基于信息网模型的动态数据划分策略

陈诗雅 刘梦赤
(武汉大学计算机学院 湖北 武汉 430072)

摘 要 为了满足大规模数据管理与查询的需要,设计并开发了基于信息网模型 INM(Information Networking Model)的分布式并行数据库管理系统。分布式环境下数据的划分方式将影响系统的可扩展性和查询分析效率。根据信息网模型的数据结构和查询特性,设计一种轻量级的数据动态划分方法。该方法结合数据的水平分割和垂直分割,以 INM 对象为单位,未存储过的数据对象直接划分到当前操作节点,并记录数据对象的存储位置,否则根据数据对象的历史位置信息将其划分到不同的存储节点。同时,单个 INM 对象可能由于其包含的关联对象增多到一定程度成为大对象,而对系统的性能造成影响,因此将此类大对象分割成多个小对象,并按照一定的策略划分到不同节点进行存储。集群中的每个处理节点被赋予一个负载阈值。随着数据量的增加,如果超过负载阈值则增加新的机器,保证系统的可扩展性和各个处理节点数据量的均衡。实验结果证明,该方法能够保障系统良好的可扩展性,同时提高数据的查询分析效率。

关键词 信息网模型 数据划分 大对象分割 负载阈值

中图分类号 TP311 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2018.11.006

DYNAMIC DATA PARTITION BASED ON INFORMATION NETWORK MODEL

Chen Shiya Liu Mengchi
(*School of Computer, Wuhan University, Wuhan 430072, Hubei, China*)

Abstract To meet the needs of large-scale data management and query, we designed and developed a distributed and parallel database management system based on information network model(INM). The way to divide data in distributed environment would have an effect on scalability and the efficiency of query analysis of the system. We proposed a lightweight dynamic data partition method according to the data structure and query characteristics of INM. The method combined the horizontal and vertical segmentation of data. The unstored data objects were directly divided into the current operation nodes, and the storage locations of the objects were recorded in units of INM objects. Otherwise, the data objects were divided into different storage nodes according to their historical location information. A single INM object might become a large object because its associated objects were increased to a certain degree, which would have an impact on the performance of the system. Therefore, these large objects were segmented into multiple small objects and divided into different nodes for storage according to certain strategies. We gave each processing node in the cluster a load threshold. As the amount of data increased, a new machine would be added if stored data exceeded the load threshold, so as to ensure the scalability of the system and the balance of the amount of data in each processing node. Experimental results show that this method can guarantee good scalability of system and improve the efficiency of query analysis of data as well.

Keywords Information network model Data partition Large object division Load threshold

---

收稿日期:2018-03-25。国家自然科学基金面上项目(61672389);国家杰出青年科学基金(外籍)项目(60688201)。**陈诗雅**,硕士生,主研领域:数据库。**刘梦赤**,教授。

0　引　言

互联网的快速发展,导致数据爆炸式的增长,同时对于数据存储的要求也不断提高,传统的集中式数据库的缺陷日益显露:系统可用性和可靠性较低,可扩展性差,导致系统无法满足日益增长的数据的存储需求。因此构建在集群上,甚至不同数据中心间的分布式并行数据库[1-2]成为全新的解决方案,它们透明地把数据分散存储到服务器集群中的不同节点上,采用并行数据处理框架高效应对不断增长的大数据,提供更好的水平扩展性和更高的可用性。因此项目组基于信息网模型[3-4]设计并开发了分布式并行数据库管理系统,系统能够通过水平扩展集群节点数量来获得更大的存储容量和更高的并发访问量。

对于分布式系统来说,合理地将整体数据分散到多台存储机上,可以有效提高系统的存储效率。而对于数据划分方案的选取,需要考虑到系统可扩展性、负载均衡等性能需求,存储数据的结构,以及划分算法的时间空间开销。比如部分分布式系统为了系统的高扩展性选取简单的基于数据的关键值进行划分的方式,Apache Hbase[5]根据行键(row key)的范围将 Hbase 表分割为多个 region,然后由 HMaster 将每个 region 分配给相应的 RegionMaster 进行管理。一致性哈希算法[6]由于其实现简单,对大规模数据集划分性能较好,且易于扩展,得到了广泛的运用。算法将系统中的物理节点和需要被存储的数据映射到哈希环上的合适位置,根据其相对位置来选取数据的存储节点[7]。而且有学者在一致性哈希方法的基础上引入虚节点[8]的概念,即每个物理节点根据其处理性能从逻辑上切分为多个虚拟节点,来保证各个处理节点间的负载均衡。文献[9]提出根据处理节点的异质性将一致性哈希和基于范围的方式相结合,即机器集群被分为 k 个节点集合,一致性哈希算法用于 k 个集合之间的数据划分,基于范围的方式用于每个节点集合中的 m 台机器间的数据划分。但是,上述这些划分方案由于其划分的随机性,对于彼此之间相互独立的数据具有较好的划分效果,但是如果数据之间相互关联,在分布式环境中以任意的方式将这些数据划分到集群中,可能会造成在一个查询任务不能在一个存储节点上完成的情况,影响数据的查询分析效率。

