本小节将介绍在单区域部署并拓展到多区域部署的CockroachDB在提高读写性能方面的几个关键技术。

用户若需要更全面的调优建议，可以查看[SQL Performance Best Practices](http://doc.cockroachchina.baidu.com/#develop/performance-optimization/sql-best-practices/)

概述

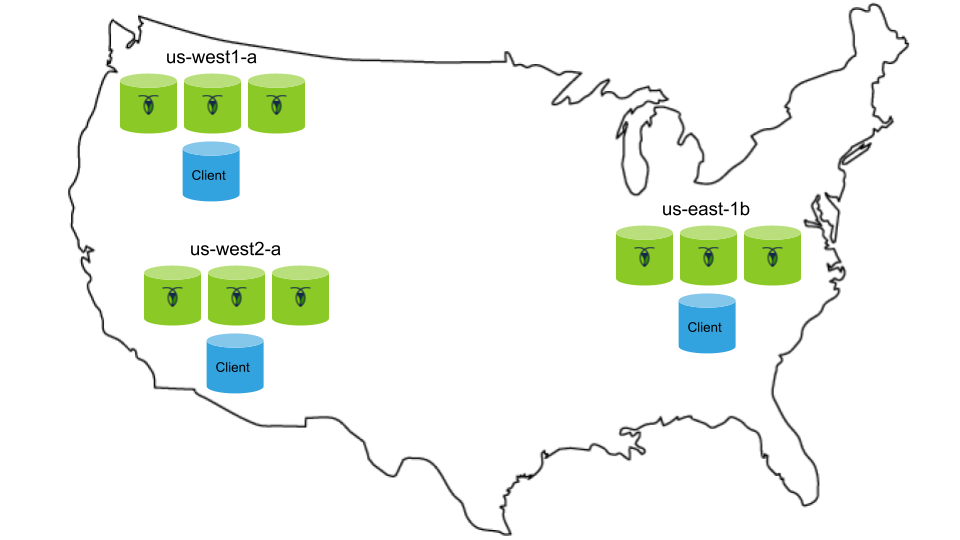
拓扑结构

本小节将在同一个GCE zone部署一个3节点集群，并配置一个额外的机器用户执行客户端应用程序负载。



**NOTE:** 在一个GCE zone内，机器之间的网路延迟需要是亚毫秒级别。

随后将拖航集群到跨3个GCE区域的9个节点，并且每个区域配置一台额外的机器执行客户端应用程序负载。

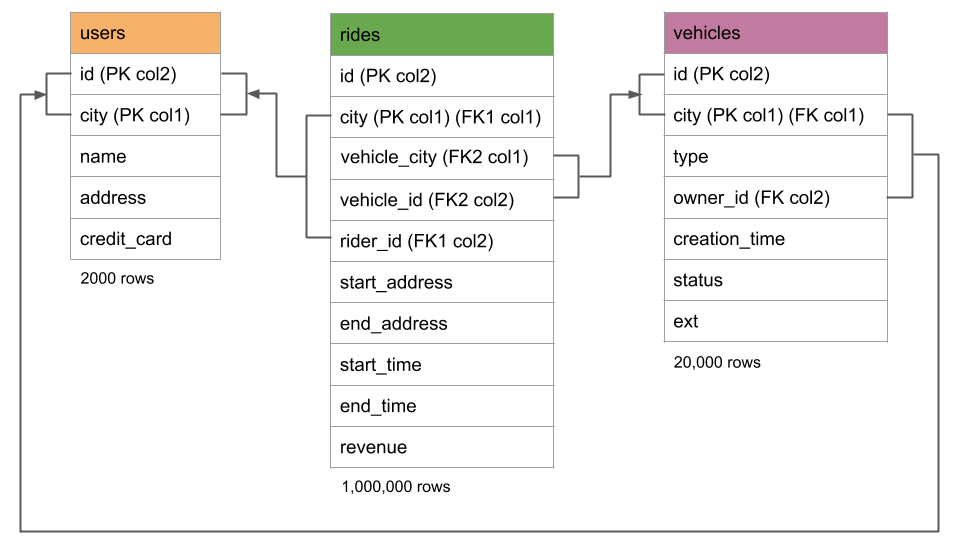


用户若需要复现本小节的性能实验结果，则：

* 对于每个CockroachDB节点，需要使用[n1-standard-4](https://cloud.google.com/compute/docs/machine-types#standard_machine_types)机器类型，其配置为4 vCPUs和15 GB内存，操作系统为Ubuntu 16.04，并挂载一个[本地SSD](https://cloud.google.com/compute/docs/disks/#localssds)硬盘。
* 执行客户端应用程序负载的机器，则需要更小的机器配置，如n1-standard-1。

Schema

Schema和数据将基于虚构的点对点共享汽车应用MovR设计的，其特性可查看[CockroachDB 2.0 demo](https://www.youtube.com/watch?v=v2QK5VgLx6E)

