Apache Cassandra 在 Facebook 的应用

在 Instagram （Instagram 是 Facebook 公司旗下一款免费提供在线图片及视频分享的社交应用软件，于2010年10月发布。）上，我们拥有世界上最大的 Apache Cassandra 数据库部署。我们在 2012 年开始使用 Cassandra 取代 Redis ，在生产环境中支撑欺诈检测，Feed 和 Direct inbox 等产品。起初我们在 AWS 环境中运行了 Cassandra 集群，但是当 Instagram 架构发生变化时，我们将 Cassandra 集群迁移到Facebook 的基础架构中。我们对 Cassandra 的可靠性和可用性有了非常好的体验，但是在读取数据延迟方面我们觉得他需要改进。

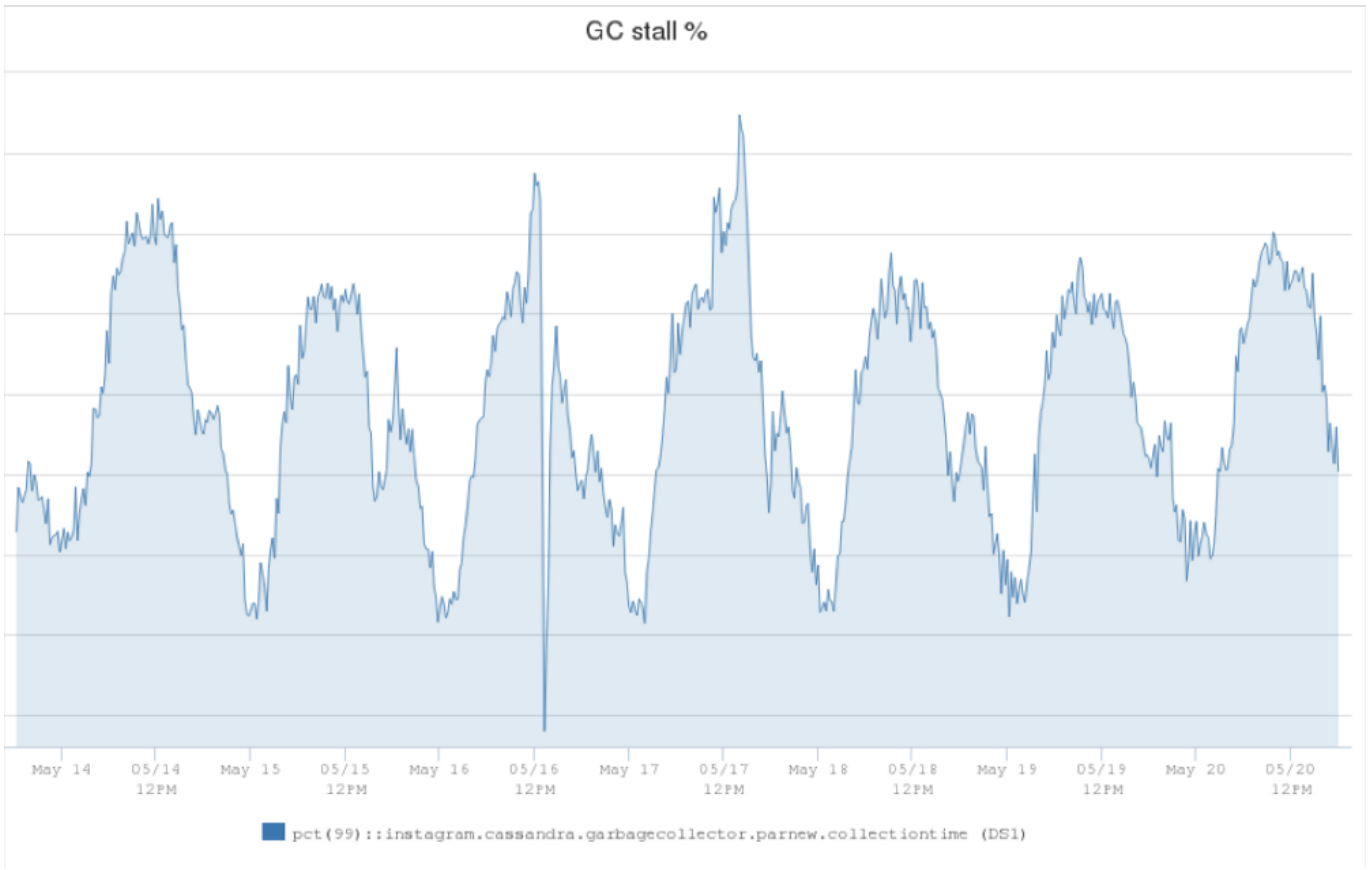
去年 Instagram 的 Cassandra 团队开始着手这个项目，以显着减少 Cassandra 的读取延迟，我们称之为 Rocksandra。
在这篇文章中，我将描述该项目的动机，我们克服的挑战以及内部和公共云环境中的性能指标。

