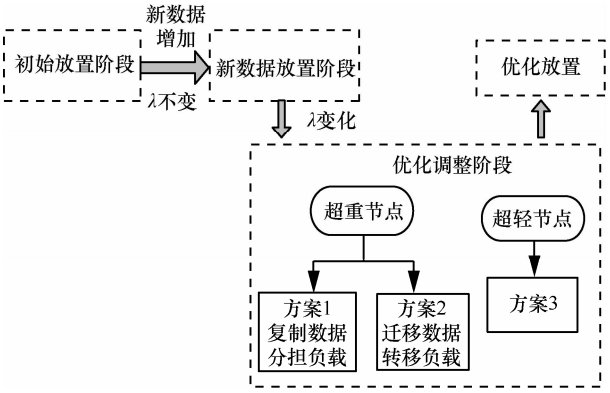

图 2 SaaS 多租户数据放置 3 个阶段

4.3 优化调整策略

云环境的动态性，导致系统负载发生变化，影响数据节点的负载均衡，使访问请求的响应时间出现延迟，违背租户 SLA 需求。优化调整阶段，针对负载的变化，通过调整数据副本的数目及位置，使所有数据节点的负载都在正常范围，满足租户 SLA 对响应时间的需求，同时最小化整体成本代价。

优化调整策略根据负载情况将数据节点分为负载超重节点和负载超轻节点（简称超重节点和超轻节点）2 种情况来处理。本文假设当节点 DN_i 的实际负载 $\lambda_{DN_i}>(1+\beta)\overline{\lambda}_{DN_i}$ 时，称为超重节点，$\lambda_{DN_i}<(1-\beta)\overline{\lambda}_{DN_i}$ 时，称为超轻节点。β 为系统设置的参数。策略中 $\lambda'^{\text{old}}_{TX_{-\text{group}y}}$、$\lambda'^{\text{new}}_{TX_{-\text{group}y}}$ 分别表示副本数目变化前、后，数据 $TX_{\text{-group}y}$ 在每个数据节点上的负载值。

4.3.1 超重节点调整策略

负载超重的节点，通过“复制数据分担负载”（方案 1）或者“迁移数据转移负载”（方案 2）进行调整，调整过程中选择成本代价最小的方案执行。具体步骤如下。

1) 超重节点 DN_i 上所有数据 $TX_{\text{-group}y}$，按照访问负载 $\lambda'_{TX_{-\text{group}y}}$ 值由大到小排序。其他节点 DN_p 按节点负载由小到大排列。图 3(a)中，负载超重节点 DN_1（粗线节点）上访问频率最高的数据为 $TB_{\text{-group1}}$（粗线数据），其他节点按访问负载排列为：DN_3，DN_4，DN_2。

2) 判断所有 $TX_{\text{-group}y}$ 的副本数目，若 $R_{\min}\leqslant$

$R_{TX\text{-group}y}<R_{\max}$，则有 2 种调整方案。方案 1 采用“复制数据分担负载”的方法，将 $TX_{\text{-group}y}$ 复制到合适的目标节点，副本数目增加，均分访问负载，使 $\lambda'_{TX_{-\text{group}y}}$ 值降低，节点 DN_i 的负载值也相应降低。方案 2 采用“迁移数据转移负载”的方式，将 $TX_{\text{-group}y}$ 复制到合适的目标节点，并在源节点删除，负载也相应的转移。2 种方案在选择目标节点时，满足目标节点与 $TX_{\text{-group}y}$ 副本所在其他节点通信代价之和最小。最后比较 2 个方案的 *COST* 值，选 *COST* 值最小的执行，确保在优化调整过程中，系统整体成本代价的最小化。若 $R_{TX\text{-group}y}=R_{\max}$，则直接采用方案 2 调整，否则，副本数目超过 $R_{\max}$ 会增加系统成本代价。

图 3(b)和图 3(c)描述了采用 2 种方案将数据 $TB_{\text{-group1}}$ 复制或者迁移到现有节点的情况，若现有节点的容量有限或者负载不满足条件，则需要加入新节点作为 $TB_{\text{-group1}}$ 的目标节点。

