# 移动应用程序的可靠，一致和高效的数据同步

学号：2017282110286

姓名：吴政楠

# 移动应用程序的可靠，一致和高效的数据同步

## 摘要

移动应用程序通常需要跨设备的去管理数据，为用户提供无缝存储、访问、协作和脱机编辑等多种功能。为了使这样做可靠，应用程序必须预见和处理大量的保持数据一致性的本地和网络故障。对于移动环境，节约使用蜂窝带宽和设备电池也是必不可少的。上面的要求为应用开发者带来了巨大的负担。我们建立了Simba，为移动应用程序开发人员提供一个高层次的统一表格和对象的局部编程抽象的数据同步服务——移动应用程序的共同需求——以可靠、一致和高效的方式透明处理数据的存储和同步。在本文中，我们提出了Simba的客户端软件作为数据同步基础设施的网关行为的详细描述。我们的评价显示了Simba在所有故障场景下一致的快速发展的健壮的移动应用程序的有效性，不同于Dropbox开发的应用程序。Simba也确实能节约使用蜂窝带宽。

## 1 引言

个人智能设备已无处不在并且用户现在可以享受在这上面的各种各样的应用程序。许多这样的应用程序是以数据为中心，这些数据经常依赖于基于云的资源来存储，共享和分析数据。除了用户界面和各种功能，开发这样一个应用程序开发者需要构建基础数据管理设施。

例如，为了提供一个高质量的笔记软件如Evernote，开发者必须建立一个支持丰富多媒体注释，数据和元数据的查询，协作，以及脱机操作的数据管理平台，同时确保在面对失败时的可靠性和一致性。此外，移动应用程序开发人员在需要满足上述要求的同时有效地利用移动设备上有限的资源，比如蜂窝带宽和电池电量。开发人员越好的处理上述问题，应用程序吸引和留住用户的可能性越大。

随着市场上的应用程序的数量和种类的迅速增长，随之而来的，有一个来自高级抽象的从业者的需求，在管理数据时隐藏复杂性和简化应用程序开发人员的各种任务。

数据同步服务已经成为开发人员的辅助工具，应用程序可以卸载它对第三方服务，如Dropbox，iCloud，或谷歌驱动的一些数据管理。起初这种服务迎合了

希望跨多个设备访问文件的最终用户，最近这种服务通过直接使用CRUD（创建，读取，更新，删除）操作为App提供的SDK。同步服务建立在数十年的分布式和移动数据同步的研究——从断开操作的基础工作，弱连接复制存储和版本管理，到最近的广域数据库复制，协作编辑以及移动设备的缓存研究工作。

数据同步的原理与机制大家都明白，我们不想重新发明，但是，数据同步服务需要实现双重性使其实现对移动应用程序的价值。首先,它必须在很少涉及的应用程序开发者的情况下，透明地处理可靠性、一致性和效率的问题，这是具有挑战性的。Dropbox的制造者还提出，为外部用户提供简单性的背后可能需要巨大的复杂性和努力。其次，数据同步服务必须提供对大多数App有益的数据模型。

虽然文件同步是常见的，实际上很多App都是操作跨结构和非结构化数据。包含表格和文件的高级数据模型对应用程序开发人员非常有用，并且透明度必须适用于此数据模型。

数据同步服务必须保存，代表应用程序，数据在存在故障时被存储和共享时结构化和非结构化数据之间的一致性。考虑到照片共享应用程序的例子

Picasa和Instagram；通常这样的应用程序将专辑信息存储在表中，并将实际图像存储在文件系统或对象存储中。在这种情况下，同步服务需要确保永远不会有从专辑到图像悬摆指针。由于移动应用可能会崩溃或由于各种原因频繁停顿，如果应用程序是在数据操作的中间（本地写或同步）发生故障，同步服务需要可靠地检测并恢复到一致状态。最近的工作已经表明，当与桌面文件系统一起使用时，几个数据同步服务也传播腐败数据。虽然服务已经在文件和表数据中存在，但都不符合上述标准。

为了更好地了解移动应用和同步服务如何在故障时保持数据一致性，我们进行了针对Android的流行移动应用程序的研究包括那些使用Dropbox，Parse和Kinvey来进行数据同步的应用程序。我们的研究表明，应用程序在数据丢失，腐败和不一致等方面管理数据不佳。

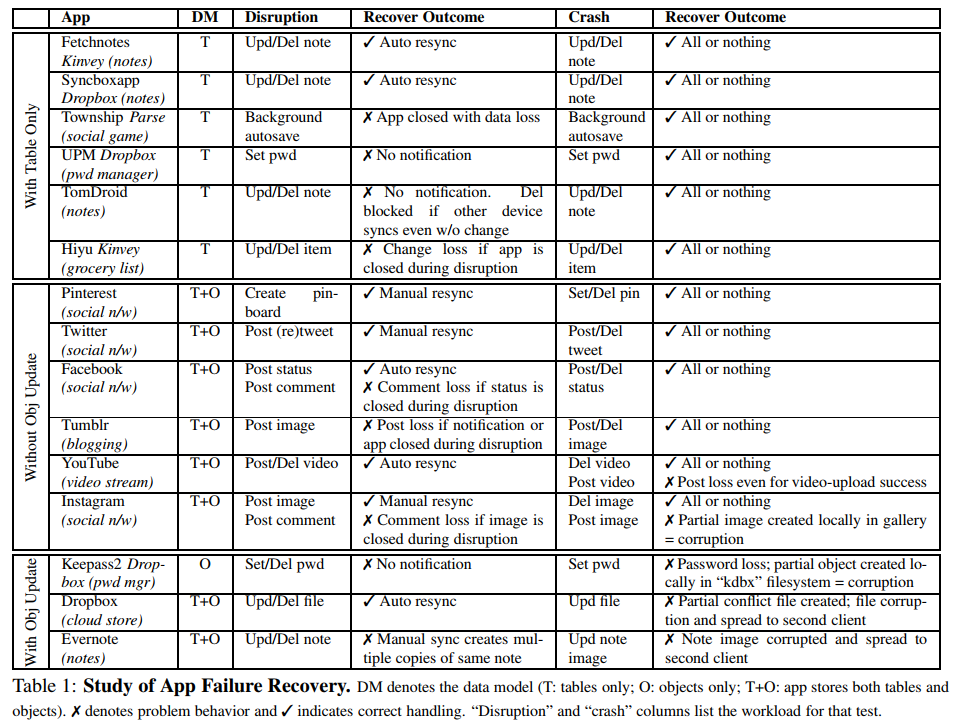
因此，我们建立了Simba来管理移动应用程序的数据，它提供了高级抽象来统一文件和表。这些表可能包含两个基本类型的列（字符串，整数等）和任意大小的对象，全部通过类似于CRUD的界面访问。为了方便调用，接口和那些已经被开发人员熟悉的iOS和Android接口保持类似。应用可以构建跨越表和对象的数据模型并且Simba确保所有数据的可靠性，一致性并且确保数据与服务器和其他移动设备的有效同步。

