支持频繁位置更新的不确定移动对象索引策略

丁晓锋 金 海 赵 娜

(华中科技大学服务计算技术与系统教育部重点实验室 武汉 430074)
(华中科技大学集群与网格计算湖北省重点实验室 武汉 430074)

摘 要 移动数据采集和处理技术的迅速发展给研究人员提出了新的应用需求,如何在频繁位置更新应用中索引不确定移动对象的当前及未来位置信息成为当前的研究热点之一. TPU 树是针对不确定移动对象的当前及未来位置信息索引的策略,其具有较高的概率域查询效率,但是其采用的传统自顶向下更新算法,存在频繁位置更新效率低下的问题. 通过在 TPU 树上增加一个记录不确定移动对象状态特征的更新备忘录(UM)内存结构,文中提出了一种支持频繁位置更新的不确定移动对象索引策略 TPU^{2}M 树,并在此基础之上提出了一种改进的基于备忘录(MMBU/I)的更新/插入算法. 代价分析和实验仿真表明,采用 MMBU/I 算法的 TPU^{2}M 树频繁更新性能大大优于 TPU 树和 AB^{x}树索引,且概率查询性能与传统索引大致相当,因此具有很好的实用价值和广泛的应用前景.

关键词 不确定移动对象;索引结构;基于备忘录更新;TPU 树;概率查询
中图法分类号 TP392 **DOI 号**: 10.3724/SP.J.1016.2012.02587

Indexing of Uncertain Moving Objects with Frequent Updates

DING Xiao-Feng JIN Hai ZHAO Na

(*Key Laboratory of Services Computing Technology and System of Ministry of Education, Huazhong University of Science and Technology, Wuhan* 430074)
(*Key Laboratory of Cluster and Grid Computing of Hubei Province, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract The rapid advances in mobile data collecting and processing technology has proposed the researchers new challenges: how to monitor the current and future positions of uncertain moving objects under frequent and high extent updates. TPU-tree is currently a popular indexing method for the current and future positions of uncertain moving objects. It can speed up the probabilistic range query efficiency, but the traditional top-down update method of TPU-tree has made its frequent updates performance very low. In this paper, we propose the TPU^{2}M-tree for moving objects with frequent updates, which is based on TPU-tree, supplemented by a memory-based update-memo structure recording the state of uncertain moving objects. Furthermore, a modified memo-based update/insert algorithm is developed for TPU^{2}M-tree. Cost analyses and experimental evaluations demonstrate that the TPU^{2}M-tree outperforms significantly any other indexing method including TPU-tree and AB^x-tree with frequent updates, while yielding similar probabilistic query performance. TPU^{2}M-tree has more practical values and comprehensive application foreground than other indexing methods.

Keywords uncertain moving objects; index structure; memo-based update; TPU-tree; probabilistic query

---

收稿日期:2009-09-23;最终修改稿收到日期:2011-09-25. 本课题得到国家自然科学基金青年项目(61100060)、国家科技支撑计划重点项目(2008BAH29B00)、中国博士后科学基金面上项目(20100471179)、湖北省自然科学基金(2011CDB037)和中央高校基本科研业务费专项资金(2011QN054)资助.**丁晓锋**,男,1982 年生,博士,讲师,主要研究方向为不确定数据管理、查询处理与优化、移动计算. E-mail: xfding@hust.edu.cn. **金 海**,男,1966 年生,博士,教授,博士生导师,主要研究领域为海量数据管理、网格计算、并行与分布式计算等. **赵 娜**,女,1982 年生,硕士,助理工程师,主要研究方向为网络架构、信息传播.

1 引 言

近年来,伴随着科学技术的不断进步,人们对数据的采集方式呈现多样化和对数据处理技术的逐步深入研究,不确定性数据得到学术界和工业界的广泛重视[1-2].在诸多应用如实时交通信息管理与导航、军事监控和民航管制中,移动终端的实时位置信息的不确定性普遍存在,如何对移动终端的不确定性实时位置信息进行监控和管理是一个至关重要的问题[3-5].传统数据库索引技术是为存储精确的数据而设计,其索引结构中存储移动对象的精确位置,因此无法有效地管理不确定性数据,从而学术界和工业界对研发新型的不确定性数据管理技术提出了要求.

针对如何高效管理移动对象实时变化的精确位置信息,研究人员提出了一系列的索引模型,根据位置信息的不同类型,大致可以分为两类[6]:一类是针对移动对象历史位置信息的索引;另一类是针对移动对象当前及未来位置信息的索引.其中包括许多基于参数化的索引方法来对移动对象当前及未来位置信息进行管理,如 TPR 树[7]及其变种 TPR* 树[8]、R^{EXP}树[9]和 STAR 树[10]等.TPR 树及其变种 TPR* 树由于沿用了传统 R 树的查询,插入及删除等算法而成为目前广泛使用的精确移动对象当前及未来位置信息索引方法,但其固有的自顶向下(top-down)更新模式,由于较大的 I/O 代价而难以满足大量并发更新的要求.R^{EXP}树借用 TPR 树的时间参数策略,通过在 R* 树上添加数据的有效期属性,提高了失效数据的删除效率,从而表现出比较好的更新性能.STAR 树在处理简单更新时也表现出很好的性能.研究人员提出能同时对移动对象历史、当前及未来位置信息进行索引的模型(BB^x树[11]、R^{PPF}树[12]).但上述索引方法包括 R 树家族[3]均不能很好地处理移动对象频繁位置更新.Lee 等人[13]提出了基于 R 树的自底向上(Bottom-Up Update,BUU)更新思想,其更新过程起始于要求更新的叶节点,无需浪费大量查找时间,从而大大提高动态更新性能,但辅助索引的维护和大量内存空间的占用导致系统稳定性较差,且不能很好地解决频繁大幅度位置更新问题.

针对如何管理和查询对象的不确定性位置信息,Tao 等人[1]提出了基于 R* 树的不确定对象索引策略 U 树,其固有的良好动态结构可以使得数据对象以任何次序更新或插入,而且对不确定数据本身的概率密度分布(Probability Density Function,PDF)没有任何限制.文献[4]针对支持不确定移动对象当前及未来位置信息索引和查询的问题,提出了一种基于 U 树的高效率当前及未来不确定位置信息检索的索引结构 TPU 树,并提出了一种改进的基于 p-bound 的域查询(Modified p-bound Based Range Query,MP_BBRQ)处理算法.实验表明,采用 MP_BBRQ 算法的 TPU 树概率域查询效率可以得到很大程度的提高.最近,Zhang 等人[5]提出来一种基于 B^x树的不确定移动对象索引策略 AB^x树,利用矩形框推论法则(Rectangle inference)和蒙特卡洛(Monte-Carlo)模拟相结合的方法预测移动对象未来的大概位置信息,并提出了高效的概率范围查询和概率 K 最近邻查询算法.但由于上述索引采用传统的自顶向下更新方法,不断变化的不确定移动对象位置信息,会使得更新过于频繁而导致系统资源枯竭,从而影响系统的响应时间和查询效率.

