# 移动分布式实时数据库系统的事务处理

雷向东，赵跃龙，陈松乔，袁晓莉

(中南大学 信息科学与工程学院，湖南 长沙，410083)

**摘 要**：提出 DMVOCC-MDA 协议处理移动分布式实时事务。协议采用多版本乐观方法，使用多版本动态调整串行次序技术。移动分布式事务局部有效性确认分为 2 个阶段：第 1 阶段在移动主机上进行，使用向后有效性确认机制，对服务器上个广播周期提交事务进行有效性确认；第 2 阶段在服务器上进行，通过局部有效性确认的移动事务，提交到服务器进行局部最终有效性确认。在全局有效性确认中对分布更新事务进行检查，以保证分布串行性。移动只读事务能无阻塞提交，降低移动只读事务的响应时间，消除移动只读事务和移动更新事务的冲突，避免不必要的移动事务重启动。在移动主机上及早地检测数据冲突，节省处理和通信资源。实验结果表明，DMVOCC-MDA 协议与其他协议相比，错过率和重启动率低，提交率和吞吐量高。

**关键词**：移动分布式实时数据库系统；多版本乐观并发控制；多版本动态调整；有效性确认；多版本数据广播
**中图分类号**：TP311 **文献标识码**：A **文章编号**：1672-7207(2007)06−1186−06

# Transaction processing in mobile distributed real-time database systems

LEI Xiang-dong, ZHAO Yue-long, CHEN Song-qiao, YUAN Xiao-li

(School of Information Science and Engineering, Central South University, Changsha 410083, China)

**Abstract:** DMVOCC-MDA (Distributed multiversion optimistic concurrency control with multiversion dynamic adjustment) protocol was presented for processing mobile distributed real-time transaction. In the protocol, multiversion optimistic concurrency control method was adopted. Technique of multiversion dynamic adjustment of serialization order was used. Local validation of mobile distributed transactions was performed in two phases. In the first phase, local validation was performed at mobile host (MH), and carried out against committed transactions at the server in the last broadcast cycle by using backward validation mechanism. In the second phase, local final validation was performed at the server. Transactions that survive in local validation must be submitted to the server for local final validation. In global validation distributed update transactions have to be checked to ensure the distributed serialization in all participants. Mobile read-only transactions can be committed with no-blocking. Respond time of mobile read-only transactions is greatly improved. Data conflicts can be eliminated between mobile read-only transactions and mobile update transactions. Unnecessary restarts of mobile transactions are avoided. The data conflict is early detected in MH. Processing and communication resources are saved. The simulation results show that the new protocol proposed has lower missing rate and restart rate, and higher commit rate and throughput compared with other protocols.

**Key words:** mobile distributed real-time database systems; muliversion optimistic concurrency control; multiversion dynamic adjustment; validation; multiversion data broadcast

**收稿日期**：2007−03−10；**修回日期**：2007−04−28
**基金项目**：国家教育部博士点基金资助项目(20030533011)
**作者简介**：雷向东(1964−)，男，湖南常宁人，博士，副教授，从事移动数据库技术、并行数据库技术研究
**通信作者**：雷向东，男，博士；电话：13786153879；E-mail: leixiangdong@mail.csu.edu.cn

在移动广播环境中，传输带宽是不对称的。从服务器到移动主机(Mobile Hosts，MHs)下行传输带宽远大于从 MHs 到服务器上行传输带宽。在这种不对称的传输带宽环境中，MHs 应尽量减少对上行传输信道的竞争，并发控制协议不应卷入 MHs 在事务执行期间进行数据冲突检测时与其他 MH 和服务器持续同步。在移动分布式实时数据库系统中，并发控制协议研究主要有 3 个方面：

**a.** 扩展锁制机制[1−2]。对每个数据项加锁的请求必须从 MH 传送到服务器，大量的加锁请求会堵塞从 MHs 到服务器上行传输信道，事务请求数据封锁可能等待很长时间，导致移动实时事务处理不可容忍地延时。此外，存在死锁问题，死锁检测开销很大。MHs 为检测数据冲突必须与服务器持续同步。