Simba由用于开发移动应用程序的SDK，用于移动设备的Simba客户端应用程序（sClient），以及Simba云服务器（sCloud）组成，所有用SimbaSDK，Simba—apps写的App，只与作为所有和sCloud交互代理的sClient的本地实例通信。在本文中，因为高级抽象透明度对Simba的影响，我们主要关注它，也因此我们主要讨论sClient；整个Simba服务在其他地方有更详细的介绍。

通过案例研究，我们展示了Simba如何让我们快速开发几款移动应用程序，显著增加开发的便利性和功能性。Simba从sClient的失败透明度中大大受益；使用Dropbox编写的应用程序无法保持整个数据对象原子性，导致在失败的情况下破坏更新和同步不一致的数据。Simba能够以编程方式整合容迟数据传输，Simba也展示出减少网络足迹和给设备增加关闭蜂窝无线电的机会的能力。

## 2 移动App可靠性的研究

我们通过系统地介绍故障——网络中断，本地应用程序崩溃和设备功率损失——研究了一些流行的移动应用的可靠性和同步服务（在Android上）并观察恢复结果，如果有的话。我们研究中的应用程序使用表和文件/对象，并依靠各种现有的服务，即Dropbox，Parse和Kinvey进行数据同步。我们设置了两个具有相同应用和初始状态的Android设备。为了模拟网络中断，我们启动了飞行模式和崩溃（1）手动停止应用程序，（2）取出电池；两次崩溃测试的结果没有差异，我们将其列出一次，如表1所示



对于网络中断测试，如果没有在重新连接后立即处理同步失败，一些应用程序（例如Hiyu，Tumblr）将会造成数据丢失。如果应用程序（或该通知）已经关闭，直到重启时才会恢复。有些应用程序（UPM，TomDroid，Keepass2）甚至没有通知用户同步失败。大多数应用程序都需要用户在同步失败后手动重新同步，这个疏忽导致数据永久处于待同步状态。一些应用程序表现出其他形式的不一致。对于TomDroid，如果第二个设备在没有变化的情况下与服务器保持同步，则删除操作将无限期地阻塞。对于Evernote，中断后手动再同步会创建同一个注释的多个副本。

对于崩溃测试，只在桌面的应用程序才能正确恢复，因为他们的崩溃一致性完全依赖于SQLite。但是，带有对象的应用程序会表现出有问题的行为，包括腐败和不一致。对于YouTube，即使对象（视频）已成功上传，该应用程序本身丢失了帖子。Instagram和Keepass2都创建了一个本地的局部对象；此外Keepass2未能恢复桌面数据导致出现指向对象的悬挂指针。Dropbox创建了一个带有局部对象（局部损坏）的冲突文件并将腐败传播到类似Evernote的第二个设备。

尽管有大量的先前研究的数据，分析工具和数据同步服务。首先，处理对象对大多数应用程序尤其有问题——在我们的研究中，没有应用程序能够在对象更新期间正确地从崩溃中恢复。其次，一些应用程序采用了更简单的方法来禁用对象一起更新，而不是确保正确的恢复。第三，在某些情况下，应用程序无法通知用户错误，导致进一步腐败。进一步的研究激励我们采取一种全面的方法来透明地处理数据同步服务中的故障并为应用程序提供一个有用的高级抽象。

## 3 Simba的应用程序开发

### 3.1 数据模型和API

数据模型：Simba的数据模型是是这样设计的，应用程序可以将其所有数据存储在一个统一的存储中而不必担心如何存储和同步。该高级抽象使应用程序有一个生成称为Simba表（简称sTable）的表和对象的数据模型。为了支持这个统一的数据管理视图，Simba，在后台，确保应用程序永远能在云和其他移动设备查看与本地存储数据一致的视图。

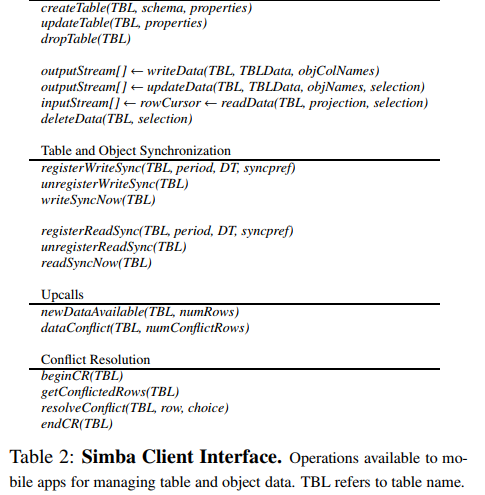
Simba中客户端—服务器一致性的单位是sTable（简称sRow）中包含表格数据及其中涉及的所有对象的单独的一列。对象不是通过sRows共享。Simba提供因果一致性，在sRow上使用全或无原子性的语义进行本地和同步操作；这是一个比现有的同步服务更强有力的保证。一个应用程序可以在Simba的支持下

有一个只有表格或只有对象的模式。由于sRow代表一个更高级的，语义上有意义的应用程序数据单元，确保它在所有情况下的一致性对开发者是非常有价值的并可以让她从复杂的事务管理和恢复代码中解放出来。图像1显示了Simba的数据模型。

Simba目前不为sRows或sTables提供原子同步。虽然有些应用程序可能会从原子多行同步中受益，我们最初的经验已经表明对于整个表同步的ACID语义会不必要的复杂化Simba设计，导致更高的性能开销，并为大多数应用程序的矫枉过正。

API：sClient的API，如表2所示，类似于流行的CRUD界面，但有四个附加功能：1）表格和对象的CRUD操作2）操作注册表进行同步3）上传新数据和

冲突4）内置冲突检测和对解决方案的支持。对象使用允许Simba支持大型对象的抽象流写入或读取；它也允许本地阅读或写一个大的对象的一部分——一个在关系数据库中BLOB（二进制大型对象）不可用的属性。



由于不同的应用可以有不同的同步需求，Simba支持由应用程序开发人员使用同步方法（registerWriteSync等）控制的按表同步策略。每个sTable可以指定一个非零的周期，这决定了变更收集同步的频率。延迟容限（DT）值可以是特定的，这为在通过网络发送之前跨应用程序合并的数据提供了额外的机会。对于延迟敏感的数据，DT可以设置为零。甚至当应用程序有不一致的时期，DT为更好地利用蜂窝无线电启用crossapp使流量一致。如果应用程序需要按需同步数据，它可以使用writeSyncNow（）和readSyncNow（）方法。Simba的延迟容忍传输机制直接受益于以前的工作。由于同步发生的背景，新数据由于同步可用或发生冲突，应用程序使用upcalls进行通知。应用程序可以随意地开始和结束冲突解决事务并且迭代冲突的行可以解决任一本地复制，服务器副本或全新的选择。