本文针对支持频繁位置更新的不确定移动对象当前及未来位置索引方法,提出了一种基于 TPU 树与更新备忘录(Update-Memo,UM)内存结构的 TPU^2M 树,并提出了一种改进的基于备忘录更新/插入(Modified Memo-Based Update/Insert,MMBU/I)算法.TPU^2M 树在基本 TPU 树结构上,增加了记录不确定移动对象状态特征的 UM 内存结构.MMBU/I 算法利用 UM 控制不确定移动对象的位置更新,在保留原有记录的情况下首先插入新记录,这样就减少了查找原有记录的磁盘 I/O,从而很大程度上提高了记录的更新效率,然后 TPU^2M 树利用空间清理器,定期清除索引树叶节点中包含的旧记录,同时维护并限定 UM 内存结构的大小,为索引树高效稳定运行提供保证.本文的主要贡献如下:

(1)本文引入了一种新颖的数据结构——UM 内存结构.通过与传统的索引结构 TPU 树相结合,提出了一种新的不确定移动对象索引结构 TPU^2M 树.

(2)本文出了一种改进的基于备忘录的更新/插入 MMBU/I 算法,采用合适的 TPU^2M 树索引,显著地提高了大量不确定移动对象并发更新的效率,并利用自底向上的更新思想来进一步提高索引树的更新效率.

(3)本文引入了空间清理器,利用合适的运行机制可显著地稳定 TPU^2M 树的运行效率.

(4)设计了详细的性能评价实验,并与传统算法进行了比较.实验结果表明,采用 MMBU/I 算法

的 TPU²M 树动态更新性能大大优于 TPU 树和 AB^{x}树索引，查询性能与同类索引大致相当.

本文第 2 节详细描述 TPU²M 树不确定移动对象索引的内部结构，及其更新与查询处理算法；第 3 节引入了空间清理器并具体讨论其对 TPU²M 树索引性能的影响；第 4 节给出不同更新策略的代价分析；第 5 节给出实验结果；第 6 节对全文工作进行总结，并介绍了未来的研究方向.

2　TPU²M 树不确定移动对象索引

U 树是 Tao 等人[1]提出的一种针对不确定对象当前位置信息管理的索引模型，丁晓锋等人[4]在此基础上提出了针对不确定移动对象的 TPU 树索引策略. U 树及 TPU 树采用传统的删除加插入两阶段更新策略，即当移动对象发出位置或速度更新请求时，算法首先从索引树根节点开始，依次比较此对象原有 MBR(VBR)与中间节点 MBR(VBR)大小，直至在叶节点中找到移动对象所在的索引项并删除，然后再执行一次自顶向下的搜索，在合适位置中插入新的移动对象记录. 自顶向下的更新模式非常简单直观，有助于索引结构的维护. 但由于所有的移动对象位置记录存储在索引树的叶节点中，采用此种模式往往从根节点开始搜索相关记录，从而具有很高的搜索定位代价，在频繁更新应用中必然导致索引树查询性能的下降.

鉴于此，我们采用类似于文献[14]中基于备忘录(Memo-Based Update，MBU)更新的思想来提高 TPU 树的动态更新和访问效率. 并利用 Lee 等人[13]提出的自底向上(BUU)的更新思想来进一步提高索引树的频繁更新效率. 根据基于备忘录更新的思想，索引结构必须能够区分相关移动物体位置的新旧记录，因此利用一个记录移动物体更新状态的更新备忘录(UM)内存结构来标识移动物体的最新位置记录.

为了将位置更新记录及时插入到索引树中，我们提出了改进的基于备忘录更新/插入(MMBU/I)算法，思想如下：首先判断其是否超出所在叶节点的 MBR(VBR)范围之内，若未超出范围，则直接更新叶节点页面及数据页面即可；否则，算法不需首先查找并删除相应的旧记录，而是允许同一对象的若干新旧位置记录共存于索引树中，即直接利用标准的 TPU 树插入算法自顶向下进行搜索，在合适的叶节点中插入新记录，并同时更新 UM 内存结构的相关内容以标识新旧记录. TPU²M 树在记录更新次数达到一定条件时激活空间清理器，然后空间清理器依据 UM 内存结构的内容清除当前叶子节点中包含的所有旧记录.

2.1　TPU²M 树索引结构

在 TPU²M 树中能否有效地管理和维护不确定移动对象的新旧记录是影响索引树性能的关键. 我们引入时间戳(time-stamp)概念来标识记录的新旧程度. 在 TPU 树基本索引结构基础上，每个 TPU²M 树叶子节点的记录项都添加了一个标识该记录插入时间的 *time-stamp* 属性，记录形式为〈*oid*，*PCR*(*pi*)，*VBR*，*ptr*，*time-stamp*〉，*oid*，*PCR*(*pi*)，*VBR*，*ptr*，*time-stamp* 分别表示不确定移动对象标识、P_i限定性区域、速度包围框、节点磁盘页面地址、时间戳.

为方便快速区分移动对象的最新记录和若干旧记录，TPU²M 树引入了辅助内存结构 UM，其包含数据项的记录形式为〈*oid*，*Slatest*，*Ntotal*〉，*oid*，*Slatest*，*Ntotal* 分别表示不确定移动对象标识、不确定移动对象 *oid* 的最新时间戳、不确定移动对象 *oid* 在索引树中的记录总数. 例如，〈100，20090623001，5〉表示标识为 100 的不确定移动对象在索引树中共有 5 条记录，其中具有时间戳‘20090623001’的记录是最新位置记录. 为进一步加快移动对象在 UM 中的查找速度，TPU²M 树索引还增加了一个建于移动对象标识 *oid* 之上的辅助索引(Hash)来定位不确定移动对象在 UM 中的位置.

