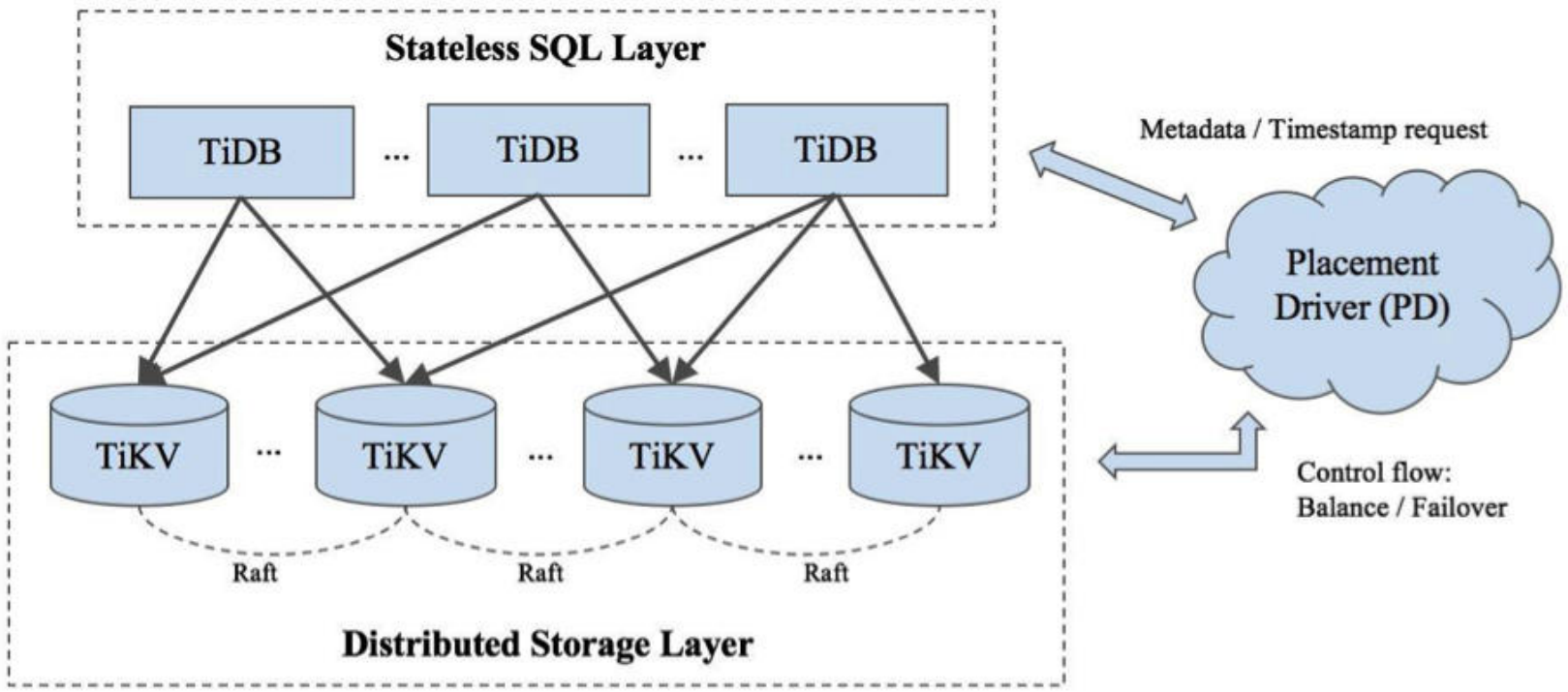
如何对分布式 NewSQL 数据库 TiDB 进行性能调优

在分布式系统中进行调优不是开玩笑的事情。分布式系统中调优比单节点服务器调优复杂得多，它的瓶颈可能出现在任何地方，单个节点上的系统资源，子组件，或者节点间的协作，甚至网络带宽这些都可能成为瓶颈。性能调优就是发现并解决这些瓶颈的实践，直到系统达到最佳性能水平。我会在本文中分享如何对 TiDB 的“写入”操作进行调优，使其达到最佳性能的实践。

TiDB 是开源的混合事务处理/分析处理（HTAP）的 NewSQL 数据库。一个 TiDB 集群拥有几个 TiDB 服务、几个 TiKV 服务和一组 Placement Deiver（PD）（通常 3-5 个节点）。TiDB 服务是无状态 SQL 层，TiKV 服务是键值对存储层，PD 则是管理组件，从顶层视角负责存储元数据以及负载均衡。下面是一个 TiDB 集群的架构，你可以在 TiDB 官方文档中找到每个组成部分的详细描述。



