# 海量存储课程作业

# 《Analysis of HDFS Under HBase:A Facebook Messages Case Study》论文阅读报告

# 院（系）名 称 ： 计算机学院

# 专 业 名 称 ： 计算机技术

# 学 生 姓 名 ： 章辉宇

# 指 导 教 师 ： 何水兵 教授

# 二〇一七年十二月

## 

## 1. 论文的基本信息

Tyler Harter, University of Wisconsin—Madison; Dhruba Borthakur, Siying Dong,Amitanand Aiyer, and Liyin Tang, Facebook Inc.; Andrea C. Arpaci-Dusseau andRemzi H.Arpaci-Dusseau, University of Wisconsin—Madison <https://www.usenix.org/conference/fast14/technical-sessions/presentation/harter>

## 2. 论文要解决的问题

摘要：我们展示了基于HBase和HDFS的Facebook Messages堆栈的多层研究。 我们收集和分析HDFS痕迹以识别潜在的改进，然后我们通过模拟评估。 消息代表了一个新的HDFS工作负载：而HDFS是为存储非常大的文件而构建的，并且主要接收I / O，90％的文件小于15MB，I / O高度随机。 我们发现热数据太大，不容易装入内存，冷数据太大，不容易装入闪存; 然而，成本模拟表明，添加一个小的闪存层比RAM或磁盘上的等价支出提高了性能。 HBase的分层设计提供了简单性，但是以性能为代价; 我们的仿真显示，如果压缩绕过复制层，则网络I / O可以减半。 最后，尽管消息是以读为主的，堆栈的几个特性（即日志记录，压缩，复制和缓存）放大了写I / O，导致写操作主宰磁盘I / O。

FM是一个消息系统，使Facebook用户可以发送聊天和类似电子邮件的消息; 它是相当受欢迎的，每天处理数百万条消息。 FM将其信息存储在HBase（以及HDFS）中，因此是一个很好的案例研究。另外，代表了一种重要的应用类型，通用后台为手机短信，聊天记录和邮件等。

## 3. 论文介绍

### 3.1 分层存储

大规模的分布式存储系统设计，实现和运行非常复杂和耗时。 因此，工程师不是从整个布料上切割新的系统，而是经常选择分层结构，在已有的系统上建立新的系统，以减轻开发和部署的负担。 众所周知，分层有很多优点。 例如，通过在分布式和复制块级存储系统之上实施，分散式文件系统[Frangipani]的构建大大简化了。 由于Petal提供可扩展的容错虚拟磁盘，Frangipani可以专注于文件系统级问题（例如锁定）; 据这位作者所说，这种两层结构的结果是“相对容易建造”[27]。 不幸的是，分层也可能导致问题，通常以性能降低，可靠性降低或其他相关问题的形式出现。 例如，Denehy表明软件RAID上的日志文件系统如何分层会导致数据丢失或损坏[5]。 同样，其他人也争论过在块设备上文件系统的低效率[10]。

HDFS设计的初衷是为了存储大文件（例如日志文件），面向批处理、顺序I/O的。然而架设在HDFS之上的HBase设计的初衷却是为了解决海量数据的随机读写的请求。把这两种设计初衷截然相反的组件怎么揉在一起的呢？这种分层的结构设计主要是为了使架构更清晰，HBase层和HDFS层各司其职；但是却带来了潜在的性能下降。 论文的重点放在一个分布式存储体系结构上：一个分布式数据库（HBase）在分布式文件系统（HDFS）之上。目标是研究这些重要系统的相互作用，分析HDFS是否是HBase的有效存储后端。

### 3.2 分析方法

为了深入了解这个等级系统，并回答这个问题，我们在一个流行的工作负载下追踪和分析它：Facebook Messages（FM）[20]。为了执行我们的分析，我们首先在特殊配置的影子群中的一部分机器上收集八天内的详细HDFS级别的跟踪。 为了测试系统变化，FM流量镜像到这个影子群集; 在这里，我们利用阴影来收集详细的HDFS痕迹。 然后，我们分析这些痕迹，将结果与以前在更传统工作负载下的HDFS研究进行比较[14,16]。