3) 重复判断执行步骤 2)，直到节点 DN_i 的负载范围正常为止。

4.3.2 超重节点优化调整算法

为方便描述，将策略中计算整体成本代价、求最小通信代价之和以及“复制数据分担负载”和“迁移数据转移负载”处理方案定义为算子。详细算子及算法如下。

算子 1　与已知节点集合通信代价和最小的节点

输入：数据节点集合 *F*，数据 $TM_{\text{-group}n}$；

输出：目标节点 DN_g

min=系统初设一个最小值；

FOR （*F* 中每个节点 DN_p）

{ *DJ*=0;

FOR (*AD* 中每个应用节点 AD_i)

IF ($R^i_{M,n}=1$) *DJ*=*DJ*+*AD*[*i*, *p*];

IF *DJ*<min THEN { min=*DJ*; DN_g=DN_p}

} RETURN (DN_g)

算子 2　整体成本代价 *COST*

输入：节点集合 *DN*，系统负载 λ；

输出：*COST*

$COST=aCost_p+bCost_w+cCost_r$，

RETURN (*COST*)

算子 3　超重节点处理方案 1

输入：$TX_{\text{-group}y}$，DN_i，*DN*，λ，DN_m；

输出：*COST*

$F=\{DN_p|DN_p\in DN,\ DN_p\neq DN_i,\ DN_p$ 按负载 λ_{DN_p} 由小到大排列$\}$；　$L=\Phi$

FOR (*F* 中每个节点 DN_p)

IF $(((S_{DN_p}+S_{TX_{-\text{group}y}})<\overline{S}_{DN_p})\ \&\&\ (M^p_{X,y}=0)\ \&\&\ ((\lambda_{DN_p}+\lambda'^{\text{new}}_{TX_{-\text{group}y}})<(1+\beta)\overline{\lambda}_{DN_p}))$ HEN $L=L\cup\{DN_p\}$

IF ($L\neq\Phi$) THEN

{ DN_m=算子 1（*L*，$TX_{\text{-group}y}$），令：$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'^{\text{old}}_{TX_{-\text{group}y}}+\lambda'^{\text{new}}_{TX_{-\text{group}y}}$，$\lambda_{DN_m}=\lambda_{DN_m}+\lambda'^{\text{new}}_{TX-\text{group}y}$ }

ELSE { 新节点作为目标节点 DN_m，令：$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'^{\text{old}}_{TX_{-\text{group}y}}+\lambda'^{\text{new}}_{TX_{-\text{group}y}}$，$\lambda_{DN_m}=\lambda_{DN_m}+\lambda'^{\text{new}}_{TX_{-\text{group}y}}$ }

COST=算子 2(*DN*，λ)

RETURN (*COST*)