在信息网模型中,现实世界中的每个实体对应于信息网模型数据库中的一个对象,实体自身的所有信息保存在一个 INM 对象之中,而对象之间通过关系进行联系。如果想查询和某个对象相关联的对象信息,则需要根据关系的指向去访问关联对象的信息。因此,当这些关联对象存储在不同节点上时,查询任务无法在一个节点上完成,需要和存储关联对象的其他节点进行通信,造成大量的通信开销。因此考虑通过减少不必要的通信开销来提高查询效率。很多图分割算法在维护数据单元之间的关联关系,减少查询任务跨分区进行带来的通信开销方面提出解决方法。比如针对小规模图的静态划分方法 KL[10]、FM[11]。基于 k-balanced 的图分割算法旨在保持划分均衡的同时,减少节点之间的通信开销。文献[12]证明此类方法是一个 NP 难问题,不具备实用性,因此也产生了很多近似算法以及多层次启发式算法[13-16]。文献[16]提出将图分割问题转化成寻找高质量大型子图的问题,通过去除一些问题节点,寻找划分效果较好的大型子图,并根据对子图的划分优化图分割问题。通过最小化切割的边数[17-19]来减少各个分区之间的通信,从而减少任务跨分区的情况,文献[20]提出一种最小切边算法,实现了距离常规有向图以及强规则有向图的有效划分。

事实上大多数图分割方法由于其自身时间和空间复杂性的限制,在实际应用中并不被采用。而且考虑到信息网模型的模型特点,除了需要关注整体数据的划分方式,还要对一些具有丰富关联关系和属性信息的 INM 大对象单独进行分割。因为在实际操作时发现,与这些大对象关联的其他对象较多,使得在大量写语句并发时以及查询此类对象时开销较大,影响整个系统的性能。因此本文提出了一种基于大对象分割的动态数据划分方法,从水平方向和垂直方面对数据进行分割,保证系统的可扩展性,并提高数据的查询分析效率。

1　系统架构

1.1　信息网模型

信息网模型 INM 是课题组提出的一种语义型数据模型,通过对象间各种关联关系来表达对象之间丰富的语义性。

信息网模型将现实中的实体抽象为类,并将实体之间可能产生的关系、类和关系所具备的特性都集成到类中,实例[3-4]则是类的实例化对象。比如:

大学 武汉大学[

@级别:{“985 工程院校”,“211 工程院校”,教育部高校},

@类型:综合院校,

@主页:“http://www.whu.edu.cn”,

normal 校训:“自强弘毅 求是拓新”,

role 校领导[@任期:4] -> {

校长[@级别:副部级]:窦贤康[@上任时间:“2008-11”,@性别:男],

党委书记[@级别:副部级]:韩进[@上任时间:“2008-11”]},

contain 学部:{工学部(武汉大学),信息学部(武汉大学),文理学部(武汉大学),医学部(武汉大学)}];

“大学”是一个类,而“武汉大学”即为该类的一个实例化对象。该对象具有“级别”等属性和“校领导”等关联关系。

在 INM 模型中,关系[3-4]是一个很重要的概念,对象之间的语义信息正是通过各种关联关系来表示的。一个关系一般连接着两个对象,比如示例中的关系 contain,它表示对象“武汉大学”有“学部”这个包含关系,该关系的目标对象(target)之一为对象“工学部”,即关系 contain 连接着“武汉大学”和“工学部”两个对象。除了 contain 关系,信息网模型中还有普通关系 normal(默认的关系类型)、角色关系 role、基于角色的关系 role-based 等多种关联关系。关系还可能具有层次性,比如“武汉大学”有以“校领导”为根的角色关系层次,“校领导”有角色子关系“校长”等。因此在写入对象“武汉大学”时,同时也会写入各种关系的目标对象,比如在写入对象“武汉大学”时,同时也会写入对象“窦贤康”、“韩进”等,而对关系的目标对象“窦贤康”等的写入、更新或者查询都需要关联到源对象“武汉大学”。

