高技术通讯 2013 年 第23 卷 第4 期:352 ~ 359

doi:10.3772/j.issn.1002-0470.2013.04.004

多租户数据库即服务模式系统资源优化分配方法①

亓开元②*** 赵卓峰** 张 东* 刘正伟*

(*高效能服务器和存储技术国家重点实验室 济南 250101)

(**北方工业大学云计算研究中心 北京 100144)

(***中国科学院计算技术研究所 北京 100190)

摘 要 针对数据库即服务(DBaaS)模式下共享服务器的多租户之间的系统资源优化分配问题进行了研究,基于虚拟化技术提出了此问题的约束规划模型和求解方法。此求解方法通过建立性能模型计算特定资源配置虚拟机所能提供的数据库服务性能,且着重考虑了副本的一致性因素;通过定义效用函数,从资源使用量和性能需求两个方面度量资源分配效果。在此基础上,充分考虑虚拟化开销因素,通过两阶段的贪心算法快速寻找问题的近似最优解。应用示例及实验表明,该方法能够在满足租户性能需求的同时优化资源使用量,而且由于着重考虑了副本一致性、虚拟化开销和调整粒度等因素,因而能使资源分配更为准确和可控。

关键词 数据库即服务,多租户系统资源共享,按需资源分配,性能模型,效用函数,两阶段贪心算法

0 引 言

随着云计算技术的发展,传统软件层次栈中的操作系统、中间件和应用纷纷在云中找到了位置,出现了基础设施即服务(infrastructure-as-a-service, IaaS)、平台即服务(platform-as-a-service, PaaS)和软件即服务(software-as-a-service, SaaS)模式。实际上,承载信息的数据库也非常适合构建在云数据中心,但要注意资源分配问题。某全国范围内服务的科技信息网[1]则是一个例子。在其前期建设中,各二级单位独自建立了数据库并提供科技信息服务,但随着应用的深入,逐渐暴露出了资源利用不均衡和维护管理困难等问题。以数据库即服务(database-as-a-service,DBaaS)模式[2]将各单位(租户)的数据库集中托管起来,通过共享服务器实现资源的优化利用,降低软硬件成本和维护管理开销,是解决上述问题的理想方案。本研究基于虚拟化技术,提出了一种在无共享架构的情况下以副本方式实现DBaaS 的资源优化分配的方法,该方法可使资源分配更准确,资源效用更高。

1 相关工作

在多个租户数据库之间共享服务器以提高系统资源利用率是 DBaaS 模式的核心追求[2]。共享方式下,若租户之间的性能隔离得不到保障,某些租户的请求负载就可能占用大部分系统资源,造成资源劫持。而 DBaaS 往往在服务水平协议(service level agreements,SLA)中声明了性能指标,满足租户性能需求是资源优化利用的前提。文献[3]通过在运行时调度各租户数据库的负载来保障性能隔离,但这种反应式方法造成了性能延迟。更主动的方式是为租户数据库分配保障性能的系统资源,这种方式需要依靠虚拟化技术实现。利用虚拟化技术(如 Xen[4]等),能够将物理服务器资源按比例分配给互不干扰的虚拟机。例如,可将一台服务器的系统资源平分给两个虚拟机,各配置 50% 的 CPU 时间、内存容量和 I/O 带宽。利用虚拟化技术进行资源分配的一般过程是确立优化目标,建立系统资源到优化目标的映射模型,搜索解空间求得最优解或可行解。文献[5,6]提出了为数据层分配虚拟资源以优化整

① 973 计划(2012CB724101)资助项目。

② 男,1984 年生,博士;研究方向:云计算,大规模数据处理;联系人,E-mail: 7hy1@163.com

(收稿日期:2011-09-05)

体性能的方法。然而,这些方法或依赖于数据库管理系统内部查询优化器[5],或局限于单台服务器或共享存储[5,6],不适合云数据中心通常采用的无共享(shared-nothing)架构。

在无共享架构下,为应对动态变化的数据库请求负载,通常采用副本或分片方式提供伸缩性。分片适用于 TB 以上海量数据的数据库服务,但在分片方式下保持关系特性将限制伸缩能力。与分片相比,副本方式具有保持关系特性和高可用性的特点。文献[7]依托 Amazon EC2[8]提出了一种以副本方式构造 DBaaS 的资源规划方法,为优化租用成本,按性能需求在 EC2 的 5 种虚拟机类型中进行选择来部署副本,但仅考虑了副本所需承受的负载,无法计算多个副本所能提供的性能,并且也没有考虑虚拟化开销[9]的影响,造成资源分配不准确。文献[10]提出了集中托管 Web 应用的多租户资源分配方法,利用排队理论建立多个应用副本的性能模型,能够针对租户负载配置虚拟机资源,但该方法基于应用层副本的无状态性假设,没有考虑副本的状态一致性。

