时态数据的变粒度分段存储策略及其效益分析*

唐常杰　于中华　游志胜　张天庆　相利民

(四川大学计算机系　成都　610064)

E-mail: chjtang@scu.edu.cn

摘要　根据时态数据库用户对数据厚今薄古的需求特点,该文提出变粒度分段存储技术,将一个对象的历史分为 3 个时期,分介质、变粒度存储.文章讨论了分段存储的特殊数据结构、时代转移算法和压缩采样算法.基于微机实用参数的定量分析表明,这一技术将时态存储密度和时态查询速度提高了一个数量级.

关 键 词　时态数据库,分段存储,时代转移算法,压缩采样,时态存储密度.

中图法分类号　TP311

20 年来,时态数据库管理系统(TDBMS)的研究者一直在为改善 TDBMS 的性能作不懈的努力.时态数据库(temporal database,简称 TDB)旨在保存被处理对象和数据库本身的历史.历史只能追加,不能删除的特点使得 TDB 中数据量极大,时空效率较低,这一特殊困难长期制约了 TDBMS 的发展.

在实用环境中,大多数 TDB 的应用都具有厚今薄古的特点,即对近期数据的查询多,要求详细;而对历史久远的数据查询少,要求粗略.根据这一事实,我们在 HBase 中引入了分段变粒度存储方案,将一个对象的历史分为远、中、近 3 个时间段,设置两个过渡期,通过时代转移算法和压缩采样算法进行数据提炼,并进行分介质、变粒度的存储,体现了与数据仓库的无缝连接.在时态存储密度和时态查询速度上获得了一个数量级时空效益. 本文讨论分段存储方案的时态结构、相关算法及其时空效益.

1　传统的时态数据库存储方案的局限

1.1　时态数据库现有存储方案

国外 TDB 研究界把时态数据库性能研究的重点放到时态索引技术上,例如,Time Split B+ Tree,Append Only Tree,Checkpoint Index,Monotonic B+ Tree[1],近年来,国内在时态数据库预测模型、支持决策[2]、前兆依赖[3]和变粒度时间轴[4]等方面取得了可喜成果.国内外已经研究的时态数据存储方案主要有以下两类.

方案 A. 对象历史不分段,时间量子为恒量,大多数 TDB 模型采用这一方案[1].

方案 B. 时间量子为恒量,"TDB=快照+历史",快照保存在高速介质(例如硬盘)中,记录的有效期结束后,记录变为历史,被转存到低速介质(例如磁带),历史不再分段[5].

方案 B 的优点是,当用户不涉及历史数据、把 TDB 降级为传统 RDB 使用时,有较高效率[5].此外,由于历史不分段,易于实现.其缺点在于,在高速介质中的数据不是历史,而只是历史的最后一瞬.在历史性应用中,时间效率至多等同于方案 A,存储效率略优.

在上述两个方案中,对象的历史不分段,都采用单一的时间量子,以下简称为单史制.

---

* 本文研究得到国家自然科学基金和国家教育部留学回国人员启动基金资助.作者唐常杰,1946 年生,教授,主要研究领域为数据库.于中华,1966 年生,博士,副教授,主要研究领域为机器翻译,数据库.游志胜,1945 年生,教授,博士生导师,主要研究领域为图像数据库.张天庆,1971 年生,讲师,主要研究领域为数据库.相利民,1961 年生,博士,副教授,主要研究领域为数据库.

本文通讯联系人:唐常杰,成都 610064,四川大学计算机系

本文 1998-07-14 收到原稿,1998-10-20 收到修改稿

1.2 符号和概念

时间量子(chronon)是时态系统选定支持的最小的时间间隔[1].系统时间起点记为 0,与起点相距 K 个时间量子的时刻记为 K,系统时间论域记为 $Sys\_T=\{0,1,2,\ldots,K,\ldots,Now,\ldots,MaxSysTime\}$.$Sys\_T$ 的子集 $\{t,t+1,t+2,\ldots,t+n=t'\}$称为区间,记为$[t,t']$,并称时间量子数 n 为该区间的长度.关于时态数据库的其他基本概念参见文献[1].

