|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分类号 |  | |  | | | 密 级 | |  |
| U D C |  | |  | | | 编 号 | | 10486 |
|  | | | | | | | | |
| 武汉大学logo  **智能手机的交易记录** | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | 研究生姓名 | | ： | 程苗 | |  | |
| 学号 | | ： | 2017282110233 | |
| 指导教师姓名、职称 | | ： | 李雪飞 副教授 | |
| 专业类别（领域） | | ： | 计算机技术 | |
|  | |  |  | |
|  | |  |  | |
|  | | | | | | | | |

目录

[摘要 3](#_Toc500169466)

[Abstract 4](#_Toc500169467)

[第一节 介绍 5](#_Toc500169468)

[第二节 背景和动机 6](#_Toc500169469)

[A.移动数据库中的事务日志开销 6](#_Toc500169470)

[B. QNVRAM概述 8](#_Toc500169471)

[第三节 XLog设计 9](#_Toc500169472)

[第四节 初步结果 10](#_Toc500169473)

[A.微观基准 10](#_Toc500169474)

[B.宏观基准：一个Leveldb例子 12](#_Toc500169475)

[第五节 讨论和未来的工作 14](#_Toc500169476)

## 摘要

移动数据库和键值存储通过预写日志提供一致性和持久性。传统的日志记录方案会将日志记录附加到日志文件的末尾，并使用fsync（）将记录刷新到持久存储中。由于底层文件系统的块大小和日志异常日志记录，日志记录延迟成为移动数据库的主要瓶颈。我们的实验结果表明，日志延迟占三星Galaxy S4智能手机整体插入延迟的90％以上。此外，我们观察到传统测井方案引起的显着的写放大（高达122×）。在本文中，我们介绍xLog，一种利用qNVRAM（一种用于移动设备的近乎非易失性存储器）的快速事务日志记录服务。根据我们的实验结果，xLog日志记录速度比传统日志记录方法快77倍，并且加快了LevelDB Put操作的速度达10.7倍。而且，xLog将传统测井方案的写放大率从122×大幅降低到1.6×以下。

关键字：数据库， 智能手机， 移动通信， 机器人， 人形机器人， 基准测试， 非易失性存储器

## Abstract

Mobile databases and key-value stores provide consistency and durability through write-ahead logging. The traditional logging scheme appends the log records to the end of the log file and flushes the records to durable storage using fsync(). Due to the large block size of the underlying file system and the Journaling of Journal anomaly, the logging latency becomes the main bottleneck of the mobile databases. Our experimental results indicate that the logging latency accounts for more than 90% of the overall insert latency on a Samsung Galaxy S4 smartphone. Moreover, we observe a significant write amplification (up to 122×) induced by the traditional logging scheme. In this paper we present xLog, a fast transaction logging service leveraging qNVRAM, a nearly non-volatile memory for mobile devices. From our experimental results, xLog logs up to 77× faster than the traditional logging scheme, and speeds up the LevelDB Put operation by up to 10.7×. Moreover, xLog drastically reduces the write amplification of the traditional logging scheme, from 122× to less than 1.6×.

**Keywords**：Databases, Smart phones, Mobile communication, Androids, Humanoid robots, Benchmark testing, Nonvolatile memory

## 第一节 介绍

智能手机和平板电脑在过去五年中无处不在。移动设备（智能手机和平板电脑）中的数据库已成为此类移动系统中数据管理的重要组成部分。SQLite数据库充当Android系统中的持久性存储层。LevelDB [1]是另一种仅支持Get（），Put（）和Delete（）等基本操作的NoSQL键值数据库，已经被各种应用广泛使用[2]。移动数据库使用预写式日志记录来确保数据持久性（原子性，一致性和持久性）。在提交时将事务日志刷新到持久存储以防止数据丢失。然而，由于智能手机中的懦弱的存储设备以及Journal of Journal（JOJ）异常导致的Android I / O堆栈的低效性，事务日志记录是实现快速响应写入事务响应时间的主要障碍[3]。最近的一项研究[4]指出JOJ异常是指文件系统对数据库日志活动进行日志记录的双重日志现象，会大大减慢移动数据库的速度。已经提出了几种解决方案来解决JOJ异常并提高性能。一个[4]是优化的Android I / O使用不同的技术，包括日志结构文件系统，外部日志和基于池-的组合堆的I / O。另一个解决方案[3]是将恢复信息集成到数据库文件本身，以便省略数据库日志。沉等人。[5]认为JOJ通过单个I / O几乎是免费的数据日志在Ext4文件系统中。尽管如此，虽然这些解决方案已被证明可以提高数据库事务吞吐量，但在事务提交时不可避免的数据冲刷到闪存存储仍然会导致特别长的日志记录延迟。我们以前的工作[6]提出了一个新的设计，利用现代智能手机的电池供电性质，使DRAM中的数据能够承受几乎所有的故障条件，这可以用来提高移动数据库的性能。