本文提出了一种在无共享架构下以副本方式实现 DBaaS 的资源优化分配的方法。该方法通过定义效用函数,综合资源使用量和性能因素衡量分配效果;针对部署副本的多个虚拟机,利用性能模型计算在不同资源配置下所能提供的数据库服务性能,其中着重考虑副本的一致性因素。在性能模型和效用函数基础上,充分考虑虚拟化开销因素,以两阶段贪心方式调整资源配置寻找问题的近似最优解。此外,通过分析资源调整粒度对分配效率和效用的影响,使得资源分配更为可控。

2 多租户资源分配方法

基于副本方式,传统数据库托管服务以物理服务器为单位部署租户的数据库副本,往往超过实际所需,资源利用率低。使用虚拟机部署租户数据库副本,可以灵活地控制租户数据库的资源分配并保障性能隔离[11]。例如,租户 A 和 B 的数据库各部署在一台物理服务器上,若租户 A 的负载规模较大,服务器不能满足其性能需求,而租户 B 服务器的资源利用率不足,这样就可以在租户 B 的服务器上划分出一部分系统资源部署租户 A 的数据库副本,在运行时将租户 A 负载路由到两个副本上(如图 1 所示),从而在提升租户 A 性能的同时实现资源优化利用。因此,DBaaS 资源分配可以转化为如下问题:如何在集群上为每个租户配置部署数据库副本的虚拟机,用最少的系统资源满足所有租户的性能需求。本节给出上述问题的约束规划模型和求解算法。

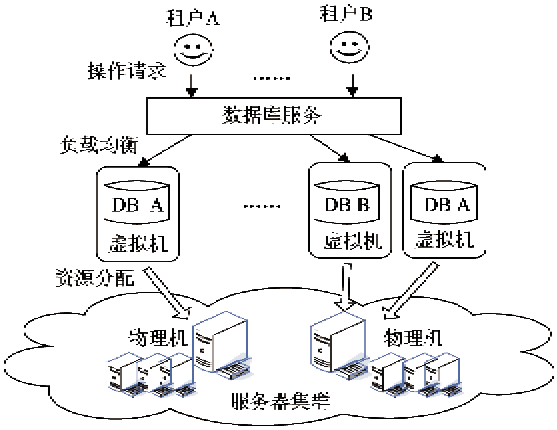

图 1 DBaaS 资源分配

2.1 问题定义

若数据中心有 n 台物理服务器为 m 个租户提供数据库服务,每个租户的数据库服务由多个部署在虚拟机上的数据库副本构成。

首先,定义 $m \times n$ 的资源配置矩阵 $\boldsymbol{X} = (x_{ij})$,其中 x_{ij} 表示租户 i 的数据库副本在服务器 j 上占用的系统资源比例。在服务器 j 上所有租户的资源占用量应该不大于该服务器资源总量,即

$$\tau_j = \sum_{i=1}^{m} x_{ij} \leqslant 1,\ j = 1,2,\cdots,n \tag{1}$$

其次,资源分配要使每个租户 i 的平均响应时间 rt_i 满足性能需求 SLA_i,即

$$rt_i \leqslant SLA_i,\ i = 1,2,\cdots,m \tag{2}$$

最后,定义效用函数度量资源分配的效果,在分配算法中作为优化目标指导资源调整。DBaaS 资源优化分配的目标是用最少的资源满足所有租户的性能需求,因此,效用函数应该包括租户 SLA 是否满足和资源使用量两方面的信息。资源效用函数定义为一个分段函数:

$$U = \begin{cases} \dfrac{1}{n}\sum_{k=1}^{n}\tau_k - 1 & \exists i \in \{1,2,\cdots,m\},\ rt_i > SLA_i \\ 1 - \dfrac{1}{n}\sum_{k=1}^{n}\tau_k & \forall i \in \{1,2,\cdots,m\},\ rt_i \leqslant SLA_i \end{cases} \tag{3}$$

其中,当存在租户 i 的平均响应时间 rt_i 没有达到 SLA_i 要求时,资源效用为负值,表示资源分配应以满

足性能需求为导向，增大租户 i 资源配置提高数据库服务性能将促使资源效用随 rt_i 的减小而增大；在所有租户的平均响应时间满足性能需求时，资源效用为正值，表示资源分配应以减少资源使用量提高资源效用为导向，继续增加资源会导致效用变小。