为了补充我们的分析，我们还对各种缓存，日志记录和其他架构增强和修改进行了大量模拟。 通过仿真，我们可以探索一系列“如果？”情景，从而更深入地了解分层存储系统的功效。

论文的分析方法包括追踪、分析和模拟，在组合分析和模拟研究中获得了许多见解。比如，写入占HDFS文件I / O的21％，然而绝大多数的写入是来自日志和压缩的HBase开销。除了这些开销外，FM写入很少，仅占“真正”HDFS I / O的1％。在堆栈中深入，模拟显示写入被放大。在HDFS复制（三倍写入）和操作系统缓存（吸收读取）之下，64％的最终磁盘负载是写入I / O。 这种写入放大（从1％到64％）强调了在分层系统中优化写入的重要性，即使对于特别是读取繁重的工作负载（如FM）也是如此。

## 4. 论文背景

### 4.1 消息框架

FM的用户与网络层进行交互，网络层由应用程序集群支持，而应用程序集群又将数据存储在单独的HBase集群中。 应用程序集群执行特定于FM的逻辑并缓存HBase行，而HBase本身负责维持大部分数据。

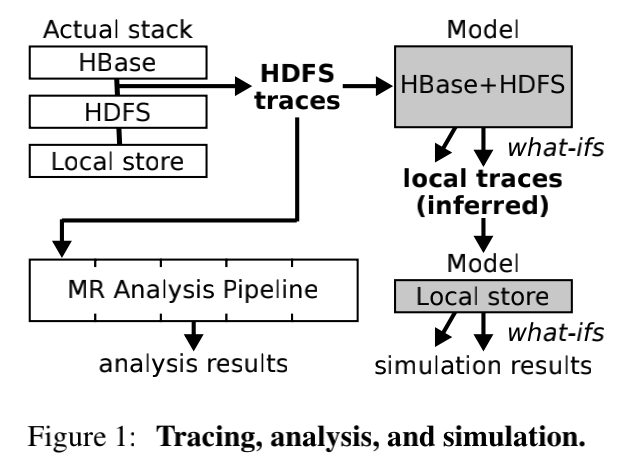
HBase将其数据存储在HDFS [24]中，这是一个类似于GFS的分布式文件系统[11]。 HDFS三重复制数据，以提供可用性和容忍失败。 这些属性免费HBase专注于高层次的数据库逻辑。 由于HBase将其所有数据存储在HDFS中，通常使用相同的机器来运行HBase和HDFS服务器，从而改善局部性。 这些集群有三种主要类型的机器：HBase主机，HDFS Name Node和许多工作机器。 每个工作人员运行两个服务器：一个HBase区域服务器和一个HDFS数据节点。 HBase客户端使用HBase主机将行密钥映射到负责该密钥的一个Region Server。 同样，一个HDFS Name Node可以帮助HDFS客户端将一个路径名和块号映射到三个Data Node中，并带有该块的副本。

### 4.2 Methodology

这部分讨论了数据追踪收集和分析，模拟，有效性和机密性。

#### 4.2.1 追踪收集和分析

之前的Hadoop跟踪研究[4,16]通常会分析默认的MapReduce或HDFS日志，这些日志会记录粗糙的文件事件（例如，创建和打开），但缺少有关各个请求（例如偏移和大小）的详细信息。 对于我们的研究，我们建立一个新的跟踪框架，HTFS（Hadoop跟踪文件系统）来收集这些细节。 尽管（例如，写入的内容）一些数据不被记录， 这使得痕迹更小，（更重要的是）保护用户的隐私。

我们从2013年6月7日起收集了8.3天的踪迹。我们收集了116GB数据，记录了52亿次记录的事件和71TB的HDFS I / O。 每台机器都有32个超级CPU内核和48GB内存，其中16.4GB分配给HBase缓存（大部分内存留给文件系统缓存，因为在HBase中使用更大的缓存会导致JVM垃圾回收） 