 注意事项：

* 有三张表，users代表注册服务的人员，vehicles代表服务的车辆，rides代表用户参与的服务时间和地点。
* 每个表都有一个复合主键，其中city作为第一个键。在单区域部署中复合主键不是必需的，但将集群扩展到多个区域时，这些复合主键可以用来根据city对[数据进行行级别的地理分区](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/#partition-using-primary-key)。本小节使用了为方便将来集群扩容而设计的schema。
* 用户导入数据时使用的IMPORT功能不支持外键，因此导入的数据不受外键约束。因而，导入之后会对需要添加外键约束的列创建必要的二级索引。
* rides表包含city字段和看似多余的vehicle\_city字段。这种冗余是有必要的，考虑到不可能将多个外键约束应用于单个列。尽管如此，用户还是需要将两个外键约束应用于rides表，并且每个约束都需要city字段作为约束的一部分。此时，冗余的vehicle\_city字段能够帮助突破[该限制](https://github.com/cockroachdb/cockroach/issues/23580)，并通过[CHECK约束](https://www.cockroachlabs.com/docs/stable/check.html)与city字段保持同步。

重要概念

理解本小节中的技术，并且能够在自己的场景中应用它们，首先需要回顾一些重要的[CockroachDB架构上的概念](https://www.cockroachlabs.com/docs/stable/architecture/overview.html)：

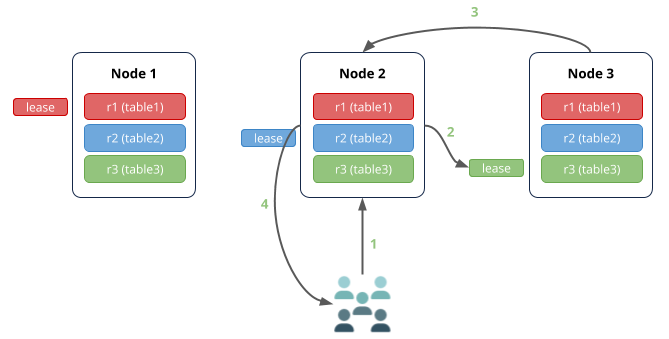
| **英文** | **中文** | **简介** |
| --- | --- | --- |
| Cluster | 集群 | 部署的内容，在逻辑上视为一个独立的应用。 |
| Node | 节点 | 运行CockroachDB的一台单独的机器，多个节点组成一个集群。 |
| Range | - | CockroachDB所有用户数据（包括数据表、索引等等）和几乎所有的系统数据都存储在一个巨大有序的key-value集合。根据连续的key划分成多个区间，每个区间是一个Range，因此每个key只会出现在一个range中。  从SQL的角度来说，一个数据表和对应的二级索引最初只有一个Range，Range中的每个key-value对表示表（因为表是按主键有序的，其也被叫做主键索引）或者二级索引的单行。当数据增长超过64MiB的时候，单个Range会分裂成2个Range，并随着数据继续增多，持续进行分裂。 |
| Replica | 冗余 | CockroachDB对每个Range做数据冗余（默认情况下是3份副本），并分别存储在不同的节点上。 |
| Leaseholder | 租约持有者 | 一个Range的所有副本当中只有一个副本持有Range租约，持有者接收和协调关于该Range的所有读写请求。  与写请求不同，读请求访问租约持有者并将结果返回给客户端，而无需与任何Range其他副本协调，能够有效地减少网络开销。所有写请求都会发送给租约持有者，因此保证了租约持有者数据是最新的。 |
| Raft Leader | Rafter领袖 | 对于每个Range，只有一个副本能够作为协调处理写请求的“领袖”。通过[Raft一致性协议](https://www.cockroachlabs.com/docs/stable/architecture/replication-layer.html#raft)，副本内容的更新需要在提交写入之前，基于Raft日志获得包括领袖的绝大多数副本的同意。绝大多数情况下，Raft领袖和租约持有者为同一副本。 |
| Raft Log | Raft日志 | 按Range变更时间有序的、得到Range副本认可的日志。日志存储在每个副本所在机器的硬盘上，是该Range一致性数据冗余的可信来源。 |

如上所述，在执行查询时，集群将请求路由到包含相关数据Range的租约持有者。如果查询涉及多个Range，则请求将转发到多个租约持有者。对于读请求，只有相关Range的租约持有者可以读取到数据。对于写请求，Raft一致性协议保证了在写入提交之前相关Range的绝大多数副本必须达成一致。