基于信息网模型的查询特性,在存储对象时,要尽量将具有关联关系的对象如“武汉大学”、“窦贤康”、“韩进”等放在一个存储节点上,避免查询时去跨越多个其他节点来获取相关对象,造成额外的通信开销。而且有些 INM 对象可能有几十万个 target,不论是对这些对象本身的读写,又或是对其关联对象的读写,都会产生不可忽视的时间开销,从而影响系统性能,因此还需要对这类对象单独进行处理。

1.2 分布式系统架构

分布式并行信息网数据库管理系统(DPINM)的设计理念之一是具有高度可扩展性,因此系统采用一个主节点(master),多个子节点(slave)的架构,如图 1 所示。

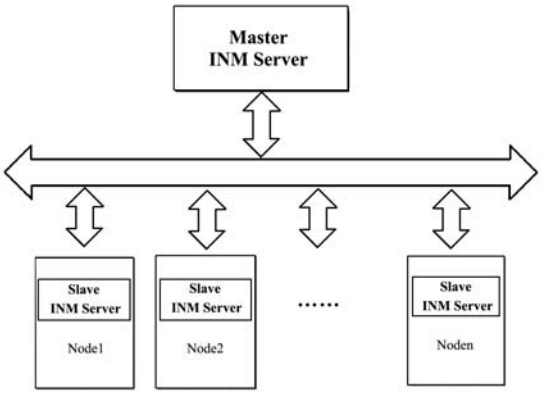

图 1 DPINMDB 系统架构

集群中的机器节点分为两类:主节点 master 和处理节点 slave。其中处理节点主要进行任务分发、数据的初始划分和元数据管理,处理节点则进行数据的具体操作。大对象分割即在处理节点 slave 上完成。为了快速有效地定位数据所在的处理节点,master 节点上会动态维护两张表:Id-Node 表和 Node-Set 表,表示数据对象存储的位置,具体的表内容将在划分方法中介绍。

2 大对象分割

信息网模型中存在部分实例对象,具有庞大的属性信息和关联关系,比如对象“美国”,其所具有的关系“电影”连接了几十、上百万个电影对象,那么在写入、更新或者查询对象“美国”的相关信息时,都需要把这个庞大的对象取出来写进去,耗时较大。而且在写入大量“美国”拍摄的“电影”时,每个写任务都需要等待前一个写任务的完成。因此不论是对大对象本身的存取复杂度,还是其关联的对象,都可能影响系统的性能。因此采取一种策略,对该类对象进行分割。

2.1 存储结构

大对象被分割后的子对象分布在不同节点上,为了定位这些子对象所在的节点,需要保存对象及其子对象所在的节点信息。因此设计了 Node-Set 表来保存对象被分割后的子对象存储在哪些节点上。在大对象分割的过程中会生成表项信息。考虑到将 Node-Set 表放在各个处理节点上不利于维护该表的一致性,可能存在不同步的情况,因此将 Node-Set 表存放在 master 节点上统一进行维护。Node-Set 表结构如表 1 所示。

表 1 Node-Set 表结构

O_1 -> id	node1, node2,…
O_2 -> id	node2, node3,…

例如对象 O_1 被分割成两个子对象 sub1、sub2,子

对象 sub1 和 sub2 分别存储在节点 1 和节点 2 上,则表项中的第一项为大对象 O_1 的 id,第二项为 sub1 和 sub2 所在的节点号 1、2。

2.2　大对象分割算法

首先要考虑的是如何尽可能地均匀分割大对象,如果分成的子对象本身大小相差较大的话,则达不到大对象分割的目的。造成对象太大的主要原因是和该对象关联的数据对象太多,如果按关系类型分割是否可行呢? 事实上并不可行,因为每个关系的目标对象数目可能差距较大,比如对象武汉大学的 normal 关系"校训"只有一个目标对象,而 contain 关系"学部"有四个目标对象,这样即使拆分之后,子对象之间的大小也有明显的不均衡。因此按照关系的目标对象数目分割是目前最合理的方案,如此分割出来的子对象之间除了关系的目标对象不同之外,其他信息基本一致。