**b.** 采用乐观并发控制机制[3−7]。乐观并发控制机制具有非阻塞和无死锁的良好特性，对实时环境非常有吸引力。但乐观并发控制机制重启动事务数量多，而在移动实时数据库中重启动移动事务代价很高。移动事务直接提交到服务器进行有效性确认，在服务器决定所有事务是否提交。MHs 需等待很长时间才知道哪些移动事务必须重启动。在具有大量 MHs 的移动实时分布式数据库中，此策略也会导致移动事务处理不可容忍地延时，且浪费处理资源和通信带宽。

**c.** 使用混合机制[8]。移动只读事务按乐观并发控制机制处理，移动更新事务必须提交到服务器进行有效性确认。若发生冲突，则事务重启动，否则使用锁机制。MHs 为检测数据冲突同样必须与服务器持续同步，且重启动事务数量多。

在此，本文作者提出 DMVOCC-MDA 并发控制协议处理移动分布式实时事务。通过使用多版本机制，移动只读事务能无阻塞提交，通过乐观方法，降低移动更新事务之间的冲突。通过多版本动态调整事务串行次序，避免不必要的事务重启动。事务有效性确认分为 2 个阶段：第 1 阶段在 MH 进行，使用向后有效性确认机制，与在服务器上个广播周期提交的事务进行有效性确认。MHs 能及时确定哪些事务由于数据冲突必须重启动。及早地检测数据冲突，节省了处理和通信资源。第 2 阶段在服务器上进行。通过 MH 上局部有效性确认事务，提交到服务器进行局部最终有效性确认。移动分布事务还必须执行全局有效性确认，以保证分布串行性。

## 1　广播有效确认信息

广播介质可以看作延迟时间很长的广播磁盘来建模[9]。服务器维护每个数据项多个版本，数据项所有版本都将广播。数据项按访问频度进行划分，访问频度相近的数据项放在同一个广播磁盘上。磁盘分成更小相等的称为块的数据单元。磁盘块的个数与磁盘的速度成反比。数据项所有版本都相继广播。热数据项位于快速磁盘上，冷数据项则位于慢速磁盘上。广播数据时，从每一个磁盘上取一块广播, 磁盘上的块按顺序取。广播周期分成多个微周期, 在1个微周期中广播所有磁盘中的一块。从而，在1个广播周期中，热数据项被频繁地广播，冷数据项广播次数则相对较少。

服务器广播在广播周期开始时，广播在服务器上个广播周期提交事务的有效确认信息。事务$T_i$的有效确认信息为($T_i$, $T_S(T_i)$, $R_{Set}(T_i)$, $W_{Set}(T_i)$)，其中：$T_S(T_i)$为$T_i$ 时间戳，$R_{Set}(T_i)$为$T_i$读集，$W_{Set}(T_i)$ 为$T_i$写集。所有提交的事务有效性确认信息由$C_{Set}$，$A_{Set}$，$C_{ReadSet}$和$C_{WriteSet}$组成，其中$C_{Set}$为上个广播周期在服务器提交的事务集合，$A_{Set}$为上个广播周期在服务器夭折的动事务集合，$C_{ReadSet}$为所有提交的事务的$R_{Set}$组成，$C_{WriteSet}$为所有提交的事务$W_{Set}$组成。