读场景

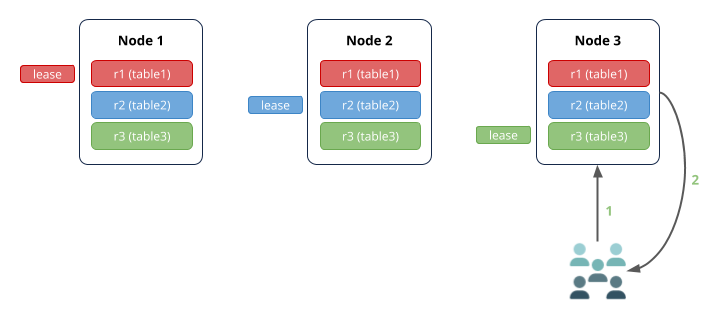
读场景集群设定如下：

* 集群由3个节点构成
* 数据库有3张小表，每张表只有一个Range的数据
* Range数据冗余3份（默认情况下）
* 查询在节点2执行，读取表3的数据



此时：

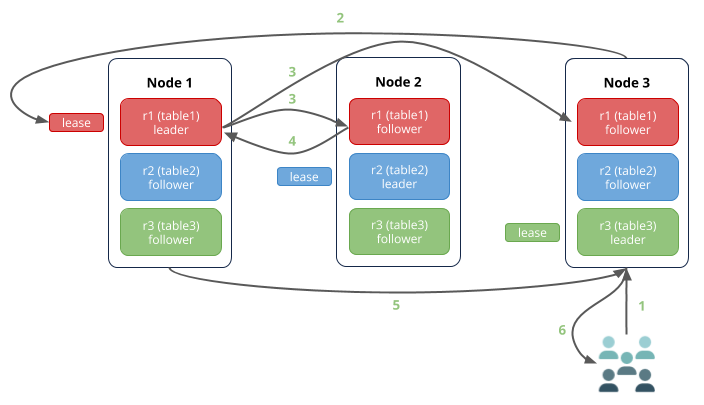
* 节点2（网关节点）接收到需要查询表3数据的请求。
* 节点3是表3的租约持有者，因此需要将请求路由到节点3。
* 节点3将数据返回给节点2。
* 节点2将查询结果返回给客户端。



如果接收、处理查询请求的节点刚好也是相关Range的租约持有者，那么此时查询处理的网络路由代价会小很多。

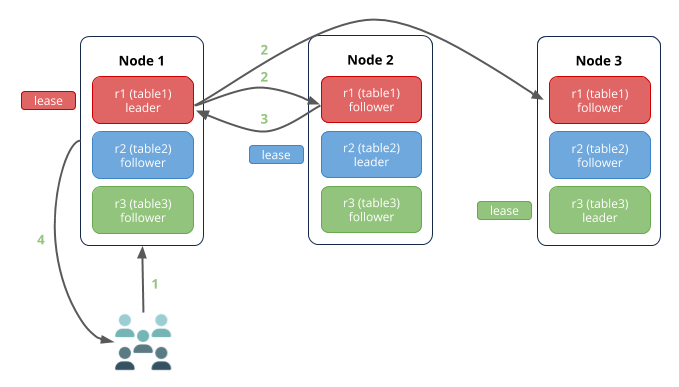
写场景

现在设想一个简单的写场景，节点3处理对表1写入数据的查询请求。



此时：

* 节点3（网关节点）接收对表1写入数据的写请求。
* 表1的租约持有者位于节点1，写请求需要路由到该节点。
* 通常情况下Raft领袖和租约持有者为同一机器，所以节点1将写操作记录追加到Raft日志、同时通知节点2和节点3上的Raft跟随者。
* 一旦一个Raft跟随者将写操作记录追加到其Raft日志（基于相同的Raft日志，绝大多数副本同意），就会通知Raft领袖，并且经过一致同意的键值写入操作会被提交。上图中，位于节点2的跟随者确认了写入，同时节点3上的跟随者也可能确认写入。此外未参与一致性投票的跟随者通常会在随后很快地同步写入。
* 节点1向节点3确认提交。
* 节点3返回结果给客户端。

 正如读场景一样，如果接收、处理查询的节点刚好也是相关Range的租约持有者，那么此时查询处理的网络路由代价会小很多。

网络和IO瓶颈

结合上述示例，有必要将网络延迟和磁盘I/O视为潜在的性能瓶颈：

* 对于读场景，网关节点和租约持有者之间的每次跳跃（hop）都会增加延迟。
* 对于写场景，网关节点和租约持有者/Raft领袖之间的跳跃，以及租约持有者/Raft领袖和Raft跟随者之间的跳跃，都会增加延迟。此外由于Raft日志条目在提交之前会被持久化存储到硬盘，因此也需要考虑磁盘的I/O时延。

单区域部署

Step 1: 配置网络

CockroachDB需要以下2个端口供TCP连接访问：

* 26257用于节点之间访问
* 8080用于开放Admin界面

由于GCE实例默认使用内部IP地址通信，内部节点之间访问不需要额外的配置。然而，如果想要从本地网络访问Admin界面，则必须要[创建防火墙规则](https://cloud.google.com/vpc/docs/using-firewalls)。

| **配置项** | **推荐值** |
| --- | --- |
| Name | **cockroachweb** |
| Source filter | IP地址范围 |
| Source IP ranges | 用户本地网络IP地址范围 |
| Allowed protocols | **tcp:8080** |
| Target tags | cockroachdb |

**NOTE:** **tag**功能能够帮助用户快速应用规则到对应机器上。

Step 2: 创建机器

在us-east1-bGCE zone里构造3节点集群，同时创建一台额外的机器用于运行客户端应用程序负载。

* 创建3台机器，对于每台机器：
* 选择us-east1-bzone
* 使用n1-standard-4机器类型，配置4 vCPUs和15GiB内存
* 使用Ubuntu 16.04系统
* [创建和挂载本地SSD](https://cloud.google.com/compute/docs/disks/local-ssd#create_local_ssd)
* 尽早的应用已配置的防火墙规则，点击**Management**、 **disk**、 **networking**、**SSH keys**，选择**Networking**标签页、并在**Networking tags**配置项填入cockroachdb
* 获得每台n1-standard-4机器的内网IP地址，用于集群节点启动。
* 在us-east1-bzone里，创建一台单独的机器用于运行客户端应用负载。其可以选择配置弱一点的机器，例如n1-standard-1。