综上所述，DBaaS 资源分配（DBaaS resource allocation，DRA）可以建模为在约束（式（1）和式（2））下最大化（式（3））的问题。在资源调整粒度为 σ（$\sigma < 1$）的情况下，DRA 可以进一步定义为如下 0/1 规划问题：

$$\max \quad U$$

$$\text{s.t.} \quad x_{ij} = \sum_{k=1}^{p} y_{ij}^{k} \cdot \sigma \quad y_{ij}^{k} = 0,1$$

$$i = 1,2,\cdots,m;\ j = 1,2,\cdots,n$$

$$\sum_{i=1}^{m} y_{ij}^{k} \leqslant 1,\ k = 1,2,\cdots,p;\ j = 1,2,\cdots,n$$

$$\sum_{i=1}^{m} x_{ij} \leqslant 1,\ j = 1,2,\cdots,n$$

$$rt_i \leqslant SLA_i,\ i = 1,2,\cdots,m \tag{4}$$

其中，$p = \lfloor \sigma^{-1} \rfloor$ 表示粒度为 σ 时物理机的虚拟资源划分数；y_{ij}^{k} 表示表示租户 i 在物理机 j 上的系统资源 x_{ij} 是否包括此物理机第 k 个虚拟资源划分；$\sum_{i=1}^{m} y_{ij}^{k} \leqslant 1$ 表示每个划分只能分配给一个租户。

定理 1 DRA 问题是 NP 完全的。

证明：若将虚拟资源看作物品，租户看作背包，DRA 问题可以转化为将 n/σ 个物品向 m 个背包中放置，在每个背包的限定条件下使得总价值最大的问题。也就是说，0/1 背包问题（0/1 Knapsack）是 DRA 问题的特殊情况，可以约化为 DRA 问题，即 0/1 Knapsack $\propto$ DRA。

另外，由于 $m \times n$ 的 DRA 问题中一个租户的资源配置有 $(1/\sigma)^n$ 种，其解空间为 $O((1/\sigma)^{mn})$，可以在多项式排序时间 $O(mn\log\sigma^{-1})$ 内验证一个解是否最优，因此 DRA 问题是一个 NP 问题。

综上，DRA 问题是 NP 的，并且可以由 NP 完全的 0/1 Knapsack 约化而成，因此，DRA 问题是一个 NP 完全问题。

为了求解 DRA 问题，本文假设：SLA 性能需求已知；各租户请求负载可预测；虚拟机系统资源可被数据库充分利用，尤其是内存，可通过配置数据库引擎的相关参数加以利用，按比例分配服务器系统资源即是划分数据库的处理能力。

基于上述假设，本文提出了一种周期性的资源分配方法。每次资源分配中，资源调整模块根据启发式规则调整虚拟机资源配置，性能分析模块根据预测负载计算这种配置方案能够提供的性能，效用计算模块综合性能和资源使用量评估当前配置方案的效果。这三者形成了一个循环搜索过程（如图 2 所示），通过启发式规则控制，最终找到问题的近似最优解。

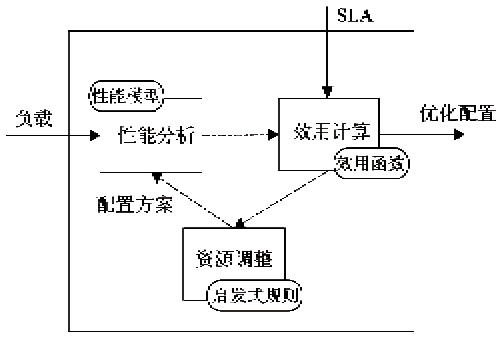

图 2 资源分配方法

2.2 性能模型

性能模型根据负载和资源配置矩阵计算数据库服务的性能，若租户 i 数据库在下一周期单位时间请求负载为 λ_i，该租户的虚拟机资源配置用矩阵 $\boldsymbol{X}$ 的行向量 $\boldsymbol{X}_i$ 表示，副本数为 $\boldsymbol{X}_i$ 中不为零的分量数，该租户的平均响应时间函数为

$$rt_i = \overline{RT}(\lambda_i, X_i) \tag{5}$$

若租户 i 所在服务器 j 对该租户数据库的读写处理能力分别为 R_{ij} 和 W_{ij}，则资源配置 x_{ij} 的读写处理能力分别为 $r_{ij} = x_{ij} \cdot R_{ij}$ 和 $w_{ij} = x_{ij} \cdot W_{ij}$。

与基于应用逻辑层的无状态性进行资源分配的方法[10]不同，数据层的有状态性决定了读写操作负载不是以完全均衡方式分配到各个副本上的。为保障一致性，负载按照读-写全（read one write all，ROWA）模式分发。对于读操作，将其发送到目标数据库的任一副本，对于写操作，将其发送到所有副本。若请求负载中读和写操作概率分别为 α 和 β，$\alpha + \beta = 1$，则读写负载的平均响应时间为

$$\overline{RT}(\lambda_i, X_i) = \alpha RT_r(\alpha\lambda_i, X_i) + \beta RT_w(\beta\lambda_i, X_i) \tag{6}$$