动机

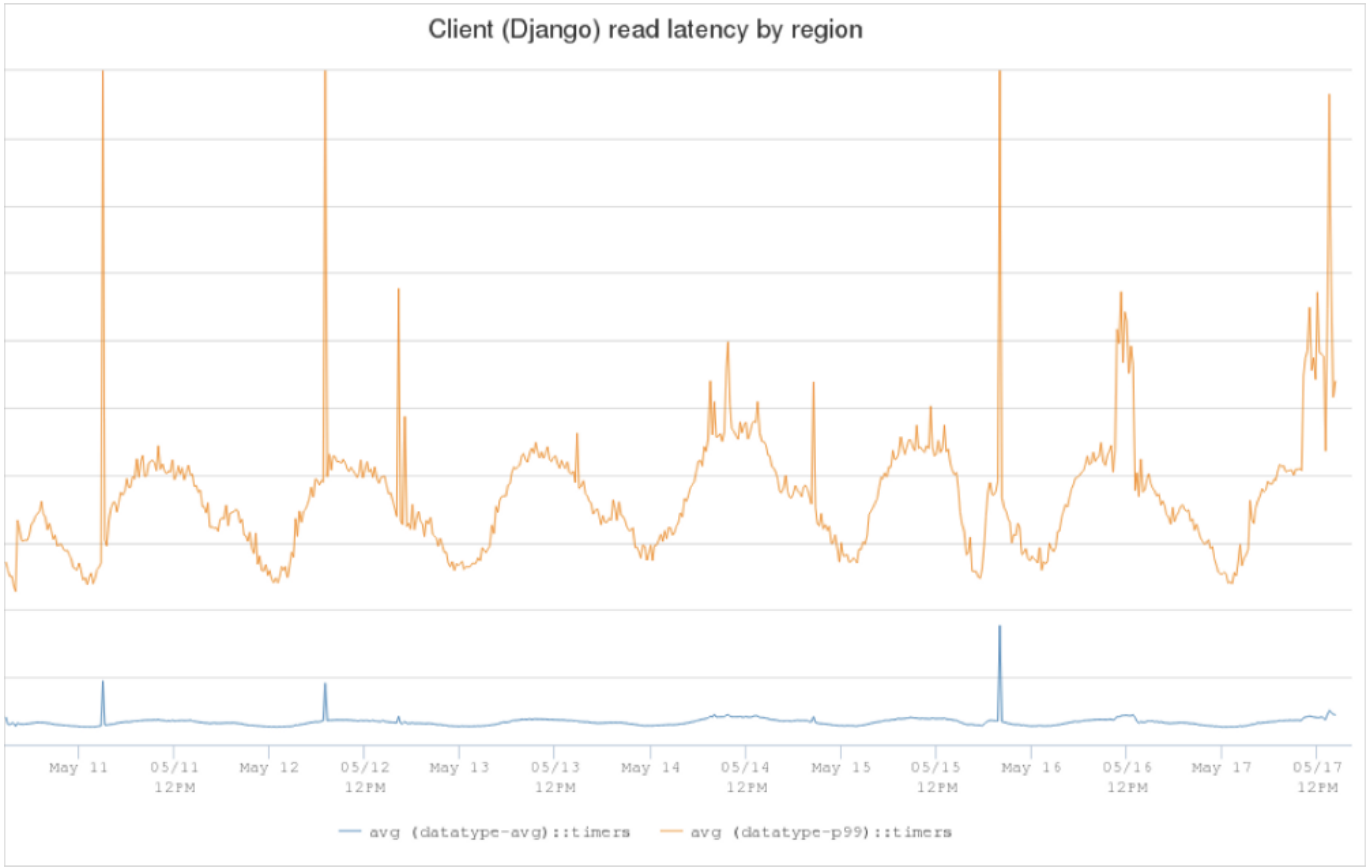
在 Instagram 上，我们大量使用 Apache Cassandra 作为通用键值存储服务。Instagram 的大多数 Cassandra 请求都是在线的，因此为了向数亿 Instagram 用户提供可靠且响应迅速的用户体验，我们有很高的 SLA 要求。