在本文中，我们提出了xLog，这是一款Android智能手机中的快速事务日志记录服务。xLog使用qNVRAM作为持久缓冲区来合并小日志记录，并显着减少SQLite和LevelDB中的事务日志记录的延迟。我们基于三星Galaxy S4智能手机的实验结果表明，xLog加速了事务日志记录77× 以及LevelDB Put操作的整体表现 10.7×，同时将写入放大从 122× 以小于 1.6×。

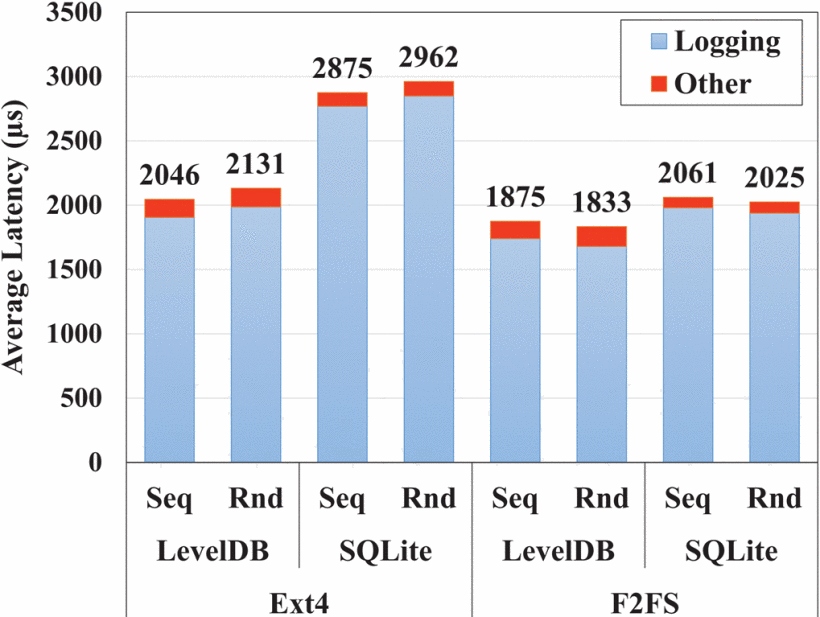
## 第二节 背景和动机

### A.移动数据库中的事务日志开销

事务日志的黄金标准是ARIES [7]，它使用细粒度的面向记录的预写日志（WAL）来恢复数据库。ARIES风格的生理日志记录结合了撤销和重做日志：在常规的ARIES系统中，撤消日志记录了操作的描述，并且重做日志记录了数据库页面的余像。作为ARIES风格日志记录的替代方法，命令logging [8]将事务逻辑（如SQL查询语句）写入预写日志。移动数据库使用这两种记录机制的变体。

SQLite的WAL使用值记录，这是不同于标准的ARIES风格的生理WAL。它只记录修改的数据库页面，在事务提交时，在SQLite WAL文件中称为框架。SQLite WAL文件中每个事务的日志记录由多个框架组成，其中最后一个框架作为提交标记。事务提交后，使用fsync（）/ fdatasync（）系统调用将日志记录刷新到eMMC闪存。在Android智能手机中，SQLite数据库的默认页面大小为4KB，与底层Ext4文件系统的块大小相同。

在LevelDB中的记录有点类似于命令记录。对于Put（）操作，LevelDB记录操作类型（kTypeValue），键和值; 对于Delete（）操作，LevelDB将操作类型（kTypeDeletion）和密钥写入日志文件。一般来说，与记录修改后的数据库页面相比，命令日志记录每个事务的写入数量要少得多，使前者比后者执行得更好。



**图1 插入操作的平均延迟**

事务日志记录在移动数据库系统上造成了很大的开销，这是由于存储设备薄弱以及Android I / O栈的效率低下造成的。为了评估在Android智能手机中登录的开销，我们在三星Galaxy S4智能手机上对SQLite和LevelDB运行基准测试。SQLite配置为使用预写式日志记录，并且LevelDB配置为同步写入模式。在基准测试的每次运行中，1000个记录（每个记录由一个整数键和一个100字节的字符串值组成）以顺序或随机键顺序插入SQLite或LevelDB。基准测试分别在Ext4和F2FS文件系统上运行。