Step 3: 创建3节点集群

对前三台n1-standard-4机器，分别执行操作：

* SSH登陆机器
* 下载CockroachDB Linux版本 [CockroachDB可执行文件](https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz)，解压并拷贝到PATH

wget -qO- https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz | tar xvz

sudo cp -i cockroach-v2.0.5.linux-amd64/cockroach /usr/local/bin

* 执行[cockroach start](http://doc.cockroachchina.baidu.com/#deploy/start-a-node/)命令

cockroach start --insecure --advertise-host=<node internal address> --join=<node1 internal address>:26257,<node2 internal address>:26257,<node3 internal address>:26257 --locality=cloud=gce,region=us-east1,zone=us-east1-b --cache=.25 --max-sql-memory=.25 --background

在任意n1-standard-4机器上，执行cockroach init命令

cockroach init --insecure --host=localhost

每个节点的CockroachDB将输出详细的帮助信息到[标准输出](http://doc.cockroachchina.baidu.com/#deploy/start-a-node/#standard-output)，例如CockroachDB版本、Web界面的URL地址、客户端SQL URL等。

Step 4: 导入Movr数据集

用户将导入美国3个东部城市（纽约，波士顿和华盛顿特区）和3个西部城市（洛杉矶，旧金山和西雅图）的Movr数据，包括用户、车辆和行程信息。

* SSH到第四个节点，这个节点没有运行CockroachDB进程
* 下载CockroachDB Linux版本[CockroachDB可执行文件](https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz)，解压并拷贝到PATH

wget -qO- https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz | tar xvz

sudo cp -i cockroach-v2.0.5.linux-amd64/cockroach /usr/local/bin

* 启动[内置SQL shell](http://doc.cockroachchina.baidu.com/#quick-start/learn-cockroachdb-sql/use-the-built-in-sql-client/)，连接CockroachDB集群其中一个节点

cockroach sql --insecure --host=<address of any node>

* 创建movr数据库并设置为默认数据库

**CREATE** **DATABASE** movr;

**SET** **DATABASE** = movr;

* 使用[IMPORT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/IMPORT/)原语，创建users、vehicles、riders表并导入数据

IMPORT TABLE users (

id UUID NOT NULL,

city STRING NOT NULL,

name STRING NULL,

address STRING NULL,

credit\_card STRING NULL,

CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC)

)

CSV DATA (

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/users/n1.0.csv'

);

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| job\_id | status | fraction\_completed | rows | index\_entries | system\_records | bytes |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| 370636591722889217 | succeeded | 1 | 0 | 0 | 0 | 0 |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

(1 row)

Time: 3.409449563s

IMPORT TABLE vehicles (

id UUID NOT NULL,

city STRING NOT NULL,

type STRING NULL,

owner\_id UUID NULL,

creation\_time TIMESTAMP NULL,

status STRING NULL,

ext JSON NULL,

mycol STRING NULL,

CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC),

INDEX vehicles\_auto\_index\_fk\_city\_ref\_users (city ASC, owner\_id ASC)

)

CSV DATA (

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/vehicles/n1.0.csv'

);

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| job\_id | status | fraction\_completed | rows | index\_entries | system\_records | bytes |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| 370636877487505409 | succeeded | 1 | 0 | 0 | 0 | 0 |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

(1 row)

Time: 5.646142826s

IMPORT TABLE rides (

id UUID NOT NULL,

city STRING NOT NULL,

vehicle\_city STRING NULL,

rider\_id UUID NULL,

vehicle\_id UUID NULL,

start\_address STRING NULL,

end\_address STRING NULL,

start\_time TIMESTAMP NULL,

end\_time TIMESTAMP NULL,

revenue DECIMAL(10,2) NULL,

CONSTRAINT "primary" PRIMARY KEY (city ASC, id ASC),

INDEX rides\_auto\_index\_fk\_city\_ref\_users (city ASC, rider\_id ASC),

INDEX rides\_auto\_index\_fk\_vehicle\_city\_ref\_vehicles (vehicle\_city ASC, vehicle\_id ASC),

CONSTRAINT check\_vehicle\_city\_city **CHECK** (vehicle\_city = city)

)

CSV **DATA** (

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.0.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.1.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.2.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.3.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.4.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.5.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.6.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.7.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.8.csv',

'https://s3-us-west-1.amazonaws.com/cockroachdb-movr/datasets/perf-tuning/rides/n1.9.csv'

);

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| job\_id | status | fraction\_completed | rows | index\_entries | system\_records | bytes |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

| 370636986413285377 | succeeded | 1 | 0 | 0 | 0 | 0 |

+*--------------------+-----------+--------------------+------+---------------+----------------+-------+*

(1 row)

Time: 42.781522085s

**TIPS:** 用户可以在Web界面的[**Jobs page**](https://www.cockroachlabs.com/docs/stable/admin-ui-jobs-page.html)上观察导入的进度，以及所有schema的更改操作，例如添加二级索引。