如图1所示，轨迹既可以进行分析，也可以进行仿真。 我们用10条MapReduce作业的流水线分析了我们的痕迹，每条作业都转换轨迹，建立索引，分割事件或输出统计数据。 事件之间复杂的依赖关系需要仔细分析正确性。 例如，流打开事件和流写入事件必须位于相同的计算分片中，以便将I / O与文件类型相关联。 此外，分片必须解决不同的路径可能引用相同的数据（由于重命名）的事实。

#### 4.2.2 模型制作与模拟试验

我们通过仿真来评估存储堆栈的变化。 我们的仿真基于两个模型（如图1所示）：分析HDFS I / O如何转换为本地I / O以及本地存储模型。

#### 4.2.3 仿真的有效性

论文需要解决三个有效性问题：忽略网络延迟是否会影响我们的结果？ 我们是否已经模拟了足够长的时间？ 单个代表性机器的仿真结果是否有意义？

首先，我们通过将随机抖动添加到请求的时间并观察统计数据的重要性如何变化来探索我们关于不断的网络延迟的假设。

其次，为了验证我们足够长的模拟时间，我们测量了如果我们在2或4天之前完成了模拟，统计数据将会有所不同，而不是使用完整的8.3天的痕迹。

最后，我们评估选择一个代表是否合理，而不是针对我们的样本中的所有九台机器进行实验。 在单台机器上运行我们的所有实验需要大约3天的时间，使用72GB RAM的24核机器，因此将我们的结果以代表为基础是可取的。

## 5. 论文分析的问题

### 5.1 FM系统的主要读写I/O负载

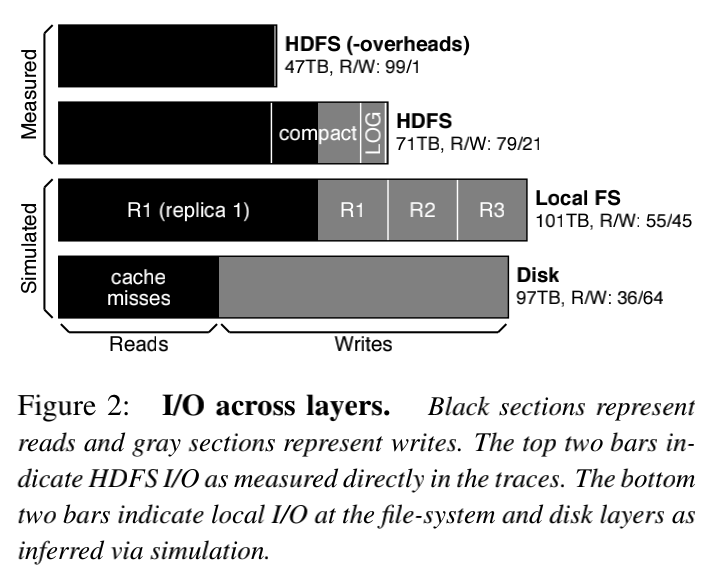


Figure 2描述了堆栈的每一层I/O构成（黑色部分表示读取，灰色部分表示写入），解释了在FM系统对外请求中读占主导，但是由于logging/compaction/replication/caching导致写被严重放大。

* 1. 在较高层次上，FM向HBase发出put（）和get（）请求。 put数据在缓冲区中累积，缓冲区偶尔会刷新到H Files（包含排序的键值对和索引元数据的HDFS文件）。 这两个操作的数据都要写到HDFS上，其中读写比99/1（如第一栏所示）。
  2. 除了核心I / O之外，HBase还会logging（保证数据持久性）和compacting（以保持读取的高效性），这两个操作都是写占主导的，所以R / W（读/写）比率下降到79/21（如第二栏所示）。
  3. FM通过使用HDFS数据备份来提高容错性。 因此，编写一个HDFS块包括写三个本地文件和两个网络传输（也就是上述对HDFS的写操作都会被转化成三倍的local file I/O和两倍的网络I/O）。从而进一步降低R / W比为55/45（如第三栏所示）。
  4. 此外，操作系统缓存可防止其中一些文件系统读取命中磁盘。 使用30GB缓存，在文件系统读取的56TB相当于仅在磁盘读取35TB。 此外，非常小的文件系统写入会导致4KB块磁盘写入，因此磁盘级别的写入会增加。 由于这些因素，读写比变成了36/64（如第四栏所示）。