## 2　多版本乐观并发控制机制

每个事务 $T_i$赋予 1 个时间戳 $T_S(T_i)$，对于每个数据项 $x$ 有一个版本序列<$x_1$，$x_2$，…，$x_m$>与之关联。每个版本$x_k$包含 3 个数据字段：**a.** $x_k$版本值；**b.** $W_{TS}(x_k)$；**c.** $R_{TS}(x_k)$。其中，$W_{TS}(x_k)$表示创建 $x_k$版本的事务时间戳，$R_{TS}(x_k)$表示所有成功读取 $x_k$版本的事务最大时间戳。更新事务执行分为 3 个阶段：读阶段，有效性确认阶段和写阶段。在读阶段，假设更新事务 $T_i$ 发出 read($x$)或 write($x$)操作。令 $x_k$表示 $x$ 的版本，其写时间戳小于 $T_S(T_i)$的最大写时间戳。若 $T_i$发出 read($x$)，则返回 $x_k$的值，并把 $x_k$放入 $R_{Set}(T_i)$中。其理由是 1 个事务读取在它之前的最近版本。若 $T_i$发出 write($x$)操作，且若 $T_S(T_i)$＜$R_{TS}(x_k)$，则 $T_i$重启动；否则，创建 $x$ 的一个新版本 $x_i$，并把 $x_i$放入 $W_{Set}(T_i)$中。同时，将创建的新版本 $x_i$存储在 $T_i$的私有工作空间中，在 $T_i$结束之前对其他事务是不可见的。其理由是：若 $T_i$试图写入其他事务读取的版本，则不允许该写操作成功。在有效性确认阶段，验证准备提交的事务写操作是否可以复

制到数据库中同时又不违反串行性。在写阶段，若 $T_i$ 通过最终有效性确认，则实际的更新就可写入数据库中。只读事务读最近提交的数据项版本，只经过读阶段和有效性确认阶段。

在移动计算环境中重启动一个移动事务开销较大。通过多版本动态调整事务串行次序，避免不必要的事务重启动。在多版本机制中，写−写操作对不再是冲突的，因为它们产生不同的版本。多版本动态调整串行次序在下列2种情况发生[10]：

**a.** 若进行有效性确认事务 $T_v$ 与事务 $T_i$ 有读−写冲突，即 $R_{Set}(T_v)\cap W_{Set}(T_i)\neq\varnothing$，则调整串行次序为 $T_v\rightarrow T_i$。$T_i$ 的写不应影响 $T_v$ 的读阶段。

**b.** 如果进行有效性确认事务 $T_v$ 与事务 $T_i$ 有写−读冲突，即 $W_{Set}(T_v)\cap R_{Set}(T_i)\neq\varnothing$，则调整串行次序为 $T_i\rightarrow T_v$。$T_v$ 的写不应该影响 $T_i$ 读阶段。

每个事务 $T_i$ 分配一个有效性确认间隔 $V(T_i)=[l, u]$ 用于调整事务串行次序，其中 $l$ 为上限，$u$ 为下限。如果事务 $T_i$ 被串行在 $T_j$ 之前，即 $T_i\rightarrow T_j$，则 $T_j$ 的有效性确认间隔$[l_j, u_j]$和 $T_j$ 的效性确认间隔$[l_j, u_j]$必须满足 $u_i<l_j$。每个事务 $T_i$ 在开始执行时赋予有效性确认间隔为[0，∞]。若事务 $T_i$ 的效性确认间隔为空，则 $T_i$ 不可能再串行调整，必须重启动。

## 3 移动事务有效性确认

### 3.1 在 MHs 上局部有效性确认

MHs通过接收上个广播周期在服务器提交的事务有效性确认信息，对MHs上的移动事务进行局部有效性确认。使用向后有效性确认策略，提交事务要在进行有效性确认的移动事务之前。因此，数据冲突检测是检查进行局部有效性确认的移动事务读集合与提交事务的写集合是否相交。在MHs上进行局部有效性确认，MHs能及时检测到数据冲突，从而立即确定哪些移动事务必须重启动。MHs上没有完整的和最新冲突事务的数据视图，例如，MHs不知道在当前广播周期开始后提交到服务器的某些冲突事务的提交信息。MHs可能自愿或不自愿与移动网络断开，造成MHs上的数据可能过时，错过接收服务器广播的有效性确认信息。因此，若移动事务通过局部有效性确认，则必须提交到服务器进行局部最终有效性确认。若移动只读事务所有读数据项通过局部向后有效性确认，则不需要执行最终向后有效性确认，即可提交，降低了移动只读事务的响应时间，减少了上行传输信道堵塞的几率。