* 逻辑上，表之间存在多个[外键](http://doc.cockroachchina.baidu.com/#develop/constraints/foreign-key/)关系

| **引用列** | **被引用列** |
| --- | --- |
| vehicles.city,vehicles.owner\_id | users.city,users.id |
| rides.city,rides.rider\_id | users.city,users.id |
| rides.vehicle\_city,rides.vehicle\_id | vehicles.city,vehicles.id |

先前我们提到，使用IMPORT无法设置属性值之间的关系，但是是有可能创建需要的二级索引。添加外键约束如下：

**ALTER** **TABLE** vehicles

**ADD** **CONSTRAINT** fk\_city\_ref\_users

**FOREIGN** **KEY** (city, owner\_id)

**REFERENCES** users (city, id);

**ALTER** **TABLE** rides

**ADD** **CONSTRAINT** fk\_city\_ref\_users

**FOREIGN** **KEY** (city, rider\_id)

**REFERENCES** users (city, id);

**ALTER** **TABLE** rides

**ADD** **CONSTRAINT** fk\_vehicle\_city\_ref\_vehicles

**FOREIGN** **KEY** (vehicle\_city, vehicle\_id)

**REFERENCES** vehicles (city, id);

* 使用\q或CTRL+D退出shell客户端

Step 5: 安装Python客户端

评价SQL性能时，最好是多次运行特定语句并查看其平均和（/或）累积时延，因此用户需要安装并使用Python测试客户端。

* 在第四台机器上，确保所有的系统软件都是最新的

sudo apt-get update && sudo apt-get -y upgrade

* 安装psycopg2驱动

sudo apt-get install python-psycopg2

* 下载Python客户端

wget https://raw.githubusercontent.com/cockroachdb/docs/master/\_includes/v2.0/performance/tuning.py | chmod +x tuning.py

涉及到的flags:

| **Flag** | **简介** |
| --- | --- |
| --host | 目标节点的IP地址，用于客户端连接 |
| --statement | 执行的SQL语句 |
| --repeat | 执行SQL语句的次数，默认为20 |

运行命令后客户端输出statement语句重复执行下的平均执行时间（以秒为单位）。 此外用户还可以使用两个另外的flags，--time打印出statement语句每次重复执行的执行时间（以秒为单位），--cumulative打印出statement语句所有重复执行的累计执行时间（以秒为单位），该flag在测试写入性能的时候特别有用。

**TIPS:** 查看更多的帮助，可以使用./tuning.py --help

Step 6: 测试/调优读性能

* [主键过滤](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#filtering-by-the-primary-key)
* [非索引列过滤，即全表查询](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#filtering-by-a-non-indexed-column-full-table-scan)
* [二级索引列过滤](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#filtering-by-a-secondary-index)
* [索引覆盖列过滤](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#filtering-by-a-secondary-index-storing-additional-columns)
* [多表JOIN](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#joining-data-from-different-tables)
* [使用IN(子查询)](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#using-in-list-with-a-subquery)
* [使用IN(给定值列表)](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#using-in-list-with-explicit-values)

主键过滤

基于主键检索单行记录，通常会在2ms或更短时间内返回：

./tuning.py --host=<address of any node> --statement="SELECT \* FROM rides WHERE city = 'boston' AND id = '000007ef-fa0f-4a6e-a089-ce74aa8d2276'" --repeat=50 --times

Result:

['id', 'city', 'vehicle\_city', 'rider\_id', 'vehicle\_id', 'start\_address', 'end\_address', 'start\_time', 'end\_time', 'revenue']

['000007ef-fa0f-4a6e-a089-ce74aa8d2276', 'boston', 'boston', 'd66c386d-4b7b-48a7-93e6-f92b5e7916ab', '6628bbbc-00be-4891-bc00-c49f2f16a30b', '4081 Conner Courts\nSouth Taylor, VA 86921', '2808 Willis Wells Apt. 931\nMccoyberg, OH 10303-4879', '2018-07-20 01:46:46.003070', '2018-07-20 02:27:46.003070', '44.25']

Times (milliseconds):

[24.547100067138672, 0.7688999176025391, 0.6949901580810547, 0.8230209350585938, 0.698089599609375, 0.7278919219970703, 0.6978511810302734, 0.5998611450195312, 0.7150173187255859, 0.7338523864746094, 0.6768703460693359, 0.7460117340087891, 0.7028579711914062, 0.7121562957763672, 0.7579326629638672, 0.8080005645751953, 1.0089874267578125, 0.7259845733642578, 0.6411075592041016, 0.7269382476806641, 0.6339550018310547, 0.7460117340087891, 0.9441375732421875, 0.8139610290527344, 0.6990432739257812, 0.6339550018310547, 0.7319450378417969, 0.637054443359375, 0.6501674652099609, 0.7278919219970703, 0.7069110870361328, 0.5779266357421875, 0.6208419799804688, 0.9050369262695312, 0.7741451263427734, 0.5650520324707031, 0.6079673767089844, 0.6191730499267578, 0.7388591766357422, 0.5598068237304688, 0.6401538848876953, 0.6659030914306641, 0.6489753723144531, 0.621795654296875, 0.7548332214355469, 0.6010532379150391, 0.6990432739257812, 0.6699562072753906, 0.6210803985595703, 0.7240772247314453]