采集监控数据

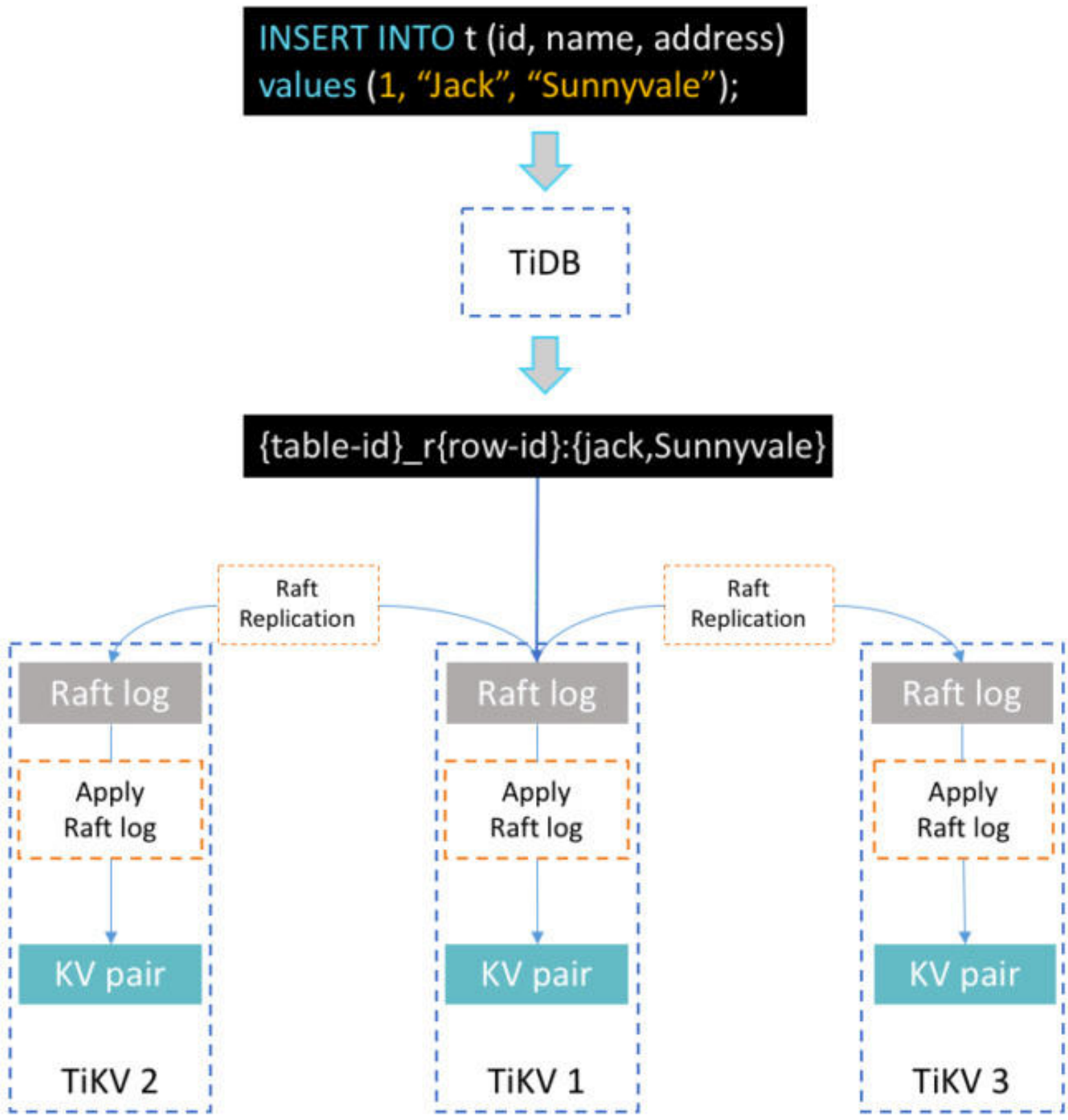
Prometheus 是一个开源的系统监测的解决方案，采集每个内部组件的监控数据，并定期发给 Prometheus 。借助开源的时序分析平台 Grafana ，我们可以轻易观测到这些数据的表现。使用 Ansible 部署 TiDB 时，Prometheus 和 Grafana 是默认安装选项。通过观察这些数据的变化，我们可以看到每个组

件是否处于运行状态，可以定位瓶颈所在，可以调整参数来解决问题。

插入 SQL 语句的写入流（Writeflow）

假设我们使用如下 SQL 来插入一条数据到表 t

```
mysql >> INSERT INTO t(id, name, address) values(1, “Jack”, “Sunnyvale”);
```



上面是一个简单而直观的简述，介绍了 TiDB 如何处理 SQL 语句。TiDB 服务器收到 SQL 语句后，根据索引的编号将语句转换为一个或多个键值对(KV)，这些键值对被发送到相关联的 TiKV 服务器，这些服务器以 Raft 日志的形式复制保存。最后，Raf 日志被提交，这些键值对会被写入指定的存储