测试的测量插入延迟的细分视图如图1所示，相应的日志文件大小和块级I / O统计数据如表1所示。“ 实际写入次数”是指写入日志文件的数据量和写入到eMMC存储设备的总次数。如图所示，日志延迟占平均插入延迟的90％以上。瓶颈在于fsync（）/ fdatasync（）系统调用。从表1我们可以看出，SQLite的日志文件大于20×比LevelDB大。对于每个插入事务，SQLite都会修改2到3个数据库页面，每个数据库页面都是4KB，并将它们写入日志文件。但是，对于相同的操作，LevelDB组成了大约130KB（4字节键+ 100字节值+记录标题）的小得多的日志记录。

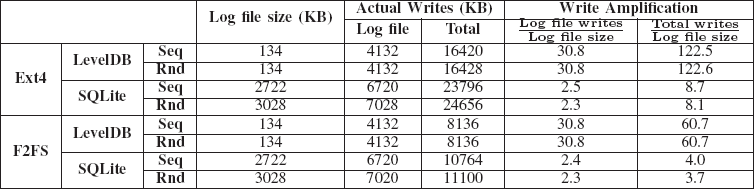
尽管如此，LevelDB和SQLite的日志记录延迟之间的差距并不像日志文件大小所暗示的那么大。在Ext4文件系统上，命令记录（LevelDB）只比值记录（SQLite）快38％; 在F2FS文件系统上，差距缩小到只有12％。这种反直觉的结果是由于底层文件系统和闪存设备使用较大的块大小（通常是4KB）。每当将日志记录追加到日志文件的末尾时，数据就会在文件系统块边界处刷新。在我们的基准测试中，一个4KB的块可以容纳大约30个LevelDB日志记录，因此对于30个Put（）操作中的每一个，相同的块将被刷新到存储设备30次。这会导致写入闪存的数据量大大超过日志文件的大小。在LevelDB基准测试中，30.8×日志文件的大小。当日志帧与文件系统块大小不一致时，SQLite中也存在此问题。更糟糕的是，更多的数据被写入文件系统元数据或日志，以确保文件系统的持久性。F2FS文件系统的总数据写入放大率可达60.7×; 在Ext4文件系统中，由于Journal Journal of Journal anomaly [4]，写入的总数据的写入放大率高达122×。

基准测试的另一个有趣的观察是，日志结构文件系统（F2FS）对记录延迟或写入放大没有明显的帮助。F2FS文件系统上的LevelDB的顺序和随机插入延迟比Ext4文件系统上的延迟只有12％和18％，而F2FS上的SQLite延迟比Ext4文件系统上的延迟缩短了28％和31％ 。其原因在于不可避免的fsync（）/ fdatasync（）系统调用由于非顺序I / O模式和高写入放大而阻止事务更快地提交。

