SLMDB: (1)kv 应用场景，b+tree特点以及问题（2）LSM特点以及问题 挑战 （3）高性能读写需求，需要做优化 （4）NVM背景特点 （5-6）介绍自己工作 （7）总结 （8）paper架构

不像关系型数据库，键值对存储不须要了解值中的数据，具有更高的灵活性，值可以是字符串，文件或者是图片，可存储内容具有多样性。现如今键值对存储技术被广泛使用在各种数据密集型应用和数据中心，如社交网络[8]、电子商务[18]和网络索引[15]。Log-Structured Merge Trees (LSM-tree)是实现键值对存储系统的常用方式，如BigTable [13], LevelDB 和RocksDB。LSM tree是主要针对的场景是写密集、少量查询的场景，核心思想是放弃部分读能力，换取写入的最大化能力。高效的写入性能主要是通过在内存中构造一个缓冲区来将写入请求变成批处理实现的，从而将对磁盘的随机写变成顺序写。为了保证数据的顺序性和提高数据访问速度，LSM-tree 实现了多层的数据结构，并通过后台线程进行合并操作来维护数据结构。但这也带来了读写放大的问题。

为了保证系统能够从故障中恢复，在写入内存之前，需要使用Write-ahead-log记录操作。在数据被持久化到磁盘后，日志会被删除。使用WAL会造成写入性能下降,所以在默认情况下，基于LSM的存储系统如LevelDB和rocksdb禁用了WAL，以此来获得较好的性能，但这样降低了数据的一致性，因为当发生crash时，可能会有部分数据被丢失。

由于LSM-tree存在读放大的问题，所以基于LSM-tree的键值对存储系统需要对事务的并发控制协议进行优化。比如在乐观并发控制中，为了不去访问磁盘的数据而带来较大的消耗，验证阶段只会在内存中进行，如果版本数据不在内存中事务就会终止，这提高了事务的终止率。

With the emergence of byte-addressable persistent memory (PM), there occurs new opportunities and research directions for improving the performance of key-value (KV) stores. PM has recently become more and more involved in key-value stores \cite{NVMRocks,DBLP:conf/usenix/KannanBGAA18,DBLP:conf/usenix/XiaJXS17}, which ehances the performance of storage system. PM technologies such as PCM [2], memristors and 3D XPoint [4] promote the rapid development of NVM. PM can not only replace disks such as HDDs and SSDs but also be connected via a memory bus and act like DRAM. PM is expected to achieve a comparable read performance with DRAM and the write latency of PM is about 10x higher than DRAM~\cite{DBLP:conf/usenix/XiaJXS17}, but the cost of PM is lower than DRAM.

在本文中，我们利用PM来代替RocksDB架构中的内存组件，使用持久性内存来存储MemTable 和Immutable MemTable。由于PM具有字节寻址和持久化特性，从而不需要write-ahead-log，因为写入就被持久化了，减少log的开销，有更强的consistency。除此之外有更快的recovery速度，因为recovery不用去扫log entry。However, modern CPUs have multiple caches, and in some cases, CPUs may reorder some of the memory write instructions to improve write performance, which may lead to inconsistent system state. Therefore, we use cacheline flush and memory fence instructions to guarantee the consistency of persistent memory.

为了优化RocksDB的并发控制，特别是OCC，we use DRAM-based cache to maintain the sequence number of outstanding transactions, so that we only need to validate the sequence number in cache in the validation phase. Since the sequence number is maintained by cache, it has no need to directly access disk and thus no need to abort transaction, which improves read amplification and reduces the radio of transaction abort.

To summarize, we make the following contributions in this paper.

We explain the trade-off between performance and consistency in RocksDB 使用持久性内存来存储MemTable 和Immutable MemTable, 从而不需要write-ahead-log，减少log的开销，有更强的consistency，更快的recovery速度。

We show the limitations of optimistic concurrency control in RocksDB and exploits a DRAM-based cache to improves read amplification and reduces the radio of transaction abort.