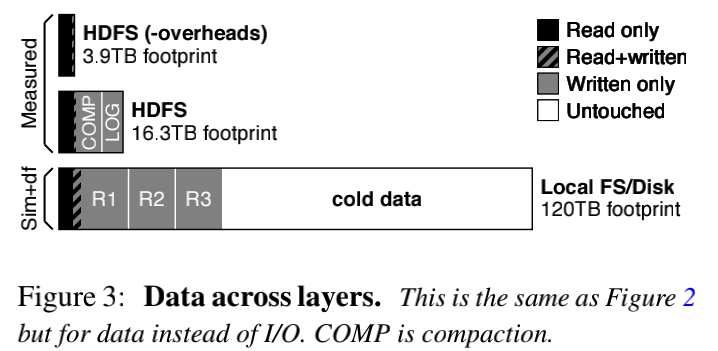


Figure 3给出了一个类似的分层概述，但是对于数据而不是I / O。 第一栏显示在跟踪期间接收到一些核心I / O的3.9TB HDFS数据（在跟踪过程中删除的数据不计算在内）。 几乎所有这些数据都被读取，一小部分被写入。 第二栏还包含仅由非核心I / O访问的数据; 非核心数据比核心数据大几倍。 第三栏显示跟踪期间在本地级别触摸了多少数据。 该栏还显示未触动的数据; 我们通过从磁盘利用率（用df测量）减去由于HDFS I / O而被推断的数据量来估计。 除了上面说的，还发现了整个系统最终存储在磁盘上有大量的cold data（占2/3），所以需要支持hot/cold数据分开存储。

结论：FM读为主导，但是日志记录，压缩，复制和缓存放大了写入I / O，导致写入操作占据了大部分的磁盘I / O。

### 5.2 FM系统的主要文件类型和大小

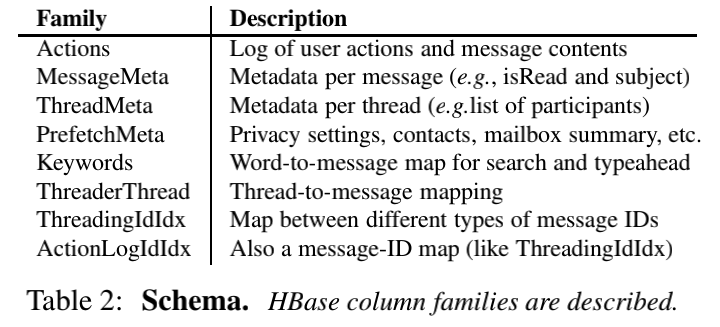


Table 2给出了FM存储的数据类型。每个用户的数据存储在同一个HBase行中; 这可以防止数据分散到不同的Region Server上。 用户的新数据将添加到该行内的新列中。 相关列被分组到由FM模式定义的族中。

Actions系列是建立在HBase之上的日志，不同的日志记录存储在不同的列中。因为Facebook聊天是建立在消息上的，所以元数据对象相对于许多消息来说是很大的（例如“hey，what’sup？”）。 因此，编写Actions的更改通常比将完整的元数据对象写入Message Meta代价小多了。