为了定量计算时态数据库的时空效率,我们引入下列概念:

(1) 记录所表达的历史区间平均长度称为平均历史跨度(span),记为 S.例如,1 000 个记录表达了 2 000 年历史,则平均历史跨度 S=2(年).它反映了 TDB 中记录对历史的概括能力.

(2) TDB 采用定长记录,记录长度记为 L(K 字节).

(3) 历史数据按时段分存于多种存储介质,介质 J 的平均存取速度记为 V_J(记录/s).

(4) 每 K 字节信息的历史跨度称为时态存储密度,记为 $Temp\_D$(区间长/K 字节)

(5) 系统每秒所能查询出的信息的历史跨度称为时态查询密度,记为 $Temp\_V$(区间长/s).

1.3 单史制中存取100个记录所需的时空资源

给定一组中等代价的系统参数:时间量子 C=1(s),记录平均时间跨度 S=1(h),每记录长 L=1(K 字节),介质平均存取速度 V=500(K 字节/s).传统单史制中存取 100 个时态记录所需的时空资源如下:

存储空间 $M=100L=100$(K 字节)　　存取时间　　$t=100/500=0.2$(s)

时间跨度 $Span=100S=100$(h)　　时态存储密度　　$Temp\_D=Span/M=1$(h/K 字节)

时态查询密度 $Temp\_V=S/t=500$(h/s),即每秒能查询出 500 小时的历史.

实践表明,上述时态存储密度和时态查询速度不能满足实用的需要.其原因是单史制忽略了用户时态数据需求近多远少的特点,在历史久远的数据上花费了过多的时间和空间.

2 变粒度,分段,分介质存储结构

为了提高时态存储和查询性能,在 Hbase 模型[6,7]中,我们把被处理对象的历史按事务时间划分为 3 个时代和两个过渡区间,并称这一存储制式为三史制,如图 1 所示.其要点如下.

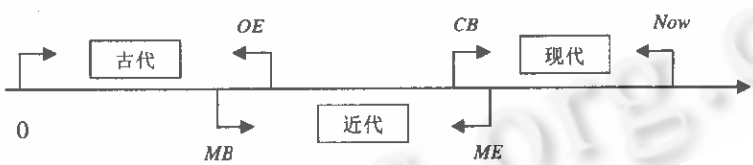

图 1　3 个时期和两个过渡区间

(1) 3 个时代.引入 6 个时刻变量 OB(或 $OldAgeBegin$),OE(或 $OldAgeEnd$),MB(或 $MidAgeBegin$),ME(或 $MidAgeEnd$),CB(或 $CurrAgeBegin$),Now(或 $CurrAgeEnd$)和时态约束条件 $0=OB<MB\leq OE<CB\leq ME<Now$.区间$[OB,OE]$,$[MB,ME]$,$[CB,Now]$分别称为古代、近代和现代.

(2) 两个过渡区.区间$[MB,OE]$和$[CB,ME]$分别称为古代-近代过渡期和近代-现代过渡期,过渡期中的历史数据在两个相关历史时代中各存一份.由时代转移模块自动而平滑地转移过渡期中超期的历史数据.

(3) 变粒度、分介质存储.用户可分古代、近代和现代设置不同的时间量子粒度.现代、近代和古代的数据分别保存在内存、硬盘和 CDROM(或磁带)中.在内存中为近代和古代设置高速缓存 M_Cache,O_Cache,二者大小比例和使用频度相同,默认为 10∶1.

(4) 使用频率. 系统能实测古代史、近代史、现代史的使用频率.初始值为古代∶近代∶现代=1∶10∶100.

上述要素全部封装在对象 Ages 中,作为 Ages 的数据成员.

用户设置 $MidAgeBegin$(或 MB),$OldAgeEnd$(或 OE)及古代-近代过渡期长度这 3 个变量中的任意两个,可以划分古代和近代及其过渡期.设置 $CurrAgeBegin$(或 CB),$MidAgeEnd$(或 ME)以及近代-现代过渡期 $MidCurrLen$ 三者中的任意二者可以划分近代和现代以及近代-现代过渡期.

