doi:10.11959/j.issn.1000-436x.2016150

面向移动计算环境的混合式数据同步机制

徐小龙，刘笑笑
（南京邮电大学计算机学院，江苏 南京 210003）

摘 要：提出混合式数据同步机制，有机融合集中式和 ad hoc 架构，设置自组织域(SOD, self-organization domain)，减少了同步数据通信量和数据同步服务器负载；提出基于节点能力值的数据分发策略，根据移动终端综合处理能力值来建立 SOD 树分发路径，实现同步数据的高效分发；还提出了基于轨迹变更的增量捕获策略，采用触发器捕获操作日志，用净化方法合并操作日志得到净增量数据。实验结果表明，混合式数据同步机制能更好地维护移动计算环境中数据的一致性，缩短同步响应时间，减少同步数据通信量，降低同步服务器负载。
关键词：移动计算；数据同步；混合式架构；同步；增量捕获
中图分类号：TP393 **文献标识码**：A

Hybrid data synchronization mechanism for mobile computing

XU Xiao-long, LIU Xiao-xiao
(College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: The hybrid data synchronization mechanism (HDSM) was proposed, which integrates centralized architecture and ad hoc architecture together, and a series of self-organization domains (SOD) were built as to cut down the traffic of synchronization data and lighten the load of synchronization server. The capacity-value-based data distribution strategy was proposed, which establishes data distribution paths of SOD tree based on the comprehensive processing capacity of mobile nodes to implement the high-efficient data transmission. The increment capture strategy based on track changes was proposed, which captures operation logs with triggers and obtains pure increment data by purification. The experimental results show that HDSM can make maintenance of data consistency better in mobile computing environment and shorten the response time of synchronization, and reduce the traffic of data synchronization and the load of synchronization server.
Key words: mobile computing, data synchronization, hybrid architecture, synchronization, increment capture

1 引言

随着各类无线通信技术的发展与普及，移动网络也日趋复杂。与固定的有线网络相比，移动网络的复杂性、动态性、弱连接性以及通信延迟与带宽相对有限等特征使移动计算环境中的数据同步机制效率难以提升，也难以实现高效数据的一致性。移动网络需要容忍移动终端的非持久性连接和非实时数据通信，为了满足移动环境下对业务数据的可靠处理需求，移动计算系统需要允许用户在离线情况下处理数据，需要支持多数据副本的分布式存储机制[1]。目前的分布式存储机制中普遍采用乐观复制(optimistic replication)[2]方法来保障业务数据的可用性和控制业务数据的一致性，以实现业务数据的最终一致性[3~6]。

近年来，国内外学者就移动计算环境中数据同步问题展开了一系列研究工作。目前的数据同步系统常采用集中式架构[4]和 ad hoc 架构[5]。在基于集

收稿日期：2015-02-08；**修回日期**：2016-07-27
基金项目：国家自然科学基金资助项目(No.61472192)；教育部科技发展中心网络时代的科技论文快速共享专项研究基金资助项目(No.2013116)；江苏省高校自然科学研究计划基金资助项目(No.14KJB520014)
Foundation Items: The National Natural Science Foundation of China (No.61472192), Special Fund for Fast Sharing of Science Paper in Net Era by CSTD (No.2013116), The Natural Science Fund of Higher Education of Jiangsu Province (No.14KJB520014)

中式架构的数据同步系统中的同步服务器上保存主数据集，各移动终端上保存主数据集的部分或全部数据；每次发起的数据同步过程都由同步服务器控制并执行，各移动终端之间不可直接通信，只能通过同步服务器间接通信；当移动终端在线或者条件允许的情况下，将自己的数据同步到同步服务器，由同步服务器分别同步到其他移动终端。与此同时，同步服务器还担负着冲突检测与处理、同步事务回滚等职责。基于 ad hoc 架构的同步系统通常将移动终端按照地域、业务逻辑等因素划分为一个个工作组，每个工作组同步或异步地完成同一个任务，分享文件及信息。

典型的移动数据同步技术包括 Microsoft 的远程数据库访问(RDA, remote data access)[7]技术、合并复制技术(MRT, merge replication technology)[8]以及 SyncML initiative 的同步标记语言(SyncML, synchronization mark up language)[9~12]等。这些移动数据同步方案一般采用传统集中式架构，难以满足大规模复杂移动计算系统的性能需求，需要另行寻找解决方法。为了更好地辅助各移动终端传输数据，提高系统的数据同步性能，有研究者提出了超级节点模型(SPM, super-peer model)[13]、基于树型结构的 SCOPE 系统[14]、基于 2 层结构的 OceanStore 系统[15]、混合树结构[16]等。为了减少同步数据量，提高同步效率，降低同步对移动网络带宽的需求，移动计算系统通常采用增量捕获策略(increment capture strategy)[17]，每次数据同步时只交换修改过的数据。典型的增量数据捕获方法有快照法(snapshot)[18,19]、触发器法(trigger)[20]、日志法(log)[21]和时间戳法(timestamp)[22]等。

移动数据同步方案存在以下一些问题需要解决：基于传统集中式架构的数据同步机制使服务器系统负载和通信量将呈线性上升趋势，容易形成系统的性能瓶颈；移动网络的弱连接性增加了数据同步失败的可能性，从而导致业务数据的不一致；基于纯粹 ad hoc 架构的数据同步方案[23,24]将全部的数据同步交由移动终端完成，增加移动终端的系统开销和电力消耗，还容易影响同步系统性能，延长同步响应时间，增大同步失败概率；现有的增量数据捕获方法面向稳定的分布式计算环境，并不适应移动计算环境，简单移植容易导致低效率、高开销，如 Log 法需要数据库有日志机制和操作日志权限、Timestamp 法要求更改数据集结构等。

针对上述问题，本文对移动计算系统中的数据同步机制展开一系列研究，主要贡献包括以下 3 个方面。

1) 本文提出了一种混合式数据同步机制(HDSM, hybrid data synchronization mechanism)，将集中式架构和 ad hoc 架构有机融合为基于自组织域(SOD, self-organization domain)的混合式体系架构，HDSM 以 SOD 为单位来管理同步进程，并且将部分同步处理逻辑分配到移动终端，减轻同步服务器负载的同时适当提升各移动终端的自主控制权，同时将同步数据通信量局域化在 SOD 内部，降低了通信开销，缩短同步响应时间。

2) 为了缩短同步响应时间，加快 SOD 中同步数据的分发速度，本文提出了一种基于节点能力值的数据分发策略(CDDS, capacity-value-based data distribution strategy)，通过构建 SOD 树，参考基于 ad hoc 架构的移动终端同步数据的分发方法[14~16]，按照 SOD 树的数据传输路径，实现了 SOD 内各移动终端间的高效数据同步。

3) 本文提出了一种基于轨迹变更的增量捕获策略(ICSTC, increment capture strategy based on track change)，采用触发器捕获操作日志，记录数据集的操作变化过程，并采用净化方法合并操作日志，实现净增量的捕获和整个操作变化过程的记录，以及在发生意外终止、严重超时等影响数据最终一致性时回滚，从而有效减少同步数据通信量和同步响应时间。

2 相关工作

移动计算系统维护数据各副本一致性一般采用乐观复制方法[2]，系统中多个节点中的数据副本都可以独立进行更新，允许各个节点的数据处理完成之后再进行一致性控制；所有的更新信息会写在一个更新事务中，通过同步协议传送到系统的其他节点，经过同步后数据最终达到一致。采用乐观复制方法能够减少网络通信量，降低同步失败所需要的开销；其缺点是当有多个移动终端对同一数据修改并且提交同步请求时会发生数据冲突。

