一种基于Hadoop的BIM云服务框架和空间位置检索算法

陈泽琳 潘运军 何浥尘 齐德昱
(华南理工大学软件学院 广州 510640)

摘 要 云平台存储和管理应用复杂的海量数据已成为必然。建筑信息模型(Building Information Modeling,BIM)是从全工程建设生命周期的视角组织相关数据并协同工作,所以BIM迫切需要云计算的支持。但是面对复杂的BIM应用,如何构建云平台的超级计算模式将是一个巨大的挑战。提出一个面向BIM应用的云服务框架,在Hadoop分布式软件框架上设计了云存储、云平台服务、应用服务和客户端应用四层结构。提出了该框架下的城市空间位置检索算法,该算法采用改进的KD树作为索引表。针对大用户群的并发访问,提出了面向空间位置检索的负载均衡算法,通过统计节点访问频度设计了数据块均衡分布策略。实验表明,该框架组织的建筑信息具有并发处理能力强、响应速度快等特点。

关键词 云计算,建筑信息模型,框架,KD树,空间位置检索,负载均衡
中图法分类号 TP316.8 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.021

Framework of BIM Cloud Services and Retrieval Algorithm of Spatial Location Based on Hadoop

CHEN Ze-lin PAN Yun-jun HE Yi-chen QI De-yu
(School of Software Engineering, South China University of Technology, Guangzhou 510640, China)

Abstract Cloud platform becomes a necessity to store and manage huge data for complex applications. The task of Building Information Modeling (BIM) is to organize the relevant data and work collaboratively during the whole life-cycle of the construction informationize. BIM is in urgent need of cloud computing. How to build a supercomputing model on cloud platform is a big challenge when facing the complex BIM application. This paper presented a framework of cloud services for BIM application. Hadoop is a distributed software framework. The four layers of the framework were designed: cloud storage, cloud platform services, application services and client applications. The retrieval algorithm of urban spatial location was proposed on the framework, which uses the improved KD tree as index table. The paper presented the load balancing algorithm for spatial location retrieval by which many groups of users access the data block concurrently. By statistical access frequency of the nodes, the strategy of balanced distribution for data block was designed. Experiments show that the framework has the characteristics of concurrent processing capability and rapid response for building information.

Keywords Cloud computing, Building information modeling, Framework, KD tree, Spatial location retrieval, Load balancing

1 引言

随着云计算(cloud computing)技术的快速发展,面向各种应用的公有云、私有云和混合云的计算架构模式已成为人们探索的研究热点之一。云计算是利用海量存储空间满足互联网上的各种服务和应用的个性化需求。BIM(Building Information Modeling)是近年来城市建筑信息化的又一次大规模行动,BIM是以建筑三维信息模型为基础,从立项、规划、投资、预算、设计、采购、建造、监理、交付管理到运营维护等过程进行全生命周期的信息资源共享和协同作业[1,2]。构建云平台上的BIM服务架构不仅是BIM应用的需要,也是建设数字城市基础设施信息化的必要保证。本文提出一种面向BIM的城市空间信息检索的服务框架,该框架基于云平台并且面向互联网和移动用户,能合理地组织云平台上BIM所需要的应用和服务,并设计了城市空间信息索引树和检索算法以及大用户群访问的负载均衡算法。

云是一种由互联的虚拟集群计算机组成的并行和分布式系统,它根据服务提供商与用户间协商确定的服务等级协议,动态提供若干统一的计算资源[3]。尽管云计算已有很多通用的处理框架,也有很多较为成熟的云基础设施和云计算平台,

到稿日期:2013-09-01 返修日期:2013-11-01 本文受国家自然科学基金(61003157),广东省自然科学基金资助项目(10451064101005155,S2011010001754),广东省科技计划项目(2012B010100030),广东省战略性新兴产业核心技术攻关项目(2011A010801002)资助。
陈泽琳(1962-),女,硕士,副教授,主要研究领域为云计算和移动计算、分布式并行系统、建筑信息模型等,E-mail:cszlchen@scut.edu.cn;潘运军(1987-),男,硕士,主要研究领域为云计算、基于位置的服务;何浥尘(1992-),男,主要研究领域为KD树算法、基于位置的服务;齐德昱(1959-),男,博士,教授,博士生导师,主要研究领域为云计算、智能计算、分布式并行系统等。