引擎。

在此过程中，有 3 类关键的过程要处理：转换 SQL 为多个键值对、Region 复制和二阶段提交。接下来让我们深入探讨各细节。

从 SQL 转换为键值对

与其他数据库系统不同，TiDB 只存储键值对，以提供无限的水平可伸缩性以及强大的一致性。那么要如何实现诸如数据库、表和索引等高层概念呢？在 TiDB 中，每个表都有一个关联的全局唯一编号，被称为“table-id”。特定表中的所有数据（包括记录和索引）的键都是以 8 字节的 table-id 开头的。每个索引都有一个名为“index-id”的表范围的唯一编号。下面展示了记录键和索引键的编码规则。

```
record key: {table-id}_r{row-id} -> record value: {encoded row content}
index key: {table-id}_i{index-id}{index-content} -> index value: {row-id}
```

Region（区域）的概念

在 TiDB 中，Region 表示一个连续的、左闭右开的键值范围 [start_key，end_key）。每个 Region 有多个副本，并且每个副本称为一个 peer 。每个 Region 也归属于单独的 Raft 组，以确保所有 peer 之间的数据一致性。（有关如何在 TiKV 中实现 Raft 一致性算法的更多信息，请参阅 PingCAP 杰出工程师唐刘的相关博文。）由于我之前提到的编码规则的原因，同一表的临近记录很可能位于同一 Region 中。

当集群第一次初始化时，只存在一个 Region 。当 Region 达到特定大小（当前默认值为96MB）时， Region 将动态分割为两个邻近的 Region ，并自动将数据分布到系统中以提供水平扩展。

二阶段提交

我们的事务处理模型设计灵感来源于 Percolator，并在此基础上进行了一些优化。简单地说，这是一个二阶段提交协议，即预写入和提交。

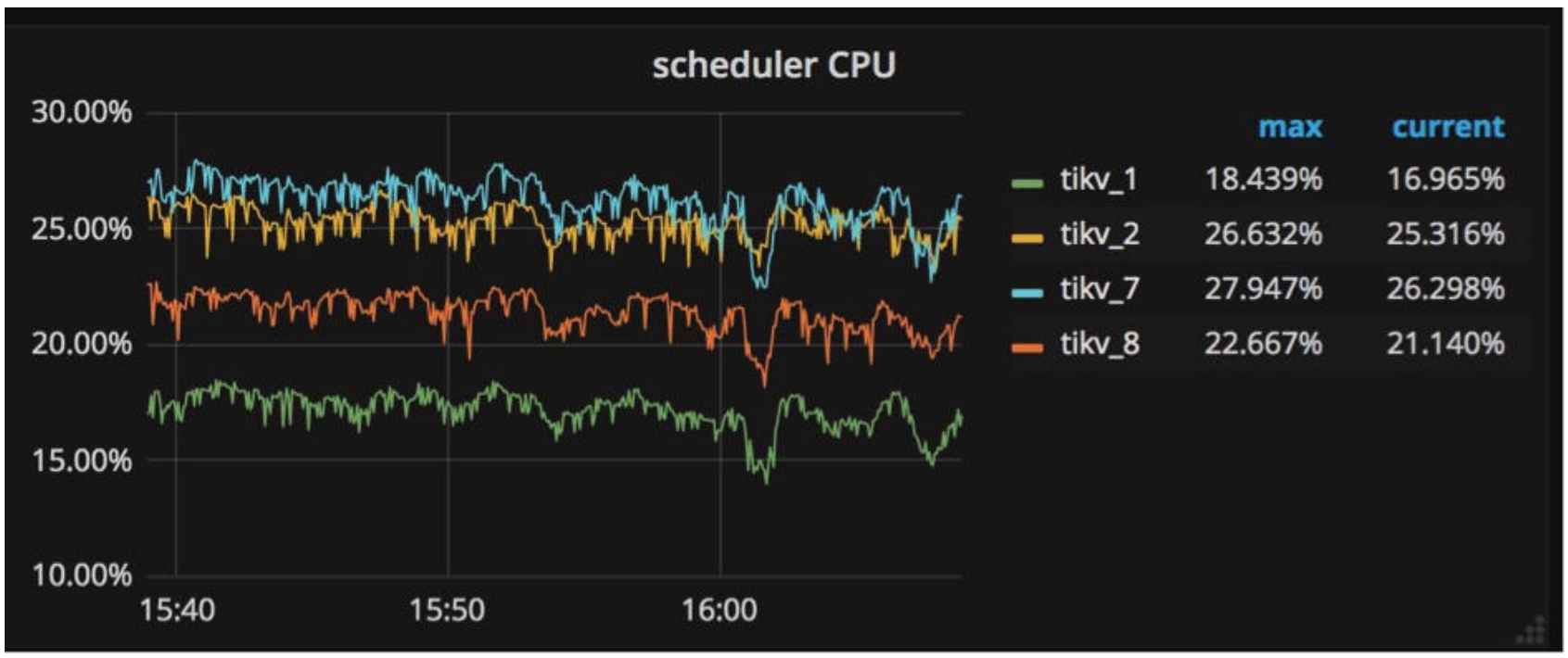
每个组件中都有更多的内容，但从宏观层次来理解足以为性能调优设置场景。现在我们来深入研究四种调优技术。