### 3.2 在服务器上局部最终有效性确认

提交到服务器的移动更新事务必须执行局部最终向后有效性确认，因为移动事务在 MHs 上执行局部有效性确认后，服务器上可能有新的事务提交。

在服务器上进行有效性确认的事务有移动事务和服务器事务。向前有效确认策略提供灵活的数据冲突解决方法，可选择有效性确认事务或冲突事务重启动，甚至可以通过强迫有效性确认事务在有效性确认阶段等待以避免一些事务被取消。因此，在服务器使用向前有效性确认。有效性确认事务串行在所有并发运行且尚处于读阶段的事务之前。

### 3.3 全局有效性确认

全局有效性确认需要检查分布数据项，这是因为分布数据项可能在有效性确认事务最后操作和其他并发事务有效性确认阶段之间被改变[11]。为了保证分布串行性，对所有移动分布更新事务进行检查，确定所读写的分布数据项是否改变。若移动分布更新事务所读写的发布数据项发生了改变，则移动分布更新事务夭折。

## 4 新协议

### 4.1 移动实时事务提交处理

传统 2PC 提交协议不适应于移动计算环境[12]，必须对 2PC 提交协议进行修改。发起移动分布式实时事务 $T$ 的 MH 称为 H-MH(Home MH)。与 H-MH 相连的基站 BS(Base Station)上的事务 $T_{co}$ 为协调者，协调各子事务的执行。若 H-MH 不能处理 $T$ 所有子事务，则抽出能处理的子事务 $T_i$， 送其余的子事务给 $T_{co}$。当 MH 迁移到新的无线单元时，$T_{co}$ 也应迁移新的单元，并通知新的协调者有关提交 $T$ 的状态信息。由于 $T_{co}$ 的位置可能发生变化，每一个参与者都必须跟踪 $T_{co}$ 位置的变化。$T_{co}$ 分配 $T$-$T_i$ 给相关的数据库服务器。数据库服务器收到子事务 $T_j$ 后，分配 $T_j$ 给相应的 MH 或固定主机 FH(Fixed Fast)。$T_{co}$ 等待所有参与者的处理结果。在 $T_j$ 处理过程中， 参与者在 $T_j$ 截止时间前可以无条件地中止 $T_j$。在 $T_j$ 执行结束后，参与者向 $T_{co}$ 传送提交 $T_j$。如果参与者不能完成 $T_j$，参与者向 $T_{co}$ 传送夭折 $T_j$。由于事务的提交需要全体一致提交，只要有参与者回答夭折其子事务 $T_i$，$T$ 便回滚。若到了 $T$ 截止时间时 $T_{co}$ 还没收子事务的提交或夭折的任何回答，则 $T$ 回滚。若所有的参与者回答提交其子事务，则 $T_{co}$ 提交 $T$。

### 4.2 MHs 上的事务处理

MHs 处理移动事务的读写请求和调整有效性确认间隔。当移动事务进行读或写请求时，它的有效性确认间隔被调整，以反映移动事务与提交事务之间的串行关系。令 $x_k$ 表示 $x$ 的版本，其写时间戳小于 $T_S(T_i)$ 的最大写时间戳。移动事务 $T_i$ 发出读数据项 $x$，选择读版本 $x_k$，$V_l(T_i)$被调整，即置 $V(T_i)$= $V(T_i)\cap[W_{TS}(x_k), \infty]$，同时将 $x_k$ 版本放入 $R_{Set}(T_i)$中。移动事务 $T_i$ 发出写数据项 $x$，若 $T_S(T_i)<R_{TS}(x_k)$，则 $T_i$ 重启动；否则创建 $x$ 的一个新版本 $x_i$，$V_l(T_i)$被调整，即置 $V(T_i)$= $V(T_i)\cap[W_{TS}(x_k), \infty]\cap[R_{TS}(x_k), \infty]$，并放入 $W_{Set}(T_i)$中。

在 MHs 进行局部部分有效性确认，使用向后有效性确认机制，以保证没有任何移动事务与上个广播周期在服务器提交事务有数据冲突。服务器在广播周期开始时广播上个广播周期在服务器提交的事务有效性确认信息。移动事务 $T_i$ 对于已提交的事务 $T_c$ 进行局部有效性确认方法如下：