例如 IBM“蓝云”计算平台[4]、微软的 Azure 云平台[5]、Amazon 的弹性计算云[6]等，但是由于应用的复杂性和多变性，云计算环境仍有很多关键问题和技术难题需要解决。云计算中面向城市空间信息的快速定位和检索技术以及访问城市建筑信息的大用户群调度和均衡算法都是 BIM 云服务要解决的难点。本文第 2 节介绍了相关研究；第 3 节将介绍提出的四层 BIM 云服务框架；第 4 节将介绍城市空间位置检索算法；第 5 节将介绍城市空间信息的负载均衡算法；最后给出了实验结果分析和结论。

2 相关研究

云计算一直与大规模数据处理紧密相关，它是以虚拟化技术为基础，可以整合数据、存储、计算和服务等分布式共享资源，具有可扩展性的协同超级计算模式[7]。数据和存储的整合是云平台的基础，Brad Calder[8]等提出了全球基础服务、云基础设施服务以及应用服务的云平台数据组织架构。Mohamed[9]等给出了部署在 Web 服务上的云平台框架。Faramarz[10]等设计了具有自适应性和分布式工作流程执行的面向服务的架构。Craig[11]等描述了 Force. com 应用部署平台和基于元数据驱动的软件架构，该平台能够支持超过 55000 个组织并提供面向开发者的开发框架和 API 等。

同时，将云计算技术运用到 BIM 应用也得到了相关领域专家的重视，Zhang[12]等比较了当前 BIM 云计算框架，包括 Revit 云和 STRATUS。文献[13,14]也分别对 BIM 云计算进行了初步探讨。Autodesk 的 BIM 360 Glue[15]是建立在云平台上的具有存储、访问、共享、检索以及协同工作等功能的工具。但是总体而言，云平台上的 BIM 应用架构研究尚处于起步阶段。

城市空间位置检索是 BIM 的一个内容，在 LBS(Location-Based Services)中也包含城市空间位置检索的研究。文献[16,17]介绍了 LBS 系统的架构和关键技术，包括云平台的 LBS 等。空间位置检索首先应建立空间数据索引，该工作早在 20 年前已开展研究，出现了 R-Tree 索引法[18,19]、KD-Tree 索引法[20,21]和 Quard Tree 索引法[22,23]等。于戈[24]等对云平台上处理图数据的关键问题和机制做了综述。文献[25]提出了一种基于栅格统计的自适应空间划分算法。Yang 和 Ma 等[26,27]提出了基于 MapReduce 架构的大规模历史轨迹数据处理算法。

云存储的负载均衡研究较多，关于 Hadoop 的负载均衡改进算法有文献[28]的实时动态分配方法、文献[29]的动态分配网络带宽的数据均衡算法、文献[30]的超负载机架的优先处理及引用排序策略的算法。

3 BIM 云服务框架

BIM 应用的特点是数据量大、数据类型复杂、资源共享要求高、应用形式多样化、协作性强、生命周期长等。云平台的分布式计算架构、支持海量数据存储和方便水平扩展等特性正适用于 BIM 城市空间信息的存储和应用服务。基于云平台的 BIM 应用架构可描述为图 1 的形式。

BIM 的城市空间信息组织在云平台上，城市空间的内容服务供应商通过开放式内容组织接口进行数据更新，BIM 的用户可以通过 Web、客户端程序或移动端 APP 等访问或操作各自的应用。云服务平台支持 BIM 全生命周期的共享资源和协同工作。

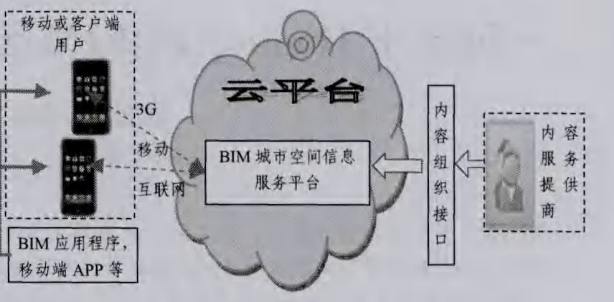

图 1 BIM 云服务平台的组织架构

在 BIM 云服务框架设计中，我们选用目前应用最成熟的云计算开源项目—Hadoop，它是开源的分布式软件框架，具有可靠、可伸缩和高效的特点[31]，其集群部署可扩展到上万个服务器，每个服务器称为一个节点，可以有独立的计算和存储功能。该框架通过 HDFS(Hadoop Distributed File System)管理分布式文件系统，每个集群有一个主服务器保存名字节点(Name Node)，其余为从服务器保存数据节点(Data Node)。该框架还提供建立在 HDFS 之上的 HBase 分布式数据库，它是面向列的关系表，不同于关系型数据库，适合于存储超大规模的较散乱的结构化数据。Hadoop 可以将应用程序的工作分解成很多工作小块(small block of work)，由 MapReduce的分布式计算模型并行处理 TB 级数据。