### 3.2 写一个Simba应用程序

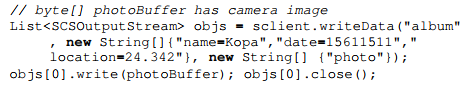
Simba的统一API简化了应用程序的数据管理。这可能可以用一个例子最好的展现。我们考虑一个照片共享应用程序，它存储和定期同步的图像，以及他们的名字，日期和地点。首先，通过指定其模式来创建一个sTable：



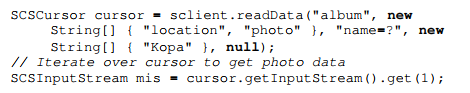
接下来，注册阅读（下载）和写入（上传）同步。应用程序每10分钟（600s）同步一次照片，DT读取和写入1分钟（60s），选择用于写入的WiFi，并允许3G进行读取同步。



一张照片可以用writedata()方法通过向输出流写入被添加到表之后。



最后，可以使用查询检索照片：



## 4 Simba设计

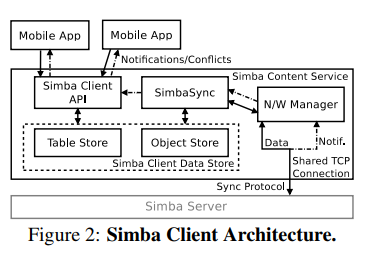
### 4.1 Simba服务器（sCloud）

该服务器是一个可扩展的云存储，管理多个应用程序，表和客户端的数据。它提供了一个用于数据同步的网络协议，基于sClient的职责来代表所有本地设备的Simba应用程序从服务器获取更新并推送任何本地修改；sClient可以向服务器注册以通知订购表的改变。

同步协议：为了讨论sClient的设计，我们需要通过网络协议来引用服务器提供的语义。服务器被期望提供持久性，行更新的原子性和多版本并发控制。因此，sClient被暴露给与服务器交换的消息中的任何数据的版本。Simba实现了一个版本向量的变体，它提供了因果一致性语义的并发控制。由于所有的客户端都同步到一个中央的sCloud，所以我们简化了版本控制方案，使每行有一个版本号，而不是一个向量。每个sRow都有一个由主键生成的唯一标识符IDrow（如果存在的话），或者是一个随机的版本Vrow。

行版本在每次更新行时都会在服务器上增加；表格中最大的行版本保持为表格版本Vtable，使我们能够快速识别哪些行需要同步。在八卦协议中使用类似的方案。由于Simba支持可变大小的可能较大的对象，因此协议消息明确标识了需要以原子方式应用的对象的部分更改的集合。

sClient允许联网的Simba应用程序继续拥有一个本地I/O模型，该模型显示出更容易编程；sClient使应用程序免受服务器和网络中断的影响，并提供更好的整体用户体验。图2显示了sClient的简化架构；（1）为所有Simba应用程序提供对其表和对象数据的访问权限（2）管理一个本地设备副本以启用断开连接的操作（3）确保容错性，数据一致性和行级原子性（4）通过网络执行所有同步相关的操作。Simba应用程序通过提供Simba客户端接口（表2）并将客户端操作转发给sClient的轻量级库（sClientLib）与sClient链接；通过上传提醒应用程序发生在后台发生的事件（例如新数据，冲突）。最后，sClient监视应用程序的活跃性，以便在应用程序崩溃的情况下释放内存资源。



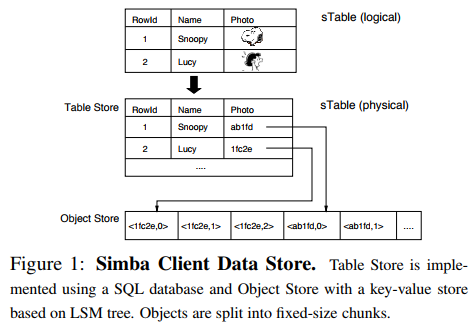
sClient数据存储（§4.2.1）提供了对表存储和对象存储的统一抽象。SimbaSync使用sCloud执行同步处理（第4.2.2节）；对于上游同步，它收集本地修改的数据，对于下游同步，它将从服务器获取的更改应用到本地存储，检测冲突并生成应用程序上传。同步协议和本地数据存储一起为所有Simba应用程序提供透明的故障处理（§5）。网络管理器处理sClient的所有网络连接和服务器通知（§6）；它通过合并和延迟容忍来提供设备蜂窝无线电的有效利用。

实现：sClient目前在Android上实现，但是，设计原则可以应用到其他移动平台，如iOS。sClient被作为一个叫做Simba内容服务（SCS）的守护进程来实现，由移动应用程序通过本地RPC访问；在Android上，我们使用AIDL接口在应用程序和服务之间进行通信。另一种方法—直接与应用程序链接—之后是Dropbox和Parse，但我们的方法允许sClient为同一设备上的所有Simba应用程序构建网络流量，从而受益于多个跨应用程序优化。尽管使用持久连接的好处早已为人所知，但各个应用程序使用TCP连接的次优方式来建立和拆卸频繁的连接。sClient的设计允许它代表多个应用程序使用一个持久的TCP连接到sCloud；同样的连接也被服务器重复使用，以提供通知，提供额外的节省，类似于Thialfi。

行为不当的应用程序可能会对其他Simba应用程序产生不利影响。在实践中，我们认为开发者已经有动力编写出色的应用程序来让用户满意。在未来，Simba可以建立类似于Android会计的细粒度数据会计，进一步阻止这种行为。

## 4.2.1 Simba客户端数据存储

sClient数据存储（SDS）负责将应用程序数据存储在移动设备的永久性存储（通常是内部闪存或外部SD卡）上。对于Simba应用程序，这意味着能够以逻辑统一的方式存储表格数据和对象。SDS的主要设计目标是启用和有效支持sRows上的CRUD操作；这要求商店支持本地数据的原子更新。此外，因为对象是变量化的，而且可能很大，所以商店也需要支持这些对象的原子同步。由于商店持续存储所有本地修改，所以它必须有效支持的频繁查询是上行同步的更改检测；SDS应该能够快速确定子对象的变化。图1显示了SDS数据布局。



对象被细分为固定大小的块，并存储在支持范围查询的键值存储（KVS）中。随机写入的优化对于移动应用程序来说是非常重要的，KVS的选择受到对附加和覆盖的良好吞吐量的需求的影响。每个块被存储为KV对，关键字是<object\_id，chunk\_number>元组。通过查找对象的第一个块并按键顺序迭代KVS来访问对象的数据。

本地状态：sClient保持附加的本地状态，持久性和易失性，用于同步和失败处理。两个持续的每行标志FlagTD（表脏）和FlagOD（对象脏）用于标识上游同步所需的本地修改的数据。为了防止部分对象同步，我们为每一行CountOO维护打开的更新对象的数量。当一个行的写事务被关闭时，它被认为是关闭的。每一行都有两个持久标志FlagSP（syncpending）和FlagCF（conflict），它们跟踪当前的同步状态。最后，内存中的脏块表（DCT）会跟踪已本地修改但尚未同步的块。这避免了在正常操作期间向商店查询这些改变的需要。