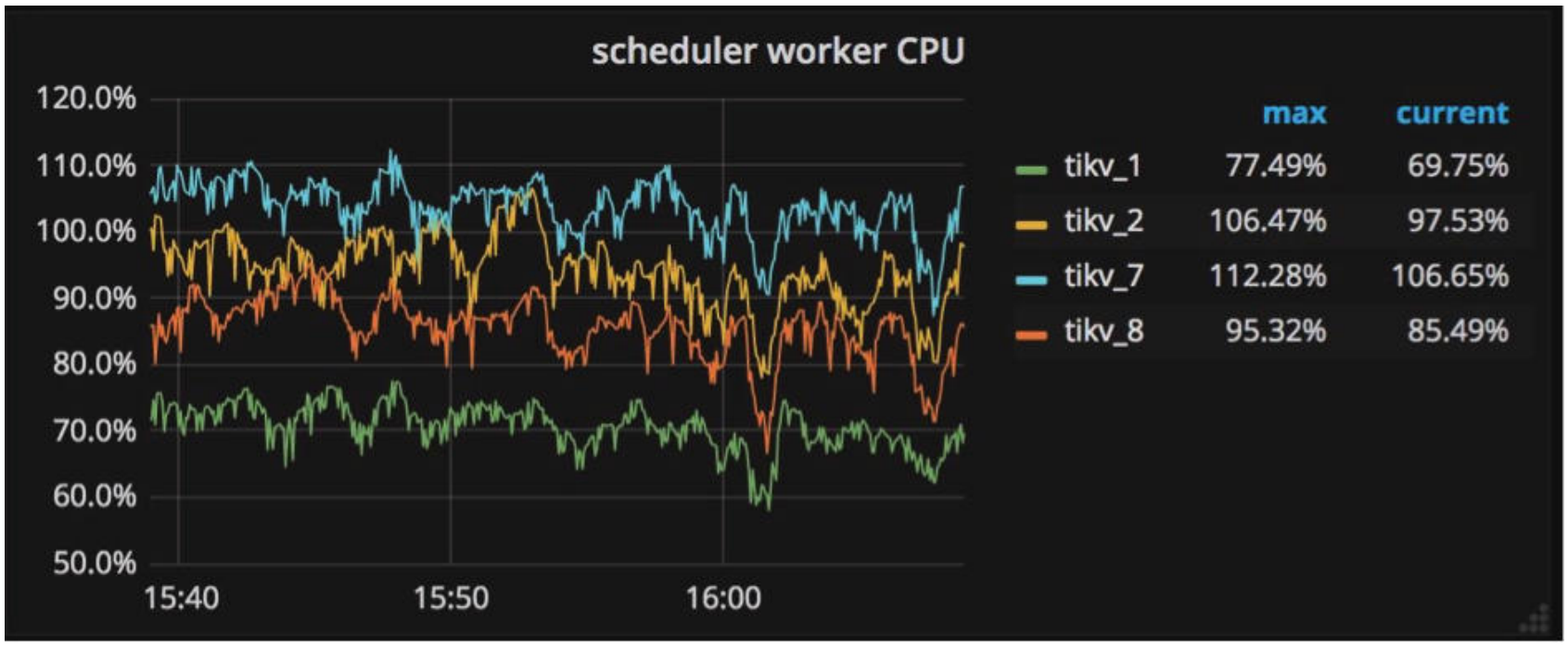
调优技巧 #1: 调度器

所有写入命令都被发送到调度器模型，然后被复制。调度器模型由一个调度线程和几个工作线程组成。为什么需要调度器模型？在向数据库写入数据之前，需要检查是否允许这些写命令，以及这些写命令是否满足事务约束。所有这些检查工作都需要从底层存储引擎读取信息，它们通过调度由工作线程来进行处理。

如果看到所有工作线程的 CPU 使用量总和超过 scheduler-worker-pool-size * 80% 时，就需要通过增加调度工作线程的数理来提高性能。

可以通过修改配置文件中 'storage' 节的 'scheduler-worker-pool-size' 来改变调度工作线程的数量。对于 CPU 核心数目小于 16 的机器，默认情况下配置了 4 个调度工作线程，其它情况下默认值是 8。参阅相关代码部分：scheduler-worker-pool-size = 4