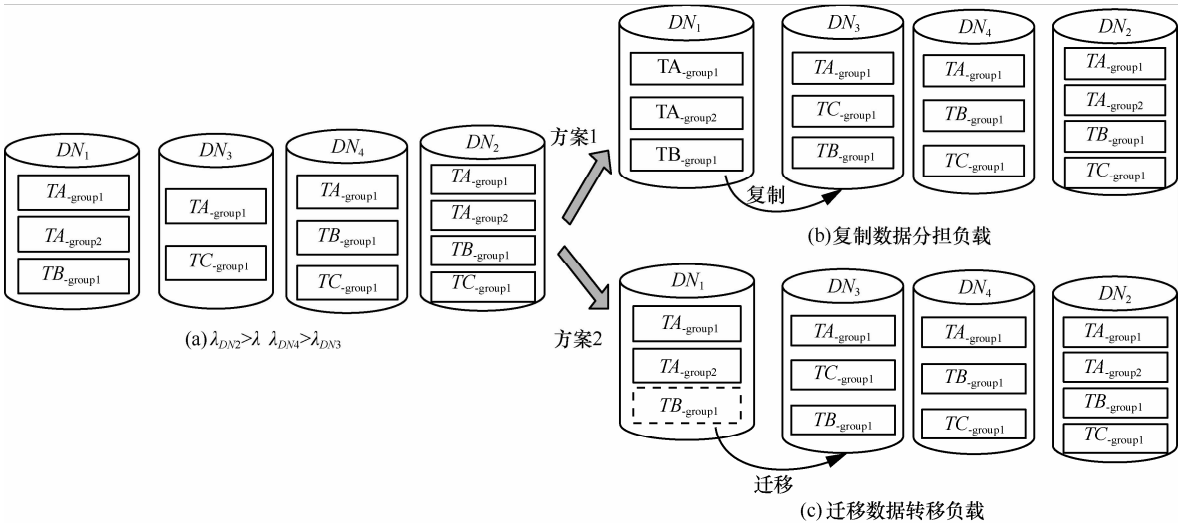

图 3　超重节点调整

算子 4 超重节点处理方案 2

输入：TX-group$_y$，DN_i，DN，λ，DN_e；

输出：$COST$

F={DN_p|D$N_p \in DN$, $DN_p \neq DN_i$, DN_p 按负载 λ_{DN_p} 由小到大排列}； $L=\Phi$；

FOR (F 中每个节点 DN_p)

IF ($((S_{DN_p}+S_{TX\_\text{group}y})<\overline{S}_{DN_p})$ && $(M^p_{X,y}=0)$ && $((\lambda_{DN_p}+\lambda'_{TX-\text{group}y})<(1+\beta)\overline{\lambda}_{DN_p})$) THEN

$L=L\cup\{DN_p\}$

IF ($L\neq\Phi$) THEN

{ DN_m= 算子 1(L， TX-group$_y$)， 令：$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'_{TX-\text{group}y}$， $\lambda_{DN_e}=\lambda_{DN_e}+\lambda'_{TX-\text{group}y}$ }

ELSE {系统加入新节点作为目标节点 DN_e，令：$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'_{TX-\text{group}y}$，$\lambda_{DN_m}=\lambda_{DN_m}+\lambda'_{TX-\text{group}y}$ }

$COST$=算子 2(DN，λ)

RETURN ($COST$)

算法 1 负载超重节点优化调整算法

输入：存在负载超重节点的系统节点集合 DN;

输出：没有负载超重的系统节点集合 DN

FOR (每个超重节点 DN_i)

{ F={TX-group$_y$| $M^i_{X,y}=1$，按 $\lambda_{TX\text{-group}y}$ 值降序排列}

$COST1$=0, $COST2$=0, DN_m=null，DN_e=null

FOR (F 集合中每个 TX-group$_y$) DO

{IF ($R_{\min}\leqslant R_{TX\text{-group}y}<R_{\max}$)

{ $COST1$=算子 3(TX-group$_y$，DN_i，DN，λ，DN_m)

$COST2$=算子 4(TX-group$_y$，DN_i，DN，λ，DN_e)

IF ($COST1\leqslant COST2$) THEN {

将 TX-group$_y$ 复制到目标节点 DN_m，并修改

$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'^{\text{old}}_{TX-\text{group}y}+\lambda'^{\text{new}}_{TX-\text{group}y}$， $\lambda_{DN_m}=\lambda_{DN_m}+\lambda'^{\text{new}}_{TX-\text{group}y}$， $M^m_{X,y}=1$ }

ELSE {将 TX-group$_y$ 复制到目标节点 DN_e，并从源节点 DN_i 删除，并修改

$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'_{TX-\text{group}y}$， $\lambda_{DN_e}=\lambda_{DN_e}+\lambda'_{TX-\text{group}y}$， $M^e_{X,y}=1$， $M^i_{X,y}=0$ } }

IF ($R_{TX\text{-group}y}=R_{\max}$)

{算子 4 (TX-group$_y$，DN_i，DN，λ，DN_e)

将 TX-group$_y$ 复制到目标节点 DN_e，

并从源节点 DN_i 删除，修改

$\lambda_{DN_i}=\lambda_{DN_i}-\lambda'_{TX-\text{group}y}$， $\lambda_{DN_e}=\lambda_{DN_e}+\lambda'_{TX-\text{group}y}$，

$M^e_{X,y}=1$， $M^i_{X,y}=0$ }

} WHILE (DN_i 不超重) }

RETURN (DN)

4.3.3 超轻节点调整策略

系统中出现超轻节点时，若整体成本代价大于删除此超轻节点后的整体成本代价，则系统进行优化调整。删除超轻节点（简称方案 3），可能会导致某些租户数据的副本数小于 $R_{\min}$，此时不能简单抛弃这些副本，而是寻找合适的目标节点将其迁移。否则，允许超轻节点的存在。具体步骤如下。