Instagram 可靠性 SLA 保持为5-9s，这意味着在任何给定时间，请求失败率应小于0.001%。为了提高性能，我们主动监控不同 Cassandra 集群的吞吐量和延迟，尤其是 P99 读取延迟。

下面的图显示了一个生产 Cassandra 集群的客户端延迟。蓝线是平均读取延迟（5ms），橙色线是 P99 读取延迟（在25ms到60ms的范围内）。



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

经过调查，我们发现 JVM 垃圾收集器（GC）对延迟峰值做出了很大贡献。我们定义了一个称为 GC 停滞百分比的度量，用于衡量Cassandra 服务器执行 GC（Young Gen GC）并且无法响应客户端请求的时间百分比。结果显示我们生产 Cassandra 服务器上的 GC 停滞百分比。在最低请求流量时间窗口期间为1.25%，在高峰时段可能高达2.5%。

这表明 Cassandra 服务器实例在垃圾收集上花费 2.5% 的运行时间，而这段时间是不能服务客户端请求的。GC 开销显然对我们的 P99 延迟有很大影响，因此如果我们可以降低 GC 停顿百分比，我们将能够显着降低P99延迟。

解决办法

Apache Cassandra 是一个分布式数据库，并用 Java 编写了基于 LSM 树的存储引擎。我们发现存储引擎中的组件，如memtable，压缩，读/写路径等，在 Java 堆中创建了很多对象，并为 JVM 产生了大量开销。为了减少来自存储引擎的GC影响，我们考虑了不同的方法，并最终决定开发C++存储引擎来替换现有的引擎。

我们不想从头开始构建新的存储引擎，因此我们决定在 RocksDB 之上构建新的存储引擎。

RocksDB 是一个开源的，高性能嵌入式键值数据库。它是用 C++ 编写的，并为 C++，C 和 Java 提供官方 API。RocksDB 针对性能进行了优化，尤其是在 SSD 等快速存储方面。它在业界广泛用作 MySQL，mongoDB 和其他流行数据库的存储引擎。

挑战

在 RocksDB 上实现新的存储引擎时，我们克服了三个主要挑战：

第一个挑战是 Cassandra 还没有可插拔的存储引擎架构，这意味着现有的存储引擎与数据库中的其他组件耦合在一起。为了在大规模重构和快速迭代之间找到平衡点，我们定义了一个新的存储引擎 API，包括最常见的读/写和流接口。这样我们就可以在 API 后面实现新的存储引擎，并将其注入 Cassandra 内部的相关代码路径。

其次，Cassandra 支持丰富的数据类型和表模式，而 RocksDB 提供纯粹的键值接口。我们仔细定义了编码/解码算法，使得我们使用 RocksDB 数据结构来构建 Cassandra 数据模型，并支持与原始 Cassandra 相同的查询语义。

第三个挑战是关于 streaming。 Streaming 在 Cassandra 这样的分布式数据库是很重要组件。每当我们从 Cassandra 集群加入或删除节点时，Cassandra 都需要在不同节点之间传输数据以平衡集群中的负载。现有的 streaming 实现基于当前存储引擎的。因此，我们必须将它们彼此分离，创建一个抽象层，并使用 RocksDB API 重新实现 streaming 传输。对于高 streaming 吞吐量，我们现在首先将数据流式传输到临时 sst 文件，然后使用 RocksDB 提取文件 API 立即将它们批量加载到 RocksDB 实例中。