RDA[7]是面向移动计算环境的集中式架构同步模型，其数据同步进程包含 push 和 pull 这 2 种操作。pull 操作将数据服务器中的数据集下载到移动终端上；push 操作将本地数据集的增量信息或者整

个数据集提交至远程数据服务器中。同一时间只能是 push 或 pull 操作，并且一次只能操作一个数据集；当执行 pull 操作的时候传输整个数据集而非增量数据，增加了同步数据通信量，延长了整个系统的同步响应时间，与此同时还增加了移动终端的存储空间。MRT[8]主要针对同步时会产生数据冲突的移动应用程序，允许分别在移动终端和同步服务器上自行更新数据，之后移动终端和同步服务器之间可以双向交换数据增量进行数据同步。SyncML[9~12]由数据同步协议、数据表示协议和传输绑定协议 3 部分组成，它是基于扩展标记语言(XML, extensible mark up language)，具有平台无关性、行业通用性和开放性等特征，为不同网络、平台及设备间的远程数据同步提供了统一的、规范的数据同步协议。

增量同步方法[17]减少了同步通信流量，但是如何获取数据增量信息成为新的难题。已存在数据同步机制中增量数据捕获方法有 Snapshot[18,19]、Trigger[20]、Log[21]、Timestamp[22]等，它们各有优缺点。Snapshot 取数据集的新旧快照进行对比以提取信息增量，这一方法适用面较广，但对存储空间要求较高，随着数据元组的增多，对比检测算法易成为系统的性能瓶颈。Trigger 利用触发器捕获增量数据，增量捕获效率高，但只应用于设有触发器机制的数据管理系统中，而且当数据集中有大量数据进行操作时，触发器对系统的性能影响较大。Log 采用分析数据库自身带有的操作日志来提取增量，不增加系统额外开销，效率高，但数据库和管理系统对日志的访问一般都有严格的权限限制；此外，数据管理系统的日志格式大都各不相同，导致日志法的使用受到诸多限制。Timestamp 在元组中设置一个时间戳字段，同步时将大于上次同步时间的所有字段提取出来即可获取增量，简单易实现，但此方法要求改变业务数据集的结构，并且难以捕获删除操作，限制了其适用范围。

3　混合式数据同步机制

3.1　混合式架构

本文提出的 HDSM 所采用的混合式架构由移动终端(MT, mobile terminal)及其移动数据系统、无线接入点(AP, access point)、同步数据服务器（SyncServer）和主数据库（SyncDB）构成，如图 1 所示。

定义 1　自组织域(SOD, self-organization domain)。若干地理位置邻近且能够通过某种无线通信方式相互通信的移动终端，为了更好地完成各自的任务而自动组织到一起，以协作方式工作的移动终端群称为自组织域。

HDSM 的混合式架构是将集中式架构和 ad hoc 架构有机融合，以 SOD 为移动终端自组织单位，SOD 中的各个移动终端通过服务性移动终端（SMT, service mobile terminal）与同步服务器进行数据同步，服务性移动终端接收并向 SOD 内部的其他移动终端转发同步数据。

在 HDSM 中，当某个移动终端接收到同步服务器发来的同步请求时，首先利用无线通信技术（如蓝牙、红外线或 Wi-Fi 等）检测邻近区域内是否存在着有同样需求的其他移动终端。若存在，则在同步服务器的协助下，移动终端以协作方式从基站下载同步数据并自主地形成 SOD。

同步服务器以 SOD 为单位来进行同步，当进

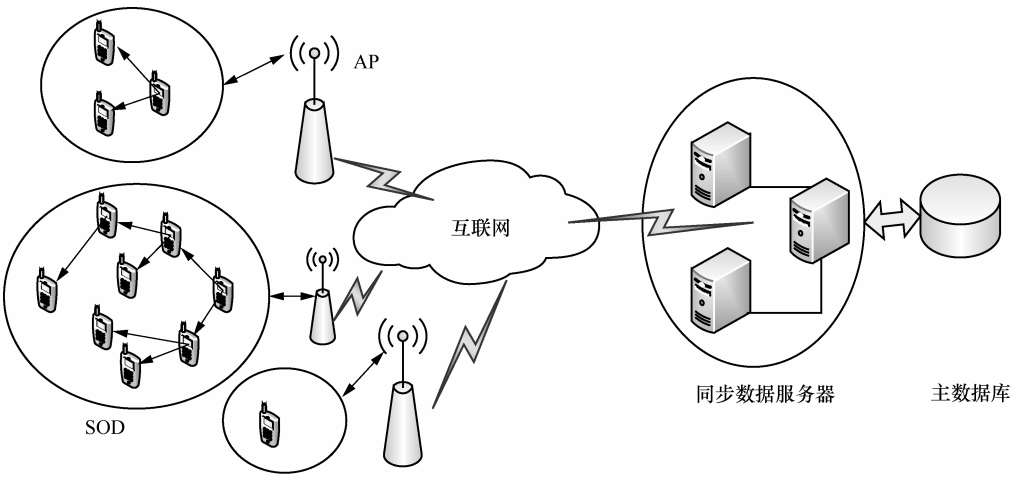

图 1　混合式网络架构

行数据同步时，同步服务器并不需要与每一个移动终端都直接进行数据同步，而是仅与 SOD 中的服务性移动终端进行直接的数据同步，其他移动终端只需要从服务性移动终端获得同步数据即可。当移动终端加入同步系统时，只是加入到地域邻近的 SOD 中，所以移动终端的加入不会给同步服务器造成负担。

3.2 基于节点能力值的数据分发策略

由于移动终端处理能力有限，当同步数据在 SOD 内部进行转发时应在保证其有效转发的同时尽量减少各移动终端转发的次数，尤其是 CPU、内存、电量等性能较差的移动终端。围绕 HDSM，本文设计一种基于节点能力值的数据分发策略 CDDS，抛弃传统多播式的一对多数据分发机制，提高同步数据传输效率的同时保证同步质量。

将一个拥有 n 个移动终端的 SOD 表示成一棵完全二叉树 $T(V)$，如图 2 所示。其中，$V=\{V_1,V_2,V_3,\cdots,V_n\}$ 表示移动终端集合。本文为完全二叉树的每个节点赋予一个权值 ω，那么 SOD 表示成的一棵带权完全二叉树则为 $T(V)=\{V_1(\omega_1),V_2(\omega_2),V_3(\omega_3),\cdots,V_n(\omega_n)\}$。以此完全二叉树为同步数据的转发路径，则每条路径上的同步数据被转发的次数最大不超过 $\lfloor \mathrm{lb}n \rfloor$，而每个移动终端转发同步数据的次数不超过完全二叉树的度，位于叶子的移动终端则不需要转发同步数据，从最大程度上减少了同步数据在 SOD 内部的转发次数，减少了数据传输延迟，降低了数据转发失败的风险。

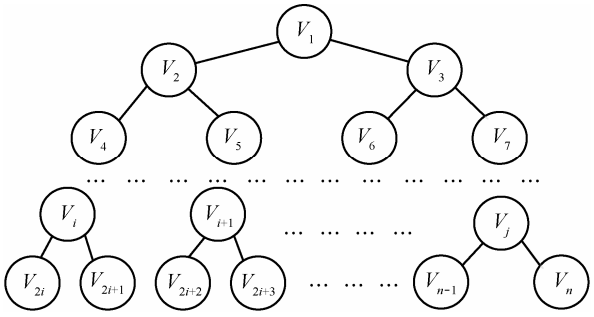

图 2 SOD 树

一棵完全二叉树叶子节点数为 $n_0=\left\lfloor \frac{n+1}{2} \right\rfloor$，叶子节点数显然大于处理能力偏低的移动终端数，CDDS 将那些处理能力偏低的移动终端分配到叶子节点上，令其不再转发同步数据，降低同步数据转发失败概率，从而提高同步成功率。