实现：我们利用SQLite来实现表格存储，并使用表示对象标识符（objectID）的附加数据类型。对象存储是使用LevelDB实现的，该数据库是基于日志结构合并（LSM）树的KVS。LevelDB满足本地附加和更新的吞吐量标准。LevelDB还具有快照功能，可用于原子同步。Android的LevelDB没有本地端口，因此我们使用Android的本地开发套件（NDK）移植原始的C++LevelDB代码。我们使用LevelDB的一个实例来保留所有表的对象，以确保顺序写入以获得更好的本地性能。由于本地状态存储在sRow的表格部分，SQLite确保了它的一致性更新。

### 4.2.2 同步处理

一个sClient独立执行上行和下行同步。上游同步是基于各个表的指定周期启动的，并且使用（FlagTD，FlagOD）中维护的本地状态来确定脏行数据；这些标志在数据收集时被重置。对于具有脏对象的行，块将逐个读取并直接打包到网络消息中。

由于收集脏数据并将其同步到服务器可能需要很长时间，因此我们使用以下技术来允许前台应用程序进行并发操作。首先，sClient从当前版本的LevelDB快照中收集对象修改。由于sClient只在关闭修改后的对象并更新本地状态（将CountOO递减1）时才同步修改后的对象，因此sClient始终确保快照上sRows的一致视图。其次，我们允许sClient在之前的同步操作正在进行时继续进行修改；如果客户端断开并且同步处于延长的持续时间，这是特别有利的。这些更改将sRow的本地标志FlagTD或FlagOD设置为在随后的同步过程中进行收集。为此，sClient维护一个同步待处理标志FlagSP，一旦收集到更改，就为脏行设置，并在服务器指示成功后重置。如果另一个同步操作在前一个同步操作完成之前开始，则已经设置了FlagSP的行将被忽略。

下游同步也由sClient响应服务器对表的更改通知而启动。客户端拉取所有具有比本地Vtable更大版本的行，暂存下游数据，直到接收到所有行的数据块，然后按照增加的Vrow顺序逐行将数据应用到sClient数据存储区。

上游同步的冲突是通过服务器上的Vrow不匹配确定的，而下游则通过检查接收行的本地脏标志来确定。为了使应用程序自动解析或向其用户展示，服务器返回的冲突数据由sClient在本地进行，并通知相关的Simba—app。sClient旨在优雅地处理冲突的更新。冲突的行被标记（FlagCF），以防止进一步的上游同步，直到解决冲突。但是，应用程序可以根据自己的方便来解决冲突，并且可以继续读取和写入本地版本的行而不同步。我们相信，这大大提高了用户体验，因为应用程序在出现冲突时不必突然中断操作。

## 5 透明故障处理

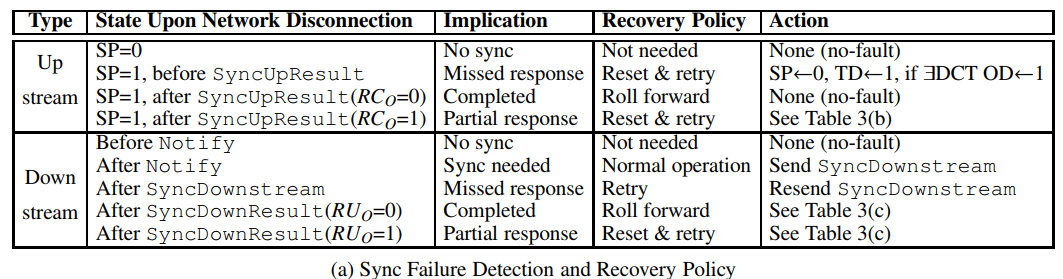
移动应用运行在拥塞的蜂窝网络，网络中断，频繁的服务和应用程序崩溃[10]以及电池损耗的环境下。移动操作系统的内存管理也可以积极破坏应用程序。

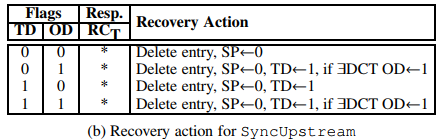
故障透明是sClient的关键设计目标，它通过三个相互关联的方面来实现。首先，机制是全面的：系统检测每种可能的故障类型，恢复使系统处于一个明确定义的状态。其次，恢复不仅仅是一个已知的状态，而是一个按照统一的数据模型服从高级一致性的系统。第三，sClient在交换可用性和恢复成本方面是明智的（在移动环境中本身可能是令人望而却步的）。除了一些优化（在§5.2中讨论），一个sClient维护足够的本地元数据，以避免为了恢复的目的向服务器分配状态。客户端是有状态的，因为它允许同步服务（拥有许多经常出现故障的移动客户端）和中央服务器来解除故障恢复，从而提高可用性。

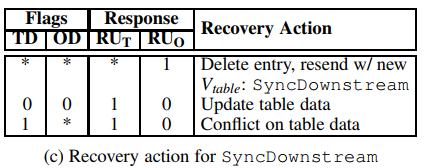
### 5.1 全面和高层次一致

sClient旨在全面处理故障，并使用状态机。每个成功的操作都将sClient从一个定义好的状态转换到另一个状态。不同类型的故障导致不同的故障状态，每个故障状态具有明确的恢复。

我们首先讨论仅影响同步操作的网络故障。如前所述，服务器对上行同步的响应可以指示成功或冲突，下行同步可以指示成功或未完成。表3（a）根据本地同步挂起状态（FlagSP）和相关的服务器响应（RCO，RCT，RUO，RUT）描述了sClient的状态。请注意，只有一部分响应可能与任何给定状态有关。在网络断开之后，对于上游或下游同步，每个唯一状态代表无故障或故障情况；对于后者，恢复政策和行动是指定的客户端。表3（b）和（c）分别规定了在上行和下行同步过程中故障的恢复动作。具体的动作是根据本地数据的脏状态和服务器响应的组合来确定的。

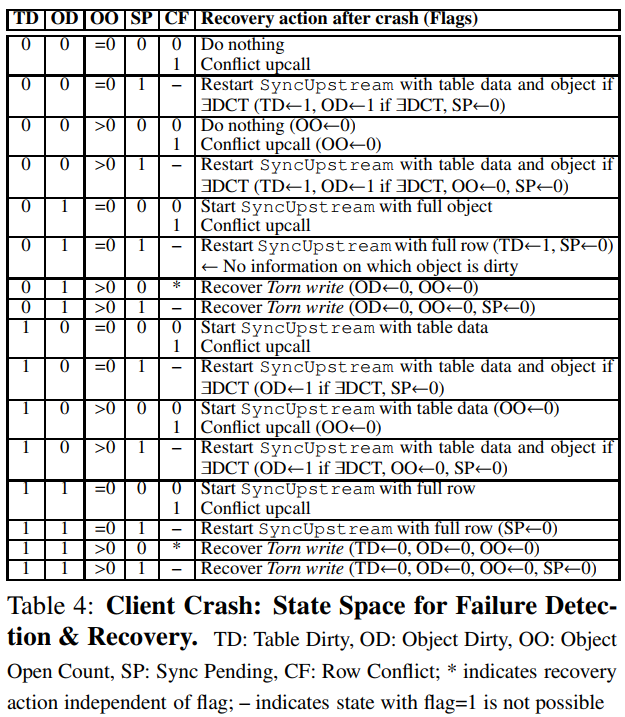






崩溃会影响同步和本地操作，无论sClient，Simba—app还是设备崩溃，SDS的状态都是一样的。sClient通过侦听器上的信号检测Simba—app崩溃，并为应用程序取消分配内存资源。表4显示了在发生崩溃后sClient重新启动时采取的恢复操作；对于Simba应用程序崩溃，恢复发生在其重新启动。