大对象阈值 objSize 设置在配置文件中,因此在对象存储到底层之前,先读取配置文件中的阈值和对象大小 n 进行比较,如果超过阈值则可能分割成 num 个子对象($num = n/\text{objSize}$)。如果对象符合分割条件则按以下算法进行分割,得到子对象集合 nodeset。大对象分割算法如下:

输入:大小超过阈值的对象 O_1。
输出:分割后的子对象集合 nodeset。
1 标记该对象被分割 setNode(id, Max)
2 新建子对象集合 subSet,将源对象 O_1 的基础信息如 id、name 等拷贝到 subSet 中的每一个对象 sub_i
3 **FOR** 源对象 O_1 的每个关系 rel_i
4 新建关系 rel 的子对象集合 vecRel
5　统计关系 rel 的所有目标对象数目 tgtNum,计算 aveTgt = (tgtNum/num) + (tgtNum% num)
6　**FOR** vecRel 中的每一个关系 initRel
7　　拷贝关系 rel 的基础信息(version、name 等)和属性到 initRel
8　　拷贝 aveTgt 个关系 rel 的 target 到 initRel
9　　**IF** initRel 有子关系
10　　重复 8 - 10 的操作
11　　**END IF**
12　**END FOR**
13　**FOR** 集合 vecRel 的每一个关系 rel_i
14　　将关系 rel_i 添加到 sub_i
15　**END FOR**
16 **END FOR**
17 删除源对象 O_1

2.3　子对象分布

由于子对象分布在不同处理节点上,在 master 节点上进行数据的初始划分时候很难抉择应该选取哪个节点作为存储节点。如果子对象随意分发到某些 slave 节点,会出现两个问题:一是某个处理节点可能已经存储过这个对象了,将子对象发送到该节点则要进行对象合并,而且合并后的对象可能又成一个大对象;二是 master 节点在对分割过的对象进行划分的时候任意选取的话,会造成各个节点上存储的子对象大小在动态变化的过程中逐渐出现较大差距,违背了子对象大小尽量均匀的原则。所以选择此前没有存储过该对象的节点作为子对象接收节点,能够有效避免对象拆分之后又合并,且各节点上的子对象大小要尽量保持动态均衡,避免频繁的维护子对象大小的均衡。

因此本文中提出的子对象分发策略为:按照节点号顺序选择,即子对象集合中的第一个子对象存储在当前操作节点 current 上,子对象 subi 发到节点 p 进行存储,p 的计算如式(1)所示:

$$p = (\text{lastNode} + i)\%L \tag{1}$$

当 $p = 0$ 时(0 表示为 master 节点),$p = p + 1$。

式中:L 表示集群节点数目。lastNode 初始值为 current,对象分割后会先去 master 节点上按照对象 id 查找 Node-Set 表,如果查找到的 Node-Set 表中的相应表项不为 NULL(说明此对象被分割过,存在一个节点位置集合 nodeset),则 lastNode 置为 nodeset 中的最后一个 node 号,即上一次子对象分发所发送到的节点号 q,设置 lastNode = q,那么本次选取 q 的下一个节点 $q+1$ 作为起始接收节点。同时还要将新的 node 信息顺序添加到 nodeset 中,并更新到 master 节点上的 Node-Set 表。

事实上,在数据量足够多的情况下,通过 random()函数随机选取 nodeset 中的后两个 node 号(上一次对象分割后子对象的接收节点)中的某一个作为对象的划分节点,可以使得两个节点上的子对象大小保持一定的动态均衡,也就是说如果其中一个节点上的子对象大小达到临界值,另一个节点上的对象大小也是在临界值附近,不会有太大的差距。因此在分割后的子对象分发过程中无需再考虑之前已经操作过的所有节点(即 nodeset 中的所有 node 值)。

3　基于大对象分割的动态数据划分方法

系统对插入语句进行处理时,会将每个 INM 对象及其信息提取出来,并为其分配全局 id 号。为了快速有效地定位数据所在的处理节点,master 节点会动态维护一张 Id-Node 表,一个对象 id 对应一个节点号,表明该对象存储在哪个节点上。表 2 给出了 Id-Node 表结构。