The remainder of this paper is organized as follows. Section 2 gives an introduction on persistent memory technologies for KV store and the structure and features of RocksDB with the limitations of WAL overhead and read amplification, explaining the motivation of our work. Section 3 discusses how to leverage persistent memory in RocksDB. Section 4 presents the design details of DRAM-based cache. Section 5 shows the design of experiments and the expected experimental results. At last, Section 6 discusses the related work and Section 7 concludes the paper.

With the emergence of byte-addressable persistent memory (PM) and commercially available Intel OPTANE NVDIMM, PM offers new opportunities and research directions for improving the performance of key-value (KV) stores. We introduce a feasible solution to improve RocksDB via DRAM-based cache and persistent memory. We replace the DRAM with PM in RocksDB and redesign the memtable, reducing write overhead and providing stronger consistency and faster recovery due to the absence of Write-Ahead-Log (WAL). To optimize the Optimistic Concurrency Control (OCC) in RocksDB, our solution exploits a DRAM-based cache to maintain the sequence number of outstanding transactions, providing validation in memory and avoiding to directly access disk and thus reducing the probability of transaction abort. The cache can also improve read amplification. For Two-Phase Locking, we simply use the cache to maintain the lock manager. Our expected experimental results show that our solution provides higher read and write performance, faster recovery and lower transaction abort radio compared with RocksDB.

LSMtree由内存和DIsk两个部分的数据结构组成。为了提高写的吞吐量，使对disk的随机写变成顺序写，rocksdb在首先内存中构造了一个memtable buffer来batch写，memtable由排序过的跳表组成，当memtable的数据达到一定大小时，就将这个memtable置成immutable，然后创建一个新的memtable来接收写入请求。并通过后台线程将immutable table刷到磁盘上。磁盘由多层SSTable组成，从L0到Ln。除了L0之外，每个级别都有一个或多个排序的SSTable文件，其中同一级别的文件的键范围不会重叠。每个级别的容量是有限的，但更高的级别可以包含更多的SSTable ﬁles，这样一个级别的容量一般比前一个级别的容量大10倍左右。为了保持这样的分层级别和数据的顺序性，当一个层的大小超过它的限制时，后端线程会对将该层与下一层进行compaction操作（merge-sort)，两个层的数据根据key值进行排序，当两个层有相同的key时，上层的value会覆盖下层的value，这样就保证了每一层数据的唯一性还有顺序性，除了L0，因为Immutable MemTable刷到disk时没有进行compaction操作，以增加写吞吐量，所以L0中的SSTa-bles可以有重叠的键范围。总的来说，LSM优化了写操作，牺牲了一定的读性能，但由于多层结构和数据的有序性，使得读操作仍然保持较好的性能。

为了保证系统能够从故障中恢复，在写入memtable之前，会将写入操作写入到Write-ahead-log中。在不可变MemTable中的KV对被最终转储到磁盘后，日志会被删除。这会造成写入性能下降。当使用一个插入8GB数据创建的数据库，值大小为1KB时，当为WAL启用fsync()时，写入性能下降了12倍以上。所以这里涉及性能与一致性的trade off，默认情况下rocksdb不会将操作记录到log中，以此来获得较好的性能，所以当发生crash时，就可能会丢掉部分数据。

因为LSM-tree多层结构，RocksDB存在读放大的问题。当需要读数据时，首先访问memtable, 然后immutable table, 如果数据不在内存中，就需要去访问磁盘， 从最低层到最高层访问SSTable。在每一层，首先使用二分查找查找数据所在的SSTable，定位到SSTable，再使用二分查找找到键的索引。所以在检索某一层时，至少需要两次二分查找。如果当前层检索不到数据，就会去下一层查找，重复以上操作，直到查找到数据。所以如果数据需要到磁盘才能读到，消耗就会很大。为了减少访问磁盘的消耗， RocksDB使用布隆过滤器来优化读操作。对于OOC，rocksdb uses the minimal sequence number in memory and global sequence to determine whether to validate in memory. 如果sequence不在内存就会把事务abort掉，避免访问磁盘。为了优化读放大问题和减少事务中止的概率，我们使cache来维护outstanding 事务的 sequence number，从而验证阶段只需要在cache 验证sequence number。