构建 SOD 树，首先是由 SOD 内部的所有移动终端通过选举法[25]产生协调者(coordinator)，然后由协调者协调 SOD 内部各移动终端来构建 SOD 树和确定服务性移动终端，具体方法如下。

1) 移动终端综合能力值 C 的确定

移动终端综合能力值主要取决于自身的 CPU 利用率和 RAM 利用率

$$C=\alpha N_{\mathrm{CPU}}(1-\overline{P}_{\mathrm{CPU}})+\beta(1-\overline{P}_{\mathrm{RAM}}) \tag{1}$$

其中，N_{CPU} 表示 CPU 的核数，CPU 核数越多，表示其并行处理能力越强；$\overline{P}_{\mathrm{CPU}}$ 和 $\overline{P}_{\mathrm{RAM}}$ 分别表示 CPU 和 RAM 的平均利用率，$\overline{P}_{\mathrm{CPU}}$ 和 $\overline{P}_{\mathrm{RAM}}$ 显示了移动终端所运行的程序消耗 CPU 和 RAM 资源的程度。α 和 β 分别为 CPU 和 RAM 的剩余利用率对于确定移动终端综合能力值 C 而设置的权值，分别反映了运行某程序对该移动终端 CPU 和 RAM 的要求。下面确定 $\overline{P}_{\mathrm{CPU}}$、$\overline{P}_{\mathrm{RAM}}$、$\alpha$ 和 β 的值，若移动终端随机选取最近限定时间区间的 10 对样本值，如表 1 所示。

表 1 随机选取的 10 对样本

时刻	$\overline{P}_{\mathrm{CPU}}$	$\overline{P}_{\mathrm{RAM}}$
t_1	P_{c_1}	P_{a_1}
t_2	P_{c_2}	P_{a_2}
t_3	P_{c_3}	P_{a_3}
t_4	P_{c_4}	P_{a_4}
t_5	P_{c_5}	P_{a_5}
t_6	P_{c_6}	P_{a_6}
t_7	P_{c_7}	P_{a_7}
t_8	P_{c_8}	P_{a_8}
t_9	P_{c_9}	P_{a_9}
t_{10}	$P_{c_{10}}$	$P_{a_{10}}$

其中，$P_{c_1},P_{c_2},\cdots,P_{c_{10}}$ 是 CPU 利用率的 10 个样本值；$P_{a_1},P_{a_2},\cdots,P_{a_{10}}$ 是 RAM 利用率的 10 个样本值，则

$$\overline{P}_{\mathrm{CPU}}=\frac{1}{10}\sum_{i=1}^{10}P_{c_i} \tag{2}$$

$$\overline{P}_{\mathrm{RAM}}=\frac{1}{10}\sum_{i=1}^{10}P_{a_i} \tag{3}$$

$$\alpha=\frac{\left\lceil 100P^* \right\rceil}{\left\lceil 100P^* \right\rceil+100} \tag{4}$$

$$\beta = 1 - \alpha \tag{5}$$

$$P^* = \frac{1}{10}\sum_{i=1}^{10}\left(\frac{P_{c_i}}{P_{a_i}}\right) \tag{6}$$

其中，P^* 表示移动终端最近限定时间区间内 CPU 和 RAM 利用率比值的平均值，它同时显示了最近限定时间区间内该移动终端运行时所消耗 CPU 和 RAM 的比率，以此确定 α 和 β 的值。利用式(2)～式(6)可以得到式(1)中的参数，再根据式(1)即可求出移动终端综合能力值 C。

2) 服务性移动终端的确定

由于服务性移动终端负责与同步服务器通信、接收同步数据并转发给其他移动终端，需要与 AP 通信，所以服务性移动终端不仅要有较强的综合处理能力，而且要有较强的通信能力。

定义 2　服务指数。用来衡量移动终端服务能力的大小，用 E 表示。

服务性移动终端的确定首先由协调者获取 SOD 内部各移动终端接收 AP 的信号强度 $S_i(i=1,2,\cdots,n)$，结合各移动终端综合能力值 $C_i(i=1,2,\cdots,n)$，计算出各个移动终端的服务指数 $E_i(i=1,2,\cdots,n)$，公式如下

$$p_i = \frac{S_i - S_{\min}}{S_{\max} - S_{\min}} \tag{7}$$

$$q_i = \frac{C_i - C_{\min}}{C_{\max} - C_{\min}} \tag{8}$$

$$E_i = \eta p_i + \lambda q_i \tag{9}$$

其中，$i=1,2,\cdots,n$。在式(7)~式(9)中，$S_{\min}$ 和 $S_{\max}$ 分别表示当前 SOD 中各移动终端接收 AP 的信号强度的最小与最大值；$C_{\min}$ 和 $C_{\max}$ 分别表示当前 SOD 中各移动终端综合能力值的最小和最大值；p_i $(0 \leqslant p_i \leqslant 1)$ 和 q_i $(0 \leqslant q_i \leqslant 1)$ 分别表示移动终端 i 归一化后的信号强度值和综合能力值，p_i 越大表示移动终端 i 接收的 AP 信号强度越大；同理 q_i 越大表示移动终端 i 的综合能力值越大；η 和 λ 分别表示移动终端接收 AP 的信号强度与移动终端综合处理能力的重要度权值。在 SOD 中，服务性移动终端不仅要有较强的综合处理能力，而且要有较强的通信能力，因此本文设定 $\eta = \lambda = 0.5$。根据式(9)可以计算出每个移动终端的服务指数 $E_i(i=1,2,\cdots,n)$，然后由协调者选取服务指数最大的移动终端(即 Max(E_i))作为服务性移动终端。

3) SOD 树的构建

以移动终端的综合能力值 $C_i(i=1,2,\cdots,n)$ 作为权值 ω_i，则带权 SOD 树的构建流程如下。

① 将 SOD 中的各移动终端按综合能力值 C 降序排列并依次编号。

② 取 SMT 为根节点，同时有序地将各个节点按照树的层次遍历顺序从上到下、从左到右逐层建立成完全二叉树。

③ 将建立好的完全二叉树的各个节点按层次遍历顺序保存在一维数组（设为数组 H）中，便于随机读取左右孩子节点信息。

3.3　数据同步模型

1) 同步方式

基于本文混合式架构，在数据同步过程中根据同步数据的流向分为以下 3 种同步方式。

① 移动终端→同步服务器，移动终端向同步服务器发起数据同步。移动终端将自己的同步数据发送给同步服务器，同步服务器解决数据冲突并合并，更新本地数据。

② 同步服务器→移动终端，同步服务器向移动终端发起数据同步。同步服务器将自己的同步数据发送给未更新过的移动终端。

③ 移动终端→移动终端，移动终端之间相互转发同步数据。位于 SOD 中的移动终端按照 CDDS 转发同步数据。

同步数据流向如图 3 所示。

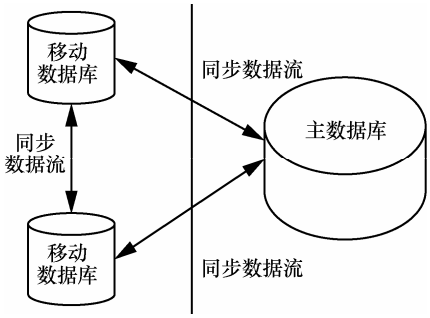

图 3　同步数据流向

2) 数据操作模型

对数据集的数据操作类型(Operation Type)可以分为 3 种：A（新增）、M（修改）、D（删除）。*Operation* (OperationType, Data)来表示一条数据记录的更新操作，Data 表示操作记录对象。