BIM 云服务层次框架如图 2 所示，该框架共分 4 层，每层的描述如下：

最底层——云存储层。该层组织计算机集群和分布式数据，要求该层的存储能力具有可扩展性。在 Hadoop 架构下，集群是由主服务器和从服务器组成的，分别对应名字节点和数据节点，数据由 HDFS 或 HBase 组织。

第二层——云平台服务层。包含云存储上的操作集，如 Hadoop 中的处理并行计算的 MapReduce、集群资源管理框架 YARN、提供分布式的高效协调服务 Zookeeper 和负载均衡服务 Balancer 等。

第三层——应用服务层。主要由 web 服务器和应用服务器组成，该层包括两个服务组和一个可装配服务模块，其中访问操作服务组主要通过访问云端数据完成 BIM 用户的访问和操作；内容支持服务组负责完成内容提供商对云端数据的更新、维护和分析；可装配服务模块是面向第三方开发商的服务扩展操作。

最顶层——应用层。可以是各种客户端程序、Web 页面或移动 APP 等，用户通过应用层完成各自的应用。

云存储的优势在于高效批量处理海量的大数据，但是云平台服务不能有效处理即时更新的数据，所以我们在第二、三层，通过构建应用的公有云、私有云、共享云等，为即时更新类的应用提供有效的服务。该框架具有以下特点：

1)层次化：四层架构的结构更清晰，应用和服务的定义和内涵更规范；

2)可扩展性：具有集群节点的可扩展性、服务的可扩展性，所以数据、业务和用户都可扩展；

3)云架构与云服务结合：云架构由 Hadoop 框架决定，云服务第二、三层构成，组成了 IaaS(基础设施即服务)、PaaS(平

台即服务)和 SaaS(软件即服务)的完整框架;

4)云服务的自动化优化管理:具有云平台的自动化管理服务,见第 5 节的负载均衡算法;

5)开放性:可装配服务模块为开发者提供了开放的框架。

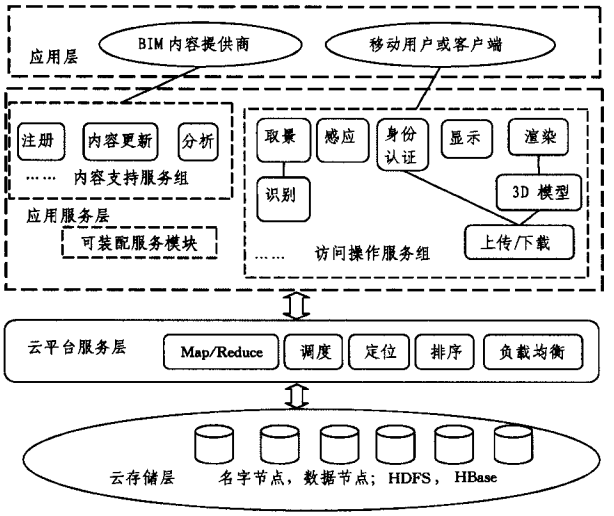

图 2 BIM 云服务层次框架

4 城市空间位置检索算法

城市空间位置检索是 BIM 应用的基础,本小节描述我们提出的城市空间位置的快速检索算法。空间信息检索取决于其数据的组织和索引方法,空间索引就是按照一定的排列顺序实现快速的空间实体检索,以下主要描述空间数据组织和空间数据索引算法。

首先,建筑信息的逻辑结构是以树形结构组织,即由国家、省、市、行政区的节点树描述空间数据。建筑信息由建筑的属性信息和几何信息构成,其中属性信息包含结构化属性信息和非结构化属性信息(如立面图、三维模型等),建筑的几何信息主要描述建筑的位置,本文采用建筑中心点和建筑轮廓点集方法描述。数据的物理存储方式是,非结构化属性信息保存在节点树中,直接以 HDFS 的文件和目录方式保存;结构化属性信息和几何信息主要用于检索,是以记录的形式存储在 HBase 数据库。以上两类数据都是以 HDFS 的数据块为单位存储在集群的节点上。