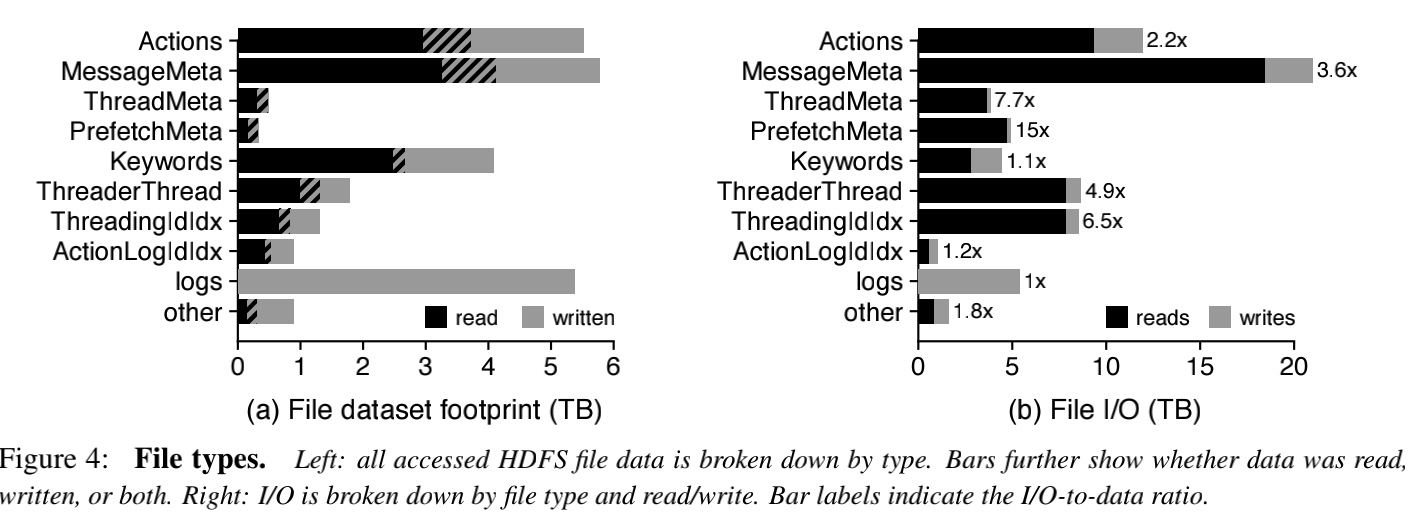


Figure 4a显示了在跟踪过程中每种类型的数据至少被访问过一次（包括后来被删除的数据）。 访问了26.5TB的总和。 虽然实际的消息（即Actions）需要很大的空间，但辅助数据（例如元数据，索引和日志）需要更多。 我们也看到，小数据都被读取和写入，表明写入应该被选择性地缓存（如果有的话）。 Figure 4b报告了每种类型的I / O。 我们观察到一些家庭每个数据接收更多的I / O，例如，Prefetch Meta的平均数据字节接收15字节的I / O，而关键字字节仅接收1.1。

结论：FM使用大量空间来存储消息，并在这些消息上执行大量的I / O操作; 然而，空间和I / O都由辅助数据（即元数据，索引和日志）占主导地位。 在跟踪期间写入和读取的数据相对较少; 这表明缓存写入没有多大作用。这对HDFS的NameNode提出了很大的挑战，因为HDFS设计的初衷是为了存储少量、大文件准备的，所有的文件的元数据是存储在NameNode的内存中的，还有有NameNode federation。

### 5.3 FM系统的主要I/O访问类型

论文探讨了不同读请求之间的三种关系：时间局部性，空间局部性和序列性。

经过研究发现73.7%的数据只被读取了一次，但是1.1%的数据被读取了至少64次。也就是说只有少部分的数据被重复读取了。但是从触发I/O的角度，只有19%的读操作读取的是只被读取一次的数据，而大部分I/O是读取那些热数据。

在HDFS这一层，FM读取数据没有表现出sequentiality，也就是说明high-bandwidth, high-latency的机械磁盘不是服务读请求的理想存储介质。而且对数据的读取也没有表现出spatial locality，也就是说I/O预读取也没啥作用。

## 6. 论文给出的解决方案

### 6.1 Flash/SSD作为cache使用

现在我们来为本地机器添加一个闪存层。 FM有一个非常大的，主要是Cold数据集（§4.1）; 将所有这些数据保存在闪存中将是浪费的，成本高达$ 10K / machine3。 我们评估这两个选择：使用一些闪存或不用闪存。