**a.** 若 $T_i$ 与提交事务 $T_c$ 有读−写冲突，即 $R_{Set}(T_i)\cap W_{Set}(T_c)\neq\varnothing$，则调整串行次序为 $T_i\rightarrow T_c$，这意味着虽然 $T_c$ 在 $T_i$ 之前提交，$T_i$ 的读应放在 $T_c$ 写之前。调整 $V(T_i)$使得 $V_u(T_i)<T_S(T_c)$，即置 $V(T_i)=V(T_i)\cap[0, T_S(T_c)]$。由于调整 $V_u(T_i)$，$C_{WriteSet}$ 为所有提交事务的写集合，对于每个数据项 $x_k\in R_{Set}(T_i)$，若 $x_j\in C_{WriteSet}$，则调整 $V_u(T_i)$到 $\min(\{W_{TS}(x_k)|\ x_k \text{ in } C_{WriteSet}\})$。$x_k$ 标记已进行有效性确认，不需要再在服务器上进行最终有效性确认。因为在当前广播周期开始后提交的事务写数据项的时间戳大于 $C_{WriteSet}$ 中所有数据项的写时间戳。

**b.** 若 $T_i$ 与提交事务 $T_c$ 有写−读冲突，即 $W_{Set}(T_i)\cap R_{Set}(T_c)\neq\varnothing$，则调整串行次序为 $T_c\rightarrow T_i$，这意味着 $T_c$ 的读不应影响 $T_i$ 的写。调整 $V(T_i)$使得 $V_l(T_i)>T_S(T_c)$，即置 $V(T_i)=V(T_i)\cap[T_S(T_c), \infty]$。由于调整 $V_l(T_i)$，所有提交事务的读集合为 $C_{ReadSet}$，对于每个数据项 $x_i\in W_{Set}(T_i)$，若 $x_j\in C_{ReadSet}$，则调整 $V_l(T_i)$ 到 $\max(\{R_{TS}(x_j)|\ x_j \text{ in } C_{ReadSet}\})$。因为提交事务不可能读活跃事务所写的数据项。

移动只读事务时，若所有读数据项通过部分向后有效性确认，则可提交，串行在当前广播周期前提交的事务之后，当前广播周期后提交的事务之前。移动更新事务如果通过部分向后有效性确认，必须提交到服务器进行最终有效性确认。因为移动更新事务串行在它到达有效性确认阶段之前、提交的事务之后及所有活跃事务之前。MHs 上移动事务执行局部有效性确认算法如下：

```
local_validation(Ti)  {
  for each xk ∈RSet(Ti)  {
    if(xj ∈CWriteSet)  {
      V(Ti)=V(Ti)∩[0, min({WTS(xj)|  xj∈CWrit eSet})];
      if(V(Ti)==[ ] )  abort Ti;
    }
    mark xk valid;
  }
  for each xi ∈WSet(Ti)  {
    if(xj ∈CReadSet)  {
      V(Ti)=V(Ti)∩[max({RTS(xj)}| xj∈CReadSet)], ∞];
      if(V(Ti)==[ ] )  abort Ti;
    }
  }
  submit Ti to the server for local final validation;
}
```

### 4.3 在服务器上事务的处理

假设事务 $T_i$ 和 $T_j$ 分别成功创建数据项 $x$ 版本 $x_i$ 和 $x_j$，版本序列定义为：$x_i<<x_j \Leftrightarrow T_S(T_i)<T_S(T_j)$。提交到服务器移动事务 $T_v$ 局部最终向后有效性确认方法如下：

**a.** $T_v$ 读数据项 $x$ 版本 $x_k$，若 $x_k$ 没进行有效性确认标记，并存在版本 $x_{k+1}$，$x_k<<x_{k+1}$，说明 $T_v$ 读版本 $x_k$ 后有提交事务写了数据项 $x$ 新版本 $x_{k+1}$，调整 $V_u(T_v)$ 到 $W_{TS}(x_{k+1})$，即置 $V(T_v)=V(T_v)\cap[0, W_{TS}(x_{k+1})]$。