在庞大的空间数据中能够快速地检索出正确的建筑物,是目前地理定位的研究内容。本文提出了一种结合结构化属性信息和改进的 KD 索引树方法来进行快速检索的算法。

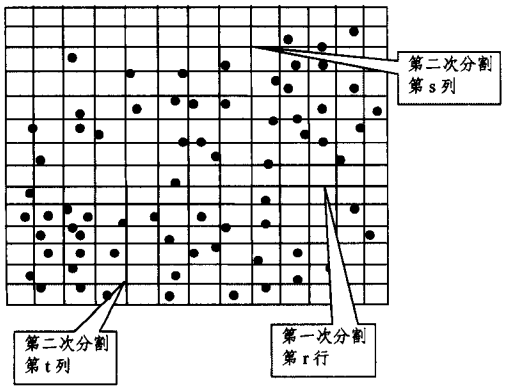

图 3 KD 树的分割线选择

如果用建筑的中心点作为 KD 树的分割面,则构造的 KD 树将过于庞大,本文改进的 KD 树是用平均建筑点数作为分割条件,构造的二叉树是一个完全二叉树,且所有叶子节点具有近似相等的建筑个数。用递归方法实现 KD 树,递归深度可以由叶子节点的建筑个数决定,适用于建筑分布稠密不均的各种城市空间情况。KD 树的分割线选择如图 3 所示。

算法 1 建立 KD 索引树

输入:城市空间的 M×N 区域栅格,栅格的建筑数 B,区域中的建筑数上限 Threshold(用于判断递归结束)

输出:KD 索引树

```
KDTree(grid,direction ) {
变量 BCurrNum 为当前分割线前的建筑总数
变量 BnextNum 为当前分割线与下一分割线之间的建筑总数
1. if(grid. BNum < Threshold) /* 当前区域的建筑数少 */
2.    return;
3. if(direction='X') {/* X 方向分裂 */
4.    BCurrNum =0;/* 分裂前按列统计建筑数 */
5.    for(i:1-M) {/* 遍历所有列 */
6.      for(j:1-N) /* 遍历所有行 */
7.        BCurrNum= BCurrNum+B[i,j]/* 统计建筑数 */
8.        If ( BcurrNum > = grid. Bnum/2) or (BcurrNum < =
          grid. Bnum/2) and((BcurrNum+BnextNum)>grid. Bnum/
          2) {
9.        用 i 列分割当前区域;
10.         左区域放到左子树,且 gridL. BNum=BcurrNum;
11.         右区域放到右子树,且 gridR. Bnum = grid. Bnum-Bcurr-
            Num;
12.       KDTree(gridL,'Y');
13.       KDTree(gridR,'Y');
14.     }
15.   }
16. }
17. if(direction='Y') {/* Y 方向分裂 */
18.   在 Y 方向上选择分割线,算法同 X 方向分裂…
19. }
20. }
```

算法 1 中第 8 行的作用是判断当前列分割的左右两个区域的建筑数是否近似相等,如果不等,继续遍历下一列;否则,执行分割并将两个区域放到 KD 树中,递归分割。

算法 2 空间位置检索算法

输入:KD 树,要检索的位置坐标 p 和半径 r:range(p,r)

输出:检索区域内的建筑集合 BSet[n]

```
1. for(每一个 KD 树的叶子节点 nodei)
2.   if(nodei 的节点区域 与 range(p,r) 相交)
3.     for(每一个 nodei 里的建筑 Bj)
4.       if(Bj 与 range(p,r) 相交)
5.         Bj≥BSet[n];/* Bj 是检索到的建筑之一 */
```

算法 1 的时间复杂度是 $O(M*N*K)$,其中 M 和 N 分别是城市划分的栅格数,K 为 KD 树递归深度。因为算法 1 是算法 2 的预处理过程,可以离线处理,且更新的频率较低,所以算法 1 的效率不会影响检索算法的效率。

算法 2 的时间复杂度是 O(Num * Threshold),其中 Num 是分割后的区域总数(即 KD 树的叶子节点数),Threshold 是区域中建筑数上限。这两个值一般不会很大,所以检索算法的效率能得到保证。

5 负载均衡算法

在分布式集群环境中，系统必然会出现数据负载不均衡的问题。数据负载均衡除了均衡各节点的存储空间使用率，更重要的是提高用户访问效率。作为云平台框架，Hadoop 本身具有负载均衡处理机制，其策略主要是将负载过重的节点的部分数据块移动到负载较轻的节点，以使各节点的负载达到相对平衡[32]。该负载均衡仅是将数据均衡地分布在集群的各节点上，控制了各节点的空间使用率，但没有考虑数据的访问频度，难以满足实际中各种具体应用的需要。