另外，当空间清理器清除叶节点中包含的旧记录之后，TPU²M 树需要以自底向上的方式动态调整树结构，这就要求各节点能够访问父节点磁盘页面，因此，我们修改了 TPU 树索引节点的记录结构，增加了一个指向父节点磁盘页面地址的物理指针 *parent-ptr*. TPU²M 树索引节点记录形式为〈*block*，*level*，*num_entries*，*entry*，…，*entry*，*parent-ptr*〉，其中 *block* 表示该节点所在的磁盘页面号；*level* 表示该节点在索引树中的层次(叶结点层次为 0，根节点层次最大)；*num_entries* 表示该节点中包含的记录项数目.

图 1 所示为支持 MMBU/I 算法的 TPU²M 树索引结构. 其中右上角为辅助内存结构 UM. 特别地，考虑到大部分不确定移动对象都会发生位置更新从而在索引树中存在多条记录，我们在 UM 的记录的部分结构中，使 *Ntotal* 标识不确定移动对象在索引树中的总记录数，这样就保证 UM 在大小相当

的情况下，避免终因 *Ntotal* 仅仅标识不确定移动对象的旧记录数所引起的记录幻像问题. 此外，内存结构 UM 为索引结构快速准确在叶子节点中定位不确定移动对象的存储位置提供了可靠保证.

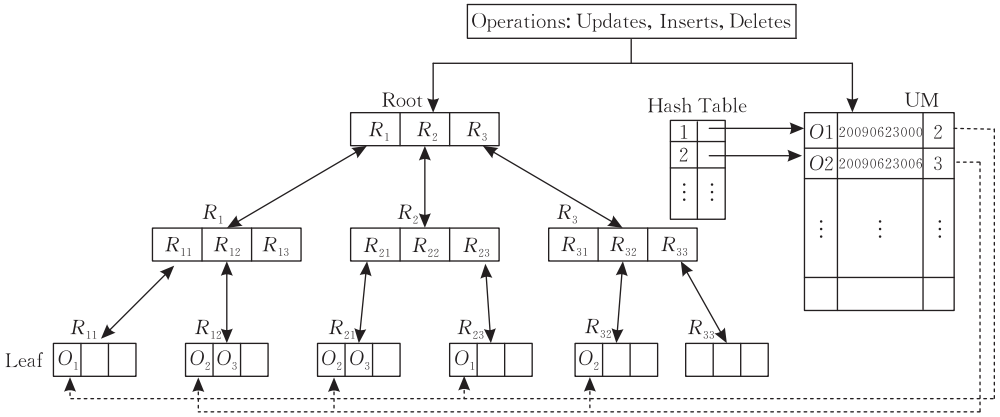


图 1 支持 MMUB/I 算法的 TPU²M 树结构

2.2 MMBU/I 算法

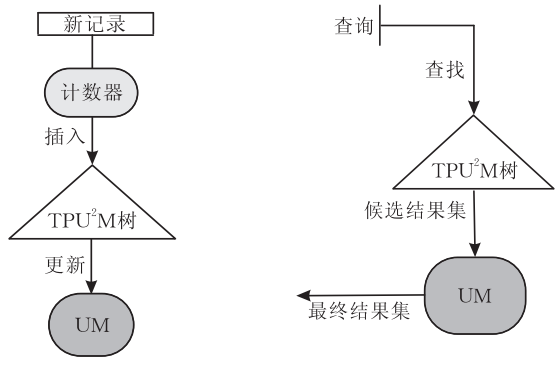
当不确定移动对象发出位置更新请求时，新的位置记录信息要求被 TPU²M 树索引，算法首先根据 UM 中的记录确定移动对象的叶节点，并判断其是否超出所在叶节点的 MBR(VBR)范围之内，若未超出范围，则直接更新叶节点页面及数据页面，否则更新过程等价于在索引树中插入新记录，因此，插入新记录和更新索引树中原有记录在 TPU²M 树中具有相同的处理过程，其关键在于如何处理产生的移动对象位置信息新记录. 图 2(a)所示为 TPU²M 树的插入处理过程. 其中关键的 MMBU/I 算法思想如下：首先给预处理的新记录分配时间戳，执行标准的 TPU 树插入算法在索引树中插入新记录，然后更新内存结构 UM 的内容，如果新记录的 *oid* 在 UM 中不存在，则插入新的 UM 数据项并设置相应 *Slatest* 为 *time-stamp*，*Ntotal* 为 1；否则，将相应 UM 数据项中的时间戳更新为 *time-stamp*，且移动对象 *oid* 的记录总数加 1. 算法 1 描述了 MMBU/I 算法.

算法 1. MMBU/I 算法.

Input: *oid*, *newPCR*, *newVBR*
Output: updated TPU² M-tree
Begin
1. *time-stamp* ← stamp counter;
2. new record = ⟨*oid*, *newPCR*, *newVBR*, *time-stamp*⟩;
3. insert the new record to the TPU² M-tree;
4. search *oid* in Update Memo UM;
5. if no entry is found in UM
6. then insert (*oid*, *time-stamp*, 1) to UM;
7. else
8. *fentry* ← entry found in UM
9. *fentry.time-stamp* ← *time-stamp*
10. *fentry.Ntotal*++;
11. end if
End

2.3 查询处理

TPU²M 树的查询处理过程类似于 TPU 树. 但由于移动对象的若干旧记录和新记录共存于 TPU²M 树中，因此需要在结果集中剔除满足条件的旧记录，保留满足条件的最新记录. 查询处理过程如下：首先利用标准 TPU 树查询处理算法，得到实际结果的候选集，然后利用 UM 作为过滤器清除候选集中的旧记录. 图 2(b)所示为 TPU²M 树的查询处理过程.



(a) 插入/更新处理过程 (b) 查询处理过程

图 2 TPU²M 树的相关处理过程

UM 过滤器的工作原理如下：比较候选集中叶子节点记录项与相应 UM 数据项的时间戳大小，若叶节点记录项的时间戳等于 UM 数据项的时间戳，则此记录为最新的不确定移动对象位置信息记录，并保存在最终结果集中，否则，此记录作为旧的移动对象位置信息记录而被清除. 算法 2 描述了 Filter 算法.

算法 2.　Filter 算法.

```
Input: leaf entry e
Output: answer-set
Begin
1.  search e.oid in Update Memo UM;
2.  fentry ← entry found in UM;
3.  if e.time-stamp == fentry.time-stamp
4.    then answer-set ← fentry;
5.  end if
6.  return answer-set;
End
```

3　空间清理器

大量移动对象位置信息旧记录的存在势必影响 TPU²M 索引树的查询和更新效率，因此，我们使用空间清理器(space cleaner)定期清除 TPU²M 树中包含的旧记录，并动态维护 UM 内存结构的大小，从而提高索引树的整体效率.