**b.** $T_v$ 写数据项版本 $x_v$，提交事务不可能读 $T_v$ 写的版本 $x_v$，调整 $V_l(T_v)$到数据项 $x$ 版本中最大 $R_{TS}(x)$，即置 $V(T_v)=V(T_v)\cap[\max(R_{TS}(x)), \infty]$。

服务器上事务执行局部最终向后有效性确认算法描述如下：

```
local_final_validation(Tv)  {
  for each xk in RSet(Tv)  {
    if (xk is not marked valid)  {
      if(∃xk+1(xk<<xk+1))  V(Tv)=V(Tv) ∩[0, WTS(xk+1)];
      if(V(Tv)==[ ])  {
        abort Tv;
        ASet=ASet ∪ { Tv};
      }
    }
  }
  for each xv in WSet(Tv)  {
    V(Tv)=V(Tv) ∩[max(RTS(x)), ∞];
    if(V(Tv)==[ ])  {
      abort Tv;
      ASet=ASet ∪ { Tv};
    }
  }
```

### 4.4 全局有效性确认

若有效性确认事务 $T_v$ 是移动更新事务，则对于 $R_{Set}(T_v)$和 $W_{Set}(T_v)$中每个分布数据项 $x_k$，$T_v$ 有效性确认间隔 $VI(T_v)$ 向前调整为 $V(T_v)= V(T_v)\cap[W_{TS}(x_k), \infty]\cap[R_{TS}(x_k), \infty]$。移动分布事务执行全局有效性确认算法如下：

```
global_validation (Tv)  {
  if(Tv is update transaction)  {
    for each xk in RSet(Tv) or WSet(Tv)  {
      if(xk is marked distributed)  {
        V(Tv)=V(Tv)∩[WTS(xk), ∞]∩[RTS(xk), ∞];
        if(V(Tv)==[ ])  abort Tv;
      }
    }
  }
  CSet=CSet ∪ {Tv };
}
```

若$T_v$在服务器通过最终有效性确认，则$T_v$的最终时间戳$T_S(T_v)$=min($C_{Time}$, $V_I(T_v)$+Δ) ($C_{Time}$表示当前系统时间)，指示事务的串行次序的位置。若$T_v$没进行任何串行调整，则$T_v$的最终时间戳为$C_{Time}$。$R_{Set}(T_v)$中每一个数据项$x_i$ 的$R_{TS}(x_i)$ 更新为$T_S(T_v)$，$W_{Set}(T_v)$中每一个数据项$x_v$的$W_{TS}(x_v)$更新为$T_S(T_v)$，同时写入数据 库中。

## 5 性能评价

通过仿真测评本文提出的 DMVOCC-MDA 协议性能。模拟的模型由服务器、MHs 和广播磁盘组成。仿真基本参数如下：每过事务读写操作为 5~10 个，只读事务概率为 70%，过界率为 5%，断接率为 5%，断接时间为 1~50 s，松弛因子为 2.0~6.0，无线蜂窝为 5 个，MH 个数为 10 个，事务间隔为 1~7 s，数据库项为 500 项，其中，热数据项占 30%，CPU 调度为 EDF (Earliest deadline first)策略。

选用 DHP-2PL(Distributed high priority two-phase locking)[1] 和 PVTO-2PC (Partial validation with timestamp ordering combined with 2PC)[6]2 项协议作为基准协议。DHP-2PL 和 PVTO-2PC 是单版本协议，服务器只保存数据的最新版本，并周期地广播每一个数据项。

图 1 所示为不同工作负载对移动事务错过率的影响。可见，DMVOCC-MDA 协议错过率比 PVTO-2PC 协议和 DHP-2PL 协议的错过率都低，其原因是移动只读事务在 MHs 有效处理，无阻塞提交。 移动事务在 MHs 进行局部有效性确认，及早地检测数据冲突，减少了移动事务错过率。另一个主要原因是多版本机制消除了移动只读事务和移动更新事务之间冲突，读请求从不失败且不必等待。图 2 所示为不同工作负载对移动事务提交率的影响。可见，DMVOCC-MDA 协议性能要优于 PVTO-2PC 协议和 DHP-2PL 协议性能，其主要原因多版本机制消除了移动只读事务和移动更新事务之间冲突，移动事务在 MHs 局部有效性确认，较早地检测数据冲突，通过多版本动态调整事务串行次序，避免不必要的事务重启动。图 3 所示为在 20% 断接率下不同工作负载对移动事务过错率的影响。图 4 所示为在 20%断接率下不同工作负载对移动事务提交率的影响。可见，DMVOCC-MDA 比其他协议更能容忍 MHs 与网络断接。