本项目的应用特点是，部分建筑数据会同时有大量用户访问，而大部分的建筑数据访问量很少。如果对这些建筑数据不区分对待，所有数据都保存相同的复本数，并平均分配到各个数据节点，则在实际访问时，会导致某些节点出现拥塞，用户等待时间过长，而另一些节点是空闲的情况。本文提出了一种提高用户响应时间的均衡算法，以保证部分常被访问的数据不会出现拥塞，提高用户访问数据的效率。

定义 1 单位时间内，用户访问数据块的次数称为访问频度，记作 f。

定义 2 单位时间内，用户访问数据块的最大次数称为最大访问频度，记作 f_{max}。

定义 3 单位时间内，用户访问数据节点的平均次数称为节点访问频度，记作 fs。

设节点 j 有 k 个数据块：$m_1, m_2, \cdots, m_k$，对应数据块的访问频度为 $f_1, f_2, \cdots, f_k$，则：

$$fs=\sum_{i=1}^{k}(\alpha \cdot f_i)/k$$

其中，$\alpha \in [0,1]$为权重因子，f_{max}越大则 α 越大，且$\sum_{i=1}^{k}\alpha=1$。

本算法的目标是如何设计负载均衡策略，使得用户访问数据块的响应速度最小。因为访问数据块的响应速度与节点的处理速度、网络带宽和访问频度有关，在不影响实际应用的情况下，我们可以假定节点的处理速度、网络带宽是常量，而节点的访问频度与访问数据块的响应速度成反比。所以，算法的负载均衡策略是使得节点的访问频度降低。

为了判断节点的访问响应情况，本算法将节点访问频度分为 3 个级别，为此我们定义集群访问频度。

定义 4 集群中所有节点的访问频度的平均值称为集群访问频度，记作 F。

设集群的节点数为 N，则：

$$F=\sum_{i=1}^{N}fs_i/N$$

对于给定的一个阈值 δ，3 个级别的节点分别定义为：

(1)繁忙节点 BusyS，当 $fs>F+\delta$；

(2)普通节点 UsualS，当 $F-\delta<fs<F+\delta$；

(3)空闲节点 IdleS，当 $fs<F-\delta$。

算法 3 负载均衡算法

```
if(fs ∈{BusyS})
/*1. 增加数据块的复本*/
for(该节点的所有数据块 i)
  if(fi> fs +Δ)&(块 i 的复本数< N)
  /*当第 i 个块是高频块*/
  then 增加该块复本数 C,并存储在空闲节点;
/*2. 移动数据块*/
if({IdleS}≠∅)/*空闲节点集不为空时*/
  for(节点 j∈{IdleS})
    if(fi> fs +Δ) &(块 i ∉节点 j)
    then 将数据块 i 移动到空闲节点 j 中;
      break;
```

算法中，Δ 为一给定的高频度常量，用于确定数据块是否属于高访问频度。

上述负载均衡算法主要由两个步骤组成，步骤一是增加高访问频度数据块的复本，每个复本存储在不同的节点，当多用户并发访问同一数据时，存储该数据复本的每个节点分别服务于一部分用户，从而分散节点数据块的访问压力，降低访问频度。步骤二是将高访问频度的数据块移动到空闲节点，从而降低访问频度。

算法经过多次迭代，高访问频度的数据块将分布在不同的节点中，且复本数趋于合理，各节点自动地持久保持数据块的均衡响应。

6 实验与结果分析

由于 Hadoop 的容错性很高，对集群中的服务器没有过高的要求，本实验环境由一台 Dell PowerEdge 2600 作为主服务器，其余为普通 PC 机，共搭建 7 个节点服务器，其中一个兼设置为 Web 服务器和 HDFS 客户端，集群部署如图 4 所示。本文实验环境是 Ubuntu Linux，Hadoop 配置文件在 conf 目录下，主服务器的名字节点配置文件为 conf/core_site.xml，初始配置 HDFS 数据块复本数为 3[33]。