表 2 Id-Node 表结构

对象 ID	存储节点号
Oi -> id	nodeNum

因此基于大对象分割的分布式数据划分方案如下：

IQL 语句在经过词法语法分析之后得到对象集合。对于对象集合中的对象 O_1，需要先根据对象 id 查询 master 节点上的 Id-Node 表，然后主节点 master 根据返回的值 x 决定选取哪个 slave 节点作为接收节点：

- $x=0$，表明该对象之前没有被处理过，则分发到当前操作节点 current 处理。
- $x=1,\cdots,L-1$，表明集群中节点 x 上已经存储了对象 O_1，则分发到 x 值对应的 slave 节点处理。
- $x=$MAX（MAX 为一个大于集群节点数目的常量值），说明此 id 的对象曾经被分割，该对象 id 对应一个 node 集合，则先读取存储在 master 节点上的 NodeSet 表，根据对象 O_1 的 id 获取相应的子对象所在节点集合 nodeset。通过 random( )函数从集合 nodeset 中的最后两个值中随机选取一个节点作为接收节点。

而且，集群中的每台机器都会被设置一个负载阈值 load，当活跃节点 current（当前进行数据存储的 slave 节点）的数据量达到 load 之后，将选择集群节点编号中的下一个 slave 节点作为活跃节点，因此将该方法命名为 Load Partition 数据划分方式。此方法虽然操作简单，但是能够较好地满足系统水平可扩展的需求。而且基于系统对于数据写入操作处理的特性，一条写入语句会生成多个具有关联关系的对象，每个对象具有顺序的唯一确定的全局对象 id，Load Partition 划分方法可以保证这些关联对象在一定时间之内能够存储在同一个节点上。对于可能成为系统性能瓶颈大对象，将其分割后分开存储，在有大量写任务并发的情况下可以提高系统效率。

4 实验分析

实验主要分析大对象分割方案对于系统数据插入和查询的效率影响，并对比使用较广泛的一致性哈希算法和基于大对象分割的动态数据划分方案下的数据查询时间。

系统分布式集群包括一个主节点 $node_0$ 和五个处理节点 $node_1\cdots node_5$。由于系统底层使用 berkeleyDB 进行数据存储，因此将大对象阈值 objSize 设置为一个 BDB 大页大小，即 16 384 Byte（16 KB）。鉴于大对象的特殊性，实验数据选定为大约 100 万条格式如下所示的电影数据：

```
Insert Movie "name"("year")[countryList:"nation"];
Insert Movie "1971 World Series"("1971")[countryList:"USA"];
```

该插入语句在进行词法语法分析时，会生成两个对象"Movie"和"Country"，其所对应的部分模式语句分别为：

```
create class Country
[
    contain cityList *:City,
    normal movieList(M:N):Movie(inverse countryList)
];
create calss Movie
[
  $@languages *:{"English","Italian","French"},
  @runTime : string,
];
```

类 Country 中的 normal 关系 movieList 和类 Movie 以 countryList 互为逆关系，因此在插入电影数据的时候，其 countryList 关系中 target（比如"USA"）的 movieList 关系也会不断进行更新。因此利用本文提出的方法对这类数据进行处理。

由于大对象一般是随着插入数据的增多逐渐出现的，因此为了方便统计数据大小，实验将一条"Movie"插入语句作为一个数据大小，数据查询测试用例如下所示：

```
query $x = nation0/movieList: $y construct $y;
```

即查询对象 nation0 下的所有电影信息。

4.1 大对象分割对数据写入效率的影响

表 3 比较了是否进行大对象分割后，数据写入所消耗的时间。通过数据对比可以看出，数据量较少的情况下，有对象分割的写入耗时要大，因为数据写入伴随着大对象分割的额外时间消耗。但是随着数据量的增加，大对象被分割的子对象增多，分割后的写入时间比未分割的情况下要少，因为前者可同时在多个节点上进行部分数据的写入操作，相比只有一个节点可写入而造成的任务等待，大对象分割后的写入耗时要小。

表 3 有无对象分割下的数据写入时间

数据量/万条	对象分割	无对象分割
20	2 min 20 s	2 min 10 s
40	5 min 12 s	5 min 56 s
60	7 min 25 s	9 min 23 s
80	10 min 20 s	14 min 35 s
100	12 min	18 min 52 s