sClient处理网络故障和崩溃，同时保持sRows的全有或全无更新语义—在所有情况下，状态机指定恢复操作，保留表格和对象数据的原子性——从而确保应用程序的高级统一视图；这是sClient对移动应用程序失败透明度的重要价值主张。如表4所示，当一个对象既脏又打开（FlagOD=1＆CountOO>0）时，崩溃可能导致行不一致，即写操作被破坏。类似地，在对象同步期间的网络中断可以导致部分同步；sClient检测并启动适当的恢复。



### 5.2 审慎

客户端平衡竞争需求：一方面，正常运行应该是有效的；另一方面，故障恢复应该是透明和便宜的。sClient保持持久状态，从本地检测并从大多数故障中恢复；对于破损的行，在本地检测之后，通过服务器协助可以有效恢复。要使恢复成本保持在较低水平，必须进行两种折衷。

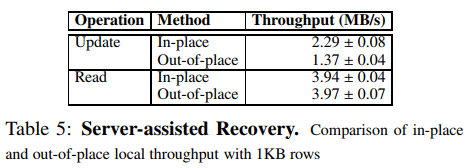
### 5.2.1 权衡：本地状态与网络I/O

当sClient从崩溃中恢复时，它可以使用FlagOD识别对象是否脏，但无法确定是完全还是部分写入永久存储；后者将需要恢复。CountOO计数器可以做出这个决定：如果它设置为零，sClient可以确保本地数据是一致的，避免使用服务器的恢复被破坏。sClient的成本是每行4字节的额外状态。但是，仍然存在一个问题：要同步这个对象，sClient仍然需要识别脏块。内存中的DCT将在崩溃后丢失，并强制sClient从服务器获取所有块，或者将所有块发送到服务器，以便逐块比较。因此，在启动同步之前，sClient支付持续DCT的小成本，以防止重新同步整个可能大的对象。一旦持续存在，DCT将用于在崩溃后同步脏块并删除恢复后的内容。如果在DCT写入磁盘之前sClient崩溃，它将发送脏对象的所有块。

### 5.2.2 权衡：本地I/O与网络I/O

如果一个对象在崩溃后有一个非零的CountOO，它确实是被撕裂的。恢复破损行的最明显的方法是不要在SDS中就地更新数据，而是总是先写出来，一旦数据成功写入，就可以将其复制到类似于预先写入日志或日志记录的最终位置。在这种情况下，sClient不是在普通操作期间支付开销，而是从Simba获得帮助。

在任何时候，Simba都有一致的观点，客户依靠这个观察来回滚或前滚到一致的状态。如果sClient在恢复期间检测到本地撕裂的行，则从服务器获取该行的一致版本；这类似于中止数据库事务的回滚。如果服务器已经取得了进展—客户本质上在前进。如果客户端断开连接，恢复将无法继续进行，但也不会阻止正常操作—只有被撕裂的行不能用于本地更新。为了比较，我们也实施了一个不合适的SDS。如表5所示，sClient能够在就地更新的情况下实现69％的更高吞吐量，而不是用于更新具有1KB对象的行的就地更新。



## 6 透明的网络效率

Simba同步是为了通过一个定制的网络协议，通过两种优化来明智地使用蜂窝带宽和设备电池：

延迟容忍和合并：通常，许多应用程序作为服务在后台运行，例如发送/接收电子邮件，更新天气，同步RSS源和新闻，以及更新社交网络。sClient被设计为一种设备范围的服务，因此多个独立应用程序的同步数据可以一起管理，并通过共享的持续TCP连接进行传输。此外，Simba支持可以在每个表格上进行控制的延迟容忍数据调度。延迟容忍和合并有两个好处。1）改进的网络覆盖范围：允许数据传输集群，减少网络活动，并提高设备关闭无线电的几率。来自服务器的控制消息受同样的措施影响。2）改进了数据压缩的范围：多个应用程序的传出数据被合并以提高压缩率。

细粒度变化检测：如果只有一个部分发生变化，则不需要同步整个对象。即使数据是按行进行版本控制的，sClient也会保持内部软状态（DCT）以检测可配置块级别的对象更改；Simba服务器对下游同步也是一样的。

实施：即使sRows是逻辑同步单元，sClient的网络管理器也会将网络消息与来自多行，跨多个表和应用程序的数据一起打包，以减少网络占用空间。Simba的网络协议是通过使用Protobufs来实现的，该Protobufs有效地对结构化数据进行编码，而TLS则用于安全的网络通信。目前的原型使用双向SSL认证与客户端和服务器证书。

## 7 评估

我们希望回答以下两个问题：

* Simba是否向应用程序提供失败透明度？
* Simba在同步和本地I/O方面表现不错吗？

我们使用三星GalaxyNexus手机和一台运行Android4.2的AsusNexus7平板电脑交换使用sClient。WiFi测试是在WPA安全的WiFi网络上进行的，而在4GLTE上进行蜂窝测试：韩国的KT和LGU+以及美国的AT＆T。我们的原型sCloud使用8个虚拟机进行设置，这些虚拟机均匀分布在2个IntelXeon服务器上，每个服务器都配有双核8核2.2GHzCPU，64GBDRAM和8个7200RPM2TB磁盘。每个虚拟机都配置了8GBDRAM，一个数据磁盘和4个CPU核心。

### 7.1 构建容错应用程序

Simba的主要目标是为构建容错应用程序提供高级抽象。评估成功虽然至关重要，但主观性很强，难以量化；我们试图通过三种定性手段来提供评估：（1）比较使用Simba和Dropbox编写同等应用程序的开发工作。（2）从头开始编写一些Simba—apps的开发工作。（3）在sClient系统故障注入时观察故障恢复。

### 7.1.1 写应用程序：Simba与Dropbox

目标：实现一个存储相册元数据和图像的照片同步应用程序。Apps将使用Dropbox中的Simba和AppD编写。我们选择Dropbox，因为它具有功能最丰富，最完整的现有系统API，也非常受欢迎。Dropbox为文件（Filestore）和表（Datastore）提供API。AppS和AppD必须为最终用户提供相同的语义：在相同的故障情况下，对相册和可靠性的一致看法；我们比较了开发两个同等应用程序的努力。

总结：实现一致性和可靠性对于AppS花费大约5个小时由1位开发人员编写和测试来说是非常简单的。然而，尽管付出了相当大的努力（3—4天），AppD并没有达到所有的目标。这里我们列出一个限制的总结：