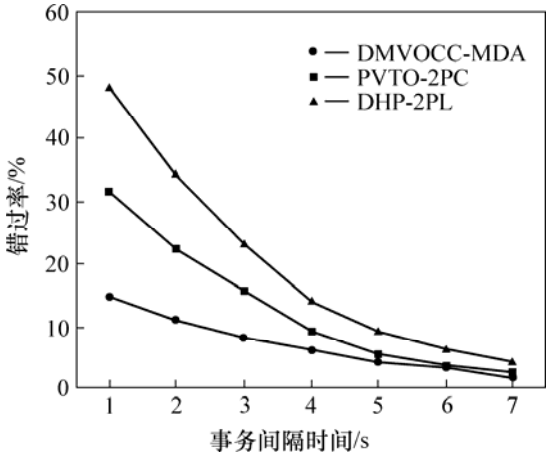


**图 1** 断接率为 5%时协议错过率与事务间隔时间的关系曲线

**Fig.1** Relationship between miss rate of protocol and think time with 5% of disconnection probability

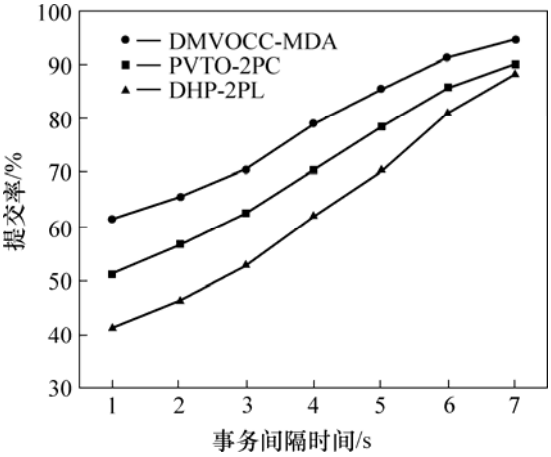


**图 2** 协议提交率与事务间隔时间的关系曲线

**Fig.2** Relationship between commit probability of protocol and think time with 5% of disconnection probability

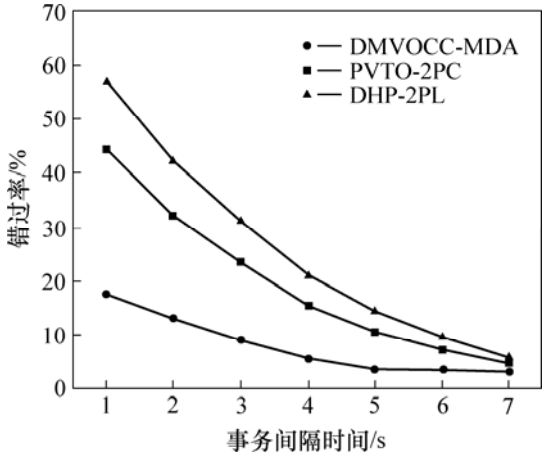


**图 3**　断接率为 20%时协议错过率与事务间隔时间的关系曲线

**Fig.3**　Relationship between miss rate of protocol and think time with 20% of disconnection probability

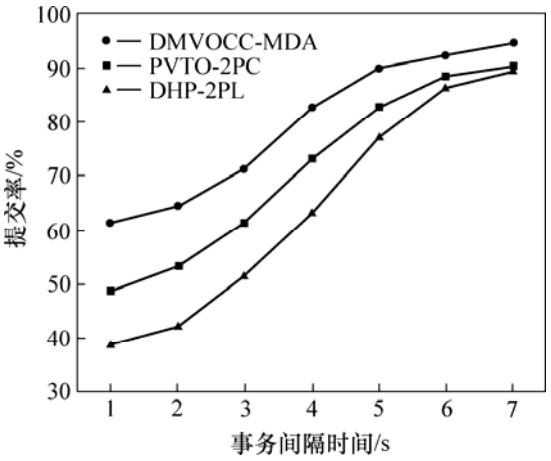


**图 4**　断接率为 20%时提交率与事务间隔时间的关系曲线

**Fig.4**　Relationship between commit probability of protocol and think time with 20% of disconnection probability

## 6　结　论

**a.** 在传输带宽是不对称的环境中，MHs 应尽量减少对上行传输信道的竞争，并发控制协议不应卷入 MHs 在事务执行期间进行数据冲突检测时与其他 MH 和服务器持续同步。

**b.** 提出了移动广播环境中 DMVOCC-MDA 并发控制协议处理移动分布式实时事务。移动只读事务能无阻塞提交，大大降低了移动只读事务的响应时间。消除了移动只读事务和移动更新事务的冲突。避免了不必要的移动事务重启动。

**c.** DMVOCC-MDA 并发控制协议与其他协议相比，错过率和重启动率低，提交率和吞吐量高。

**参考文献：**

[1] Lam K Y, Kuo T W, Tsang W H. Concurrency control in mobile distributed real-time database systems[J]. Information Systems, 2000, 25(4): 261−286.

[2] LIAO Guo-qiong, LIU Yung-sheng, WANG Li-na. Concurrency control of real-time transactions with disconnections in mobile computing environment[C]//Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing. Washington: IEEE Computer Society Press, 2003: 205−212.

[3] Lee S K, Kitsuregawa M, Hwang C S. Efficient processing of wireless read-only transactions in data broadcast[C]// Proceedings of 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/ E-Business Systems RIDE-2EC. Washington: IEEE Computer Society Press, 2002: 101−111.