调优技巧 #2：raftstore进程与apply进程

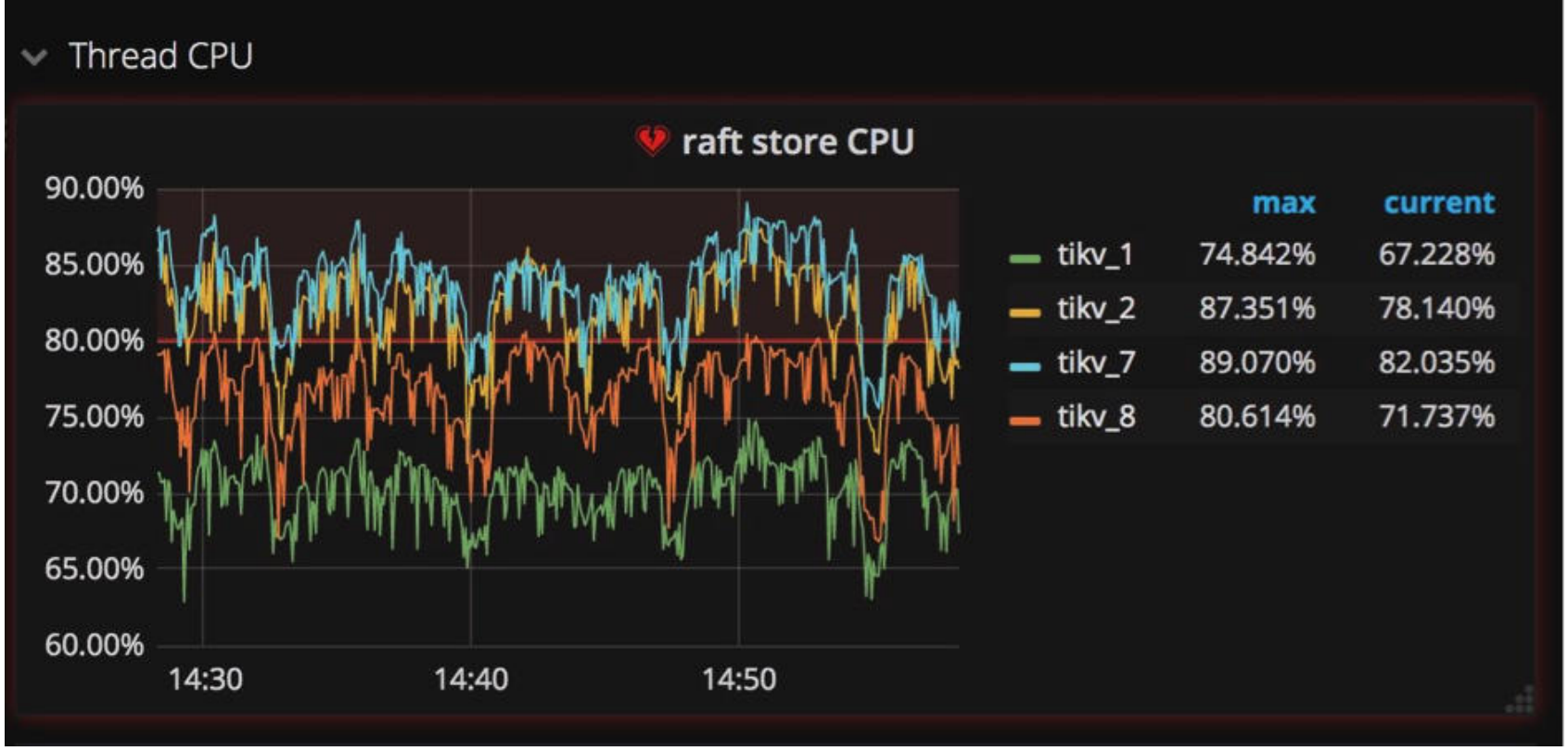
像我前边提到的，我们在多节点之间使用Raft实现强一致性。在将一个键值对写入数据库之前，这个键值对首先要被复制成Raft log格式，同时还要被写入各个节点硬盘中保存。在Raft log被提交后，相关的键值对才能被写入数据库。