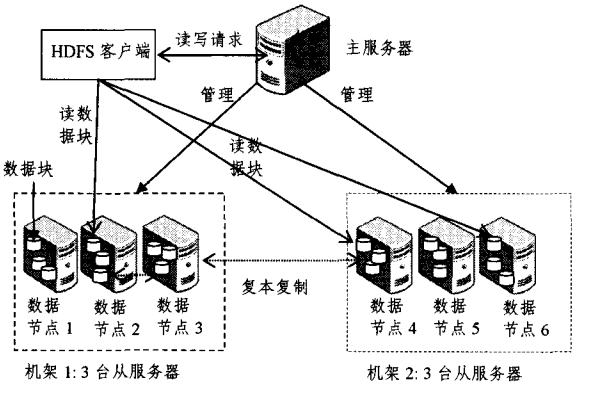

图 4 实验中的 Hadoop 集群部署

为了提高空间位置检索算法的实际运行效率，我们对 KD 树的存储结构进行了优化，将 KD 树的叶子节点组织在一个区域映射表中，表的每个元素包括区域坐标、该区域的建筑物总数、该区域的建筑表地址等。因为每次检索都要查找区域映射表，我们将区域映射表制作 6 个复本，并设置缓冲区保存，提高了多用户并发访问的时间，并减少了与硬盘的 I/O 读写时间。表 1 给出了不同用户数在有缓存和无缓存时打开区域映射表的时间。

表 1 不同用户数在有缓存与无缓存时的打开文件时间

用户数/缓存情况		打开区域映射表文件的时间	平均打开文件的时间	最坏情况打开文件的时间
50 个用户	无缓存	0.136s	0.143s	0.287s
	有缓存	0.084s	0.094	0.145s
200 个用户	无缓存	0.334s	0.413s	0.583s
	有缓存	0.169s	0.231s	0.363s

本实验使用美国芝加哥城市的建筑数据[34]，有建筑实体

N=801020 个，城市面积约 14311 平方公里。

如果取 Threshold 值为 4000，则分割的区域数为 BC=[N/Threshold] = 200，生成 KD 树的递归深度为 k=[$\log_2 BC$]=8；如果取 Threshold 值为 2000，则 BC=400，k=9；如果取 Threshold 值为 1000，则 BC=800，k=10。实验中我们假设栅格大小为 0.5 * 0.5=0.25 平方公里，则栅格数为 57244。表 2 给出了不同 Threshold 值时改进的 KD 树生成时间。

表 2 改进的 KD 树生成时间

建筑数	城市面积	Threshold	栅格大小	KD 树生成时间
801020	14311km²	4000	0.25km²	34.579s
801020	14311km²	2000	0.25km²	42.351s
801020	14311km²	1000	0.25km²	51.986s

因为 KD 树是城市建筑的空间位置索引，所以其生成时间不影响空间位置的查找。

表 3 是不同用户数在不同区域分割（Threshold 值不同）的情况下空间位置的检索响应时间。

表 3 空间位置检索时间

用户数	Threshold	检索半径	跨越区域个数	时间
50	4000	1000m	1	0.81s
			4	1.66s
	2000	1000m	1	0.63s
			4	1.53s
	1000	1000m	1	0.67s
			4	1.55s
200	4000	1000m	1	6.24s
			4	17.23s
	2000	1000m	1	5.02s
			4	15.78s
	1000	1000m	1	5.57s
			4	16.31s

从上述结果看，在同一个城市情况下，如果区域分割的 Threshold 值较大，则分割的区域数较少，检索时比较区域数的时间较少，但区域内的建筑比较耗时较长。反之，如果区域分割的 Threshold 值较小，则分割的区域数较多，检索时比较区域数的时间较长，但区域内的建筑比较的时间较少。所以找到一个 Threshold 的临界值可以得到最佳检索时间。

结束语 本文提出了四层结构的面向 BIM 应用的云服务框架，该框架具有可扩展性、开放性、服务自动化优化管理以及云架构和云服务相结合等特点。城市空间位置检索是 BIM 应用的基础，本文设计了城市空间信息的云存储组织，提出了空间位置检索算法，建立了改进的 KD 树索引。本文还提出了面向空间位置检索的负载均衡算法。实验结果表明，算法能有效地处理多用户并发访问，检索的响应速度较理想。

构建 BIM 云服务框架是一项庞大的任务，本论文只是完成了初步工作，下一步将继续完善该框架，进一步构建协同作业、业务流程和安全控制等服务模式。

参 考 文 献

[1] Harris D A, et al. National BIM Standard-United States Version 2. National Institute of Building Sciences[OL]. [2012-08]. http://www.Nationalbimstandard.org/

[2] Azhar S, Hein M, Sketo B. Building Information Modeling (BIM): Benefits, Risks and Challenges[C]// Proceedings of the 44th ASC National Conference. 2008: 5-13

[3] Rajkumar B, Yeo C S, Venugopal S. Market-oriented Cloud Computing: vision, hype and reality for delivering IT services as computing utilities[C]// Proc of the 10th IEEE International Conference on High Performance Computing and Communications. 2008: 5-13