#### 6.1.1不用闪存的性能

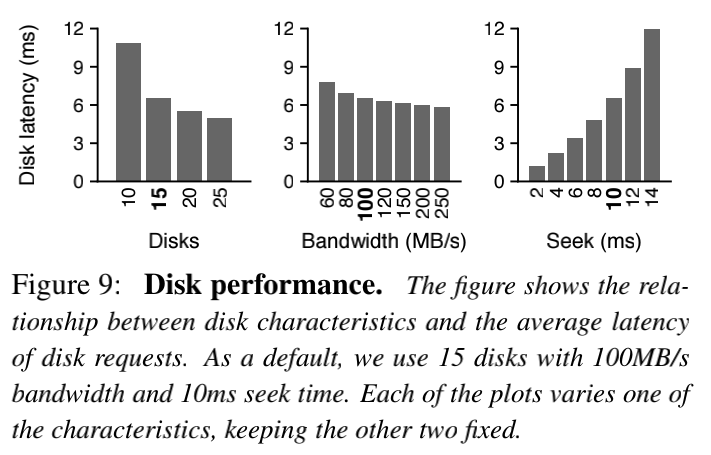
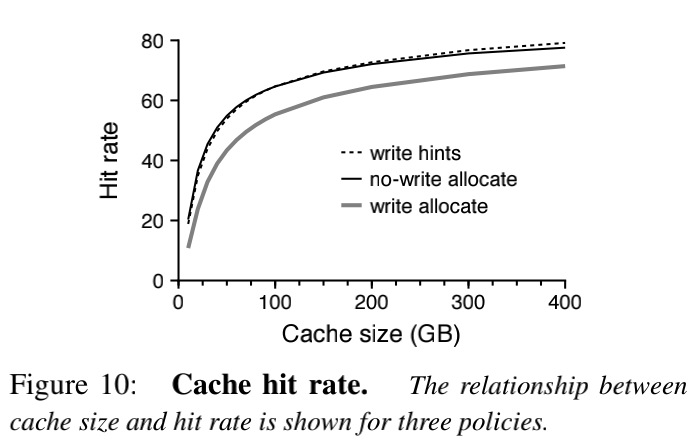


Figure 9显示了平均磁盘延迟作为各种磁盘因素的函数。 第一张图显示了15张以上的磁盘，增加了更多的磁盘迅速收益递减。 第二个显示高带宽磁盘也没有什么优势，正如之前观察到的高度随机工作负载（§4.4）所预期的那样。 但是，第三个图表显示延迟是一个主要的性能因素。事实上，较低的延迟有助于提供更多的磁盘，这表明工作负载具有相对较少的并行性，即能够快速完成一些事情比能够同时完成许多事情要好。



Hot数据可以舒适地放在纯RAM缓存中吗？论文测量10-400GB范围内的缓存大小的命中率。 尝试了三种不同的LRU策略：写分配，不写分配和写提示。Figure 10显示，对于每个策略，当高速缓存大小增加到大约200GB时，命中率显着增加，并在此处开始平稳（但不平坦）; 这表明工作集非常大。 早些时候（第4.2节），我们发现写入和读取之间几乎没有重叠，并得出结论，即使写入数据应该被选择性地缓存。 Figure 10证实：缓存所有写入是最差的策略。

结论：FM工作负载具有相对较少的顺序性或并行性，因此添加更多磁盘或更高带宽磁盘的效用有限。 幸运的是，相同的数据经常被重复读取（§4.4），所以一个非常大的缓存（即几百GB的大小）可以为近80％的读取提供服务。 非常大的高速缓存的实用性表明，将至少一些Hot数据存储在闪存中可能是最具成本效益的。