三史制与"TDB=快照+历史"方式[1,5]有着本质的不同.HBase 模型的当前期不是快照,而是[*CB*,*Now*]区间的历史,这保证了能高速分析当前期中对象间的联系,且三史制特有的过渡区及稍后要介绍的时期转换算法保证了多时期、多介质的平滑、高效的过渡.

例 1:下列语句中前两句用相对时刻动态地划分时期,而第 3 句采用绝对时刻划分时期.

Set Ages.MidAgeBegin=Ago (5 years)

Set Ages.Old_Mid_Len=60 Days

Set Ages.CurrAgeBegin="12-31-1996"

第 1 句以执行时间作为基准,以其 5 年之前的时刻作为近代的开始时刻,第 2 句设置古代-近代过渡期为区间[*MB*,*MB*+60 Days].其中 Ago(5 years)是一个时态宏,TDBMS 将根据系统内部日历及闰年法则将其转化为执行时刻的 5 年前那一时刻.因此用 Ago(...)宏来划分的时代间隔是动态的.下面的时代转移算法将考虑由动态间隔而引起的困难.

例 1 中的 Years,Months,Days 为 TDB 中预定义的区间长度宏,而"12-31-1996"为系统时刻值,TDBMS 自动地把长度宏扩展为时间量子.例如,Days=24×3600(s).而将系统时刻自动转为从数字 0 时刻开始的时间量子数.

3　时代转移算法

时代转移算法是三史制能安全平稳地过渡数据的关键.如例 1 所示,时代划分点和过渡期可能随时间变化.新、旧两过渡期的时态关系可能相邻、交叉或包含.下面以近代-现代过渡期为例,分析过渡期变化和数据转移方向的关系.

考查近代-现代过渡期,分别记旧、新过渡期为[*CB*,*ME*]和[*CB*′,*ME*′].4 个端点的全排列有 4! =24 种,其中满足 *CB*<*ME*,及 *CB*′<*ME*′的有下列 6 种:

(1) *CB*<*ME*<*CB*′<*ME*′　(2) *CB*<*CB*′<*ME*<*ME*′　(3) *CB*<*CB*′<*ME*′<*ME*

(4) *CB*′<*ME*′<*CB*<*ME*　(5) *CB*′<*ME*<*CB*<*ME*′　(6) *CB*′<*CB*<*ME*<*ME*′

逐一考查上述情形,易知前 3 种的共性为 *CB*<*CB*′,即现代开始时间的新值 *CB*′在时间轴上后移.这是实用中最常见的情形,需要将当代史中[*CB*,*CB*′]区间的历史转移到近代史中.后 3 种的共性为 *CB*′<*CB*,这种情形一般在用户强制改变过渡期时才出现,需要将近代史中[*CB*′,*CB*]区间的历史复制到内存的当代史中.

三史制通过系统预设的触发器,定期(期限一般小于过渡期长度的一半)启动时代转移模块,将过渡期中的超期部分自动地移动或复制到正确的时期中.下面用以 C++的风格书写的算法来描述现代-近代过渡期数据过渡处理过程.

算法 1. Curr_MidAge Transfer Algorithm (现代-近代转移算法)

输入: 现代史、近代史.

输出: 现代史、近代史.