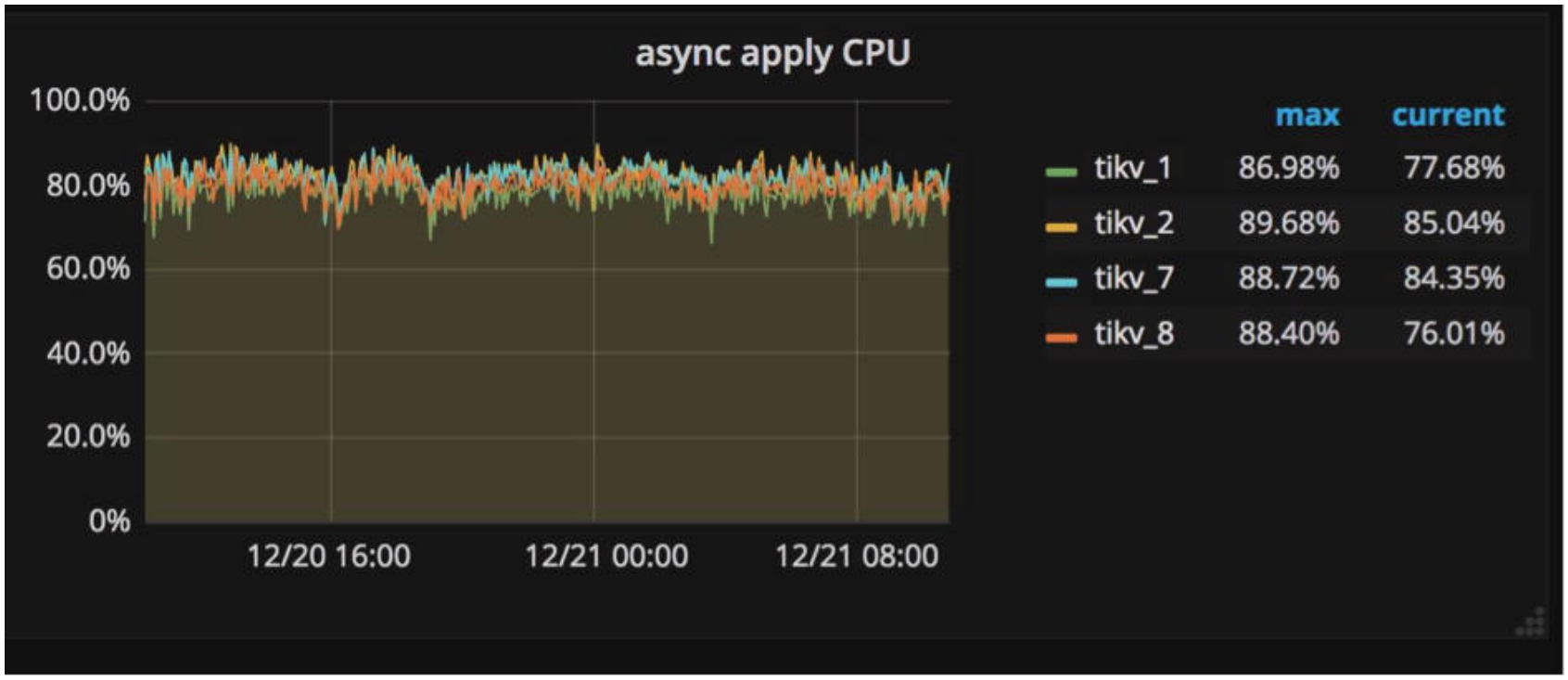
这样就产生两种写入操作：一个是写Raft log，一个是把键值对写入数据库。为了在TiKV中独立地执行这两种操作，我们创建一个raftstore进程，它的工作是拦截所有Raft信息，并写Raft log到硬盘中；同时我们创建另一个进程apply worker，它的职责是把键值对写到数据库中。在Grafana中，这两个进程显示在TiKV面板的子面板Thread CPU中（如下图所示）。它们都是极其重要的写操作负载，在Grafana中我们很容易就能发现它们相当繁忙。

为什么需要特别关注这两个进程？当一些TiKV服务器的apply或者raftstore进程很繁忙，而另一些机器却很空闲的时候，也就是说写操作负载不均衡的时候，这些比较繁忙的服务器就成了集群中的瓶颈。造成这种情况的一种原因是使用了单调递增的列，比如使用AUTOINCREMENT指定主键，或者在值不断增加的列上创建索引，例如最后一次访问的时间戳。