空间清理器利用清除(Clean)算法处理索引树中存在的移动对象位置信息旧记录，空间清理器的工作原则主要有两条：懒散处理(lazily)和批量删除(batches)，即索引树因更新或插入操作产生的移动对象旧记录，并没有被清理器当即删除，而是根据一定清理周期，在索引树当前操作的叶节点内批量删除该节点包含的所有移动对象旧记录. Clean 算法负责清除指定叶节点中包含的所有移动对象旧记录. Clean 算法首先将指定叶节点中的所有记录项与相应 UM 数据项作比较，删除判定出的旧记录，并对相应 UM 数据项作修改，然后根据该节点清除后的情况，动态调整索引树结构，算法 3 描述了 Clean 算法.

算法 3.　Clean 算法.

```
Input: leaf node N
Output: cleaned TPU²M-tree
Begin
1.  for each entry e in N
2.    search e.oid in UM;
3.    fentry ← entry found in UM;
4.    if e.time-stamp == fentry.time-stamp
5.      then ec++; //ec counts the latest entry
6.    else
7.      delete e from N
8.      fentry.Ntotal--;
9.        if fentry.Ntotal==0
10.         then delete fentry from UM;
11.       end if
12.   end if
13. end for
14. if ec < MINentries //the minimum number of entries in a leaf node
15. then reinsert the remaining entries of N into the TPU²M-tree;
16. else adjust the N and N's ancestors in a bottom up manner;
17. end if
End
```

另外，我们引入检查率(inspection ratio，ir)来衡量空间清理器检查叶节点的频率. 检查率 ir 定义为：在时间段 T 内，被清理的叶节点个数 C 与 TPU²M 树更新次数 U 的比率，即 $ir=C/U$. 其大小设置直接影响到 TPU²M 树的清理周期，对索引树整体性能有至关重要的影响. 较低的检查率虽然提高了 TPU²M 树的更新效率，但因此导致的过多移动对象位置信息旧记录会影响 TPU²M 树的查询效率；相反，检查率偏高将引起清理器频繁的扫描叶节点，在不同的触发机制下会影响到索引树的更新效率. 因此，检查率 ir 的设置在 TPU²M 树更新和查询效率之间存在平衡，我们将在代价分析和实验仿真中进一步说明.

4　代价分析

为方便对 TPU²M 树索引性能和不同的更新策略进行代价分析，参照文献[14]我们约定所需代价参数如下：L 为索引树中叶子节点的个数；E 为辅助内存结构 UM 中记录项的大小；H 为索引树的高度；M 为索引树中不确定移动对象的总个数以及移动对象更新总次数 U.

4.1　失效率与 UM 大小

根据空间清理器在 TPU²M 树中的运行机制，存在于叶子节点中的所有旧记录，在空间清理器的一次运行之后将被全部清除，而每个叶子节点在索引树累计至 L/ir 次插入/更新操作才被清理器扫描一次. 因此，在最坏情况下索引树中包含旧记录的个数为 L/ir.

作为 TPU²M 树中重要性能指标之一的失效率(obsolete ratio，or)，我们定义为索引树中旧记录个数占移动对象总数量的比率. 较高的失效率意味着过多移动对象位置信息旧记录存在索引树中，因而影响 TPU²M 树的查询效率. 失效率在索引树中

旧记录个数达到顶峰时具有上限值 $L/(ir\times M)$，因此很容易得到 TPU^{2}M 树中失效率的平均值为 $L/2(ir\times M)$. 假设每一个不确定移动对象旧记录占用内存结构 UM 一个数据项，则 UM 的上限尺寸为 $(L\times E)/ir$，且 UM 在索引树中的平均大小为 $(L\times E)/2ir$. 不难发现失效率 *or* 和 UM 大小都与叶子节点个数 L 成正比，而 L 远远小于移动对象总个数 M，因此失效率与 UM 在都保证了较小的值，使得 UM 在内存中得以有效运行，从而确保了索引树高效稳定的性能.

为进一步证明本索引的有效性，既新增内存结构 UM 的空间复杂度对索引性能的影响是甚微的，我们假设不需要删除索引树中冗余的旧记录，且移动对象的总数 M 保持不变，若一个 float 或 int 型数占 4 个 Bytes，那么 UM 的大小等于 $M\times(4+4+4)$Bytes$=12M$(Bytes)，当 M 取实验系统中最大的值 10 Million 时，UM 所占的最大内存为 120 MB，远小于系统提供的内存，因此，空间复杂度的增加对索引性能的影响是微小的.

4.2 更新代价

我们依次分析并给出 3 种不同更新策略：自顶向下更新、自底向上更新，以及基于备忘录更新所需的磁盘 I/O 次数，如表 1 所示. 其中自顶向下更新代价由两部分构成：(1) 查询并删除旧记录的代价；(2) 插入新节点的代价. 由于索引树包含节点之间的区域重叠性，查询一条记录在最好情况下也要访问 H 个节点，在最坏情况下要访问 $L\times(L-1)/4$ 个节点. 搜索到旧记录后要进行删除操作并写回磁盘，在没有节点下溢或上溢的情况下至少需要一次写磁盘. 插入新记录需要首先访问 H 个节点以搜索到合适的叶子节点，然后对该叶子节点进行读写磁盘操作. 因此，自顶向下更新策略的一次更新在最好情况下需要 $2\times(H+1)$ 次磁盘 I/O，最坏情况下需要 $L\times(L-1)/4+H+2$ 次磁盘 I/O.