性能指标

经过大约一年的开发和测试，我们已经完成了第一个版本的实现，并成功将其推广到 Instagram 中的几个生产 Cassandra 集群。在我们的一个生产集群中，P99读取延迟从60ms降至20ms。我们还观察到该集群上的GC停滞从2.5%下降到0.3%，这减少了10倍！

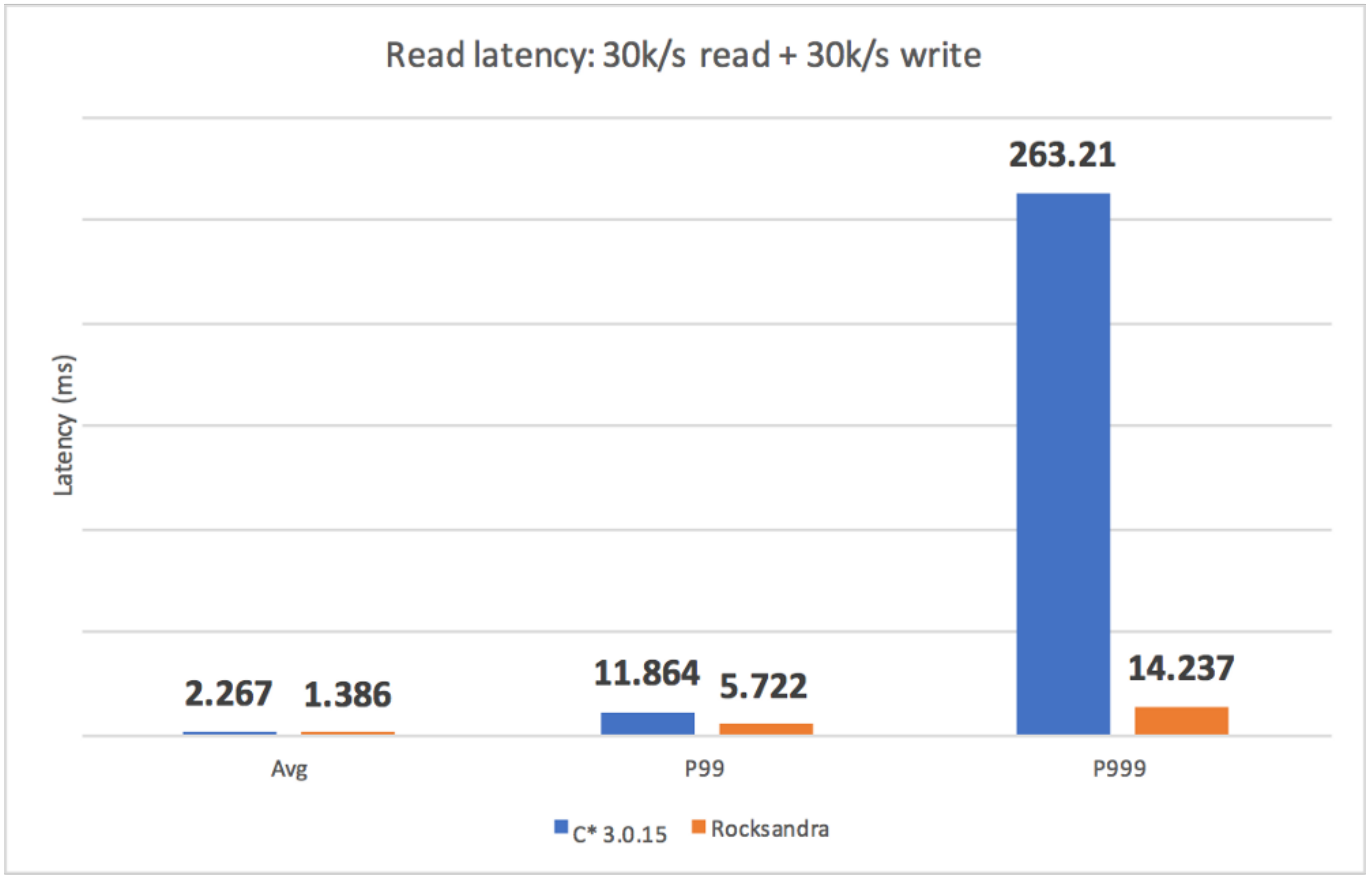
我们还想验证 Rocksandra 在公共云环境中是否表现良好。我们在 AWS 环境中使用三个 i3.8 xlarge EC2 实例部署了一个 Cassandra 集群，每个实例具有 32 个核，244GB内存和带有4个nvme闪存盘的raid0。