① *Operation* (A, Data)：表示增加一条记录。

② *Operation* (M, Data)：表示修改一条记录。

③ *Operation* (D,Data)：表示删除一条记录。

在数据同步的过程中，为了判断移动终端与同步服务器是否处于数据一致状态，需要给每次同步操作设置一个同步标志来唯一标记同步活动，本文用同步版本号(VersionID)表示，同步版本号成为同步机制的关键。由此，可定义一次同步操作为一次同步版本的更新，用同步版本矢量 ***SyncVector*** (*Operation*, *VerionID*) 表示，其中，*VerionID* 与 *Operation* 是一对多的关系，即在一次同步版本更新中可以有多个更新操作。

为了保证同步操作的正常进行，移动终端和同步服务器需要同时保存上次同步版本号 *LastVersionID*。当发起同步的时候，首先比较 *LastVersionID*，若相等则说明上次同步操作成功，数据处于一致状态，可以进行数据同步；若不等则说明上次同步操作失败，本次同步操作终止或者转向其他解决方法。同步版本号有效地保证了移动终端和同步服务器数据的一致性。

数据同步处理模型如图 4 所示。本文提出的同步处理模型与传统的同步处理模型不同之处是移动终端不仅与同步服务器进行同步，还需与其他移动终端进行同步操作，从而实现了同步功能转移，减少了同步服务器的负担。

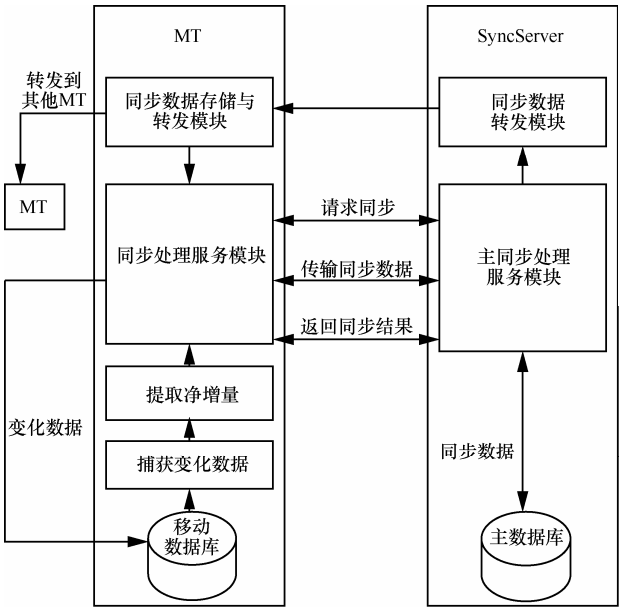

图 4 移动数据同步模型

移动终端组成模块包括：操作日志捕获模块，负责捕获操作记录的变化过程，并记入操作日志；操作日志净化模块，负责读出操作日志的内容，净化合并，消除冗余记录数据；同步处理服务模块，负责同步请求与响应、同步数据发送与接收、冲突处理与合并等；同步数据存储与转发模块，负责存储与转发同步服务器或其他移动终端发来的同步数据。

同步服务器组成模块包括：主同步处理服务模块，负责处理同步请求队列、数据冲突处理与合并、ID 映射处理、数据回滚等；同步数据转发模块，负责向移动终端转发同步数据。

基于同步处理模型，同步处理过程中遵循以下原则。

① 当一次同步更新成功完成后 *VersionID* 自增 1，并且将本次同步版本号赋值给上次同步版本号，即 *LastVersionID=VersionID*。

② 为了减少同步冲突的情况，移动终端登录后立即检测同步服务器端同步更新情况，若服务器端的相关数据有更新则立刻发起同步并更新到本地。

③ 移动终端在发起并成功完成一次同步后可按需删除本地操作日志，而同步服务器需保存每个版本的同步数据(即同步版本矢量)，直到同步系统中所有移动终端的相关数据都达到一致状态才可删除。

④ SOD 中的服务性移动终端接收到同步数据后按照 CDDS 转发给其他移动终端，同时根据 *LastVersionID* 判断是否进行同步操作并按照 SOD 树路径逆向逐跳返回同步结果。

⑤ SOD 中各移动终端接收到同步数据后进行冲突检测并以服务器优先的原则更新本地数据。

3.4 典型应用场景

根据以上同步模型和原则，本文以一个典型应用场景来进一步详细描述同步流程。现有 W_A 和 W_B 这 2 个 SOD，其中，W_A 中有 A_1、A_2、A_3 这 3 个移动终端，W_B 中有 B_1、B_2、B_3、B_4 这 4 个移动终端，同步服务器和移动终端的所有数据都处于一致状态，并且上次同步版本号 *LastVersionID* 都为 0。现在 A_1 新增一个数据集 *X*，并且要将此信息同步到 SyncServer 以及其他移动终端。具体同步流程如下。

Step1 A_1 增加一个数据集 *X* 并请求同步，则 A_1 的同步版本号增 1，即 *VersionID-*A_1= *LastVersionID-*A_1+1=1。A_1 向 SyncServer 发送同步请求消息 (*VersionID-*A_1=1，*LastVersionID-*A_1=0)。SyncServer 接收到后从本地数据集中取出与 A_1 最近一次同步

的版本号(*LastVersionID-A*$_1$=0)，与之比较发现相等，并且 *VersionID-A*$_1$ > *LastVersionID-A*$_1$。于是 SyncServer 允许 A_1 的同步请求并且向 A_1 请求同步版本矢量。

Step2 A_1 收到 SyncServer 的请求后，发送本次同步版本矢量 ***SyncVector-A***$_1$(*Operation*,1) 给 SyncServer。

Step3 SyncServer 收到同步版本矢量后更新本地数据，完成数据同步并且保存版本矢量内容到本地。此时 SyncServer 上的 *LastVersionID-A*$_1$ 增 1，即为 1。

Step4 同时，移动终端 B_1 增加一个数据集 Y 并请求同步，则 B_1 的同步版本号增 1，即 *VersionID-B*$_1$= *LastVersionID-B*$_1$+1=1。B_1 向 SyncServer 发送版本同步请求消息 (*VersionID-B*$_1$=1，*LastVersionID-B*$_1$=0)。SyncServer 接收到后从本地数据集中取出与 B_1 最近一次同步的版本号（*LastVersionID-B*$_1$=0），与之相比较发现相等，并且 *VersionID-B*$_1$> *LastVersionID-B*$_1$。于是 SyncServer 允许 B_1 的同步请求并且向 B_1 请求同步版本矢量。

Step5 B_1 收到 SyncServer 的请求后，发送本次同步版本矢量 ***SyncVector-B***$_1$(*Operation*,1) 给 SyncServer。

Step6 SyncServer 收到 B_1 的同步版本矢量。由于 A_1 在 B_1 之前已完成与 SyncServer 同步，所以 SyncServer 比较同步版本矢量 ***SyncVector-B***$_1$ (*Operation*, 1)和 ***SyncVector-A***$_1$(*Operation*,1)，并进行冲突检测与合并，之后更新本地数据。此时将 SyncServer 上的 *LastVersionID-B*$_1$ 加 1，即为 1。SyncServer 根据需要将合并后的同步数据发送给 A_1 和 B_1，A_1 和 B_1 再次更新本地数据或者回滚同步数据。