上述模型中，R_{ij} 和 W_{ij} 可通过黑盒测试法[11]确定，读操作平均响应时间函数 RT_r 和写操作平均响应时间函数 RT_w 通过排队模型确定。在 ROWA 模式下，设租户 i 数据库请求负载符合泊松分布，数据库的处理能力服从负指数分布，则

a）读操作请求相当于通过一个异构的 M/M/c

系统,请求到达速率 $\lambda_r = \alpha\lambda_i$,$c$ 个数据库副本的读处理能力 $\mu_{rj} = r_{ij}$, $j = 1,2,\cdots,c$, 由生灭过程求解公式[12],可得

$$p_k = \begin{cases} \prod_{i=1}^{k} \frac{\lambda_r}{\mu_{r1} + \mu_{r2} + \cdots + \mu_{ri}} p_0, & k = 1,2,\cdots,c \\ \prod_{i=c+1}^{k} \frac{\lambda_r}{\mu_{r1} + \mu_{r2} + \cdots + \mu_{rc} + (i-c)\mu_{rc}} p_0 & \\ k = c+1, c+2, \cdots & \end{cases} \quad (7)$$

其中,p_k 为系统中有 k 个请求的概率。

b) 写操作相当于通过只有一个数据库副本的 M/M/1 系统,请求到达速率 $\lambda_w = \beta\lambda_i$, 写处理能力由所有副本中处理能力最差的决定 $\mu_w = \min\{w_{ij}, j = 1,2,\cdots,c\}$, 由生灭过程,可得

$$p_k = \left(\frac{\lambda_w}{\mu_w}\right)^k p_0 \quad k = 1,2,\cdots \quad (8)$$

其中,p_k 为系统中有 k 个请求的概率。

根据排队理论,若 c 个副本的处理能力之和大于请求到达速率,队列将处于稳态,由式(5)(6)可以分别求得读写操作的平均队列长度和正在处理请求数量,以及平均响应时间 RT_r 和 RT_w。

2.3 分配算法

在 DRA 问题中,如果使用动态规划、分枝限界等精确性算法,在问题规模扩大时会导致组合爆炸,因此应采用启发式算法寻找近似最优解。本文基于贪心方式,提出了一种快速寻找资源配置近似最优解的搜索算法即两阶段贪心(two-phase greedy, 2PG)算法。2PG 算法分为两个阶段,约束阶段以满足 SLA 为目标,以贪心方式为每个性能需求未满足的租户增加资源,直到满足 SLA;优化阶段以减少资源使用量为目标,以贪心方式减少租户资源获得效用增量,直到资源效用不再增加。此外,在搜索过程中还引入了以下启发规则:

(1) 若有 m 个租户,n 台服务器 ($m < n$), 初始解 X_0 的构造方法是在 n 台服务器上为每个租户各创建两个数据库副本,每台服务器的系统资源被虚拟机平分。为了保障可用性,在资源调整中每个租户的副本都不能少于两个。

(2) 每次调整时,先确定需增加资源和可减少资源的租户。对于租户 i,当前配置为 X_i,下一周期的负载为 λ_i, 若 $rt_i = \overline{RT}(\lambda_i, w_i, X_i) < SLA_i$, 则 i 可减少资源,反之,需增加资源。若服务器 j 至少有一个可减少资源的租户,则称 j 可调。

(3) 虚拟化将引入额外的资源开销,实验表明虚拟化开销与虚拟机数量 n 成正比[11],即 $cost = c \cdot \omega$, 其中 ω 为虚拟化开销因子。为减少虚拟化开销,在每次资源调整时应先调整租户已有数据库副本所在虚拟机,再新建虚拟机。此外,虚拟化开销应均摊到各个租户上,因此,式(1)中还要考虑由于服务器 j 虚拟化而引入的额外开销 $cost_j$, 即

$$\tau_j + cost_j \leqslant 1, \ j = 1 \cdots N \quad (9)$$

2PG 算法

输入: 租户数 m, 服务器数 n, 租户性能需求向量 SLA, 租户负载向量 λ, 租户读操作概率向量 α, 租户数据库向量 w, 虚拟化开销因子 ω, 资源调整粒度 σ,