4.2　大对象分割对数据查询效率的影响

表 4 比较了大对象分割和不分割,查询该大对象的耗时对比。通过数据对比可以看出,在插入数据量较少的情况下,两者的查询耗时相差无几,但是随着数据的写入,数据量增大,对象分割后的查询耗时相比之下越来越短。因为对象被分割成几部分分别存储,查询任务可以在子对象的所有存储节点上并行执行,并各自返回结果,所以查询时间相对来说会有所减少。

表 4　有无对象分割下的数据查询时间

数据量/万条	对象分割/ms	无对象分割/ms
20	102	124
40	170	225
60	236	355
80	278	500
100	305	753

4.3　存储节点上的数据量变化

表 5 给出了不同写入数据量下,执行大对象分割后各个处理节点上的数据量。为了方便统计,节点上的数据量以存储的数据对象的个数表示。由于数据采用逐节点写入的方式,并非每个节点上都会存储数据,因此表中某些节点上的数据对象为 0。对表中数据进行分析,由于大对象分割的进行,缩小了数据对象之间的大小差距,因此每个节点上的数据对象数目能够稳定在 30 万个左右。但是对于已经存储过数据的节点集合,其中最后一个节点上的数据量可能和其他节点上的数据量相差较大,因为这些节点上存储的大部分可能都是其他数据对象的子对象。

表 5　不同数据量下存储节点的存储量

数据量/万条	Node1	Node2	Node3	Node4	Node5
20	200 005	0	0	0	0
40	301 580	98 512	0	0	0
60	301 580	263 520	31 542	0	0
80	301 580	300 826	154 320	43 200	0
100	301 580	300 826	300 815	96 820	0

4.4　动态数据划分方法的性能

该部分实验用于测试基于大对象分动态数据划分方法的性能。考虑到信息网模型数据的特性,以及哈希方法被使用的广泛性和合理性,实验将选取一致性哈希算法和本文提出的方法在本系统上进行性能对比。实验基于一个包含大约 120 万个对象的数据集,数据集中的数据从各所高校的网页中抽取获得,并根据信息网模型的格式转换生成。表 6 给出了实验所需的测试用例,Q_1 是查询一个对象的信息,不涉及跨对象的情况,Q_2 和 Q_3 涉及到不同对象之间的查询跳转。

表 6　测试用例

对象信息	查询语句	含义描述
Q1	Query $x = 大学 construct $x;	查询某所高校的信息
Q2	query $x = 大学/教学部门: $y construct $y;	查询高校所有的教学部门
Q3	query $x = 大学/教学部门: $y/教授:$z construct $y/$z;	查询高校的所有教授

表 7 给出了两种划分方式下的查询时间对比,通过分析可知,随着查询复杂度的增大,查询过程中对象跳转次数也随之增加,查询时间越来越大。相比于一致性哈希算法的随机划分,本文提出的动态划分算法由于能够保证具有关联关系的对象尽可能的处在同一节点,减少查询过程中跨节点的情况,再加之大对象分割减少了此类对象的时间消耗,从而大大降低整个查询的耗时。

表 7　两种划分方式下的查询时间对比

对象信息	一致性哈希	Load Partition
Q1	3 ms	3 ms
Q2	62 ms	28 ms
Q3	1 s 412 ms	325 ms

5　结　语

本文基于信息网模型的特点,考虑了对系统性能可能产生影响的因素,设计了一种基于大对象分割的分布式数据划分方案。一方面将大小超过所设大对象阈值 objSize 的对象分割成多个子对象,且子对象分布在不同的存储节点上,实验证明对大对象进行分割可以提高数据存储和查询的效率。另一方面在主节点上根据 Id-Node 表的信息对数据对象进行初始划分,集群中的每个存储节点均设置了负载阈值 load,在数据动态增加的过程中如果存储的数据量超过该节点的负载阈值,则选取一台新的服务器作为存储节点。本文提出的划分方法能够满足系统水平可扩展性的需求,且在一定程度上保证了具有关联关系的对象集中存储,提高分布式环境下的查询效率。

方案目前对于大对象的拆分只考虑了关系的目标对象,对于部分对象属性太大的情况则没有做出相应处理。虽然此方案可以保证子对象在一定程度上保持大小动态平衡,但是随着一些数据更新、删除操作的进行,各个子对象的大小会发生一些变动,因此还需要关注分割后的子对象的大小情况。对于一些小的子对象还需要进行合并或者标记之后等待后续处理,本文提出的方案目前对这种情况没有做出很好的处理,在进一步优化的过程中会定时去检查子对象的大小,并做出相应的调整。

参 考 文 献

[1] Sokolinsky L B. Survey of Architectures of Parallel Database Systems[J]. Programming & Computer Software, 2004, 30(6):337 - 346.