#### 6.1.2 使用闪存作为高速缓存

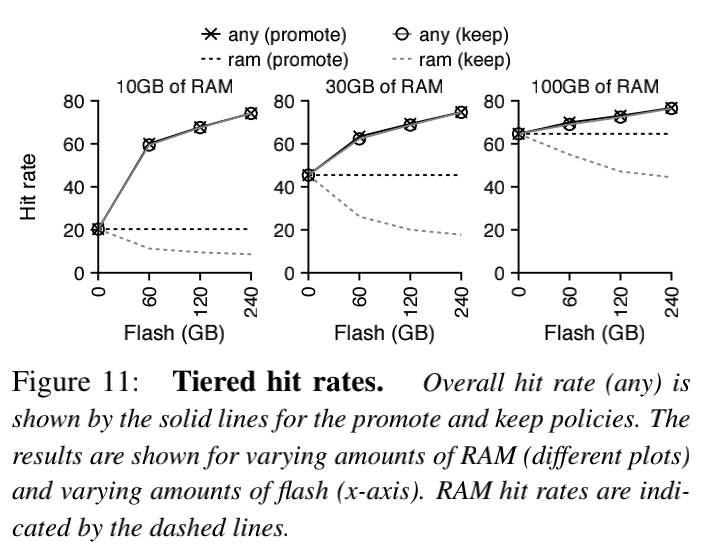


Figure 11显示了12个闪存/ RAM混合的命中率。 例如，中间图表显示了30GB内存时的命中率：没有任何闪存，45％的读取命中了缓存；但是若有60GB的闪存，命中率达到了63％。这些图说明如果拿足够大的Flash做二级缓存，cache命中率会明显提高。

另外论文探讨了Flash/SSD寿命的问题，在内存和Flash中shuffling数据能够使得最热的数据被交换到内存中，从而提升读性能，但是会降低Flash的寿命,但是随着技术的发展这个问题带来的影响可能越来越小。

说完加速读的cache，接着讨论了Flash作为写buffer是否会带来性能上的提升。由于HDFS写操作只要数据被DataNode成功接收到内存中就保证了持久性（因为三台DataNode同时存储，所以认为从DataNode的内存flush到磁盘的操作不会三个DataNode都失败），所以拿Flash做写buffer不会提高性能。虽然加写buffer会使后台的compaction操作降低他与前台服务的I/O争用，但是会增加很大复杂度，所以还是不用了。最后论文给出了结论就是拿Flash做写buffer没用。

论文通过计算发现，在这个存储栈中加入Flash做二级缓存不但能提升性能达3倍之多，而且只需要增加5%的成本，比加内存性价比高很多。

### 6.2 分层架构的缺点和改进方案

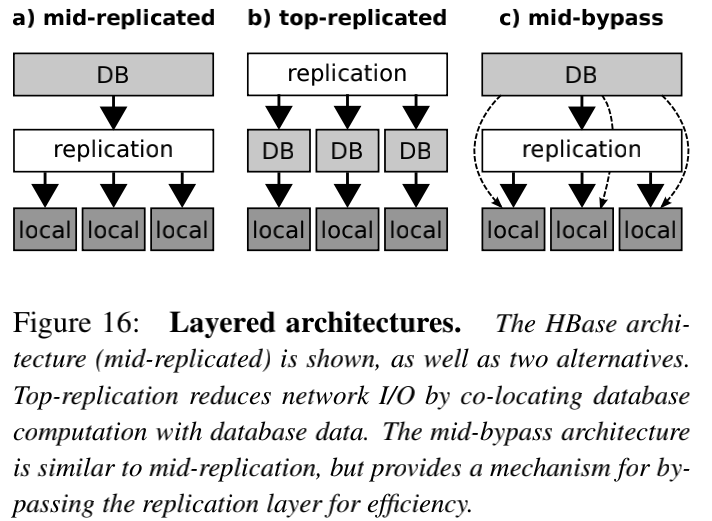


Figure 16表明，一般分布式数据库系统分为三个层次：db layer/replication layer/local layer。这种分层架构的最大优点是简洁清晰，每层各司其职。例如db layer只需要处理DB相关的逻辑，底层的存储认为是available和reliable的。

HBase是图中a)的架构，数据的冗余replication由HDFS来负责。但是这个带来一个问题就是例如compaction操作会读取多个三备份的小文件到内存merge-sorting成一个三备份的大文件，这个操作只能在其中的一个RS/DN上完成，那么从其他RS/DN上的数据读写都会带来网络传输I/O。