[4] IBM 'SmartCloud' Computing Platform[OL]. [2013]. http://www.ibm.com/cloud-computing/us/en/managed-cloud-services.html

[5] Microsoft 'Windows Azure' Cloud Platform [OL]. http://www.windowsazure.com/, 2013

[6] Amazon Elastic Compute Cloud (Amazon EC2)[OL]. http://aws.amazon.com/cn/ec2/, 2013

[7] Luis M V, Luis R-M, Juan C, et al. A Break in the Clouds: Toward a Cloud Definition[J]. ACM SIGCOMM Computer Communication Review, 2009, 39(1): 50-55

[8] Calder B, Wang Ju, Ogus A, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency [C]// Proc of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP'11). New York: ACM, 2011: 143-158

[9] Mohamed, Wahib, Munawar A. A Framework for Cloud Embedded Web Services Utilized by Cloud Applications[C]// 2011 IEEE World Congress on Services. 2011: 265-271

[10] Esfahani F S, Murad M A A, Sulaiman M N B, et al. Adaptable Decentralized Service Oriented Architecture [J]. The Journal of Systems and Software, 2011, 84(10): 1591-1617

[11] Weissman C D, Bobrowski S. The Design of the Force.com Multitenant Internet Application Development Platform[C]// Proceedings of the 2009 ACM SIGMOD International Conference. New York, USA, ACM, 2009: 889-896

[12] Zhang L, Issa R. Comparison of BIM Cloud Computing Frameworks[J]. Computing in Civil Engineering, 2012: 389-396

[13] Zhen Bo-bi, Wang Hui-qin. BIM Application Research Based on Cloud Computing[J]. Applied Mechanics and Materials, 2012 (170-173): 3565-3569

[14] 何清华，潘海涛，李永奎，等. 基于云计算的 BIM 实施框架研究[J]. 建筑经济，2012，5：86-89

[15] Autodesk 360 cloud-based platform and BIM 360 Glue[OL]. [2013]. http://www.autodesk.com/360-cloud, http://www.autodesk.com/products/bim-360-glue/overview

[16] 周傲英，杨彬，金澈清，等. 基于位置的服务：架构与进展[J]. 计算机学报，2009，34(7)：1155-1171

[17] Lee D K, Zhu M, Hu H. When Location Based Services Meet Databases[J]. Mobile Information Systems (MIS), 2005, 1(2): 81-90

[18] Guttman A. R-Trees: A dynamic index structure for spatial searching[C]// Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. Boston, 1984: 47-57

[19] Beckmann N, Kriegel H P, Schneider R, et al. The R-Tree: An efficient and robust access method for points and rectangles[C]// Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data. Atlantic City, 1990: 322-331

[20] Bentley J L. Multidimensional binary search trees used for associative searching[J]. Communications of theACM, 1975, 18(9): 509-517

（下转第 117 页）

备较好的代表性。

由于用户进行每步操作时，浏览器都会和 Scratch 发生交互，因此用户在网页上的操作次数便成为我们需要考察的因素。本实验中，选取点击(click)这一典型的浏览器事件进行测试，分别对两个网站进行 20 次和 60 次点击后(百度的测试包含输入关键词搜索后对结果页的点击)，记录内存消耗情况和 CPU 线程数，如表 2 所列。

表 2　Scratch 性能测试

测试网站	是否开启 Scratch	CPU 线程数	内存	Js 使用内存
百度(点击 20 次)	不开启 Scratch	1.5	25400k	3635k
http://www.baidu.com	开启 Scratch	1.5	25980k	6400k
百度(点击 60 次)	不开启 Scratch	1.5	26500k	3028k
http://www.baidu.com	开启 Scratch	1.5	27500k	6500k
雅虎(点击 20 次)	不开启 Scratch	2	80000k	14521k
http://www.yahoo.com	开启 Scratch	2	82000k	26400k
雅虎(点击 60 次)	不开启 Scratch	2.5	74500k	16000k
http://www.yahoo.com	开启 Scratch	3	84000k	26700k

表 2 中，内存表示该 Web 应用进程(Chrome 中每个 Web 应用分配一个进程)所占用的全部内存；由于 Scratch 以 JavaScript 实现并在运行时注入到被测应用中，因此我们单独记录了 Js 所占用的内存(需要注意的是，这部分还包含网页自身的 Js 文件解析所需的内存)。可以发现，在百度的测试结果中，Scratch 开启与否对内存和 CPU 几乎没有影响，而随着点击次数上升，使用 Scratch 后 Js 使用内存则有小幅上升(最多也仅有 1M 左右)，这是因为 Scratch 会对每次操作记录。但总体上看，开启 Scratch 后对用户浏览网页的影响很小。