表 1 3 种更新策略的代价分析

更新策略	一次更新所需磁盘 I/O 次数		
	最好情况	最坏情况	平均情况
自顶向下	$2\times(H+1)$	$L^2/4+H+2$	$L^2/8+1.5H+2$
自底向上	3	$H+6$	$H/2+4.5$
备忘录	$H+1$	$H+1$	$H+1$

当新记录插入到旧记录所在的原始叶子节点时，自底向上更新策略取得最小更新代价 3 次磁盘 I/O：读取辅助索引以定位到原始叶子节点，然后读取并写回该叶子节点. 当新记录插入到与旧记录所在原始叶子节点不相关的叶子节点时，自底向上更新策略取得最大更新代价 $H+6$ 次磁盘 I/O：读取辅助索引以定位到原始叶子节点，然后读取并写回该叶子节点，访问 H 个节点以搜索到合适的叶子节点，然后对该叶子节点进行读写磁盘操作以插入新记录，最后写回辅助索引以定位新记录. 基于备忘录的更新策略只需考虑将新记录直接插入到叶子节点的代价，因此，无论是最好或者最坏情况下该更新策略均需要 $H+1$ 次磁盘 I/O. 需要指出，由于 TPU^{2}M 树中使用的空间清理器在索引树当前操作节点中进行清理工作，因此清除旧记录并不需要多余的磁盘 I/O. 索引树的高度通常在 4 层左右，因此基于备忘录更新在平均情况下具有最小的磁盘 I/O 次数.

5 实验仿真与性能评估

为切实评价 TPU^{2}M 索引树在不确定移动对象频繁位置更新情况下的索引性能，我们设计了一组实验，对文中提出的算法给出了验证分析，同时对 TPU 树，AB^{x}树和 TPU^{2}M 树进行了性能比较.

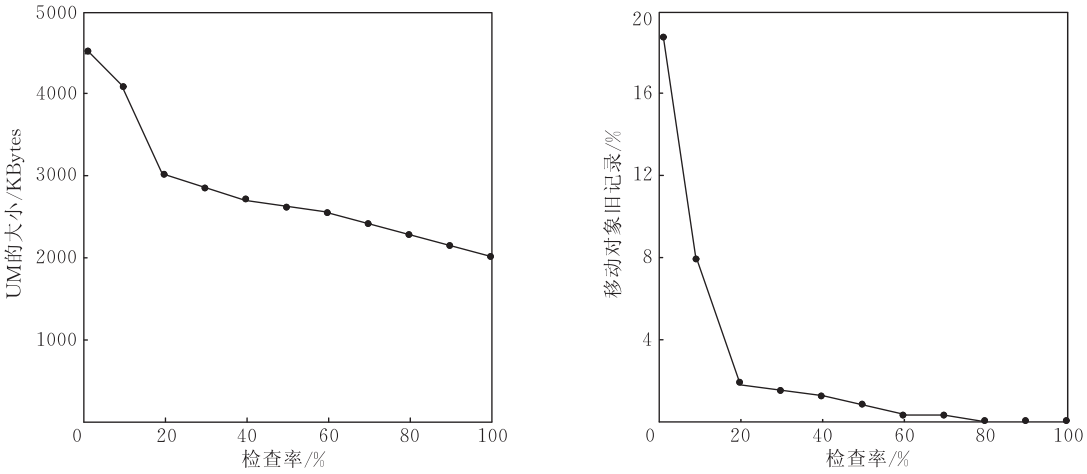
实验数据集采用文献[15]中基于道路网络的移动对象产生器随机生成. 在 10000 km×10000 km 空间区域内模拟 1 M～10 M 移动对象的运动情况，其中默认值为 1 M. 每个移动对象的不确定区域是半径为 50 km 的圆圈，概率密度分布为平均分布或高斯分布. 在初始时刻 t 每个移动对象选择一个目的地开始运动，到达目标之后发出位置更新请求并重新随机选择一个新目的地运动. 其中，移动对象运动目的地设为 5000 个 MBR，移动速度在[20,50]之间均匀分布. 值得注意，移动对象可以从系统中消失，也可以加入新的移动对象，但移动对象总数小于系统设定的值. 实验中的主要参数如表 2 所示，其中粗体字为默认值.

表 2 常用实验参数及取值

参数	值
不确定对象的数量	1 M～10 M，**1 M**
数据空间	[0,10000]
不确定半径	50
检查率	[0,1]，**20%**
更新次数	[0,1000 k]，**200 k**
移动距离	[0,250]，**50**

对移动对象数据集，我们基于 Gist 分别构建 TPU 树和 TPU^{2}M 树索引，节点大小均设置为 1 KB，中间节点扇出两者均为 27，叶子节点的扇出随

U-catalog 的大小而变化，并设定 *U-catalog*=10，页面缓存大小均为 100 KB. 基于 B^x 树构建 AB^x 树索引，节点大小设置为 1 KB. 索引树根节点常驻缓存中，并使用最近最少使用(LUR)缓存替代策略. 实验硬件环境是：Intel(R) Pentium(R) Dual-Core 2.50 GHz 的 CPU，内存为 2048 MB RAM；软件环境是：Windows XP 操作系统和 Visual C++ 6.0 集成开发环境.



(a) 对UM大小的影响　　(b) 对失效率的影响

图 3　检查率的影响

检查率 *ir* 的大小直接影响到辅助内存结构 UM 的大小，TPU^2M 索引树的查询和更新时间以及失效率 *or*. 图 3 比较了不同 *ir* 大小情况下内存结构 UM 的大小和失效率. 正如第 4 节中分析的结果：UM 的大小以及失效率均与检查率成反比增长. 从实验可以看出，随着检查率的逐渐增加，内存结构 UM 的大小与索引树的旧记录比率呈下降趋势. 这是由于较大的检查率会引起空间清理器更加频繁地删除 UM 中多余的记录项和清理叶子节点中过时的移动对象旧记录，很大程度减少内存结构的大小与 TPU^2M 树的旧记录占用率，从而为移动对象新记录开辟更多新的插入空间. 不难发现，在检查率增至 20%时，移动对象旧记录比率急剧下降至 2%，此为 UM 尺寸的大小和 TPU^2M 树产生较为理想的更新与查询效率提供了最佳时机，因此在下面的实验当中，我们均设定检查率 *ir*=20%.