Average time (milliseconds):

1.18108272552

**NOTE:** 在session中首次读取表或索引时，查询会比通常情况下慢。这是因为相关节点在首次查询时，需要花费更多的时间用于将表的schema和索引等加载到内存当中。如上所示，第一次查询花费24ms，此后的查询花费时间在1毫秒以内。

只返回部分属性列也可以获得更快的查询速度：

./tuning.py --host=<address of any node> --statement="SELECT rider\_id, vehicle\_id FROM rides WHERE city = 'boston' AND id = '000007ef-fa0f-4a6e-a089-ce74aa8d2276'" --repeat=50 --times

Result:

['rider\_id', 'vehicle\_id']

['d66c386d-4b7b-48a7-93e6-f92b5e7916ab', '6628bbbc-00be-4891-bc00-c49f2f16a30b']

Times (milliseconds):

[1.2311935424804688, 0.7009506225585938, 0.5898475646972656, 0.6151199340820312, 0.5660057067871094, 0.6620883941650391, 0.5691051483154297, 0.5369186401367188, 0.5609989166259766, 0.5290508270263672, 0.5939006805419922, 0.5769729614257812, 0.5638599395751953, 0.5381107330322266, 0.61798095703125, 0.5879402160644531, 0.6008148193359375, 0.5900859832763672, 0.5190372467041016, 0.5409717559814453, 0.51116943359375, 0.5400180816650391, 0.5490779876708984, 0.4870891571044922, 0.5340576171875, 0.49591064453125, 0.5669593811035156, 0.4971027374267578, 0.5729198455810547, 0.514984130859375, 0.5309581756591797, 0.5099773406982422, 0.5550384521484375, 0.5328655242919922, 0.5559921264648438, 0.5319118499755859, 0.5059242248535156, 0.5719661712646484, 0.49614906311035156, 0.6041526794433594, 0.5080699920654297, 0.5240440368652344, 0.49591064453125, 0.5681514739990234, 0.5118846893310547, 0.5359649658203125, 0.5450248718261719, 0.5650520324707031, 0.5249977111816406, 0.5669593811035156]

Average time (milliseconds):

0.566024780273

非索引列过滤，即全表查询

用户基于非主键或非索引列检索单行记录时，性能往往很差。

./tuning.py --host=<address of any node> --statement="SELECT \* FROM users WHERE name = 'Natalie Cunningham'" --repeat=50 --times

Result:

['id', 'city', 'name', 'address', 'credit\_card']

['02cc9e5b-1e91-4cdb-87c4-726b4ea7219a', 'boston', 'Natalie Cunningham', '97477 Lee Path\nKimberlyport, CA 65960', '4532613656695680']

Times (milliseconds):

[31.939983367919922, 4.055023193359375, 3.988981246948242, 4.395008087158203, 4.045009613037109, 3.838062286376953, 6.09898567199707, 4.03904914855957, 3.9091110229492188, 5.933046340942383, 6.157875061035156, 6.323814392089844, 4.379987716674805, 3.982067108154297, 4.28009033203125, 4.118919372558594, 4.222869873046875, 4.041910171508789, 3.9288997650146484, 4.031896591186523, 4.085063934326172, 3.996133804321289, 4.001140594482422, 6.031990051269531, 5.98597526550293, 4.163026809692383, 5.931854248046875, 5.897998809814453, 3.9229393005371094, 3.8909912109375, 3.7729740142822266, 3.9768218994140625, 3.9958953857421875, 4.265069961547852, 4.204988479614258, 4.142999649047852, 4.3659210205078125, 6.074190139770508, 4.015922546386719, 4.418849945068359, 3.9381980895996094, 4.222869873046875, 4.694938659667969, 3.9060115814208984, 3.857851028442383, 3.8509368896484375, 3.969907760620117, 4.241943359375, 4.032135009765625, 3.9670467376708984]

Average time (milliseconds):

4.99066352844

用户可以使用cockroach的内置命令[EXPLAIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/EXPLAIN/)查看查询计划，进一步了解查询性能差的原因：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT \* FROM users WHERE name = 'Natalie Cunningham';"

+------+-------+---------------+

| Tree | Field | Description |

+------+-------+---------------+

| scan | | |

| | table | users@primary |

| | spans | ALL |

+------+-------+---------------+

(3 rows)

spans|ALL行在name属性列未建立二级索引，所以CockroachDB将根据主键city/id有序地扫描users表的每一行记录，直到查找出name值符合条件的记录。

二级索引列过滤

为了加速上述查询，可以在对应属性列name上创建二级索引：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="CREATE INDEX on users (name);"

建立二级索引后，查询加速：

./tuning.py --host=<address of any node> --statement="SELECT \* FROM users WHERE name = 'Natalie Cunningham'" --repeat=50 --times

Result:

['id', 'city', 'name', 'address', 'credit\_card']

['02cc9e5b-1e91-4cdb-87c4-726b4ea7219a', 'boston', 'Natalie Cunningham', '97477 Lee Path\nKimberlyport, CA 65960', '4532613656695680']

Times (milliseconds):