要优化这样的场景并消除瓶颈，必须避免在单调增加的列上设计主键和索引。

在传统单节点数据库系统上，使用AUTOINCREMENT关键字可以为顺序写入带来极大好处，但是在分布式数据库系统中，使所有组件的负载均衡才是最重要的。





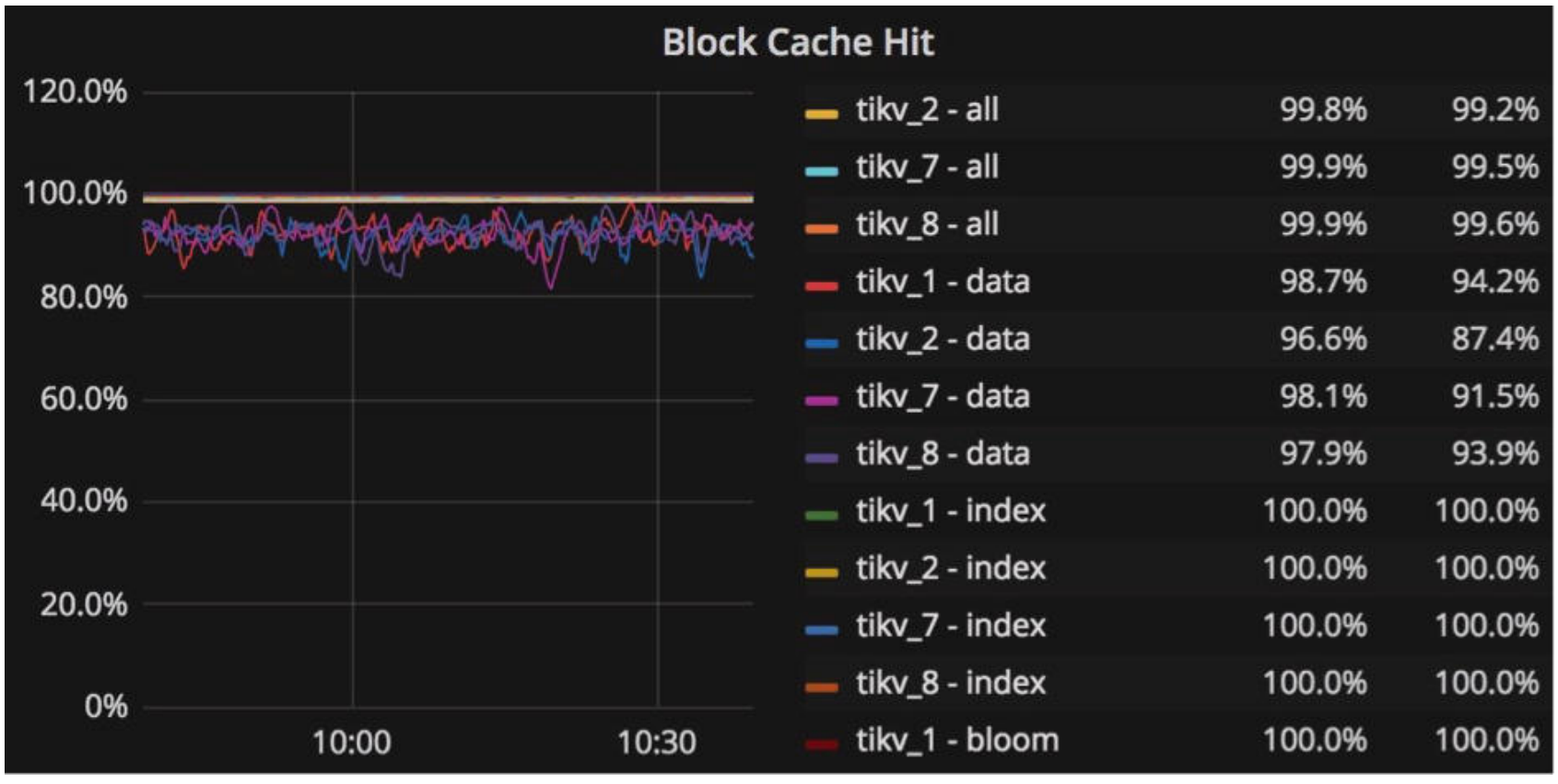
调优技巧 # 3：RocksDB

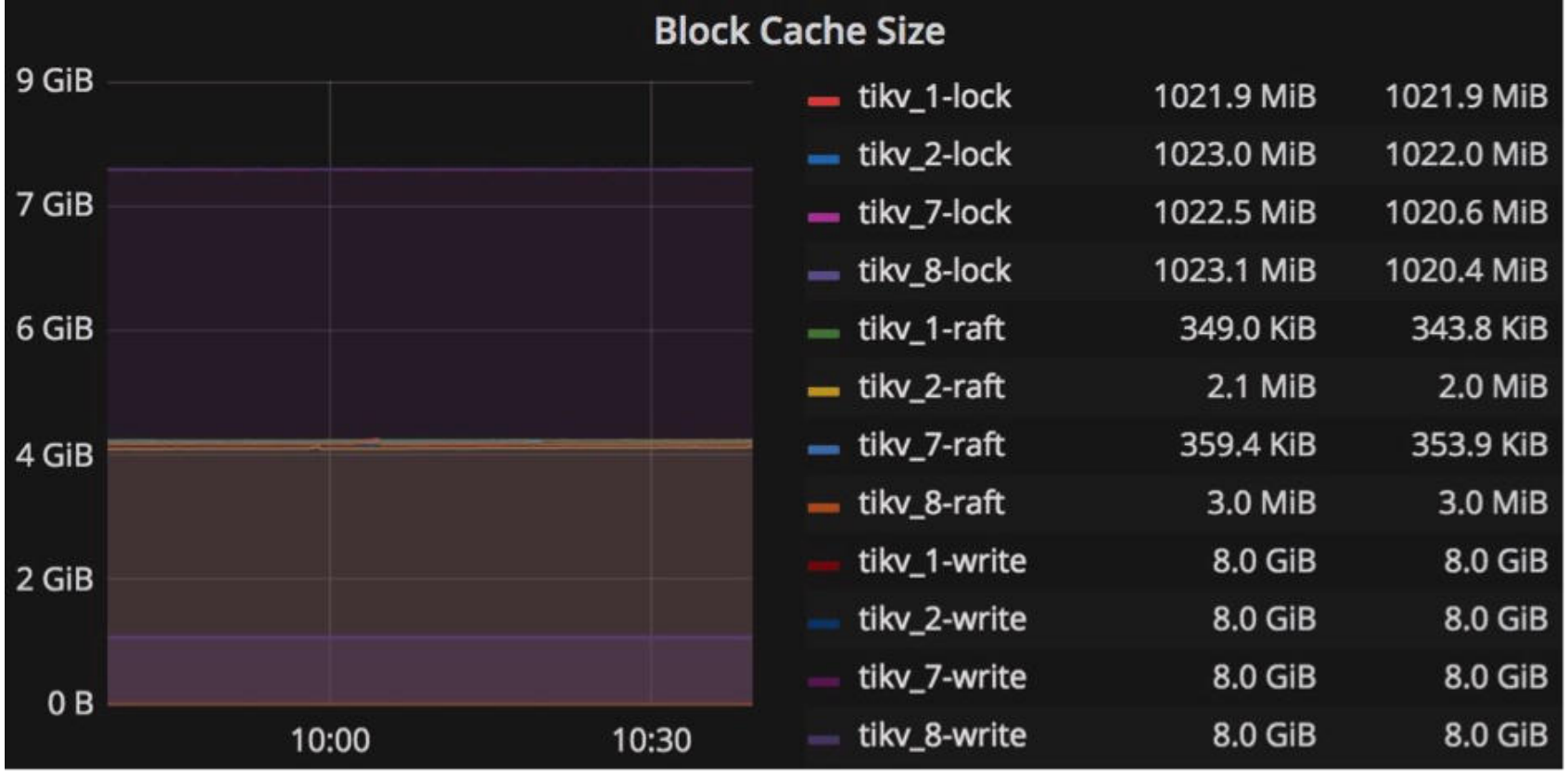
RocksDB 是一个高性能，有大量特性的永久性 KV 存储。 TiKV 使用 RocksDB 作为底层存储引擎，和其他诸多功能，比如列族、范围删除、前缀索引、memtable 前缀布隆过滤器，sst 用户定义属性等等。 RocksDB 提供详细的性能调优文档。