输出: 资源配置矩阵 X

```
allocation()
1.  X0 = new Config(m); //初始化矩阵 X0;
2.  X = constraint(X0); //约束阶段
3.  X = optimization(X); //优化阶段
4.  return X;
constraint ()
5.  while hasNextUnmarkedTenant() //存在未满足性能需求的租户
6.    t = nextUnmarkedTenant(); //选择下一未标记租户
7.    TN = selectTunableNodesByTenant(t);
        //确定租户 t 所在的可调服务器
8.    OTN = selectTunableNodesWithoutTenant(t);
        //确定其他可调服务器
9.    IN = selectIdleNodes();//确定空闲服务器
10.   for each j in TN//在租户 t 所在可调服务器上调整
11.     TT = selectTunableTenants(j);
        //确定服务器 j 上可减少资源的租户
12.     for each i in TT
13.       repeat
14.         X = addResouce(X0,σ,t,i);
            //以粒度 σ 增加租户 t, 减少租户 i 的资源
15.         rt_i = RT(λ_i,w_i,X_i); rt_t = RT(λ_t,w_t,X_t);
            //计算调整后性能
16.         ΔU = U(X) - U(X0); //计算效用增量
17.         if ΔU = 0 and rt_t > SLA_t and rt_i ⩽ SLA_i
18.           X0 = X; //租户 t 性能未满足
19.         else if ΔU = 0 and rt_t ⩽ SLA_t and rt_i ⩽ SLA_i
            //租户 t 性能满足,租户 i 性能仍满足
20.           X0 = X; mark(t); goto 6;
              //标记租户性能需求满足,继续
21.         else if ΔU > 0
22.           X0 = X; return X0; //租户性能需求都满足,返回
            else break;//租户 t 未满足,继续调整
23.   for each j in OTN//在其他服务器上进行调整
24.     if canCreateReplica(j, t,ω)
        //在 j 新建 t 的副本,均摊 ω 后租户性能满足
25.       X0 = createReplica(X0, j,t);
          //在 j 上新建 t 的副本,并计算虚拟化开销
26.       TT = selectTunableTenants(j);
            //确定服务器 j 上可减少资源的租户
27.       for each i in TT
28.         repeat
29.           X = addResouce(X0,σ,t,i); rt_i = RT(λ_i,w_i,X_i);
30.           rt_t = RT(λ_t,w_t,X_t); ΔU = U(X) - U(X0);
31.       if ΔU = 0 and rt_t > SLA_t and rt_i ⩽ SLA_i X0 = X;
```

```
32.        else if ΔU = 0 and rt_t ≤ SLA_t and rt_i ≤ SLA_i
33.                X_0 = X; mark(t); goto 6;
34.        else if ΔU > 0 X_0 = X; return X_0;
35.        else break;// t 未满足,继续调整
36.      for each j in IN //在空闲服务器上进行调整
37.          X_0 = createReplica(X_0, j, t);
38.          while hasResouce(j) //服务器 j 仍有资源空闲
39.              X = addResouce(X_0,σ,t,null);
                //以粒度 σ 增加 t 的资源,得到新矩阵 X_1;
40.          ΔU = U(X) - U(X_0); X_0 = X;
41.          if ΔU > 0 and rt_t ≤ SLA_t mark(t); goto 6;
42.  return X_0;
optimization ()
43.   TT = selectTunableTenants();//确定可减少资源租户
44.  for each t in TT
45.      repeat
46.          X = reduceResouce(X_0,σ,t,null); //以粒度 σ 减少 t
             的资源
47.          ΔU = U(X) - U(X_0);
48.          if ΔU > 0 X_0 = X; //性能需求仍旧满足
49.          else break;//此租户优化调整结束
50.  return X_0;
```

下面通过定理 2 说明 2PG 算法的正确性。

定理 2 2PG 算法是正确的。

证明:在约束阶段,为每个性能需求未满足的租户按已有副本可调服务器(10 – 23 行)、其他可调服务器(24 – 35 行)和空闲服务器(36 – 41 行)的顺序增加资源,直到性能需求满足。由于在可调服务器资源不足的情况下才使用空闲服务器,因此能保证尽量少地占用服务器。同时,在可调服务器的每次调整过程中,要保证减少资源的租户性能需求仍旧满足(17、19、32 和 33 行)。

此外,若存在性能需求不满足的租户,资源效用为负值,在可调服务器上进行调整时,由于没有资源增加,所以每次调整后资源效用的绝对值不变,此时若效用增量等于 0,说明仍有租户性能未满足,则继续调整此租户(17 行和 32 行)或下一未标记租户(19 行和 33 行);若效用增量大于 0(23 行和 42 行),说明所有租户性能需求得到满足,资源效用成为正值。在空闲服务器上进行调整时,由于使用资源增加所以每次调整后资源效用的绝对值变小,因此在租户性能未满足的情况下效用增量也大于 0,只有到租户性能需求满足时(41 行),才能结束此租户的调整。