[3.4589767456054688, 1.6651153564453125, 1.547098159790039, 1.9190311431884766, 1.7499923706054688, 1.6219615936279297, 1.5749931335449219, 1.7859935760498047, 1.5561580657958984, 1.6391277313232422, 1.5120506286621094, 1.5139579772949219, 1.6808509826660156, 1.708984375, 1.4798641204833984, 1.544952392578125, 1.653909683227539, 1.6129016876220703, 1.7309188842773438, 1.5811920166015625, 1.7628669738769531, 1.5459060668945312, 1.6429424285888672, 1.6558170318603516, 1.7898082733154297, 1.6138553619384766, 1.6868114471435547, 1.5490055084228516, 1.7120838165283203, 1.6911029815673828, 1.5289783477783203, 1.5990734100341797, 1.6109943389892578, 1.5058517456054688, 1.5058517456054688, 1.6798973083496094, 1.7499923706054688, 1.5850067138671875, 1.4929771423339844, 1.6651153564453125, 1.5921592712402344, 1.6739368438720703, 1.6529560089111328, 1.6019344329833984, 1.6429424285888672, 1.5649795532226562, 1.605987548828125, 1.550912857055664, 1.6069412231445312, 1.6779899597167969]

Average time (milliseconds):

1.66565418243

用户可以使用cockroach的内置命令[EXPLAIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/EXPLAIN/)查看查询计划，了解带索引的新查询（1.66ms）相较于不带索引的查询（4.99ms）性能提升的原因：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT \* FROM users WHERE name = 'Natalie Cunningham';"

+------------+-------+-------------------------------------------------------+

| Tree | Field | Description |

+------------+-------+-------------------------------------------------------+

| index-join | | |

| ├── scan | | |

| │ | table | users@users\_name\_idx |

| │ | spans | /"Natalie Cunningham"-/"Natalie Cunningham"/PrefixEnd |

| └── scan | | |

| | table | users@primary |

+------------+-------+-------------------------------------------------------+

(6 rows)

EXPLAIN查询结果表明，CockroachDB优先查询二级索引（table | users @ users\_name\_idx）内容。考虑到二级索引按name值有序，查询可以直接跳到相关值spans | /"Natalie Cunningham"-/"Natalie Cunningham"/PrefixEnd所在位置。然而该查询需要返回不在二级索引中的属性列，因此CockroachDB将会通过二级索引获取name值对应的主键city / id的值，查询主索引中该主键值，返回完整行。

回想一下[Ranges和租约持有者的设计](http://doc.cockroachchina.baidu.com/#deploy/performance-tuning/#important-concepts)，如果users表很小且小于64MiB，即主索引和二级索引只有一个Range、对应一个租约持有者。如果users大于64MiB，即主索引和二级索引对应多个Range和多个租约持有者，当多个Range的租约持有者位于不同节点上，将带来更多的网络消耗，增加查询时延。

索引覆盖列过滤

用户如果有一个按特定列值筛选且返回该表部分列的查询，则可以通过将这些列覆盖[存储](https://www.cockroachlabs.com/docs/stable/indexes.html#storing-columns)在二级索引中来提高性能，避免查询扫描主索引。

例如，检索用户姓名和信用卡号码：

./tuning.py --host=<address of any node> --statement="SELECT name, credit\_card FROM users WHERE name = 'Natalie Cunningham'" --repeat=50 --times

Result:

['name', 'credit\_card']

['Natalie Cunningham', '4532613656695680']

Times (milliseconds):

[2.338886260986328, 1.7859935760498047, 1.9490718841552734, 1.550912857055664, 1.4331340789794922, 1.4619827270507812, 1.425027847290039, 1.8270015716552734, 1.6829967498779297, 1.6028881072998047, 1.628875732421875, 1.4889240264892578, 1.497030258178711, 1.5380382537841797, 1.486063003540039, 1.5859603881835938, 1.7290115356445312, 1.7409324645996094, 1.5869140625, 1.6489028930664062, 1.7418861389160156, 1.5971660614013672, 1.619100570678711, 1.6379356384277344, 1.6028881072998047, 1.6531944274902344, 1.667022705078125, 1.6241073608398438, 1.5468597412109375, 1.5778541564941406, 1.6779899597167969, 1.5718936920166016, 1.5950202941894531, 1.6407966613769531, 1.538991928100586, 1.8379688262939453, 1.7008781433105469, 1.837015151977539, 1.5687942504882812, 1.7828941345214844, 1.7290115356445312, 1.6810894012451172, 1.7969608306884766, 1.5821456909179688, 1.569986343383789, 1.5740394592285156, 1.8229484558105469, 1.7371177673339844, 1.7681121826171875, 1.6360282897949219]

Average time (milliseconds):

1.65812492371

创建了name属性列的二级索引，CockroachDB仍然需要查询主索引获取credit\_name值：

cockroach sql \

--insecure \

--host=<address of any node> \

--database=movr \

--execute="EXPLAIN SELECT name, credit\_card FROM users WHERE name = 'Natalie Cunningham';"