每个 TiKV 服务器下面都有两个 RocksDB 实例：一个存储数据，我们称之为 kv-engine ，另一个存储 Raft 日志，我们称之为 raft-engine 。kv-engine 有4个列族：“default”、“lock”、“write”和“raft”。大多数记录存储在“default”列族中，所有索引都存储在“write”列族中。你可以通过修改配置文件关联部分中的 block-cache-size 值来调整这两个 RocksDB 实例，以实现最佳性能。相关部分是： [rocksdb.defaultcf] block-cache-size = “1GB” 和

[rocksdb.writecf] block-cache-size = “1GB”

我们调整 block-cache-size 的原因是因为 TiKV 服务器频繁地从“write”列族中读取数据以检查插入时是否满足事务约束，所以为“write”列族的块缓存设置合适的大小非常重要。当“write”列族的 block-cache 命中率低于 90% 时，应该增加“write”列族的 block-cache-size 大小。“write”列族的 block-cache-size 的默认值为总内存的 15% ，而“default”列族的默认值为 25% 。例如，如果我们在 32GB 内存的机器上部署 TiKV 节点，那么对于“default”列族，“write”列族的 block-cache-size 的值约为 4.8GB 到 8GB 。在繁重的写入工作量中，“default”列族中的数据很少被访问，所以当我们确定“write”列族的缓存命中率低于 90% （例如50%）时，我们知道“write”列族大约是默认 4.8 GB的两倍。为了调优以获得更好的性能，我们可以明确地将“write”列族的 block-cache-size 设置为 9GB 。但是，我们还需要将“default”列族的 block-cache-size 大小减少到 4GB ，以避免 OOM（内存不足）风险。你可以在 Grafana 的“RocksDB-kv”面板中找到 RocksDB 的详细统计信息，以帮助进行调优。





RocksDB-kv 面板

调优技巧#4: 批量插入

使用批量插入可以实现更好的写入性能。从 TiDB 服务器的角度来看，批量插入不仅可以减少客户端与 TiDB 服务器之间的 RPC 延迟，还可以减少 SQL 解析时间。在 TiKV 内部，批量插入可以通过将多个记录合并到一个 Raft 日志条目中来减少 Raft 信息的总数量。根据我们的经验，建议将批量大小保持在 50～100 行之内。当一个表中有超过 10 个索引时，应减少批量处理的大小，因为按照上述编码规则，插入一行类似数据将创建超过 10 个键值对。

总结

我希望本文能够在使用 TiDB 时帮助你了解一些常见的瓶颈状况，以及如何调优这些问题，以便在“写入”过程中实现最优性能。综上所述：

1. 不要让一些 TiKV 节点处理大部分“写入”工作负载，避免在单调增加的列上设计主键和索引。
2. 当 TiKV 调度模型中的调度器的总 CPU 使用率超过 scheduler-worker-pool-size*80％ 时，请增加 scheduler-worker-pool-size 的值。
3. 当写入任务频繁读取’write’列族并且块缓存命中率低于 90％ 时，在 RocksDB 中增加 block-cache-size 的值。

4. 使用批量插入来提高“写入”操作的性能。

我们的许多客户，从电子商务市场和游戏，到金融科技、媒体和旅行，已经在生产中使用这些调优技术，以充分利用 TiDB 的设计、体系结构和优化。期待在不久的将来分享他们的使用案例和经验。