附加效果: 将过渡期中的超期部分转移到相应时期.

```
HBase::Curr_MidAge_Transfer(CurrAge,MidAge)          //作为对象 HBase 的成员函数
{ CB_Old=CurrAge.GetOld_CB();                        //获取旧的"现代"始点
  CB_New=CurrAge.GetNew_CB();                        //获取新的现代始点
  ME_Old=MidAge.GetOld_ME();                         //获取旧的"近代"终点
  ME_New=MidAge.GetNew_ME();                         //获取新的近代终点
  if (Old_CB<New_CB)                                 //现代开始时间自然后移
    { for (each Record in([Old_CB,New_CB]∩CurrAge))  //过渡期间移向近代
        { Record.Verify();                           //校正历史数据
          If ( Record.Is_not_in(MidAge))
            { MidAge.Append(Record);                 //追加该记录到近代史
              CurrAge.DeleteRecord(Record));         //从现代史删除该记录
```

```
            };
        }                                          //End of for
        MidAge.CompressNewData();                  //近代史压缩新数据,见下面的解释(1)
    }                                              // end of if
  else if (New_CB<Old_CB)                          //见解释(2)
    { while (Record.Is_in ([New_CB,Old_CB]∩MidAge))
        {  if (Record.Is_not_in(CurrAge))
           CurrAge.ExpandChronon_Append(Record);   //复制到现代史, 见解释(3)
        };                                         // end of while
    }                                              // End of else
};
```

解释

(1) *Compress_NewData* 是 *MidAge* 对象的成员函数,其作用是压缩采样,细节将在下一节介绍,暂时可将其视为什么都不做,但返回 True 的平凡函数.

(2) (*New_CB<Old_CB*)一般只在强制扩大过渡期时出现,此时不需从([*New_CB*,*Old_CB*]∩MidAge)删除 Record,合理冗余便于存取.

(3) 扩展时间量子以适应现代区间的时间粒度.例如,*MidAge* 和 *CurrAge* 中的时间量子分别为 month 和 Day,*MidAge* 中的元组(*Temperature*=8,[96.1,96.2])表示 96 年 1 月平均温度为 8 度,扩展时间量子后为(*Temperature*=8,[96.1.1,96.2.1]),表示 96 年 1 月每天温度为 8 度.

命题 1. 设过渡期中的时间粒子数为 N,则时代转移算法的计算量为 $O(N)$.

证明:在时代转移算法的 for 循环中,循环条件 Record in([*Old_CB*,*New_CB*])取真值的次数不超过 N,循环体中的插删工作量可以用常数因子 $C1$ 估计,同理,在 While 循环中,循环条件取真值的次数不超过 N,循环体中 *CurrAge.ExpandChronon_Append*(*Record*)函数在扩展时态粒子时,如上述的解释(3)中所示.只是把生命周期细化,计算机量用常数 $C2$,估计,总的计算机量不超过 $N(C1+C2)$,所以计算量为 $O(N)$. □

4 变粒度时间轴与时态数据的浓缩

大多数应用对历史信息的要求具有远略近详的特点.为此,我们提出了变粒度时间轴.最初的目的是扩大 TDB 的覆盖区间[6],在三史制中发展为变粒度存储的基础.Hbase 中,现代史、近代史和古代史的时间粒子分别为 Second,Day 和 Year.用户可通过 TDML 语句修改时间量子默认值.例如,通过语句 CurrAge.Set(Chronon, "1 Hour")可以把现代时间量子设置为 1 小时.变粒度时间轴导致了变时间粒度存储,引出了时代转移过程中的压缩或扩展,是三史制中提高效率的关键.

时态数据的浓缩(又称为压缩采样)在时代转移过程中进行.当过渡期中数据从现代向近代转移,或从近代向古代转移时,TempoTransfer 算法中的 CompressNewData 函数自动调用用户预选的方法采样、压缩.压缩采样包括相对位置抽取法、均值法和统计法.

(1) 相对位置抽取法.由用户规定抽取点的相对位置 *Percent*,0≤*Percent*≤100,当 *Percent*=50 即为典型的中点法.过渡期中数据从现代向近代转移时,触发器调用算法 1,并通过延迟约束机制,将其中的 CompressNewData 连接(bounding)到下面的同名函数(即算法 2),注意算法中的"移动"函数 Move_One_New_Record_To_WorkSpace 把对象复制到目的地,同时要删去源数据.