图中b)的架构就是把replication层放到了DB层的上面，Facebook举的例子是Salus，不过我对这个东西不太熟悉。我认为Cassandra就是这个架构的。这个架构的缺点就是DB层需要处理底层文件系统的问题，还要保证和其他节点的DB层协调一致，太复杂了。

图中c)的架构是在a的基础上的一种改进，Spark使用的就是这个架构。HBase的compaction操作就可以简化成join和sort这样两个RDD变换。

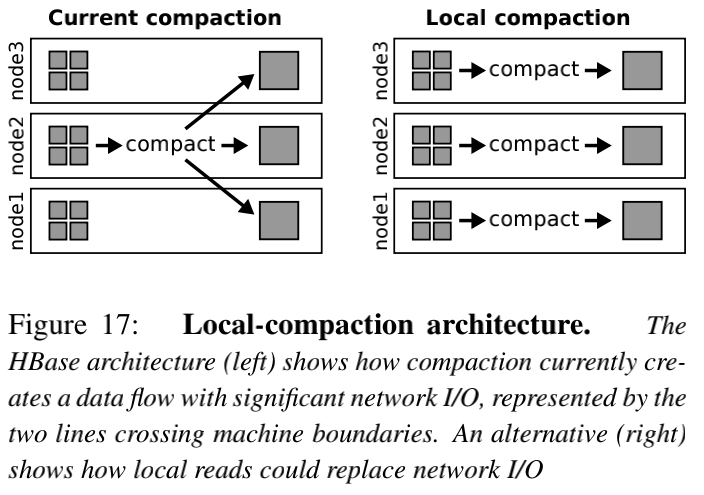


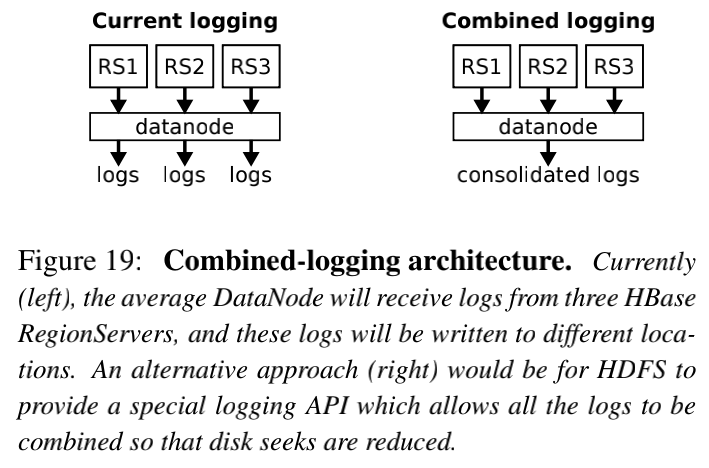
Figure 17展示了local compaction的原理，原来的网络I/O的一半转化成了本地磁盘读I/O，而且可以利用读cache加速。我们都知道在数据密集型计算系统中网络交换机的I/O瓶颈非常大，例如MapReduce Job中Data Shuffle操作就是最耗时的操作，需要强大的网络I/O带宽。加州大学圣迭戈分校(UCSD)和微软亚洲研究院(MSRA)都曾经设计专门的数据中心网络拓扑来优化网络I/O负载，相关研究成果在计算机网络顶级会议SIGCOMM上发表了多篇论文，但是由于其对网络路由器的改动伤筋动骨，最后都没有成功推广开来。

Figure 19展示了combined logging的原理。现在HBase的多个RS会向同一个DataNode发送写log请求，而目前DataNode端会把来自这三个RS的log分别写到不同的文件/块中，会导致该DataNode磁盘seek操作较多（不再是磁盘顺序I/O，而是随机I/O）。Combined logging就是把来自不同RS的log写到同一个文件中，这样就把DataNode的随机I/O转化成了顺序I/O。

## 7. 问题和总结

这篇论文主要分析了Facebook Messages系统使用HBase中遇到的一些问题和解决方案。作者使用自己编写的分析工具，很好地还原了HBase/HDFS框架中数据的读写过程，并提出十分有效的解决措施，给用HBase做在线存储的研究人员提供了参考。