[4] LI Guo-hui, YANG Bing, CHEN Ji-xiong. Efficient optimistic concurrency control for mobile real-time transactions in a wireless data broadcast environment[C]//Proceedings of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications. Washington: IEEE Computer Society Press, 2005: 443−448.

[5] Brayner A, Alencar F. A semantic serializability based fully distributed concurrency control mechanism for mobile multi-database systems[C]//Proceedings of 16th International Workshop on Database and Expert Systems Applications. Washington: IEEE Computer Society Press, 2005: 1085−1089.

[6] Lee V C S, Lam K W, Son S H. On transaction processing with partial validation and timestamp ordering in mobile broadcast environments[J]. IEEE Transactions on Computers, 2002, 51(10): 1196−1211.

[7] Janaki R D, Maluk M M A, Devanathan V R. A framework for concurrency control in real-time distributed collaboration for mobile systems[C]//Proceedings of 23rd International Conference on Distributed Computing Systems Workshops. Washington: IEEE Computer Society Press, 2003: 478−483.

[8] Cho S H, Lee J M, Hwang C S, et al. Hybrid concurrency control for mobile computing[C]//Proceedings of High-Performance Computing on the Information Superhighway, HPC-Asia ’97. Seoul, 1997: 478−483.

[9] Pitoura E, Chrysanthis P K. Multiversion data broadcast[J]. IEEE Transactions on Computers, 2002, 51(10): 1224−1230.

[10] LEI Xiang-dong, ZHAO Yue-long, YUAN Xiao-li. Transaction processing in mobile database systems[J]. Chinese Journal of Electronics, 2005, 14(3): 491−494.

[11] Jan L. Performance of distributed optimistic concurrency control in real-time databases[C]//7th International Conference on Information Technology. Berlin: Springer-Verlag, Lecture Notes in Computer Science, 2004, 3356: 243−252.

[12] Kumar V, Prabhu N, Dunham M H, et al. TCTOT—A timeout-based mobile transaction commitment protocol[J]. IEEE Transactions on Computers, 2002, 51(10): 1212−1218.