1.Dropbox不提供任何机制来持续地互操作表和对象存储。

2.Dropbox数据存储实际上甚至不提供同步期间的行级别原子性（只有列级别）！

3.Dropbox没有处理撕裂行的机制，可能会同步不一致的数据。

4.Dropbox在后台执行冲突解决，防止用户干预。

方法论：我们简要地描述了我们为克服这些限制所作的努力，并使AppD等同于AppS；测试是在2个Android智能手机上完成的——一个是作者，另一个是读者。我们成功了1，2，但3，4失败了。

✓1.跨商店的一致性：我们将AppD图像存储在数据存储中的文件存储和相册信息中；为了说明依赖关系，我们创建一个额外的数据存储列来存储图像标识符。为了检测文件修改，我们在AppD中维护Dropbox监听器。

写入：在写入器上添加新图像时，读取器上的应用程序会接收表和文件的单独更新，因为Dropbox不提供行原子性，所以Simba元数据列可能在应用程序数据之前同步。为了在Simba元数据之前处理图像或专辑信息的乱序到达，我们设置标志以指示表格和对象同步完成；当Simba元数据到达时，我们检查这个标志来确定整行是否可用。之后读取器显示图像。

更新：更具挑战性。由于读者不知道更新的列，以及是否更新了任何对象，因此需要采取额外的步骤来确定同步结束。我们创建一个单独的元数据列（MC）来跟踪对数据存储区的更改；MC在作家处存储更新的应用程序列表。我们还会在其他列之前发布MC的同步，以便读者了解已同步的列。由于Dropbox不提供行同步的原子性，读取器会检查MC是否每次更新表和对象列。

删除：一旦作者删除表格和对象列，阅读器上的两个听众最终都会得到通知，之后数据将在本地被删除。

✓2.表+文件的行原子性：对于每一列更新，数据存储创建一个单独的同步消息并发送整个行；因此不可能在同步时区分更新的列和它们的行版本。因此，使用Dropbox进行原子同步需要更多元数据来跟踪更改；我们为每一列创建一个单独的表格作为解决方法。例如，对于具有一个表格和一个对象列的应用程序表格，除了MC之外还需要创建两个额外的表格。

对于更新，写入者在MC中列出待同步的表格和对象列（例如，<col1，col3，obj2>）并发出同步。阅读器接收每个更新的通知，并等待直到收到MC中的所有列。在MC之前收到列更新的情况下，我们记录事件，并在收到MC后重新访问。处理新的写入和删除是相似的，为简洁起见省略。

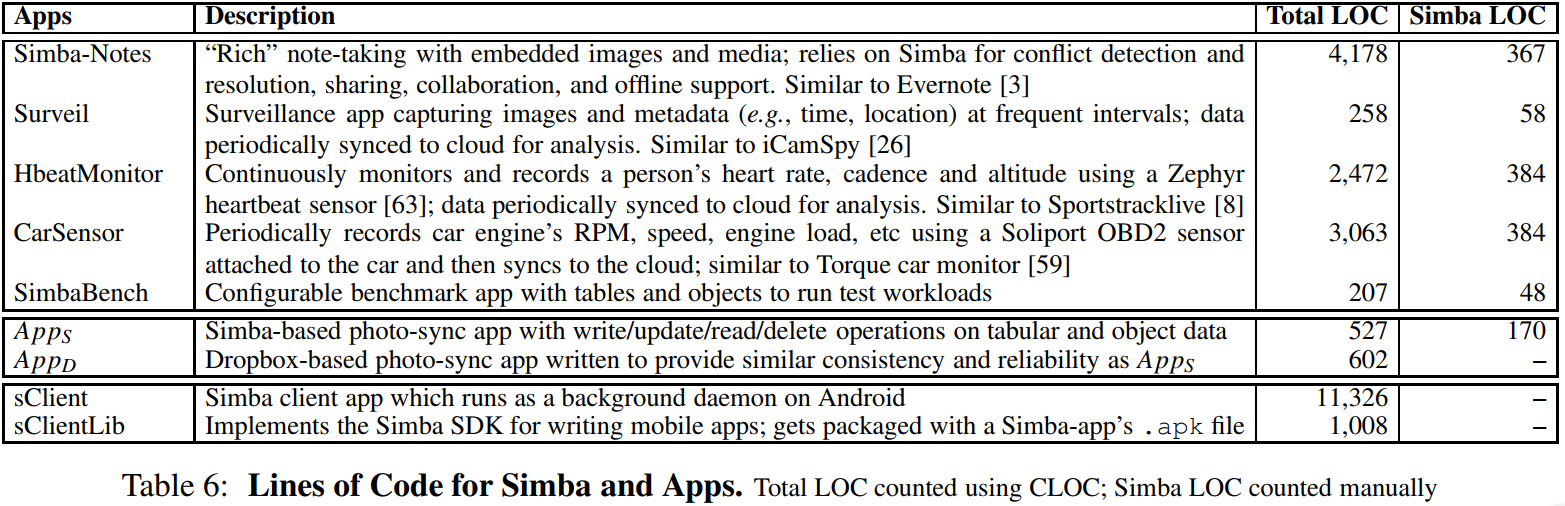
✗3.失败情况下的一致性：在失败情况下提供一致性对于AppD尤其棘手。为了防止破损行得到同步，AppD需要一个单独的持久标志来检测崩溃后的行不一致，以及所有的恢复机制来正确处理崩溃，如§5所述。由于AppD还不知道需要恢复的行中的特定对象，因此需要持久的数据结构来识别撕裂的对象。

✗4.一致的冲突检测：Dropbox为数据提供透明的冲突解决方案；因此，检测应用程序数据模型中出现的更高级别的冲突将留给应用程序。由于在更新对象之前没有检查潜在冲突的机制，因此我们需要为AppD中的每个对象创建一个持久脏标志。此外，如果冲突解决方案出现在“alwaystheirs”策略的后台，则应用程序的本地数据可能无法恢复。为了从不一致性中恢复，AppD需要将数据记录在外，需要单独的本地永久性存储。

满足3.和4.隐含重新实现AppD中的大多数sClient功能，并没有尝试。

### 7.1.2 其他Simba应用程序

我们基于现有的移动应用程序编写了一些Simba应用程序，发现这个过程很简单；这些应用程序对失败很有效，在测试时保持一致。由于Simba处理数据管理，因此平均编写应用程序需要4到8个小时，具体取决于GUI。表6提供了应用程序的简要说明及其与Simba相关的代码行（LOC）。