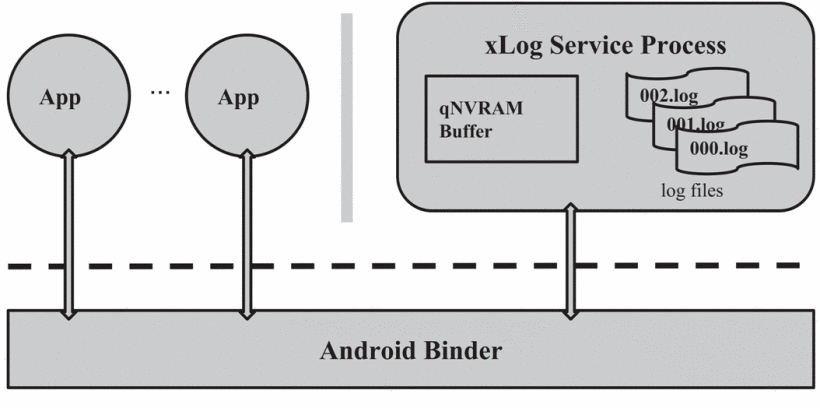
### B. QNVRAM概述

qNVRAM [6]是智能手机中的一种非易失性存储器，它利用移动设备的“电池供电”特性在qNVRAM在几乎所有故障条件下都是非易失性的。智能手机有四种不同的故障模式，分别是（1）应用程序崩溃，（2）应用程序挂起，（3）自动重启，（4）系统冻结。qNVRAM管理一块位于固定位置的物理内存，并根据应用程序的请求将物理内存映射到应用程序的地址空间。当应用程序在崩溃或被用户杀死（即故障模式（1）或（2））后重新启动时，qNVRAM应用程序分配被重新映射到它的地址空间，以便应用程序可以从qNVRAM中恢复; 当自启动（即故障模式（3））发生时，物理内存中的数据不会丢失，因为智能手机中的DRAM不会丢失功率，因此可以检索qNVRAM中的数据; 当用户执行硬复位（即，故障模式（4））时，qNVRAM将数据刷新到闪存，以便下次智能手机重新启动时将数据重新加载到qNVRAM。只要电池没有被物理拉出， qVNRAM中的数据可以被认为是非易失性的，鉴于越来越多的智能手机和平板电脑中的电池不可移动，qNVRAM在实际应用中可以被认为是非易失性的。

表1：基准测试中的日志文件大小和块I / O统计数据



## 第三节 XLog设计



**图2 xlog的体系结构**

xLog的架构如图2所示。xLog作为Android平台中的系统服务进程运行，并将一块qNVRAM作为刷新缓冲区来合并写入日志文件。应用程序通过Android Binder进程间通信（IPC）与xLog进行通信。将包含通用序列号和校验和的有关日志记录信息的小标题添加到每个记录以进行恢复。qNVRAM用作环形缓冲区，缓冲区头部的偏移量和环形缓冲区中的有效数据大小存储在qNVRA的前8个字节M，其中每个是32位整数。这两个变量是在崩溃后恢复环形缓冲区中的日志记录的关键。更新一个32位的变量需要一个单一的存储指令，这可以被认为是原子的[9]。

xLog有一个记录器线程，用于处理来自应用程序的记录请求，以及一个刷新工作线程，它定期将环形缓冲区中的日志记录刷新到日志文件中。所有传入的请求都由记录器线程序列化，并逐个插入环形缓冲区（相应的CPU缓存刷新到qNVRAM）。在xLog恢复期间，日志文件和环形缓冲区中的日志记录将使用通用序列号进行连接。

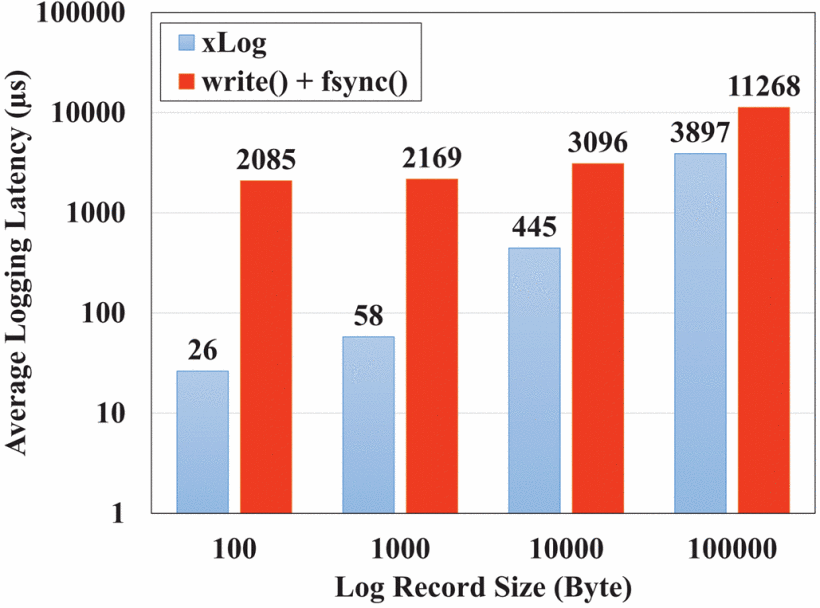
所有需要事务记录的应用程序的xLog日志保存数据。选择这种集中设计的原因是双重的。首先，突发模式在移动设备中很常见[1]，突发I / O通常由单个应用程序引起。由于移动数据库是嵌入在应用程序自己的进程中的，所以每个应用程序只能分配一小块qNVRAM作为刷新缓冲区，因为整个qNVRAM大小限制。然而，在XLOG的qNVRAM可以做出更大，更好地处理突发的I / O。另一方面，集中式采集设计有助于提高效率和qNVRAM利用率。背景应用程序可能会被lowmemkiller杀死，因此需要一个死亡通知机制来确保qNVRAM在应用程序过程结束后，应用程序分配的应用程序将被收回。但是，xLog没有这样的问题：xLog服务进程始终在运行，如果意外终止，它将立即重新启动。