在性能测试过程中，我们也对网页变化性进行了简单实验。在百度和雅虎的所有 160 次操作中，每次回放时页面都会发生一定的变化，而 Scratch 都能够准确定位到变化后的节点(如果存在的话)，初步验证了本文所提节点匹配算法的有效性。

结束语　本文实现了一个基于 Chrome 浏览器的用户记录捕捉与回放工具，解决了网页动态结构与内容变化对捕捉回放技术带来的挑战，通过在线和离线两种模式保证回放的一致性和正确性。

目前 Scratch 在用户脚本上的编辑还不太完善，不便于用户共享和复用。未来将结合自然语言处理技术，提高记录脚本的可读性、可编辑性和可定制性。同时，在更广的范围内如 Web 应用自动化测试、基于 Web 的在线协同等尝试应用 Scratch。

参 考 文 献

[1] Eclipse. org. TPTP[OL]. http://www.eclipse.org/tptp

[2] Kojouharov C, Solodovnik A, Naumovich G. JTutor: an Eclipse plug-in suite for creation and replay of code-based tutorials[C]// Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology Exchange(Eclipse'04). ACM, New York, NY, 2004: 27-31

[3] Leshed G, Haber E M, Matthews T, et al. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise[C]// CHI. 2008

[4] Mickens J, Elson J, Howell J. Mugshot: Deterministic Capture and Replay for JavaScript Applications[C]//NSDI. 2010

[5] Lau T, Bergman L, Castelli V, et al. Sheepdog: Learning Procedures for Technical Support[C]//IUI. 2004: 109-116

[6] Little G, Lau T A, Cypher A, et al. Koala: Capture, Share, Automate, PersonalizeBusiness Processes on the Web[C]// ACM. 2007: 109-116

[7] Chrome Extension Structure [OL]. http://developer.chrome.com/extensions/getstarted.html

[8] ECMAScript Standard [OL]. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

(上接第 111 页)

[21] Robinson J T. The KDB Tree: A Search structure for large multidimensional dynamic indexes[C]// Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data. Ann Arbor, 1981: 10-18

[22] Samet H. The quad tree and related hierarchical data structures [J]. ACM Computing Surveys, 1984, 16(2): 187-260

[23] Finkel R A, Bentley J L. Quad Trees a Data Structure for Retrieval on Composite Keys[J]. Acta Informatica, 1974, 4(1): 1-9

[24] 于戈，谷峪，鲍玉斌，等. 云计算环境下的大规模图数据处理技术[J]. 计算机学报，2011，34(10)：1753-1767

[25] 王结臣，张辉，吴文周，等. 一种平面散乱点集的自适应空间划分算法[J]. 武汉大学学报，2012，37(007)：770-774

[26] Yang B, Ma Q, Qian W, et al. TRUSTER: Trajectory data processing on clusters[C]// Proceeding of the 14th International Conference on Database Systems for Advanced Applications. Brisbane, 2009: 768-771

[27] Ma Q, Yang B, Qian W, et al. Query processing of massive trajectory database on MapReduce[C]// International Conference on Information and Knowledge Management. Hong Kong, China, 2009: 9-16

[28] Ye Xiang-long, Huang Meng-xing, Zhu Dong-hai, et al. A novel blocks placement strategy for Hadoop [C]// Proceedings of the 11th International Conference on Computer and Information Science. Washington D C: IEEE, 2012: 3-7

[29] 林伟伟，刘波. 基于动态带宽分配的 Hadoop 数据负载均衡方法[J]. 华南理工大学学报：自然科学版，2012，40(9)：42-47

[30] 刘琨，肖琳，赵海燕. Hadoop 中云数据负载均衡算法的研究及优化[J]. 微电子学与计算机，2012，29(9)：18-22

[31] Borthakur D. The Hadoop Distributed File System: Architecture and Design[EB/OL]. [2012-01-03]. http://hadoop.Apache.org/common/docs/stable/hdfs_design.html

[32] Apache Hadoop. Rebalance Data Blocks [EB/OL]. http://issues.Apache.org/HADOOP1652, 2009

[33] White T. Hadoop: The Definitive Guide [M]. Beijing: Tsinghua University Press, 2010

[34] GitHub Inc. Building Footprints in Chicago [EB/OL]. https://github.com/Chicago/osd-building-footprints, 2013