Step7 同时，SyncServer 向 W_B 中的 B_3 发送同步数据请求，B_3 收到请求信息后与邻近的移动终端组成 SOD 并且按照 CDDS 构造 SOD 树。假设 B_3 被选为 SMT，则将 SOD 中未同步过的移动终端同步版本号信息((*VersionID-B*$_2$=1, *LastVersionID-B*$_2$=0), (*VersionID-B*$_3$=1, *LastVersionID-B*$_3$=0), (*VersionID-B*$_4$=1, *LastVersionID-B*$_4$=0))发送到 SyncServer。SyncServer 判断 B_2 和 B_4 的 *LastVersionID* 是否正确，并且将判断结果和同步数据发送给 B_3。B_3 接收到数据后首先按照 CDDS 将同步数据分发到 B_2 和 B_4。同步数据在 SOD 分发的过程中，根据判断结果，SyncServer 判定 *LastVersionID* 不一致的移动终端将舍弃本次同步数据。然后，B_2、B_3 和 B_4 按照服务器优先的原则进行冲突处理并更新到本地数据集，各移动终端同步成功后保存各自 *LastVersionID* 为 1，并按照 SOD 树路径逆向逐跳返回同步结果到 SyncServer。

Step8 W_A 中其他各移动终端同样以 Step7 的方式更新同步数据并向 SyncServer 返回同步结果。

Step9 SyncServer 接收到各移动终端的同步结果后更新本地各移动终端的 *LastVersionID*，至此各移动终端和 SyncServer 的同步版本号均为 1，成功完成本次同步。

当移动终端 A_1 和 B_1 与 SyncServer 同步成功后，可删除本地操作日志，只需要在 SyncServer 上保存相应的同步版本矢量即可；当同步系统中所有移动终端即 A_1、A_2、A_3、B_1、B_2、B_3、B_4 都完成同步并使系统处于一致状态时，则可以将 SyncServer 上保存的同步版本矢量删除，并且将各移动终端的 *LastVersionID* 置 0。这样做可以避免移动终端或者 SyncServer 保存大量的同步数据，浪费存储空间。

4　增量捕获策略

4.1　数据变更轨迹捕获

本文提出了基于轨迹变更的增量捕获策略 ICSTC 来捕获增量数据，以适应移动计算环境中移动终端写操作频次较低的特点。ICSTC 分为 2 个步骤：1) 采用触发器捕获数据变更轨迹，如图 5 所示；2) 对捕获到的数据变更轨迹进行净增量处理。

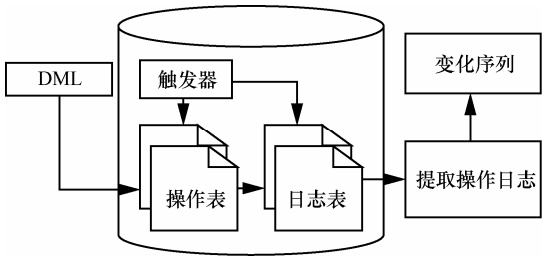

图 5　触发器捕获增量

ICSTC 策略包括以下内容。

1) 在操作数据集上定义 3 个触发器，分别是新增触发器、修改触发器和删除触发器，这些触发器用来捕获数据变更轨迹即记录的变化过程。

2) 设置操作日志来记录数据变更轨迹。操作日志除了拥有操作数据集的所有字段外，还包括“操

作类型”和“同步版本号”字段，“操作类型”取A、M、D这3种类型之一，“同步版本号”为上次同步版本号加1，用以标识本次同步版本号。

3) 定义一个向量来记录本移动终端的唯一编号和上次同步版本号，作为本次能否进行同步和本次同步版本号的依据。

4) 当操作数据集数据有变动的时候即可触发对应的触发器，将数据变更轨迹记录在操作日志中，并标明操作类型和本次同步版本号。

5) 当发起本次同步时，根据本次同步版本号从操作日志中取出本次全部数据变更轨迹，经过净增量处理后传送到同步服务器。

由于操作日志中的数据记录逐渐增多，随着系统的运行必然会累积大量的冗余数据（即已经同步过的数据），这样不利于数据查询和维护等操作，给数据库服务器带来更大的性能开销。所以，当确定移动终端与同步服务器同步成功后，即可将本地操作日志清除。

4.2 净增量处理

由于移动环境的弱连接特点和移动同步的需要，对增量数据进行净增量处理。

定义 3 净增量处理。指将作用在操作数据集上的一系列操作等价合并，压缩操作步骤，使操作序列在压缩后的操作作用下和原始操作作用下的最终状态是一致的。

净增量处理不仅能减少增量数据实际大小，减少数据传输时间，而且可以减少同步服务器对数据的加载维护时间，使整个同步过程更加稳定、高效。

为了更好地表达关系型数据库中各种操作变化过程及其结果关系，本文引入关系集合的概念，则一张表的关系可以用关系集合 R 来表示，表中元组就是关系集合中的元素。若一张表中有 n 个元组，则表示为关系集合 $R=\{r_1,r_2,\cdots,r_n\}$，其中，$r_1,r_2,\cdots,r_n$ 代表元组。

定义 4 操作函数。关系集合 X 经过一系列操作后变成 X'，本文可以描述为 X 经过函数 $\Psi(X)$ 作用后变成 X'，即 $\Psi(X)=X'$，记函数 Ψ 为 X 到 X' 的一个操作函数。由操作函数的定义和增量的定义可知，操作日志中某一元组的一系列操作即为该元组的一个操作函数。

定义 5 等价操作函数。对于关系集合 A，在操作函数 f_1 的作用下得到 A',即 $f_1(A)=A'$，若有另一操作函数 f_2 使 $f_2(A)=A'$，则操作函数 f_1 和 f_2 相对于 A 即为等价操作函数，记作 $f_1(A)\sim f_2(A)$。

定义 6 最优等价操作函数。与 f_1 等价的操作函数并不唯一，若 f_2 是 f_1 所有等价操作函数中操作步骤最少的，则 f_2 是 f_1 的最优等价操作函数。

对增量数据进行净增量处理是根据 $f_1(X)=X'$ 找到其最优等价操作函数 f^* 使

$$f^*(X)=X' \tag{10}$$

对数据集的一系列操作本质上是对各个元组的操作。操作日志用关系集合 L 表示，对操作数据集中第 i 个元组的操作集合用 $L_i(1\leqslant i\leqslant n,i\in N)$ 来表示，所以操作日志关系集合可表示为 $L=(L_1\cup L_2\cup\cdots\cup L_n)$。由此，对关系集合 L 的净化处理，可以转化为对每一个元组的操作集合 L_i 的净化处理，即

$$\begin{aligned} f^*(L)&=f^*(L_1\cup L_2\cup\cdots\cup L_n)\\ &=f^*{}_1(L_1)\cup f^*{}_2(L_2)\cup\cdots\cup f^*{}_n(L_n) \end{aligned} \tag{11}$$

在关系型数据库中，对同一元组的任何操作序列都可以分解为以下几种操作类型，如表2所示。

表 2 操作类型

编号	操作类型	意义
1	A	新增
2	M	修改
3	D	删除
4	N	无操作
5	A-M	新增后修改
6	A-D	新增后删除
7	M-M	修改后再修改
8	M-D	修改后删除

例如，一个元组经过一系列操作“新增—修改—修改—删除”,则可由表2中基本操作类型“1-2-2-3”组合成；也可以由复合操作类型“5-7-8”组合成。

由定义3可知，对操作日志的净化就是对操作数据集中每一个元组的操作序列的压缩。首先将操作日志中的操作序列按照元组分组，再将各个元组的一系列操作全部拆分为基本操作类型和复合操作类型，最后合并运算。运算规则如表3所示。

表 3　同一元组操作合并运算规则

操作类型	A	M	D	N
A	/	/	/	A
M	A	M	/	M
D	NULL	D	/	D
N	A	M	D	N

表 3 中“/”表示不存在此种合并操作，NULL 表示操作合并结果为空，运算规则可以描述如下。

1) $OP_{\mathrm{A}} \cup OP_{\mathrm{M}} = OP_{\mathrm{A}}$，1) $OP_{\mathrm{A}} \cup OP_{\mathrm{D}} = \mathrm{NULL}$，