xLog管理一组日志文件，每个客户端一个。为不同的日志记录客户端使用单独的日志文件，而不是所有客户端的单个日志文件，简化了设计并提供了有界的恢复时间，但是在刷新qNVRA时可能需要更多的fsync（）/ fdatasyncM缓冲。考虑到一个使用xLog服务的应用程序在发生崩溃后重新启动的恢复场景，它将从上一个检查点请求来自xLog的日志记录，并重播它们以从不一致的状态中恢复。为了获取这个应用程序的日志记录，xLog只需要重放与这个应用程序相对应的日志文件，而不是重放一个与所有应用程序相关的大型日志文件。

## 第四节 初步结果

### A.微观基准

我们在三星Galaxy S4智能手机上运行一系列微基准测试，以评估xLog的原始性能。微基准测试中的xLog服务进程分配10MB qNVRAM作为环形缓冲区。日志文件存储在一个单独的分区中，格式化为Ext4文件系统。使用xLog或使用write（）+ fsync（）将日志记录直接写入日志文件的传统日志记录方案记录总计100MB的日志记录。单个日志记录的大小从100字节到100千字节不等。我们测量本地xLog在服务进程内的日志延迟，而不涉及Android Binder IPC。

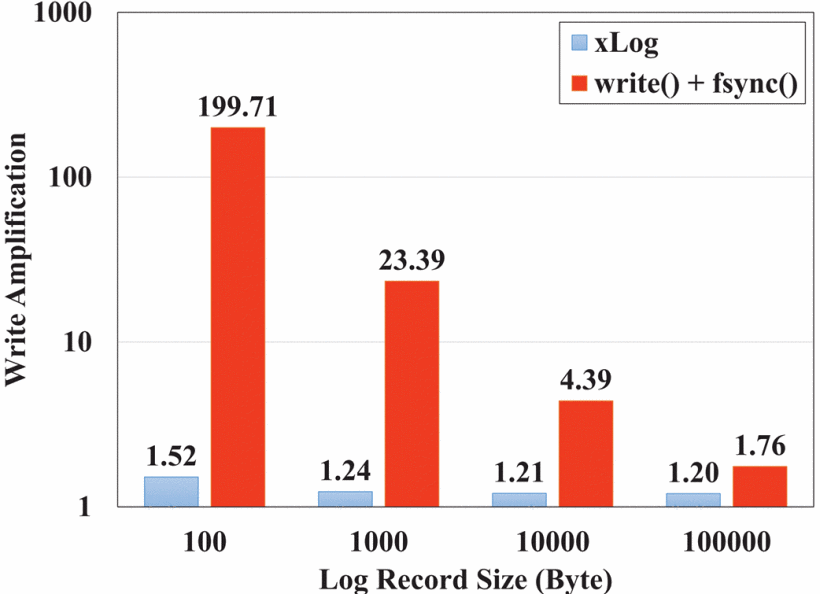


**图3 xlog的微基准性能**

xLog和基线日志记录方案的平均延迟如图3所示。从图中我们可以看到，当日志记录规模较小时，xLog远远优于传统的日志记录方法：xLog中100字节记录的平均记录延迟仅为传统方案的1.3％。随着日志记录的大小增加，两种方法之间的差距由于写入放大减少而变窄。然而，xLog仍然比write（）+ fsync（）产生更好的性能：xLog的日志记录性能是37×,7× 和 3× 当日志记录大小分别增加到1KB，10KB和100KB时，比使用write（）+ fsync（）的基线方案更快。