我们使用 NDBench 作为基准测试，并使用这个框架中的默认表模式：

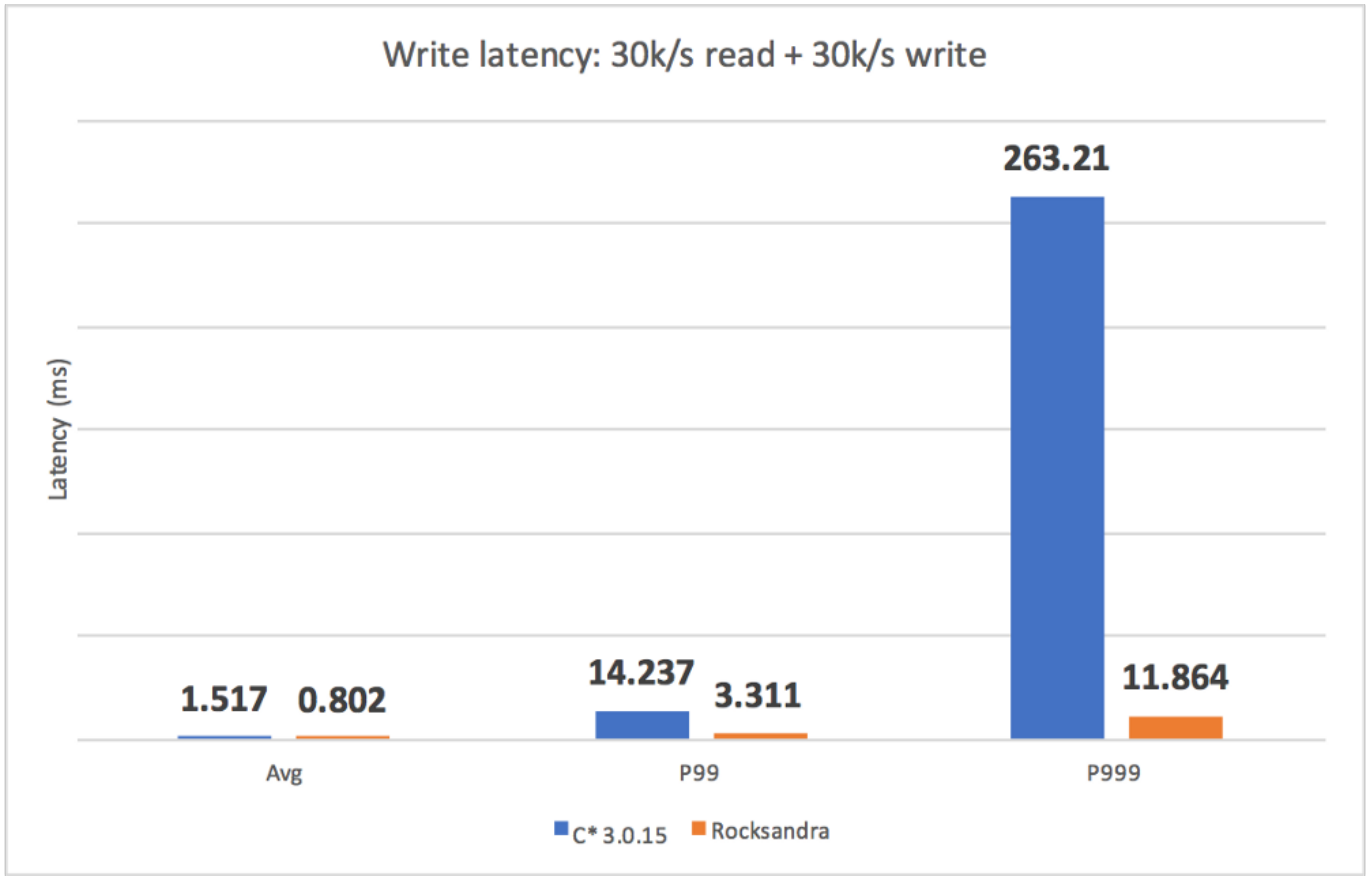
```
TABLE emp (
 emp_uname text PRIMARY KEY,
emp_dept text,
emp_first text,
emp_last text
 )
```