算法 2. 相对位置定点抽取算法

```
MidAge::CompressNewData()                  //近代史对象 MidAge 的成员函数
 { SmallChronon=CurAgeChrono;              //现代史时间量子;
   BigChronon=MidAge.Chrono;               //近代史时间量子;
   ArraySize=BigChronon/SmallChronon;      //时间量子压缩倍数
   WorkSpace=new RecordType [ArraySize];   //按最大需要分配采样数组空间
```

```
TotalLeftRec=GetTotalNewRecNum();                    //待处理的记录总数
Finish=False;Count=0;                                //初始化循环变量
While (not Finish)
  { Move_One_New_Record_To_WorkSpace ();
    RecordLifeSpan=GetNewRecordLifeSpan();           //获取新记录生命周期
    Count=Count+RecordLifeSpan/SmallChronon;         //编程要点 1
    TotalLeftRec--;                                  //待处理的记录总数减 1
    If (Count=ArraySize)                             //移满一组、开始抽样
      { Sample=Take_Sample(WorsSpace,Percent);       //在指定点抽样
        MakeCompresedFlag(Sample);                   //作已压缩标记
        MidAge.Append(Sample);                       //追加到近代史
        Count=0;                                     //准备循环,压缩另一组
        Clear(WorkSpace);                            //准备循环
      };
    Finish=(TotalLeftRec==0);
    //编程要点二,如剩下的不足一组,留待下一次压缩
  }
}
```

(2) 均值法.下面用例子加以说明.

例 2:设现代史时间量子为日,近代史时间量子为周,过渡期为 10 日,则时代转移算法先把过渡期中数据移到近代史中,然后调用 *CompressNewData*.处理过程是,对新移入的数据和上次未压缩取样的记录,每 7 个一组移到工作空间作 7 日均值,然后移回近代史,最后不足 7 个的留待下一次转移时处理.只需在算法 2 中修改 *Take_Sample*(...)函数为 7 日均值函数即可.

(3) 统计方法.在 *Take_Samle* 函数中采用其他统计函数,如均值、极大值、极小值、回归值等等.在 HBase 方案中,由用户指定方法名称,直接用 OLE 调用商品化统计软件包中相应模块.

时代转移程序中包含了时态浓缩.三史制中的现代史是标准的时态数据库方式,而近代史、古代史中的数据已经是提取、加工、浓缩后的数据,在某种程度上具有简单数据仓库的特色.在此意义上,时代转移技术实现了数据库和数据仓库的无缝连接.

5　与传统存储制式的比较

为了比较时空效率,引入下列符号和约定.

设 TDB 中已按三史制划分出历史时期 T_i,i=1,2,3 分别表示现代、近代和古代.

(1) 历史区间 T_i=[B_i,E_i],数据应用概率为 P_i,其中 $0 \le P_i \le 1$,$P_1+P_2+P_3=1$.

(2) 历史区间 T_i 的时间量子为 C_i,在区间 T_i 中,每个记录的平均历史跨度记为 S_i.

(3) T_i 时代对应的存储介质上平均存取速度为每秒 V_i 个记录.

(4) 三史制在微机环境的特有参数(接近实用)如下:

数据使用频度 P_1=0.9,P_2=0.07,P_3=0.03;每记录长 L=1(K 字节);

时间量子 C_1=1 秒,C_2=1 日,C_3=1 月;

记录平均时间跨度 S_1=1 小时,S_2=1 日,S_3=1 月.

介质平均存取速度 V_1=10 000(内存中),V_2=500(硬盘中),V_3=50(活动硬盘).

下面计算三史制下存取 100 个记录的时空资源.按数据使用概率,100 个记录中分布在 T_i 时代中有 $100P_i$ 个记录,其历史跨度为 $100P_iS_i$.以下的$\sum$求和都是从 i=1 到 i=3.对于 100 个记录,有:

总跨度为 $S=100\sum P_iS_i$;

存取总时间为 $t=100\sum(P_i/V_i)$(秒);

总存储空间 $M=100L$(K 字节);

时态存储密度 $Temp\_D=S/M=(\sum P_iS_i)/L$;

时态查询密度 $Temp\_V=S/t=(\sum P_iS_i)/(\sum(P_i/V_i))$.