1) 将超轻节点 DN_i 上所有数据 TX-group$_y$ 进行分类。副本数在（$R_{\min}$, $R_{\max}$]范围内的归入集合 $L1$，副本数目等于 $R_{\min}$ 的归入集合 $L2$。并计算此时系统的成本代价，记为 $COST1$。

图 4(a)中超轻节点 DN_2（粗线节点）上数据 TA-group$_1$ 副本总数为 4，大于 3（即 $R_{\min}$），归入集合 $L1$，数据 TC-group$_1$ 副本总数为 3，归入集合 $L2$。

2) 分别处理 $L1$ 和 $L2$ 内的数据。找到 $L1$ 内各数据其他副本所在的节点，若将 $L1$ 内数据 TX-group$_y$ 删除，数据 TX-group$_y$ 各副本因为副本数目的减少，会分摊更多负载，导致其他副本所在节点的负载值增加，若因此出现超重节点，则继续进行调整。$L2$ 内的数据，将其迁移到最合适的目标节点。调整后计算此时系统的成本代价，记为 $COST2$。

如图 4(b)所示，若将超轻节点删除（虚线节点表示删除），则 $L1$ 内数据 TA-group$_1$ 可以直接删除，$L2$ 内数据 TC-group$_1$ 迁移至满足条件的目标节点 DN_1。

3) 比较 $COST$ 值。若 $COST2$ 值最小，则系统可以删除超轻节点，并修改相应信息。否则，系统保持超轻节点不变。

4.3.4 超轻节点优化调整算法

为方便算法描述，将步骤 2 定义为简单算子。算子及算法详细描述如下。

算子 5 超轻节点上的数据处理

输入：$L1'$, $L2'$,λ，DN_m；输出:$COST$

FOR each (TX-group$_y$) in $L1'$

{存储 TX-group$_y$ 副本的所有其他节点 DN_p，令 $\lambda_{DN_p}=\lambda_{DN_p}-\lambda'^{\text{old}}_{TX-\text{group}y}+\lambda'^{\text{new}}_{TX-\text{group}y}$ }

FOR each (TX-group$_y$) in $L2'$

{迁移 TX-group$_y$ 至满足如下条件的节点 DN_p,

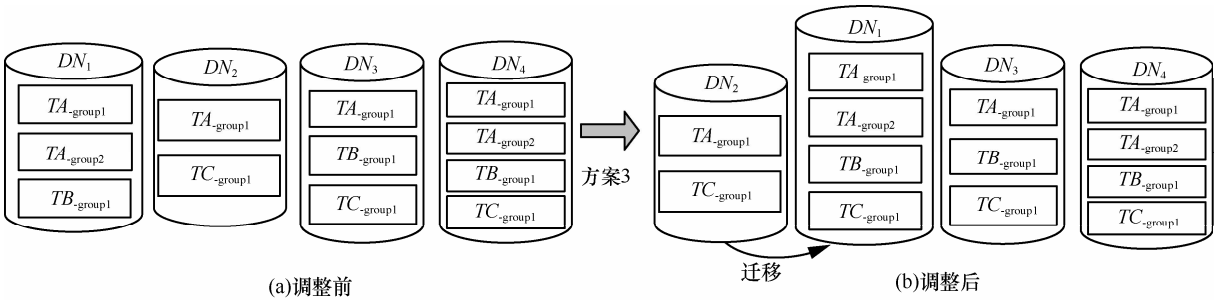

图 4　超轻节点调整图例

$((S_{DN_p}+S_{TX\text{-group}y})<\overline{S}_{DN_p})\ \&\ \&((\lambda_{DN_p}+\lambda'_{TX-\text{group}y})<(1+\beta)\overline{\lambda}_{DN_p})\ \&\ \&(M^p_{X,y}=0)$ 令 $\lambda_{DN_p}=\lambda_{DN_p}+\lambda'_{TX-\text{group}y}$ }

IF（$\exists DN_i\in DN,\lambda_{DN_i}>(1+\beta)\overline{\lambda}_{DN_i}$）算法 1（*DN*）