我们将 250M 6KB 行数据预先加载到数据库中（每个服务器在磁盘上存储大约500GB的数据）。我们在 NDBench 中配置了128个读客户端和128个写客户端。

我们测试了不同的工作负载并测量了 avg/P99/P999 读/写延迟。如我们所见，Rocksandra 提供了更低且一致的读/写延迟。

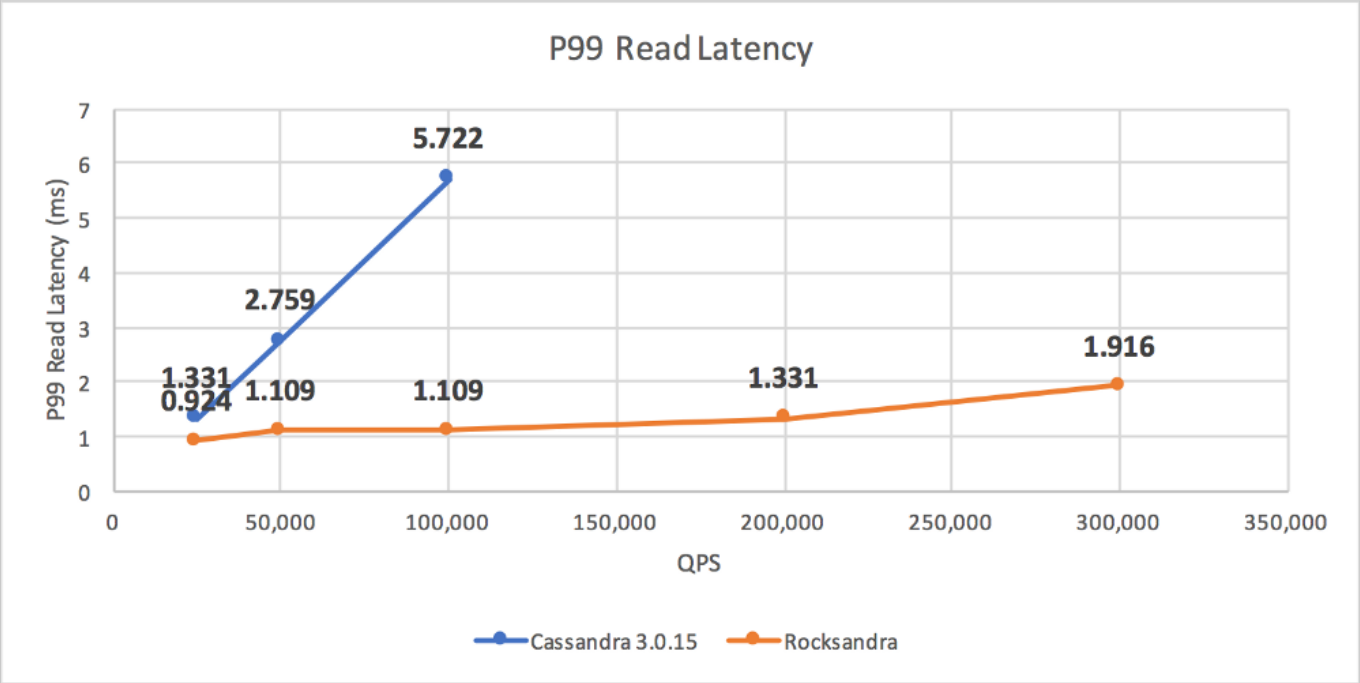


如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

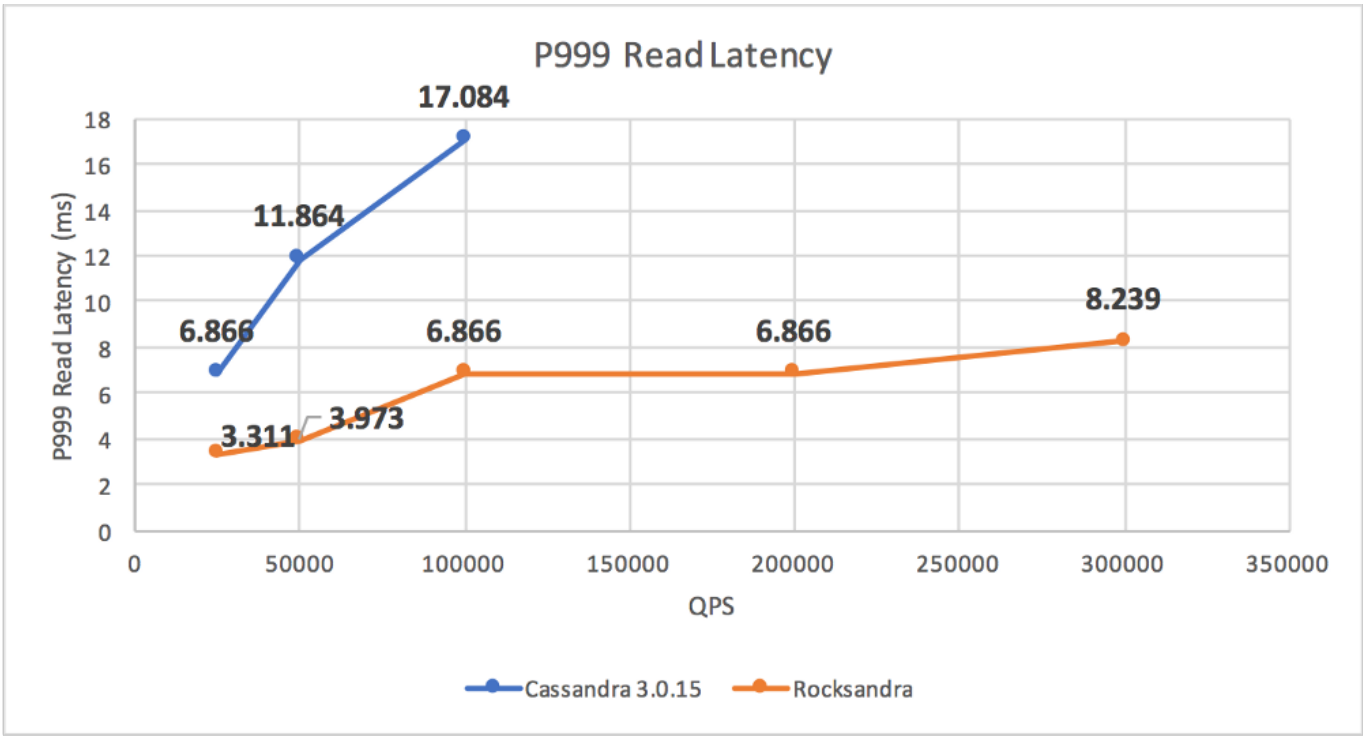


如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

我们还测试了一个只读工作负载并观察到，在类似的P99读取延迟（2ms）下，Rocksandra读取吞吐量提高了10倍（Rocksandra为300K/s，C * 3.0为 30K/s）。



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

未来工作

我们开源了 Rocksandra
代码库和基准框架，您可以从Github下载（
https://github.com/Instagram/cassandra/tree/rocks_3.0），在您自己的环境中试用！

下一步，我们正在积极开发更多 C* 功能，如二级索引，修复等。我们还在开发一个 C*
可插拔存储引擎架构，以便将我们的工作贡献给Apache Cassandra 社区。

本文翻译自：Open-sourcing a 10x reduction in Apache Cassandra tail latency

微信公众号和钉钉群交流

为了营造一个开放的 Cassandra 技术交流，我们建立了微信公众号和钉钉群，为广大用户提供专业的技术分享及问答，定期在国内开展线下技术沙龙，专家技术直播，欢迎大家加入。

微信公众号：

Cassandra技术社区

微信扫描二维码，关注我的公众号

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

钉钉群

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公众号：iteblog_hadoop

钉钉群入群链接：https://c.tb.cn/F3.ZRTY0o