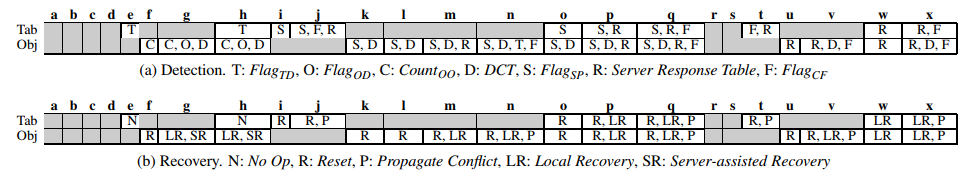
### 7.1.3 可靠性测试

在发布本地，同步和冲突处理操作时，我们注入了三种故障，网络中断，Simba—app崩溃和sClient崩溃。表7简要显示了sClient采用的技术。对于给定的工作负载（a—x），灰色单元表示未受影响或无效的情况，例如读取操作。检测中的非空细胞意味着所有病例都被计入，恢复中相应的非空细胞意味着采取了纠正措施。没有空单元格表示sClient正确检测到并从我们测试的所有常见故障中恢复。

该表显示了sClient在读，写，同步和同步操作失败时的检测和恢复策略。操作是：读取选项卡b：getobjreadstreamc：读取objd：读取tab+obje：写入tabf：getobjwritestreamg：写入objh：写入tab+obji：syncup选项卡j：syncupresultfor只有选项卡k：syncupobj只有l：只发送objfrag只有syncupm：syncupresult只有objn：只有objfrag只有objsyncupresulto：syncuptab+objp：syncupresulttab+objq：getobjfragtab+objsyncupresultr：notifys：syncdownt：syncdownresult仅用于制表符u：syncdownresult仅用于objv：仅用objfragforobjsyncdownresultw：syncdownresult用于制表符+objsyncdownresultx：用于制表符的objfrag+objsyncdownresult。

检测：表7（a）中的每个单元列出用于检测故障后表和对象状态的标志。sClient保持适当的本地状态，并从服务器响应，正确检测所有故障。当FlagTD在同步开始时切换时，表格数据的变化由FlagTD（T）进行写入，FlagSP（S）进行同步检测。然后，sClient对服务器的响应数据（R）进行检查。通过检查FlagCF（F）来识别同步冲突。类似地，通过添加用于同步的DCT（D），由CountOO（C）和FlagOD（O）检测到写入流和对象更新的使用。

恢复：表7（b）中的每个单元列出了sClient从no—op，reset，propagate和本地或服务器辅助恢复中采取的恢复操作。No—op（N）意味着不需要恢复，因为数据已经处于一致的状态。当检测到冲突，但数据一致时，sClient向寻求解决方案的用户传播（P）警报。在当地国家的帮助下，大多数情况下sClient在本地恢复（LR）；对于一个被撕裂的行，sClient依靠服务器辅助恢复（SR）。在某些情况下，sClient需要重置标志（R）以标志恢复的成功完成或无故障状态。



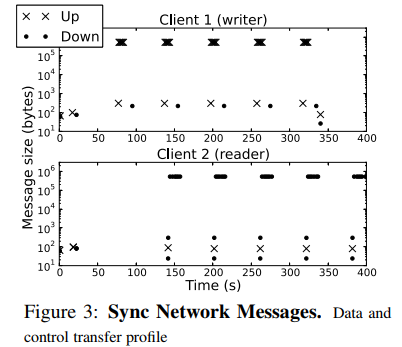


该表显示了sClient在读，写，同步和同步操作失败时的检测和恢复策略。操作是：读取选项卡b：getobjreadstreamc：读取objd：读取tab+obje：写入tabf：getobjwritestreamg：写入objh：写入tab+obji：syncup选项卡j：syncupresultfor只有选项卡k：syncupobj只有l：只发送objfrag只有syncupm：syncupresult只有objn：只有objfrag只有objsyncupresulto：syncuptab+objp：syncupresulttab+objq：getobjfragtab+objsyncupresultr：notifys：syncdownt：syncdownresult仅用于制表符u：syncdownresult仅用于objv：仅用objfragforobjsyncdownresultw：syncdownresult用于制表符+objsyncdownresultx：用于制表符的objfrag+objsyncdownresult。

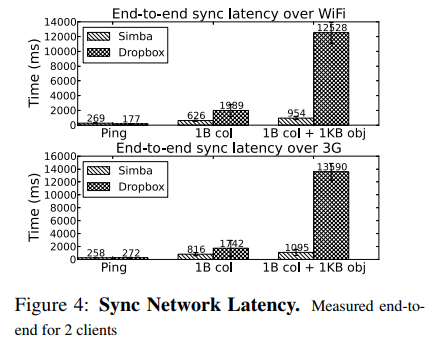
### 7.2 性能和效率

### 7.2.1 同步性能

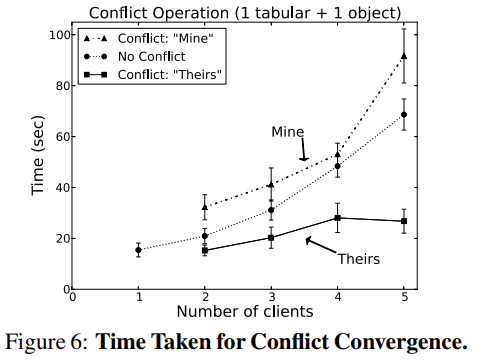
我们想验证Simba是否达到了定期同步的目的。图3显示了运行SimbaBench（表6）的两个移动客户端的客户端—服务器交互。在客户端1上，它每10秒创建一个具有100字节表数据和（50％可压缩）1MB对象的新行。客户端1还注册了60秒的定期上行同步。客户端2读取订阅同一个表也是60秒的时间。从图中可以看出，上游和下游同步的网络交互显示了短时间的活动爆发，随后是较长时间的不活动。客户端2的读取订阅计时器仅丢失第一个上游同步（77s—95s），所以第一个下游同步发生在大约一分钟后（141s—157s）。对于其余的实验，下游消息立即跟随上游消息确认Simba符合这个目标。



我们希望评估Simba的同步性能，以及如何与Dropbox进行比较。图4比较了Simba和Dropbox在WiFi和4G上的端到端同步延迟；y轴是5次试验的标准偏差所用的时间。对于这些测试，我们运行两种情况，两种情况都是在两个客户端之间同步一个行：1）只有一个1字节的列；2）有一个1字节的列和一个1KB的对象。这两个客户都在南韩。Dropbox服务器位于加州（通过IP地址验证），而Simba服务器位于美国东海岸。作为基准，我们还测量了从客户端到服务器的ping延迟。图4显示网络延迟（“Ping”）是总同步延迟的一小部分。对于这两项测试，Simba的表现都比Dropbox好得多。在情况1）下，大约100％至200％，以及情况2）大于1200％。由于Dropbox是专有的，我们并不是要求完全理解它的功能。它很可能会超载或限制流量。实验表明，即使在控制同步到应用程序时，Simba也表现良好。