3) $OP_{\mathrm{M}} \cup OP_{\mathrm{M}} = OP_{\mathrm{M}}$，4) $OP_{\mathrm{M}} \cup OP_{\mathrm{D}} = OP_{\mathrm{D}}$，

5) $OP_{\mathrm{J}} \cup OP_{\mathrm{N}} = OP_{\mathrm{N}} \cup OP_{\mathrm{J}} = OP_{\mathrm{J}}, \mathrm{J} \in \{\mathrm{A,M,D,N}\}$。

其中，OP 表示对元组的一次操作。

对于操作日志关系集合 L 中对操作数据集第 i 个元组的操作集合 $L_i(1 \leqslant i \leqslant n, i \in N)$，假设 $L_i = \{\ell_1, \ell_2, \cdots \ell_k \cdots \ell_n\}(1 \leqslant k \leqslant n, k \in N)$，$OP_{\Gamma}(\ell_k), \Gamma \in \{\mathrm{A,M,D}\}$ 表示对第 i 个元组的第 k 次操作类型是 Γ，结果是 ℓ_k，则元组的操作序列集合可以分为 5 个类型，分别采用以下 5 种净化公式。

$$f^*_i(L_i) = OP_{\mathrm{A}}(\ell_1) \cup OP_{\mathrm{M}}(\ell_2) \cup \cdots \cup OP_{\mathrm{M}}(\ell_k) \cup \cdots \cup OP_{\mathrm{M}}(\ell_n) = OP_{\mathrm{A}}(\ell_n) \tag{12}$$

$$f^*_i(L_i) = OP_{\mathrm{A}}(\ell_1) \cup OP_{\mathrm{M}}(\ell_2) \cup \cdots \cup OP_{\mathrm{M}}(\ell_k) \cup \cdots \cup OP_{\mathrm{D}}(\ell_n) = \mathrm{NULL} \tag{13}$$

$$f^*_i(L_i) = OP_{\mathrm{M}}(\ell_1) \cup OP_{\mathrm{M}}(\ell_2) \cup \cdots \cup OP_{\mathrm{M}}(\ell_k) \cup \cdots \cup OP_{\mathrm{M}}(\ell_n) = OP_{\mathrm{M}}(\ell_n) \tag{14}$$

$$f^*_i(L_i) = OP_{\mathrm{M}}(\ell_1) \cup OP_{\mathrm{M}}(\ell_2) \cup \cdots \cup OP_{\mathrm{M}}(\ell_k) \cup \cdots \cup OP_{\mathrm{D}}(\ell_n) = OP_{\mathrm{D}}(\ell_n) \tag{15}$$

$$f^*_i(L_i) = OP_{\mathrm{D}}(\ell_1) = OP_{\mathrm{D}}(\ell_1) \tag{16}$$

根据对元组的操作序列类型，在式(12)~式(16)中采用相应的净化公式即可求得此元组的净化结果。所以对操作日志关系集合的最优等价操作函数为

$$f^*(L) = \bigcup_{k=1}^{n} f^*_k(L_k) \tag{17}$$

在得到操作日志后，即可用式(17)净化操作日志，去除冗余记录，得到净增量同步数据，以减少同步过程中所需要传输的同步数据量，缩短同步响应时间。

5　实验验证与性能分析

5.1　实验环境

为了验证本文提出的混合式移动数据同步机制及相关策略的性能，本文在实际的网络环境中构建了实验平台，展开了一系列实验验证，并与基于集中式架构的移动数据同步技术等进行了对比分析各项性能指标。实验平台的软件硬件配置参数如表 4 所示。

表 4　实验平台的软件硬件配置参数

类别	数量	参数
MT	12	OS-Android 4.0 及以上；CPU-单核、双核及四核；RAM-1 GB 及 2 GB
AP	4	150 Mbit/s
SyncServer	1	OS-Windows 8.1；CPU-T6400、2.00 GHz；RAM-4 GB
Server DB	1	MySQL 5.6
Mobile DB	12	SQLite 3

本文采用 5 000 条数据记录进行实验，并用变化记录数比率(PCT, percentage of changed tuples)作为横坐标值来观察各项性能指标。

$$PCT = \left(\frac{m}{n}\right) \times 100\% \tag{18}$$

式(18)中 m 表示变化的记录数，包括经过增加、修改、删除操作的记录数，n 表示记录总数。

5.2　实验结果与性能分析

实验 1　ICSTC、Trigger 和 Snapshot 时间开销对比。

如图 6 所示，ICSTC 和 Trigger 时间开销相近，均远小于 Snapshot。随着 PCT 的增加，ICSTC 和 Trigger 的时间开销均缓慢增长，而 Snapshot 时间开销呈波动变化，且变化幅度较小。由此可以看出，Snapshot 时间开销与增量数据大小无关，而只与记录总数（即总数据量）有关，且时间开销较大，时间复杂度为 $O(n^2)$。Snapshot 需要一张快照表来保存快照信息。Trigger 方法利用触发器捕获变化记录的关键字及其操作类型并保存在一张数据表中，然后根据关键字获取变化记录的最终状态，忽略记录的操作过程，其时间复杂度为 $O(n)$。Trigger 将新增和修改这 2 个操作状态都记录为修改状态，当同步服务器处理增量数据的时候必须区分这 2 种操作状态，增加了同步机制的复杂性。ICSTC 利用触发器捕获变化记录的操作日志并且通过净化算法得到净增量，策略简单易行且效率高，其时间复杂度为 $O(n)$。ICSTC 同样需要额外的空间来保存操作日志。

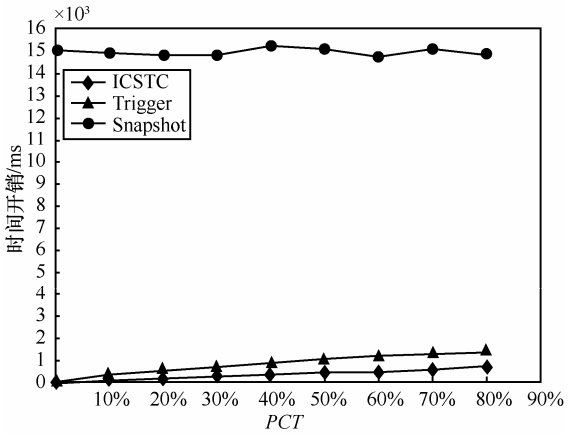

图 6 ICSTC、Trigger 和 Snapshot 时间开销对比

实验 2 混合式、集中式架构的数据同步机制同步响应时间对比。

实验开展了 3 轮：第 1 轮将 12 个移动终端分配到 1 个 SOD 中；第 2 轮将 12 个移动终端平均分配到 2 个 SOD 中；第 3 轮将 12 个移动终端平均分配到 4 个 SOD 中。通过实验对比测试 HDSM 和基于集中式架构的数据同步机制（CDSM, centralized data synchronization mechanism）的同步响应时间的变化情况。

如图 7 所示，HDSM-R_1、HDSM-R_2、HDSM-R_3 分别代表第 1 轮、第 2 轮和第 3 轮的同步响应时间的变化情况。随着 *PCT* 的增加，HDSM 和 CDSM 的同步响应时间均线性递增。其中，HDSM-R_2、HDSM-R_3 和 CDSM 的同步响应时间相差不大，且相对稳定，而 HDSM-R_1 的同步响应时间相对较长。在 HDSM-R_1 中，一个 SOD 中有 12 个移动终端，构成的 SOD 树分为 4 层，按照 CDDS 分发数据无疑增加了 SOD 内部同步数据转发次数和移动终端的负担。由于移动终端能力的限制，SOD 内部移动终端数不宜过多，否则将影响同步响应时间；SOD 内部移动终端数应根据具体应用场景、用户需求和移动终端能力来灵活确定最佳数量。由 HDSM-R_2、HDSM-R_3 和 CDSM 可知，HDSM 同步响应时间比 CDSM 同步响应时间略微长点，最大不超过 2 s，且随着 *PCT* 的增加，HDSM 同步响应时间有短于 CDSM 同步响应时间的趋势。