这些公式提供了调整三史制下各项性能的方法.修改 P_i, S_i, V_i 的大小,能得到不同的性能.

传统存储方案本质上是单史制[1,5],单史制可看成是三史制的特例.在三史制中,令 $T_2=\varnothing$,$T_3=\varnothing$,$P_1=1$,$P_2=0$,$P_3=0$(即只保留现代史阶段);令 $V_1=V_2$(即用硬盘存储),则三史制退化为单史制.将上述参数代入三史制公式,得到单史制下的结果与第 1 节中的结果一致.

表 1 存取 100 记录的比较

	存取总时间(s)	时间总跨度(h)	存储空间(K 字节)	时态存储密度(h/K 字节)	时态查询密度(h/s)
三史制	0.083	2 418	100	24.18	291 312
单史制	0.2	100	100	1	500

按 HBase 在微机环境中的参数和上述公式,在两种不同体制下存取 100 个记录的资源比较见表 1.它表明,三史制的时间跨度提高 20 倍以上,时态存储密度提高 24 倍,时态检索速度提高 58 倍.上述计算中忽略了 TDBMS 内部 cache 的正效应[7]和时态转移的代价.其原因是,一方面系统中不常作时代转移(例如,一日一次或一周一次);另一方面,由于与整个历史相比,过渡期不长,一般只占 5%,又根据命题 1,时代转移算法的计算量为 $O(N)$,因此时代转移算法耗时不多,可以忽略.

时空效益本质上来自两方面.其一,时代转移算法对数据进行了提炼压缩,存所应存,略所可略,使得时态信息密度加大.其二,当用户对历史性数据的需求是厚今薄古型时,大多数运算在高速介质中进行.

参考文献

1 Tansel A, Clifford J, Dadia S *et al*. Temporal Databases Theory, Design and Implementation. Redwood, CA: Benjamin/Cummings Publishing Company, 1997. 418~455

2 张师超,严小卫.历史数据库支持决策的研究.计算机研究与发展,1998,35(4):329~322
(Zhang Shi-can, Yan Xiao-wei. A study of historical database for decision making. Computer Research and Development, 1998,35(4):329~322)

3 Tang Chang-jie, Yin Bo-wen. The premonitory dependency of historical database and its mathematical properties. Science in China (series A),1989,32(6):758~768

4 唐常杰,吴子华等.时态数据库的变粒度时间轴.见:冯玉才编.全国第 12 届数据库会议论文集.武汉:华中理工大学出版社,1994. 419~423
(Tang Chang-jie, Wu Zi-hua *et al*. The time axis with variable granularity in temporal database. In: Feng Yu-cai ed. Proceedings of the 12th Conference of Database in China. Wuhan: Huazhong Polytechnic University Press, 1994. 419~423)

5 Ahn I, Snodgrass R. Partitioned storage for temporal databases. Information Science, 1989,(49):103~146

6 Tang Chang-jie, Xiong Min. The temporal mechanisms in Hbase. Journal of Computer Science and Technology, 1996,11(4):365~371

7 唐常杰,张天庆等.数据库管理系统内部结构及其 C 语言实现.成都:电子科技大学出版社,1995.37~63
(Tang Chang-jie, Zhang Tian-qin *et al*. The inner structure of database management system and its implementation in C Language. Chengdu: Press of University of Electronic Science and Technology of China, 1995. 37~63)

Segmented Storage with various Chronons in Temporal Databases and Efficiency Analysis

TANG Chang-jie YU Zhong-hua YOU Zhi-sheng ZHANG Tian-qing XIANG Li-min

(*Department of Computer Science Sichuan University Chengdu* 610064)

Abstract Based on the usage frequency of the data in temporal database, the storage method with three historical segments is proposed in this paper. The object history is divided into three segments, and stored in different media with various chronons. The special data structure for segmented storage is discussed, the age transfer algorithm and the compress sampling algorithm are proposed. The performance study shows that the time and space efficiency are promoted one magnitudes.

Key words Temporal database, segmented storage, age transfer algorithm, compressed sampling, temporal storage density.