5.1　更新代价

为全面验证 TPU^2M 树在更新策略上的最优性，我们分别比较了不同更新次数、更新幅度以及不同的移动对象个数对索引树更形性能的影响. 其中，图 4(a)比较了不同索引方法在每隔 100 k 次更新后的更新性能. 可以看出，TPU^2M 树的更新性能在频繁更新中相对于传统索引 TPU 树和 AB^x 树具有绝对的优势. 图 4(b)比较了在不同移动更新幅度情况下 TPU^2M 树与 TPU 树和 AB^x 树动态更新所需要的平均磁盘 I/O 次数. 特别地，不确定移动对象位置更新保持较高的频率，平均更新次数在 100 k 左右. 可以看出，TPU^2M 树的更新所需平均磁盘 I/O 维持在 50 次左右不变，具有很好的动态更新性能. 而 TPU 树和 AB^x 树则始终保持着相对较高的磁盘 I/O 次数. 这是由于 TPU^2M 树的更新过程利用 UM 内存结构，在索引树中无需查找旧记录而直接插入不确定移动对象的新记录. 而且，TPU^2M 树在较小位置更新幅度的情况下，可以采用更新代价较低的自底向上方法，因此更新代价可以得到进一步减少. 而 TPU 树和 AB^x 树由于传统的自顶向下更新过程，使得更新代价相对最高. 图 4(c)比较了在逐渐增加不确定移动对象数量的情况下 3 种索引的更新所需平均磁盘 I/O 次数. 不难看出，TPU 树和 AB^x 树的更新代价随着不确定移动对象个数的增加趋于增加的趋势，这是由于所查找定位节点个数的增加使得更新代价逐渐增加，而 TPU^2M 树的更新代价基本不受移动对象个数的影响. 这是由于 TPU^2M 树的更新代价由插入新记录和清除旧记录两部分组成，而这两阶段的代价正如第 4 节分析显示并不受移动对象个数的影响. 因此 TPU^2M 索引树具有良好的可扩展性. 图 4(d)给出了在不同移动对象数量情况下 TPU^2M 树需要辅助结构内存空间的大小. 不难发现，内存结构 UM 大小与不确定移动对象个数成线性增长，所以内存结构 UM 的大小相对于索引树 TPU^2M 的大小同样是可扩展的.