在优化阶段,2PG 算法在每次资源调整时(44 – 49 行),减少租户的资源使用量优化资源效用(48 行),当资源配置不能满足租户性能需求时(49 行),资源效用变为负值,效用增量不大于 0,此租户的优化调整结束。定理 2 表明,通过约束和优化两个阶段,2PG 算法不仅保证了所有租户的性能需求得到满足,而且优化了资源的使用量,因此能够找到 DRA 问题的近似最优解。用此算法时,若调整粒度为 σ,一个租户的资源调整最多有 n/σ 次,故此算法的复杂度为 $O(mn/\sigma)$,大大降低了问题的求解复杂度。

3 实现与评价

本节介绍 DBaaS 平台的实现,并通过科技信息数据库服务的应用示例来验证本文方法。

3.1 DBaaS 平台实现和应用示例

基于 Xen 构建了 DBaaS 平台(如图 3 所示),其中,请求处理模块对租户的操作请求进行解析和结果处理,负载管理模块路由租户的请求,监控模块跟踪虚拟机状态、资源使用量和性能指标,并根据日志预测租户负载。资源分配模块定时执行分配算法,控制模块负责虚拟机和数据库的参数配置和生命周期控制(创建、启动和关闭等)。

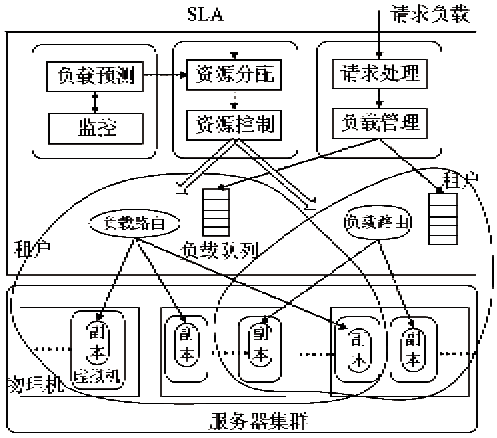

图 3 DBaaS 平台统体系结构

在应用示例中,将 DBaaS 平台应用于科技信息网,为二级单位提供科技信息数据库服务。实验环境搭建在由服务器(2 ×4 核 2.0 GHz CPU、16 GB 内存和 500 GB 硬盘)组成的集群上,虚拟化设施使用 Xen 4.1.4(ω = 2.5%),虚拟机映像预装 CentOS 6.3 和 MySQL 5.5.25,在一台服务器(双核 3.0 GHz CPU 和 4 GB 内存)上进行资源分配,使用 Apache Jmeter 2.9 模拟租户请求并监控性能指标。示例中为 3 个租户提供数据库服务,各租户数据规模、性能需求和服务器的处理能力如表 1。为应对租户负载的动态变化,每 2h 进行一次资源分配,租户在每个

周期的负载如图4所示,负载操作类型(少量读、大量读、少量插入、少量更新、大量插入和大量更新)服从负指数分布(α = 20%)。

表1 租户特征

租户	数据量(GB)	性能需求(ms)	读处理能力(time/s)	写处理能力(time/s)
A	50	15	87.3	35.4
B	100	15	71.6	28.9
C	300	15	58.7	19.2

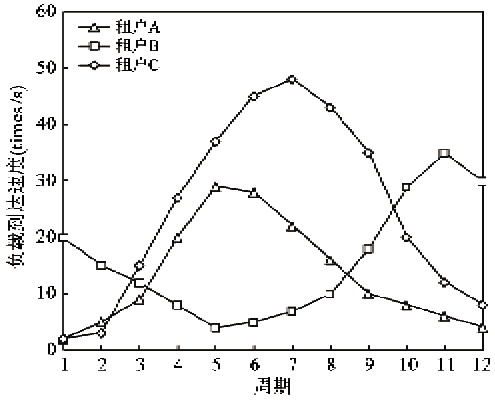

图4 租户负载变化

3.2 资源分配效果

在实验1中,对比共享方式、基于物理机分配和基于虚拟机分配三种方法,取参数α = 10%, ω = 2.5%,σ = 5%

表2显示了租户A、B和C在第7周期的平均响应时间和服务器使用情况。共享方式为租户A、B和C在4台服务器上各部署了一个数据库副本,但由于租户之间的性能隔离得不到保障,租户A和C的请求负载抢占了过多的系统资源造成租户B的性能需求不满足。基于物理机分配方式为租户部署数据库副本使用了6台服务器,可以看到每个租户的性能需求都得到满足,但是存在资源过度分配。使用基于虚拟机分配方法,可以看到只需要4台服务器就可以满足所有租户的需求。图5显示了3种方法在各个周期的效用。共享方式的资源效用在租户SLA满足的情况下是50%,但在第8、14周期会发生租户C对资源的劫持,导致租户A或B的性能得不到满足,因而资源分配效用呈现负值。另外两种分配方式在每个周期都能满足租户需求,基于虚拟机分配的资源效用要高于基于物理机分配,使用更少的资源就能满足所有租户需求。