[2] Hinton G E, Anderson J A. Parallel Models of Associative Memory: Updated Edition[M]. Psychology Press, 2014.

[3] Liu M, Hu J. Information Networking Model[C]//International Conference on Conceptual Modeling. Springer-Verlag, 2009:131 - 144.

[4] 胡捷, 刘梦赤. 信息网模型 INM 研究[M]. 北京: 科学出版社, 2011: 15.

[5] Vora M N. Hadoop-HBase for Large-Scale Data[C]//International Conference on Computer Science and Network Technology. IEEE, 2012:601 - 605.

[6] Devine R. Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm[C]//Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms. Springer-Verlag,1993: 101 - 114.

[7] Mohammad S, Breβ S, Schallehn E. Cloud Data Management: A Short Overview and Comparison of Current Approaches[J]. Plos One, 2013, 5(9): e12906.

[8] Chen C, Tsai K C. The Server Reassignment Problem for Load Balancing in Structured P2P Systems[J]. IEEE Transactions on Parallel & Distributed Systems, 2008, 19(2): 234 - 246.

[9] Chen Z, Yang S, Tan S, et al. Hybrid Range Consistent Hash Partitioning Strategy—A New Data Partition Strategy for NoSQL Database[C]//Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 2013:1161 - 1169.

[10] Kernighan B W, Lin S. An efficient heuristic procedure for partitioning graphs [J]. Bell System Technical Journal, 2014, 49(2):291 - 307.

[11] Fiduccia C M, Mattheyses R M. A linear-time heuristic for improving network partitions[C]//Proceedings of the 19th Design Automation Conference. IEEE, 1982:175 - 181.

[12] Garey M R, Johnson D S, Stockmeyer L. Some Simplified-Gareyd NP-Complete Problems[C]//ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA. 1974:47 - 63.

[13] Kim M, Candan K S. SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices [J]. Data & Knowledge Engineering, 2012, 72(1):285 - 303.

[14] Arora A, Kaur K. Enhanced Multilevel Hybrid Algorithm for Graph Partitioning[J]. International Journal of Computer Applications, 2015, 120:16 - 19.

[15] Rahimian F, Payberah A H, Girdzijauskas S, et al. A Distributed Algorithm for Large-Scale Graph Partitioning[J]. Acm Transactions on Autonomous & Adaptive Systems, 2015, 10(2):1 - 24.

[16] Lim Y, Lee W J, Choi H J, et al. Discovering large subsets with high quality partitions in real world graphs[C]//International Conference on Big Data and Smart Computing. IEEE, 2015:186 - 193.

[17] Ghaffari M, Kuhn F. Distributed Minimum Cut Approximation[M]//Distributed Computing. Springer Berlin Heidelberg, 2013:1 - 15.

[18] Guttmann-Beck N, Hassin R. Approximation Algorithms for Minimum K-Cut [J]. Algorithmica, 2000, 27 (2): 198 - 207.

[19] Nagamochi H, Ibaraki T. A fast algorithm for computing minimum 3-way and 4-way cuts[J]. Mathematical Programming, 2000, 88(3):507 - 520.

[20] Ashkboos S, Omidi G R, Shafiei F, et al. Minimum edge cuts of distance-regular and strongly regular digraphs[EB]. eprint arXiv:1702.01253, 2017.

(上接第 15 页)

[20] Xie X, Liu F, Lu B, et al. Mixed Obfuscation of Overlapping Instruction and Self-Modify Code Based on Hyper-Chaotic Opaque Predicates[C]//Tenth International Conference on Computational Intelligence and Security. IEEE Computer Society, 2014:524 - 528.

[21] Guillot Y, Gazet A. Automatic binary deobfuscation[J]. Journal in Computer Virology, 2010, 6(3): 261 - 276.

[22] 段钢. 加密与解密[M]. 3 版. 北京:电子工业出版社, 2008: 471 - 490.

[23] 李士群, 王崑声, 经小川, 等. 基于模糊层次分析法的软件保护有效性评价[J]. 计算机技术与发展, 2018(4): 133 - 140.