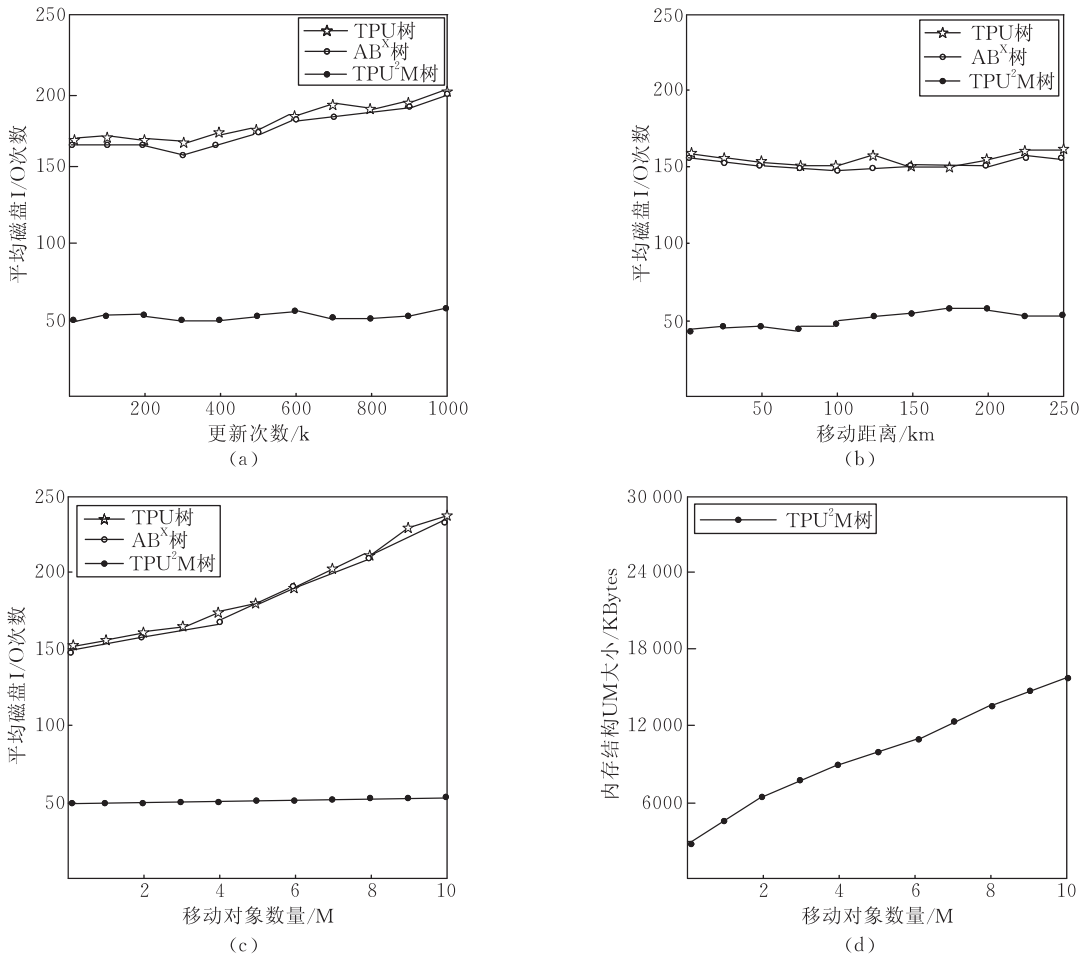


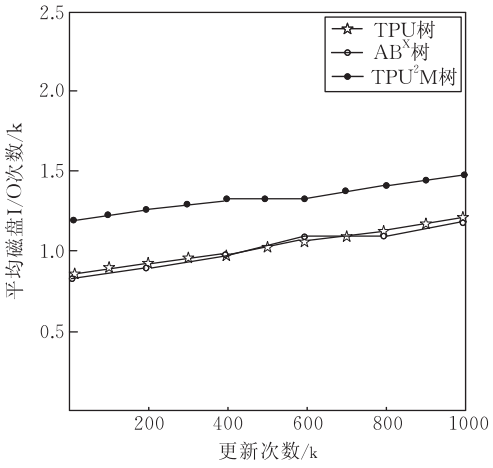
图 4 更新性能比较

5.2 查询代价

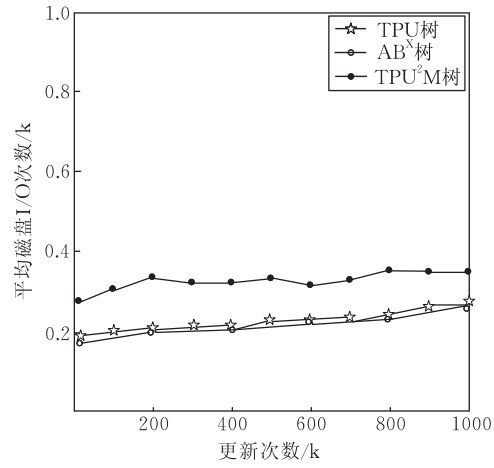
索引树中旧记录的存在势必影响 TPU^{2}M 树的概率查询效率. 图 5 显示了 TPU^{2}M 树与 TPU 树和 AB^{x}树 3 种索引结构在回答 200 个窗口查询所需要的平均磁盘 I/O 次数. 查询窗口范围大小记为 $qRlen$,速度矢量记为 $qVlen$,窗口查询时间记为 $qTlen$,概率阈值记为 Pc(Pc 的默认值设为 0.6). 其中,图 5(a)和(b)比较了 3 种不同索引方法在每隔 100 k 次更新后的查询性能,可以看出,TPU^{2}M 树在频繁更新下的查询所需平均磁盘 I/O 次数只是略高于 TPU 树和 AB^{x}树. 图 5(c)和(d)比较了 3 种不同索引方法在更新幅度不断增加情况下的查询性能. 可以看出,即使发生了较大位置幅度的频繁更新,TPU^{2}M 树一直保持着较好的查询性能,平均磁盘 I/O 次数只是略高于 TPU 树和 AB^{x}树. 这是由于较低的旧记录比率 2%保证了 TPU^{2}M 树中包含有限的不确定移动对象旧记录. 值得一提的是,TPU^{2}M 树的查询性能在不确定移动对象的更新幅度小于 100 km 之前,基本上与 TPU 树和 AB^{x}树具有相同的概率查询效率,并只在位置更新幅度达到 100 km 时略微高于 TPU 树和 AB^{x}树. 这是由于 TPU^{2}M 树在位置更新幅度达到 100 km 之前,大部分采用自底向上的更新方法动态扩展叶子节点及父节点大小以适应新的位置信息记录,之后则删除索引树中多余的旧记录,从而保证了内存结构 UM 具有较小的容量,进而使得内存结构和索引树本身更加紧凑,具有与 TPU 树和 AB^{x}树相当的概率窗口查询处理效率.

5.3 最近邻查询

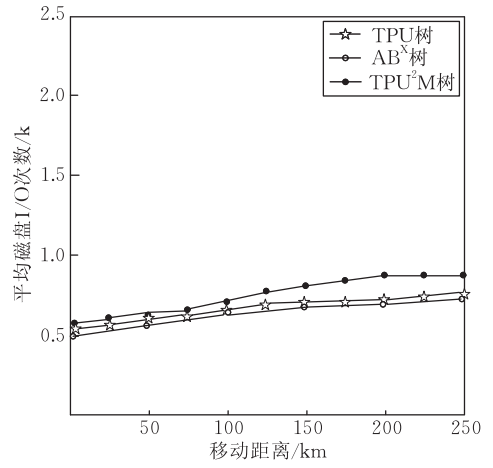
图 6 显示了 TPU^{2}M 树与 TPU 树和 AB^{x}树 3 种索引结构在回答 50 个最近邻查询所需要的平均磁盘 I/O 次数. 由于不确定移动对象是随机地分布在一个区域中的,因此最近邻查询返回的是当前时间点上有可能成为查询点最近邻居的移动对象. 指定的概率阈值记为 Pc(Pc 的默认值设为 0.5),那么查询结果是返回所有最近邻概率大于等于 0.5 的移动对象. 其中,图 6(a)比较了 3 种不同索引方法在每隔 100 k 次更新后的最近邻查询性能,跟窗口查询具有相似的结果,TPU^{2}M 树在频繁更新下的查询所需平均磁盘 I/O 次数略高于 TPU 树和 AB^x



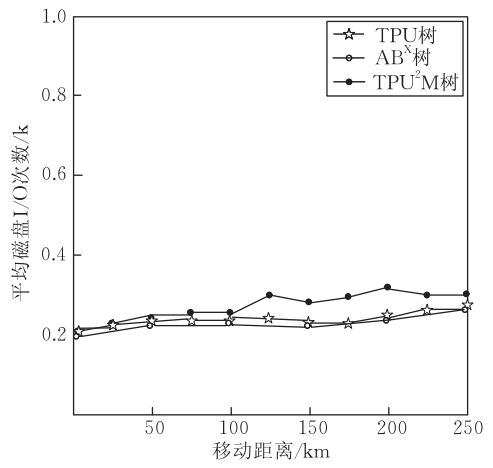
(a)($qRlen$=100,$qVlen$=5,$qTlen$=50)



(b)($qRlen$=400,$qVlen$=5,$qTlen$=1)

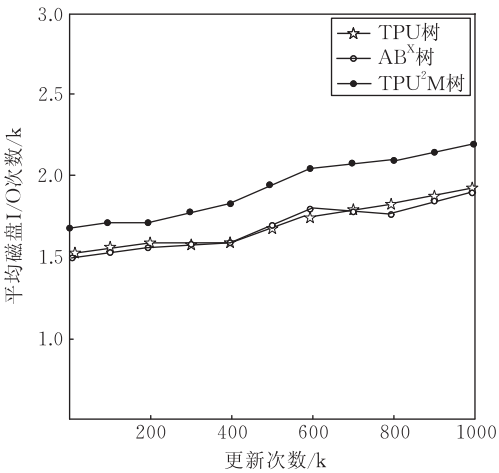


(c)($qRlen$=100,$qVlen$=5,$qTlen$=50)

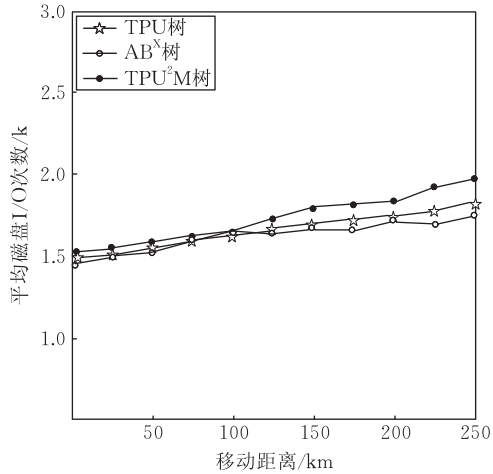


(d)($qRlen$=400,$qVlen$=5,$qTlen$=1)

图 5　查询性能比较



(a)



(b)

图 6　最近邻查询性能比较

树.不难发现,由于最近邻查询的计算复杂度高于窗口查询,因此 3 种索引结构在处理最近邻查询时的平均磁盘 I/O 次数也较高.图 6(b)比较了 3 种不同索引方法在更新幅度不断增加情况下的最近邻查询性能.跟窗口查询相似,即使移动对象发生了较大位置幅度的频繁更新,TPU²M 树一直保持着较好的查询性能,平均磁盘 I/O 次数只是略高于 TPU 树和 ABˣ树.不难发现,TPU²M 树的最近邻查询性能在更新幅度不超过 100 km 之前,基本上与 TPU 树和 ABˣ树具有相同的处理效率.这是由于 TPU²M

树在位置更新幅度未达到 100 km 时，基本上采用自底向上的更新方法动态扩展叶子节点及其父节点以适应新的位置信息记录，随后删除多余的旧记录，从而保证了内存结构 UM 较小的容量，使得内存结构和索引树更加紧凑，从而具有与 TPU 树和 AB^x 树相当的概率最近邻查询处理能力.