表2 第7周期资源分配效用

测试指标	共享方式	基于物理机	基于虚拟机
租户A性能(ms)	10.3	13.1	13.6
租户B性能(ms)	25.7	8.4	14.0
租户C性能(ms)	11.1	6.3	14.5
服务器使用量	4	6	4

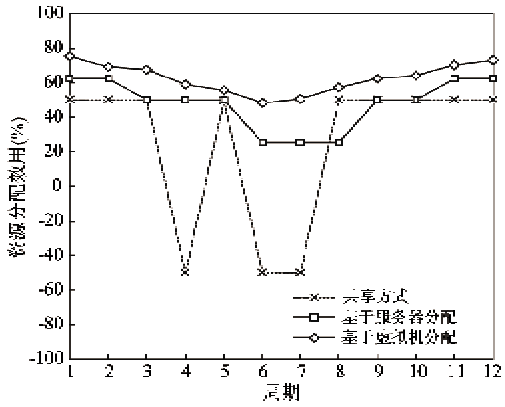

图5 总体资源分配效果

实验1表明本文提出的方法能够在满足租户性能需求的同时优化资源使用量,达到共享服务器系统资源降低成本的目的。

3.3 固有开销影响

实验2分析了副本一致性对资源分配效果的影响,取参数α = 20%,ω = 2.5%,σ = 5%。由图6可知,未考虑副本一致性的资源分配效用虽然绝对值较大,但由于忽略了写操作的同步,造成分配的资源满足不了租户需求,因而资源分配效用呈现出负值。而考虑了副本一致性的方法能够准确地为租户进行资源分配,满足租户需求。

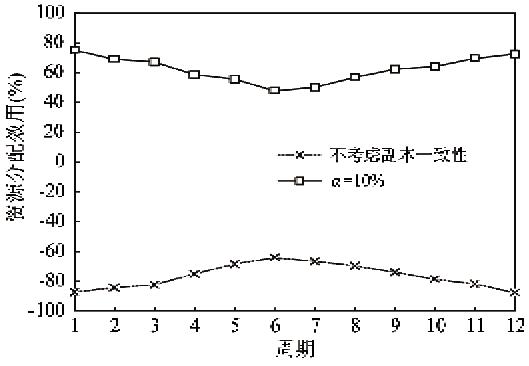

图6 副本一致性因素影响

实验3分析虚拟化开销对资源分配效果的影响,取参数α = 10%,ω = 2.5%,σ = 5%。由图7可

知,若不考虑虚拟化开销,实际分配的资源满足不了租户需求,因而资源分配效用呈现负值。而考虑虚拟化开销的方法能够准确地为租户分配资源,满足租户需求。实际上,虚拟化开销加大会使资源的需求量加大,降低分配效用。

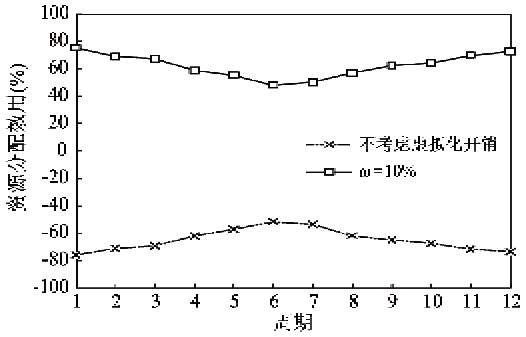

图 7 虚拟化开销影响

基于副本和虚拟化构建的数据库服务引入了两方面的固有开销,通过上述实验可以验证本文提出的方法通过考虑副本一致性和虚拟化开销的影响,使得资源分配结果更为准确。

3.4 调整粒度分析

文献[5]和[7]分别提出了以 5% 的细粒度调整资源配置的方法和以固定虚拟机类型为单位进行分配的粗粒度方法。由 2PG 算法分析可知,在服务器和租户规模确定的情况下,调整粒度 σ 影响资源分配效率。若进行一次资源调整的平均耗时为 v,资源分配要在 L 时间内完成,即要使 $(mn/\sigma)\cdot v \leqslant L$,则需 $\sigma \geqslant mn\cdot v/L$,也就是说提高资源分配效率,需增大资源调整粒度。然而,虚拟化开销类似,增加资源调整粒度也会降低资源分配效用。实验 4 通过测试在不同粒度下进行第 7 周期资源分配的耗时和效用验证了上述规律,如图 8 所示。在实际应用中,可以根据效率和效用需求选择资源调整粒度。