DN=*DN*−{DN_m}；*COST*=算子 2(*DN*，λ)

RETURN (*COST*)

算法 2　超轻节点优化调整算法

输入：系统节点集合 *DN*，系统负载 λ；

输出：系统节点集合 *DN*

FOR (每个超轻节点 DN_i)

{ *COST*1= 算子 2(*DN*，λ)

add all $TX_{\text{-group}y}$ satisfy ($R_{\min}<R_{TX\text{-group}y}\leqslant R_{\max}$) into *L*1，

add all $TX_{\text{-group}y}$ satisfy ($R_{TX\text{-group}y}=R_{\min}$) into *L*2

*L*1'=*L*1，*L*2'=*L*2，$DN_m=DN_i$

*COST*2=算子 5（*L*1'，*L*2'，λ，DN_m）

IF　(*COST*2<*COST*1)　THEN

{删除源节点 DN_i，更新节点负载及放置信息 }

ELSE　系统保持此超轻节点 }

RETURN　(*DN*)

4.4　算法复杂性分析

N 是系统现存的节点数。优化调整阶段，对于超重节点的调整算法，时间复杂度为 $O(N^2D)$，*D* 为数据节点上以 group 为单位的数据数目，对超轻节点的调整算法，时间复杂度为 $O(N^3D)$。所以本文提出的 SaaS 多租户数据放置的优化调整策略的时间复杂度为 $O(N^3D)$。

5　实验分析

本节通过模拟实验将本文提出的 SaaS 多租户数据放置的优化调整策略（以下简称本文策略）与传统数据放置算法中的贪婪放置策略和随机放置策略进行对比实验，从节点负载（可以看出对租户 SLA 访问时间的响应）及整体成本代价分析本文策略的性能。

5.1　实验设置与环境

本文实验数据采集自课题组项目-职业资格服务体系公共基础支撑平台中公共服务应用的数据，各应用通过 PaaS 平台部署成为 SaaS 应用，被诸多租户租赁使用。数据采用共享模式进行存储，使用 MySQL 数据库。实验环境如下。

数据节点：HP 服务器/DELL 服务器，CPU 为 4×2.26 GHz/2×2.26 GHz，内存为 4 GB/2 GB，硬盘 300/100 GB；Master 节点：DELL 服务器，CPU 为 4×2.26 GHz，内存为 8 GB，硬盘 500 GB。

数据放置采用贪婪策略的核心思想是将数据优先放置到现存剩余空间最大的节点中，若没有满足条件的节点，则产生新节点。而随机放置策略是选择容量不超上限的节点，随机产生一个有效解。

5.2　结果分析

1) 实验一验证放置策略对平均响应时间的影响。实验部署一个 SaaS 应用，5 个数据节点和 20 个租户。分别采用 3 种不同的放置策略将数据放置在 5 个数据节点。

实验采用 FCFS（先来先服务）策略响应各租户查询请求。根据 3.2 节的定义，节点负载不超过阈值 $\overline{\lambda}_{DN_i}$ 时，存储在节点上的租户请求都能在要求的响应时间内完成，符合租户 SLA 需求。$R_{\min}$ 取 3，$R_{\max}$ 取 5，租户初始数据量 10 000~40 000 条记录/租户。数据节点的维护成本取值为 10，单位为 min^{-1}，统计一段时间内系统的负载 λ，即数据访问请求数目，令读写负载数值相同。不同放置策略对平均响应时间的影响如图 5 所示。