实验 3 混合式、集中式架构数据同步机制的同步成功率对比。

实验开展了 3 轮：第 1 轮将 12 个移动终端分配到 1 个 SOD 中；第 2 轮将 12 个移动终端平均分配到 2 个 SOD 中；第 3 轮将 12 个移动终端平均分配到 4 个 SOD 中，分别通过实验验证 CDSM 和 HDSM 的同步成功率。

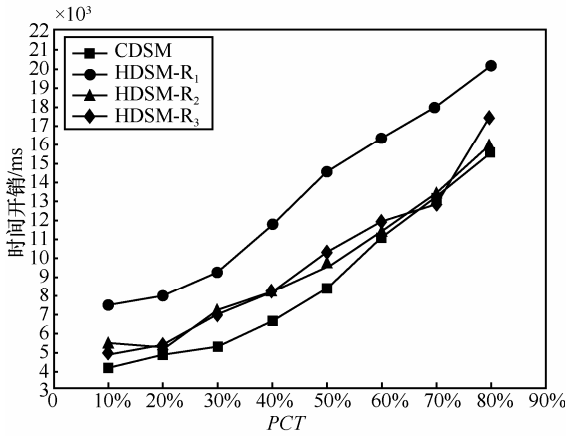

图 7 HDSM 和 CDSM 同步响应时间对比

如图 8 所示，HDSM-R_1、HDSM-R_2 和 HDSM-R_3 分别代表 HDSM 第 1 轮、第 2 轮和第 3 轮实验中的同步成功率的变化情况。CDSM 和 HDSM 的同步成功率随着 *PCT* 的增大均有下降的趋势，即同步数据增量的增大直接延长了系统同步响应时间，从而增加了系统同步成功的不确定因素，直接导致了系统同步成功率的下降，其中，HDSM-R_1 和 CDSM 下降的最为明显，且下降波形幅度变化较大。在 HDSM-R_1 中，虽然 SOD 数量少能减轻同步服务器的开销，但是在移动终端总数一定的情况下，SOD 个数的减少将增加 SOD 内部移动终端的数量。本文提出的 CDDS 按照 SOD 树路径转发并逆向逐跳返回结果，需要限制 SOD 内部移动终端总数，否则将延长同步数据在 SOD 内部转发的时间，从而影响整个系统的性能。HDSM-R_2 和 HDSM-R_3 的同步成功率下降幅度较小且相对稳定，再结合 HDSM-R_1 来看，

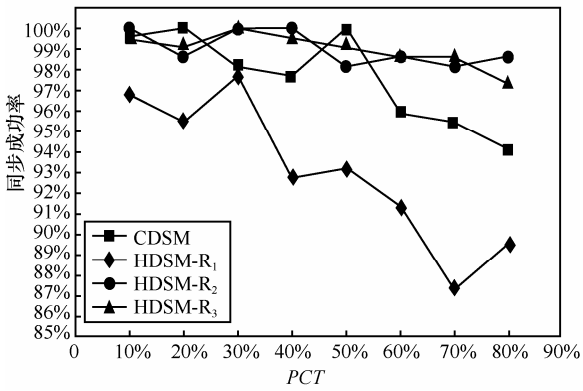

图 8 CDSM 和 HDSM 同步成功率对比

证实了 CDDS 对 SOD 内部移动终端数量的限制要求。由 HDSM-R_2、HDSM-R_3 和 CDSM 曲线可以看出当 SOD 内部移动终端数恰当时，HDSM 的同步成功率比 CDSM 高且稳定。

实验 4　混合式、集中式架构的数据同步机制同步数据通信量对比。

实验开展了 2 轮：第 1 轮将 12 个移动终端随机分配到 2 个 SOD 中；第 2 轮将 12 个移动终端随机分配到 4 个 SOD 中，分别测试 HDSM 与 CDSM 同步数据通信量。

如图 9 所示，HDSM-R_1 和 HDSM-R_2 分别代表第 1 轮和第 2 轮同步数据通信量变化情况。HDSM 同步数据通信量和 CDSM 同步数据通信量都随着 *PCT* 的增大而逐渐增多，但 CDSM 同步数据通信量远大于 HDSM 同步数据通信量。由 HDSM-R_1 和 HDSM-R_2 的通信量可知，当移动终端总数一定时，SOD 数量越少，则同步数据通信量越少，同步服务器的开销也越小。但 SOD 个数减少的同时，SOD 内部移动终端数增多，容易影响 SOD 内部同步数据的传输时间，从而影响整个同步进程的同步响应时间。在实际应用中，移动终端所处的 SOD 数量可由移动终端相对位置确定。实验结果表明 HDSM 比 CDSM 能更好地降低同步数据通信量。

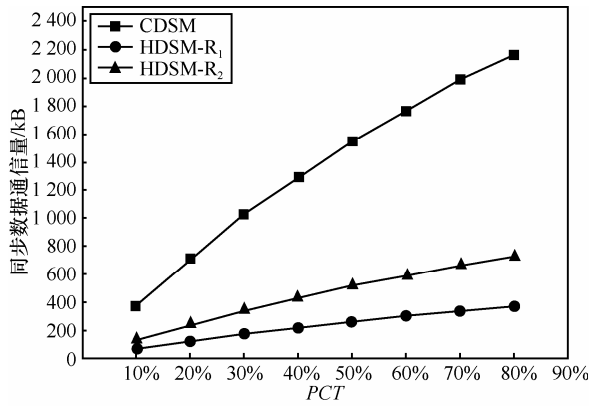

图 9　CDSM 和 HDSM 同步数据通信量对比

实验 5　混合式、集中式架构数据同步机制的 CPU 和 RAM 开销对比。

本实验以 *PCT* 为 50%的数据集作为实验数据集，同时将 12 个移动终端随机分配到 4 个 SOD 中，以同步服务器启动时刻为零时刻，分别测出 CDSM 和 HDSM 在整个数据同步过程中的 CPU 和 RAM 的使用率。

如图 10 所示，同步进程启动时刻是 2.5 s，CDSM 和 HDSM 的同步进程结束时刻分别是 12 s 和 11.5 s。从结束时间来看，CDSM 和 HDSM 同步响应时间相差不大。同步服务器启动时，CPU 消耗较高，随后下降为相对平稳状态。当同步进程启动后，HDSM 的 CPU 利用率整体上要小于 CDSM 且波动幅度相对平稳，在整个同步进程中，HDSM 比 CDSM 节省同步服务器的 CPU 开销。

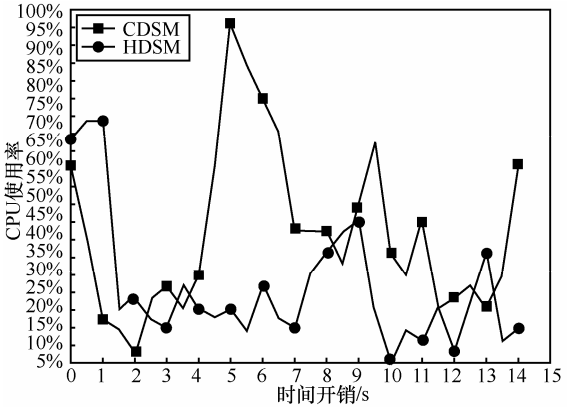

图 10　CDSM 和 HDSM 的 CPU 使用率对比

如图 11 所示，在同步进程中，CDSM 和 HDSM 对 RAM 资源的消耗相近，都在 50%~55%范围内。CDSM 和 HDSM 的 RAM 使用率都几乎成直线，说明两者在整个系统同步进程中对内存的使用均比较稳定。