6 结 论

本文在不确定移动对象当前及未来位置索引技术 TPU 索引树基础之上，针对不确定移动对象频繁位置更新带来的系统效率低下问题，提出了一种支持频繁大幅度位置更新的移动对象索引策略——TPU^2M 树. 与传统的索引方法相比，其增加了一个记录移动对象状态特征的更新备忘录 UM 内存结构，并提出了一种改进的基于备忘录(MMBU/I)更新/插入算法. 实验仿真表明，采用 MMBU/I 更新算法的 TPU^2M 索引树频繁更新效率大大高于 TPU 树和 AB^x 树，并且查询效率与同类索引大致相当. 在基于位置的服务、移动计算等具有频繁更新要求的应用领域，TPU^2M 索引树具有较高的实用价值和广泛的应用前景. 下一步的研究方向是在不确定移动对象的环境中，将问题扩展到支持局部位置更新及高级概率查询如 Top-K 查询、Skyline 查询等.

参 考 文 献

[1] Tao Y F, Cheng R, Xiao X K, Wang K N, Kao B, Prabhakar S. Indexing multi-dimensional uncertain data with arbitrary probability density Functions//Proceedings of the 31th International Conference on Very Large Databases. Trondheim, Norway, 2005: 922-933

[2] Zhou Ao-Ying, Jin Che-Qing, Wang Guo-Ren, Li Jian-Zhong. A survey on the management of uncertain data. Chinese Journal of Computers, 2009, 32(1): 1-16(in Chinese)
(周傲英，金澈清，王国仁，李建中. 不确定性数据管理技术研究综述. 计算机学报，2009，32(1)：1-16)

[3] Ding X F, Lu Y S. Indexing the imprecise positions of moving objects//Proceedings of the ACM SIGMOD PhD Workshop on Innovative Database Research. Beijing, China, 2007: 45-52

[4] Ding Xiao-Feng, Lu Yan-Sheng et al. An U-tree based indexing method for uncertain moving objects. Journal of Software, 2008, 19(10): 2676-2705(in Chinese)
(丁晓锋，卢炎生等. 基于 U-tree 的不确定移动对象索引策略. 软件学报，2008，19(10)：2676-2705)

[5] Zhang M H, Chen S, Jensen C S, Ooi B C, Zhang Z J. Effectively indexing uncertain moving objects for predictive queries//Proceedings of the VLDB 2009. France, 2009: 1198-1209

[6] Zhang Ming-Bo, Lu Feng et al. The evolvement and process of R-tree family. Chinese Journal of Computers, 2005, 28(3): 289-300(in Chinese)
(张明波，陆锋等. R 树家族的演变和发展. 计算机学报，2005，28(3)：289-300)

[7] Saltenis S, Jensen C S, Leutenegger S et al. Indexing the positions of continuously moving objects//Proceedings of the ACM SIGMOD. New York, 2000: 331-342

[8] Tao Y F, Papadias D, Sun J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries//Proceedings of the VLDB 2003. San Francisco, 2003: 790-801

[9] Saltenis S, Jensen C S. Indexing of moving objects for location-based services//Proceedings of the ICDE. San Jose, California, USA, 2002: 463-472

[10] Procopiuc C M, Agarwal P K, Har-Peled S. STAR-tree: An efficient self-adjusting index for moving objects//Proceedings of the 4th International Workshop on Algorithm Engineering and Experiments. San Francisco, CA, USA, 2002: 178-193

[11] Lin D, Jensen C S, Ooi B C, Saltenis S. Efficient indexing of the historical, present, and future positions of moving objects//Proceedings of the MDM. Ayia, Napa, Cyprus, 2005: 59-66

[12] Pelanis M, Saltenis S, Jensen C S. Indexing the past, present, and anticipated future positions of moving objects. ACM Transactions on Database Systems, 2006, 31(1): 255-298

[13] Lee M L, Hsu W, Jensen C S et al. Supporting frequent updates in R-trees: A bottom-up approach//Proceedings of the VLDB 2003. Berlin, 2003: 608-619

[14] Xiong X P, Aref W G. R-tree with updates memos//Proceedings of the ICDE. Washington DC, USA, 2006: 22

[15] Brinkhoff T. A framework for generating network based moving objects. Geoinformatica, 2002, 2(6): 153-180

DING Xiao-Feng, born in 1982, Ph. D., assistant professor. His current research interests include uncertain data management, query processing and optimization.

JIN Hai, born in 1966, Ph. D., professor, Ph. D. supervisor. His research interests include large scale data management, grid computing and computer virtualization.

ZHAO Na, born in 1982, M. S., assistant engineer. Her research interests include network architecture and information communication.

Background

This research was supported by the National Natural Science Foundation of China (Grant No. 61100060), the Key Project in the National Science & Technology Pillar Program of China (Grant No. 2008BAH29B00), the China Postdoctoral Science Foundation funded project (Grant No. 20100471179), the Natural Science Foundation of Hubei Province (2011CDB037) and the Central Colleges of Basic Scientific Research and Operational Costs (Grant No. 2011QN054). In a kind of location-dependent applications, the problem of frequently updating the position information for kinds of uncertain moving objects is becoming increasing important. TPU-tree is currently one of the most popular indexing methods for such kinds of uncertain moving objects, but the adopted traditional top-down update method has made their frequent updates performance very low. In this paper, we proposed the TPU^{2}M-tree for uncertain moving objects with frequent updates, which is based on TPU-tree, supplemented by a memory-based update-memo structure recording the state of uncertain moving objects. Also a modified memo-based update/insert algorithm is developed for TPU^{2}M-tree. Extensive experiments have been conducted to verify the effectiveness of our proposed update/insert algorithm for frequently updated uncertain moving objects.