本文策略在数据放置过程中，考虑多租户数据特征，由图 5 可看出，查询的平均响应时间与随机策略和贪婪策略比较起来要明显降低。

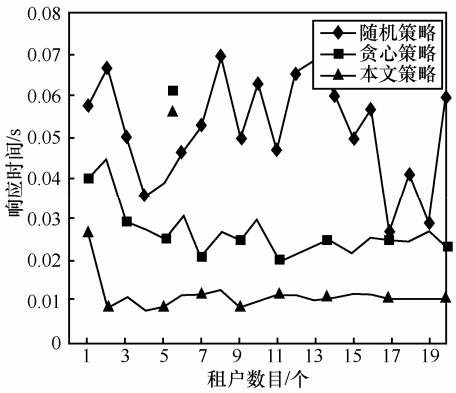

图 5 比较 3 种策略的平均响应时间

2) 实验二验证系统负载发生变化过程中，几种策略在整体成本代价和数据节点负载方面的性能对比。

随着系统数据量的增长，系统规模扩展为 8 个节点包含 30 个租户，3 种不同的放置策略将数据放置在 8 个节点。在此过程中若系统负载 λ 变化，出现某些租户访问频率增加，导致产生负载超重节点。根据 3.1 节所述，节点 DN_i 的性能目标是希望实际负载 λ_{DN_i}，保持在正常范围满足 $(1-\beta)\bar{\lambda}_{DN_i} \leqslant \lambda_{DN_i} \leqslant (1+\beta)\bar{\lambda}_{DN_i}$。本实验中令 β=10%，若所有节点 DN_i 的实际负载值 λ_{DN_i} 与理想负载值 $\bar{\lambda}_{DN_i}$ 差别小于 10%，则认为系统节点负载均衡。因此，实验定义每个节点 DN_i 的负载差别率 $diff_i = \dfrac{|\bar{\lambda}_{DN_i} - \lambda_{DN_i}|}{\bar{\lambda}_{DN_i}}$，系统负载最大差别率 $Diff_{\max}=\max(diff_i)$，利用 $Diff_{\max}$ 来衡量放置策略对系统所有节点负载的影响。3 种策略对系统整体成本代价及 $Diff_{\max}$ 值的影响如图 6 和图 7 所示。

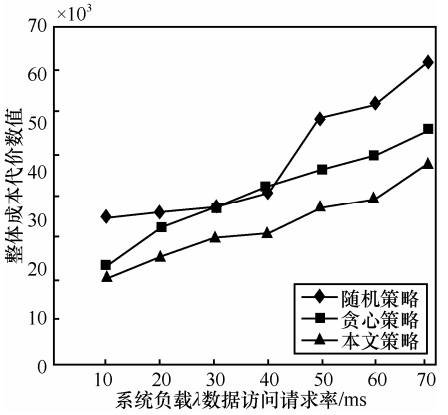

图 6 λ 变化对整体成本代价的影响

图 6 中，3 种数据放置策略对应的整体成本代价都呈现明显的上升趋势。当新节点加入系统过程时，整体成本代价会明显升高。本文策略对应的成本代价要低于随机策略和贪婪策略。图 7 中，3 种策略的系统最大负载差别率都呈现上升趋势，由于本文策略在数据放置过程中考虑节点的负载并进行优化调整，其差别率要低于其他 2 种放置策略。

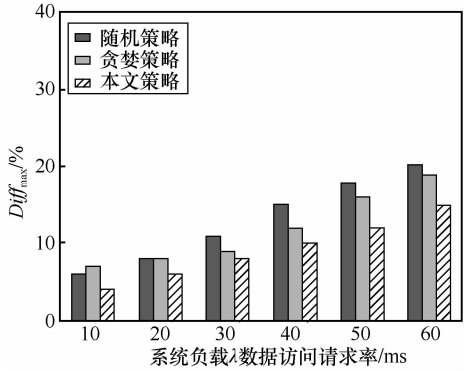

图 7 λ 变化对 $Diff_{\max}$ 的影响

综上分析，随机数据放置策略算法简单，数据放置过程中没有考虑多租户数据的特点，容易造成同一租户数据跨节点存储，产生大量访问成本。贪婪数据放置策略，主要从优化节点存储空间角度考虑，但数据节点的负载性能没有取得较好效果。本文策略，针对多租户数据特征，将数据以 group 为单位进行放置，避免因数据分布存储带来的分布式访问代价，并且根据节点负载情况进行放置优化，在满足所有租户 SLA 需求的情况下，最小化整体成本代价。

6 结束语

本文针对数据副本在云中多节点放置问题，提出了支持 SaaS 应用多租户数据副本放置的优化调整策略和算法。该策略充分考虑多租户数据特征和节点访问负载情况，在满足所有租户 SLA 需求的条件下，降低服务提供商数据维护的成本代价，满足双方目标寻找最优数据副本放置。