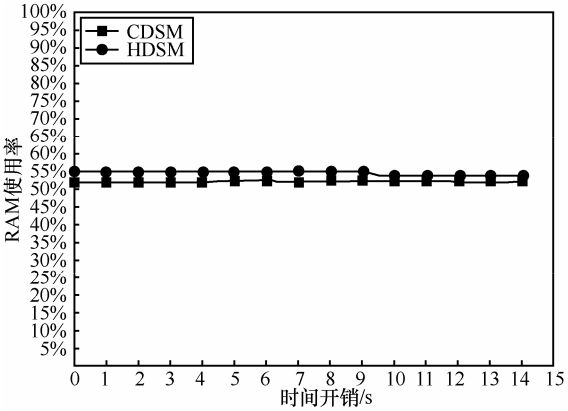

图 11　CDSM 和 HDSM 的 RAM 使用率对比

6　结束语

本文面向移动计算环境提出了一种混合式数据同步机制 HDSM，以 SOD 为单位来管理同步进程，还提出了配套的数据分发策略 CDDS 和增量捕获策略 ICSTC，不仅实现高效的数据同步，维护移动数据的一致性，而且节省了数据同步通信量，减

少同步通信费用，降低同步服务器开销。HDSM 的真正实际应用还需要进一步解决以下问题：1) 数据冲突处理，HDSM 没有给出移动数据同步过程中出现的数据冲突处理策略，该策略也是移动数据同步机制中的一个重要研究方向；2) 单用户多终端，HDSM 在处理数据同步时没有将单用户多终端考虑在内，而在移动计算和移动终端日益普及的今天，HDSM 必须要考虑这个问题，以进一步提高系统性能；3) 移动终端数量限制，HDSM 对于 SOD 内部移动终端数量有限制，数量的增长将影响同步响应时间，后续需要进一步优化算法和结构，在 SOD 中容纳更多的移动终端，提升机制性能。

参考文献：

[1] LI D W, HUANG W J, HU J H, et al. A distributed redundant real-time data storage mechanism[J]. Journal of Shanghai Jiaotong University, 2014, 48(7): 948- 952.

[2] BOUAJJANI A, ENEA C, HAMZA J. Verifying eventual consistency of optimistic replication systems[C]//The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. San Diego, CA, c2014: 285- 296.

[3] YAN F, RISKA A, SMIRNI E. Fast eventual consistency with performance guarantees for distributed storage[C]//The 32nd International Conference on Distributed Computing Systems Workshops. Macau, c2012: 23-28.

[4] FENG J L, QIAO X Q , LI Y. The research of synchronization and consistency of data in mobile environment[C]//The 2nd IEEE International Conference on Cloud Computing and Intelligent Systems. Hangzhou, c2012: 869-874.

[5] CHANG B J, CHEN C S, LIANG Y H, et al. A distributed ad hoc network based on increasing reliability and scalability for internet applications[C]//The 2006 International Conference on Wireless Communications and Mobile Computing. New York, c2006: 1453-1458.

[6] MAO H X, HUANG K, SHU X L. Research of cloud storage and data consistency strategies based on replica redundant technology[C]//The 2014 International Conference on Computer, Intelligent Computing and Education Technology. Hong Kong, c2014: 1053-1056.

[7] ITANI Z, DIAB H, ARTAIL H. Efficient pull based replication and synchronization for mobile databases[C]//The 2005 International Conference on Pervasive Services. IEEE, c2005: 401-404.

[8] AJILA S A, AL-ASAAD A. Mobile databases-synchronization & conflict resolution strategies using SQL server[C]//The 2011 IEEE International Conference on Information Reuse and Integration. Las Vegas, c2011: 487-489.

[9] PASCUAL V S, XHAFA F. Evaluation of contact synchronization algorithms for the Android platform[J]. Mathematical and Computer Modelling, 2013, 57(11): 2895-2903.

[10] HAO Y J, YAN C. Design and implementation of Android contacts synchronization system based the SyncML protocol[C]//The 5th International Conference on Intelligent Networking and Collaborative Systems. Xian, c2013: 747-750.

[11] LIN S, LI Y, HE J, et al. Synchronization research of data based on SyncML and Huffman Coding[J]. Information Technology and Industrial Engineering (Set), 2013, 48: 241-247.

[12] LI J, LI J. Data synchronization protocol in mobile computing environment using SyncML and Huffman coding[C]//The 2012 International Conference on Wavelet Active Media Technology and Information Processing. Chengdu, c2012: 260-262.

[13] XHAFA F. Data replication and synchronization in ad hoc collaborative systems[C]//The 26th IEEE International Conference on Advanced Information Networking and Applications. Fukuoka, c2012.

[14] CHEN X, REN S, WANG H, et al. Scope: scalable consistency maintenance in structured ad hoc systems[C]//INFOCOM 2005. San Diego, CA, c2005: 1502-1513.

[15] KUBIATOWICZ J, BINDEL D, CHEN Y, et al. Oceanstore: an architecture for global-scale persistent storage[J]. ACM Sigplan Notices, 2000, 35(11): 190-201.

[16] LI Z, XIE G, LI Z. Efficient and scalable consistency maintenance for heterogeneous peer-to-peer systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2008, 19(12): 1695-1708.

[17] YU C, TAN G, YU Y. Make driver agent more reserved: an aim-based incremental data synchronization policy[C]//The 9th IEEE International Conference on Mobile ad hoc and Sensor Networks. Dalian, c2013: 198-205.

[18] ARDEKANI M S, SUTRA P, SHAPIRO M, et al. On the scalability of snapshot isolation: Euro-par 2013 parallel processing [M]. Berlin: Springer Berlin Heidelberg, 2013.

[19] WANG W J. Conservative snapshot-based actor garbage collection for distributed mobile actor systems[J]. Telecommunication Systems, 2013, 52(2): 647-660.

[20] YANG G. Data synchronization for integration systems based on trigger[C]//The 2nd IEEE International Conference on Signal Processing Systems. Dalian, c2010:310-312.

[21] HU Y, DESSLOCH S. Extracting deltas from column oriented NoSQL databases for different incremental applications and diverse data targets[J]. Data & Knowledge Engineering, 2014, 93: 42-59.

[22] YUN Z, MU Z, FULING B. A tiered replication model in embedded database based mobile geospatial information service[C]//The 4th IEEE International Conference on Wireless Communications, Networking and Mobile Computing. Dalian, c2008: 1-4.

[23] HU Y, FENG M, BHUYAN L N. A balanced consistency maintenance protocol for structured ad hoc systems[C]//INFOCOM 2010. San Diego, CA, c2010: 1-5.

[24] DAFEI Y, BIN C, ZHOU H, et al. Replication strategy in peer-to-peer geospatial data grid[C]//The 2007 IEEE International Geoscience and Remote Sensing Symposium. Barcelona, c2007: 5013-5016.

[25] CHEN J, FAN J, SUN Y. Data dissemination and query in mobile social networks[M]. Berlin: Springer, 2012.

作者简介：

徐小龙（1977-），男，江苏盐城人，博士，南京邮电大学教授，主要研究方向为计算机软件、网络计算、信息安全、agent 技术等。

刘笑笑（1988-），男，江苏宿迁人，南京邮电大学硕士生，主要研究方向为移动计算与数据同步技术等。