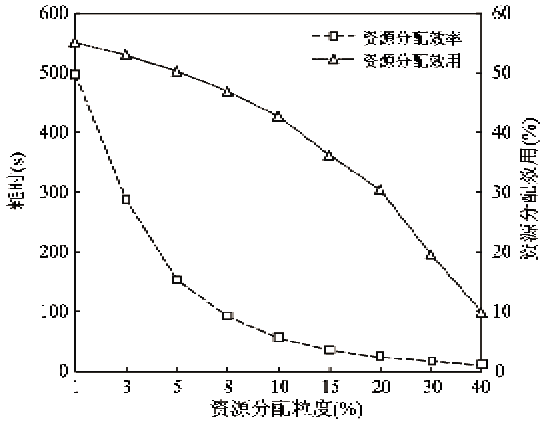

图 8 资源调整粒度分析

4 结 论

本文基于虚拟化技术,提出了一种无共享架构下以副本方式构造 DBaaS 的资源优化分配方法:建立特定虚拟资源配置数据库服务的性能模型,定义度量资源分配效果的效用函数,在此基础上,以贪心方式快速寻找问题的近似最优解。针对副本和虚拟化技术引入的固有开销,性能模型和搜索算法考虑了副本一致性和虚拟化开销因素,使得资源分配更为准确,资源效用更高。此外,通过总结虚拟资源分配方法呈现出的两种粒度趋势和分析资源调整粒度对分配效率和资源效用的影响,使得分配过程更为可控。以副本方式构建的 DBaaS 具有保持关系数据库特性和增强数据库服务可用性的特点,但是数据规模扩展性有限。下一步工作主要针对分片方式 DBaaS 的多租户资源分配方法。

参考文献

[1] 王卓昊,赵卓峰,房俊等. 一种 SaaS 模式下的服务社区模型及其在全国科技信息服务网中的应用. 计算机学报,2010,33(11):2033-2043

[2] Hacigumus H, Iyer B, Mehrotra S. Providing database as a service. In: Proceedings of International Conference on Data Engineering, Washington, USA, 2002. 29-39

[3] 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

[4] Xen. http://www.xen.org/:Citrix,2012

[5] Soror A, Minhas U F, Aboulnaga A, et al. Automatic virtual machine configuration for database workloads. *ACM Transaction on Database System*, 2010, 35(1): doi:10.1145/1670243.1670250

[6] Padala P, Shin K G, Zhu X Y, et al. Adaptive control of virtualized resources in utility computing environments. In: Proceedings of European Conference on Computer Systems, New York, USA, 2007. 289-302

[7] Rogers J, Olga P, Celintemel U. A generic auto-provisioning framework for cloud databases. In: Proceedings of the 26th International Conference on Data Engineering, Washington, USA, 2010. 63-68

[8] EC2. http://aws.amazon.com/ec2/:Amazon, 2012

[9] Minhas U F, Yadav J, Aboulnaga A, et al. Database systems on virtual machines: How much do you lose? In: Proceedings of the 24th International Conference on Data Engineering, Washington, USA, 2008. 35-41

[10] Wang X Y, Lan D J, Wang G, et al. Appliance-based

autonomic provisioning framework for virtualized outsourcing data center. In: Proceedings of the IEEE International Conference on Autonomic Computing, Washington, USA, 2007. 29-38

[11] Aboulnaga A, Salem K, Soror A, et al. Deploying database appliances in the cloud. *IEEE Data Engineering Bulletin*, 2009, 32(1):13-20

[12] 刁在筠,郑汉鼎,刘家壮等. 运筹学. 北京:高等教育出版社, 2000. 242-255

Optimization of multi-tenant DBaaS systems' resource allocation

Qi Kaiyuan[*,***], Zhao Zhuofeng[**], Zhang Dong[*], Liu Zhengwei[*]

([*] State Key Laboratory of High-end Server & Storage Technology, Jinan 250101)

([**] Cloud Computing Research Center, North China University of Technology, Beijing 100144)

([***] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract

The problem of optimizing the resource allocation of a multi-tenant DBaaS (database-as-a-service) system was studied, and its constraint programming model and solving approach were proposed. The problem was solved in such a way below: A performance model was established to map the virtualization resource into the database service performance, and a utility function was defined to measure the effect of resource allocation in terms of resource utilization and performance satisfaction. Based on the performance model and utility function, a two-phase greedy algorithm was used to search the near-optimal solution. The application example and experiments show that the above-mentioned method can save the resource cost while meeting the performance demand, and make the allocation more accurate and controllable, because of its taking replication consistency, virtualization overhead and tuning granularity into consideration.

Key words: database-as-a-service, multi-tenant server consolidation, on-demand resource allocation, performance model, utility function, two-phase greedy algorithm