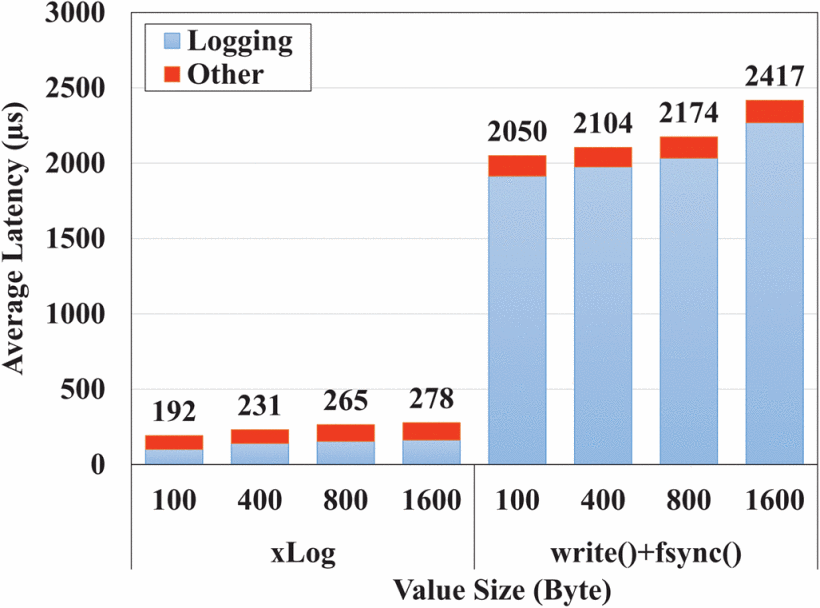
请注意，在微基准测试中，两个连续的日志记录请求之间没有间隔时间。因此，flush工作线程跟不上logger线程：logger线程需要等待flush工作线程为环境缓冲区腾出空间来存放新的日志记录。在现实世界的情况下，I / O请求以突发模式到达，这使xLog在下一次I / O突发到来之前留出足够的时间来刷新日志记录以记录文件。因此，即使日志记录与qNVRA一样长，日志记录的延迟也要小得多M缓冲区足够大以处理I / O突发。

xLog还通过将较小的写入操作分配到较大的I / O中来解决写入放大问题。微基准测试的结果如图4所示。当日志记录规模较小（每个记录100字节）时，传统日志记录方案（write（）+ fsync（））引起的写入放大率非常高(199.7×)。但是，xLog的写入放大率仅为1.5×。大部分开销来自xLog添加的记录头。随着记录尺寸的增加，这个开销将被分摊。



**图4 写xlog的放大**

### B.宏观基准：一个Leveldb例子



**图5 xlog和传统日志记录方案的Leveldb性能**

为了评估移动数据库xLog的性能优势，我们修改了LevelDB中的LogWriter，以便它使用xLog服务写入日志记录，而不是将它们附加到日志文件。使用xLog和基线传统日志记录的LevelDB的性能如图5所示。我们改变了LevelDB中的键值对的值大小，以评估具有不同日志记录大小的性能。从图中可以看出，当数值较小（100字节）时，xLog通过提高LevelDB的整体性能10.7×。随着值大小的增加，随着记录等待时间的增加，加速度会稍微下降。尽管如此，xLog依然能够提升LevelDB的性能9.1×, 8.2× 和 8.6× 当值大小分别是400字节，800字节和1600字节时。

与来自微基准测试的本地xLog的延迟相比，通过Android Binder IPC进行的xLog日志记录会导致严重的延迟。当值大小为100字节时，Android Binder IPC调用的开销约为100μs双向往返，不包括本地xLog的日志记录延迟。Binder IPC调用将涉及多个上下文切换和任务切换/唤醒，因为客户端（LevelDB）和服务器（xLog）都会从/向Binder内核驱动程序提取/传递数据。考虑到本地xLog只需要20μs来记录一个100字节的记录，IPC调用就成为整个日志记录延迟的主要瓶颈，从而降低了xLog的性能优势。尽管如此，xLog Binder IPC的日志记录延迟仍比写入日志文件低一个数量级。

## 第五节 讨论和未来的工作

在本文中，我们介绍xLog，这是一款使用qNVRAM作为缓冲区的快速事务日志记录服务，适用于Android智能手机。我们以前的工作已经证明，基于qNVRAM的持久缓冲区缓存可以显着提高SQLite数据库的性能。但是，xLog是执行持久性开销的更一般的解决方案。持久缓冲区缓存只能应用于使用分页存储并就地更新的应用程序。例如，LevelDB不能从持久化页面缓存中受益，因为它使用了日志结构化合并树（LSM-Tree），并且所有新记录被附加到已排序的表文件中。然而，xLog可以用来加速SQLite和LevelDB。xLog可用于加速现有数据库，以及构建新的持久性存储类型。使用xLog，我们可以构建一个高性能的持久内存数据库，并且不需要担心强制执行持久性的开销。我们计划在更多应用程序中探索xLog的性能优势，包括常规数据库（例如SQLite）和内存数据库，这些数据库通常通过检查点和日志记录来实现持久性。