SaaS 模式的另外一个特点是允许租户在系统运行过程中按需定制，一旦发生定制模式的改变，对于新定制产生的表及其相应的数据如何更好地选择副本单位并进行最优放置，是在接下来的研究中需要深入探讨的问题。

参考文献：

[1] CRAIG D W, CRAIG D W. The design of the force.com multitenant

internet application development platform[A]. The International Conference on Management of Data and 28th Symposium on Principles of Database Systems[C]. Rhode Island, 2009.889-896

[2] WU L L, SAURABH K G, RAJKUMAR B. SLA-based resource allocation for software as a service provider in cloud computing environments[A]. The 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing[C]. United States, 2011.195-204.

[3] Multi-tenant data architecture[EB/OL]. http://msdn.microsoft.com/en-us/library/aa479086.aspx, 2006.

[4] WOLFGANG L, KAI U S. Web-scale Data Management for the Cloud[M]. Germany: Springer New York Heidelberg Dordrecht London, 2013.

[5] KONG L J. Research on key technology in multi-tenant cloud data management for SaaS application delivery platform[D]. Jinan: Shandong University, 2011.

[6] SHVACHKO K, KUANG H, RADIA S. The Hadoop distributed file system[A]. The IEEE 26th Symposium on Mass Storage Systems and Technology[C]. United states, 2010.105-107.

[7] DeCADIA G, HASTORUN D. Dynamo: Amazon's highly available key-value store[A]. The 21st ACM SIGOPS Symposium on Operating Systems Principles[C]. New York, 2007.

[8] AL-MISTARIHI H H E, YONG C H. On fairness, optimizing replica selection in data grids[J]. IEEE Transactions on Parallel and Distributed Systems, 2009, 20(8):1102-1111.

[9] KALPAKIS K, DASGUPTA K, WOLFSON O. Optimal placement of replicas in trees with read, write, and storage costs[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 12(6):628-637.

[10] TANG X, XU J. Qos-aware replica placement for content distribution[J]. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(10): 921-932.

[11] RASHEDUR M, KEN B, REDA A. Replica placement stragegies in data grid[J]. Journal of Grid Computing, 2008, 6(1): 103-123.

[12] WU N, ZHANG S D, KONG L J. Dynamic adaptive model of the multi-tenant data replication based on queuing theory[A]. Proceedings of 2nd International Conference on Computer Science and Network Technology[C]. United States, 2012.1755-1759.

[13] RAHMAN R M, BARKER K, ALHAJJ R. Replica placement design with static optimality and dynamic maintainability[A]. The 6th IEEE International Symposium on Cluster Computing and the Grid[C]. Singapore, 2006.434-437.

[14] 李晓娜, 李庆忠, 孔兰菊. 基于共享模式的 SaaS 多租户数据划分机制研究[J]. 通信学报, 2012,33(Z1):110-120.
LI X N, LI Q Z, KONG L J. Research on multi-tenant data partition mechanism for SaaS application based on shared schema[J]. Journal on Communications, 2012, 33(Z1):110-120.

[15] AMZA C, COX A L, Z Waenepoel W. Conflict-aware scheduling for dynamic content application[A]. The 4th USENIX Symposium on Internet Technologies and Systems[C]. Seattle, WA, USA, 2003. 328-347.

[16] LIU, Z Y, HACIGUMUS H, *et al*. PMAX: Tenant placement in multitenant database for profit maximization[A]. The 16th International Conference on Extending Database Technology, EDBT 2013[C]. United States, 2013.442-453.

[17] LI X N, LI Q Z, KONG L J. Research on multi-tenant data placement strategy for SaaS application based on SLA-COST[J]. Journal of computational information systems, 2014, 10(17):7499-7506.

作者简介：

李晓娜（1980-），女，山东淄博人，青岛大学讲师，山东大学博士生，主要研究方向为云计算、多租户数据管理。

李庆忠（1965-），男，河北威县人，山东大学教授、博士生导师，主要研究方向为大规模网络数据管理和 Web 数据集成。

孔兰菊（1978-），女，山东菏泽人，山东大学讲师，主要研究方向为数据库、数据管理。

闫中敏（1977-），女，山东济南人，山东大学副教授，主要研究方向 Web 数据集成、数据库。