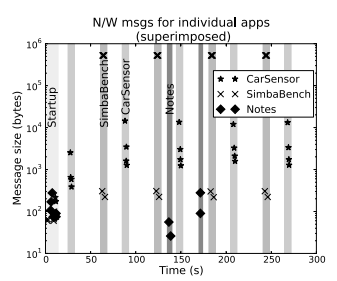
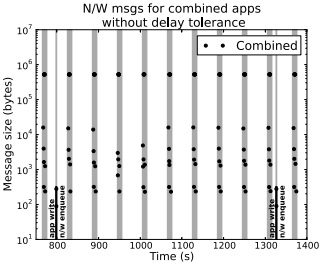
我们要测试Simba如何快速解决与多个作者的表的冲突。图6显示了这种行为。X轴显示了客户的数量（冲突所需的最少2个客户），y轴显示5次试验的平均收敛时间（秒）和标准差。对于“他们”，每次选择服务器的副本，因此不需要传播任何更改；对于“我的”，每次选择本地副本并重新同步回服务器。显示“无冲突”的情况下，建立一个基线—一个正常的同步仍然需要改变同步到服务器；“我的”总是和“他们”总是分别代表最糟糕的情况和最好的情况，典型的使用情况介于两者之间。该图表明，对于合理的客户数量（即5个），随着冲突解决回合数量的增加，与基准同步相比，即使在选择服务器的副本时也不会产生显着的开销；选择本地副本时，冲突解决速度相当快。



### 7.2.2 网络效率

我们想评估Simba对网络效率的影响。这个实验选择了三个应用程序来定期生成数据：CarSensor应用程序处于重播模式，每秒产生约250字节的行，SimbaBench设置为每10秒创建1MB行（可压缩50％），以及一个模拟SimbaNotes行为的应用程序使用平均值为300s的泊松分布生成~300字节的数据，并使用固定种子来产生随机数。CarSensor和SimbaBench以60秒的定期上行同步运行。图5显示了应用程序的数据传输配置文件的散点图；y轴是日志级别的消息大小，xaxis是以秒为单位的时间。彩带意味着描绘活动的时间集群。“启动”带显示一次性Simba认证和设置，以及表的同步注册消息。我们单独运行Simba应用程序（a），（b）与SimbaNotes的DT=0同时运行，（c）与Simba—Notes的DT=60s同时运行。

图5（a）显示了应用程序单独运行时的数据传输配置文件的超级拼版，以模拟运行不协调的应用程序的行为。从图中还可以看出，虽然不协调的定时器有可能重合，但这不太可能；特别是当数据传输时间比较长时。像Simba—Notes这样的非周期性应用程序也会导致不协调的传输。不协调的转移意味着频繁的无线电活动和由于尾巴时间较长而消耗的能量。在图5（b）中，所有应用程序都同时运行。Simba—Notes生成的事件被注释。我们看到，CarSensor和SimbaBench的网络传输是同步的，但是Simba—Notes在不规则时间仍然会导致网络传输（细节代表Simba—Notes的网络传输）。在图5（c）中，我们进行类似于（b）的实验，但是这次Simba—Notes采用了60s的延迟容限；其网络活动将延迟到下一个60年代定期计时器以及所有未决的同步活动（注意不存在的细频带）。由此产生的数据传输是集群化的，增加了无线电被关闭的几率。（b）和（c）中的x轴开始于大约800秒，因为我们在几分钟的应用程序启动后测量。

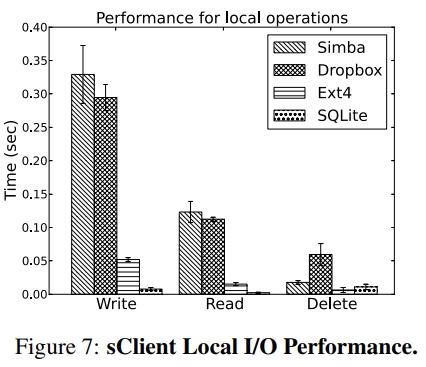


### 

### 

### 7.2.3 本地I/O性能

我们的目标是确定sClient的本地性能是否可以持续运行，特别是因为存储可以成为移动应用性能的主要贡献者。SimbaBench会针对sClient和Dropbox（CoreAPI），为包含一个1MB对象的一行数据写入，读取和删除操作。图7显示了5次试验的平均时间和标准差；对于写入和读取，sClient比Dropbox慢10％左右，主要是由于IPC开销，因为sClient是Android上的后台服务，而Dropbox直接访问文件系统。sClient通过懒惰删除执行更好的删除—数据只被标记为删除，但只有在同步完成后才能被删除。sClient和Dropbox都对Ext4和SQLite执行了几个额外的操作；我们仅提供此比较作为基准。



## 8 相关工作

数据同步和服务：同步已经在便携式设备的背景下进行了很多研究，包括关于断开操作，弱一致性复制存储和数据分级的开创性工作。

就失败透明度而言，Bayou通过写入日志提供了对其崩溃恢复的有限讨论，但它不处理对象。LBFS以回写方式自动提交文件，防止崩溃或中断时的损坏，但不处理表。我们发现，对于大多数应用程序来说，处理表格和对象数据之间的依赖关系是不一致的最大来源。

在现有的服务中，Dropbox是最全面的，但是仍然不支持对象和表的同步原子性，为多种错误条件打破了失败的透明度。iCloud还为键值界面和文件同步提供单独的机制。Mobius为表同步存储提供了CRUDAPI，但根本不支持对象。与Simba类似，Parse和Kinvey是移动后端即服务，提供GUI集成，管理和有限的数据管理。他们只支持表格，并提供最后作者的语义，这是不适用于许多应用程序。没有同步服务提供延迟容忍传输。

容错：ViewBox将桌面FS与数据同步服务集成，以仅同步本地数据的视图；本文还介绍了Dropbox如何通过校验和传播本地文件损坏的ViewBox地址。Simba专注于为应用程序提供透明的故障处理；而ViewBox只适用于文件，Simba跨越文件和表格。

存储统一：以前的桌面文件系统工作已经考虑了数据库集成，但没有网络同步或统一的API。InversionFS使用Postgres来实现具有事务保证和细粒度版本控制的文件系统。TableFS使用单独的存储池来存储元数据（LSM树）和文件，通过元数据操作来提高自身的性能。KVFS将文件数据和文件系统元数据存储在构建在VT—Trees之上的单个键值存储中，这是一种LSM树的变体，可以为各种大小的对象进行高效的存储；VT—Trees将来可以用来构建性能更好的sClient数据存储。

移动数据传输：最近的研究已经表征和优化了移动环境下的数据传输，尤其是小型零星传输的不利影响。SPDY扩展HTTP以实现更好的压缩，并通过单一连接复用请求以节省往返时间。这个大规模的网络研究激发了Simba的网络协议。

9 结论

构建高质量的以数据为中心的移动应用程序总是要求开发人员构建可靠和高效的数据管理基础架构—这是一项非常适合的任务。移动应用程序开发人员不需要担心网络和数据管理的复杂性，而是能够专注于他们最擅长的功能—实现用户界面和功能—并为用户提供出色的应用程序。我们构建了Simba，使开发人员能够快速开发和部署健壮，高效的移动应用程序；通过其移动客户端守护程序sClient，它提供了与适应大量移动应用程序的灵活策略同步的背景数据，同时透明地处理故障并有效利用移动资源。