+-----------------+-------+-------------------------------------------------------+

| Tree | Field | Description |

+-----------------+-------+-------------------------------------------------------+

| render | | |

| └── index-join | | |

| ├── scan | | |

| │ | table | users@users\_name\_idx |

| │ | spans | /"Natalie Cunningham"-/"Natalie Cunningham"/PrefixEnd |

| └── scan | | |

| | table | users@primary |

+-----------------+-------+-------------------------------------------------------+

(7 rows)

删除并重新创建name属性列的二级索引，同时索引覆盖credit\_name列：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="DROP INDEX users\_name\_idx;"

cockroach sql --insecure --host=<address of any node> --database=movr --execute="CREATE INDEX ON users (name) STORING (credit\_card);"

此时查询二级索引覆盖后不再需要查找主索引：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT name, credit\_card FROM users WHERE name = 'Natalie Cunningham';"

+-----------+-------+-------------------------------------------------------+

| Tree | Field | Description |

+-----------+-------+-------------------------------------------------------+

| render | | |

| └── scan | | |

| | table | users@users\_name\_idx |

| | spans | /"Natalie Cunningham"-/"Natalie Cunningham"/PrefixEnd |

+-----------+-------+-------------------------------------------------------+

(4 rows)

查询结果显示性能提升，由1.65ms（无索引覆盖）提升到1.04ms（带索引覆盖）：

./tuning.py --host=<address of any node> --statement="SELECT name, credit\_card FROM users WHERE name = 'Natalie Cunningham'" --repeat=50 --times

Result:

['name', 'credit\_card']

['Natalie Cunningham', '4532613656695680']

Times (milliseconds):

[1.8949508666992188, 1.2660026550292969, 1.2140274047851562, 1.110076904296875, 1.4989376068115234, 1.1739730834960938, 1.2331008911132812, 0.9701251983642578, 0.9019374847412109, 0.9038448333740234, 1.016855239868164, 0.9331703186035156, 0.9179115295410156, 0.9288787841796875, 0.888824462890625, 0.9429454803466797, 0.9410381317138672, 1.001119613647461, 0.9438991546630859, 0.9849071502685547, 1.0221004486083984, 1.013040542602539, 1.0149478912353516, 0.9579658508300781, 1.0061264038085938, 1.0559558868408203, 1.0788440704345703, 1.0411739349365234, 0.9610652923583984, 0.9639263153076172, 1.1239051818847656, 0.9639263153076172, 1.058816909790039, 0.949859619140625, 0.9739398956298828, 1.046895980834961, 0.9260177612304688, 1.0569095611572266, 1.033782958984375, 1.1029243469238281, 0.9710788726806641, 1.0311603546142578, 0.9870529174804688, 1.1179447174072266, 1.0349750518798828, 1.088857650756836, 1.1060237884521484, 1.0170936584472656, 1.0180473327636719, 1.0519027709960938]

Average time (milliseconds):

1.04885578156

多表JOIN

二级索引对于多表JOIN操作也是十分重要。

例如统计指定日期下具有行程的用户的数量，则需要将rides表的rider\_id属性和users表的id属性进行JOIN操作，并统计成功匹配数量：

./tuning.py --host=<address of any node> --statement="SELECT count(DISTINCT users.id) FROM users INNER JOIN rides ON rides.rider\_id = users.id WHERE start\_time BETWEEN '2018-07-20 00:00:00' AND '2018-07-21 00:00:00'" --repeat=50 --times

Result:

['count']

['1998']

Times (milliseconds):

[1663.2239818572998, 841.871976852417, 844.9788093566895, 1043.7190532684326, 1047.544002532959, 1049.0870475769043, 1079.737901687622, 1049.543857574463, 1069.1118240356445, 1104.2020320892334, 1071.1669921875, 1080.1141262054443, 1066.741943359375, 1071.8858242034912, 1073.8670825958252, 1054.008960723877, 1089.4761085510254, 1048.2399463653564, 1033.8318347930908, 1078.5980224609375, 1054.8391342163086, 1095.6230163574219, 1056.9767951965332, 1082.8359127044678, 1048.3272075653076, 1050.3859519958496, 1084.2180252075195, 1082.1950435638428, 1101.97114944458, 1079.9469947814941, 1065.234899520874, 1051.058053970337, 1105.48996925354, 1119.469165802002, 1089.8759365081787, 1082.5989246368408, 1074.9430656433105, 1067.4428939819336, 1066.5888786315918, 1069.6449279785156, 1067.9738521575928, 1082.4880599975586, 1037.9269123077393, 1042.2871112823486, 1130.7330131530762, 1150.7518291473389, 1165.3728485107422, 1136.9531154632568, 1120.3861236572266, 1126.8589496612549]

Average time (milliseconds):

1081.04698181

使用[EXPLAIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/EXPLAIN/)子句分析JOIN语句的查询计划：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT count(DISTINCT users.id) FROM users INNER JOIN rides ON rides.rider\_id = users.id WHERE start\_time BETWEEN '2018-07-20 00:00:00' AND '2018-07-21 00:00:00';"

+---------------------+----------+-------------------+

| Tree | Field | Description |

+---------------------+----------+-------------------+

| group | | |

| └── render | | |

| └── join | | |

| │ | type | inner |

| │ | equality | (id) = (rider\_id) |

| ├── scan | | |

| │ | table | users@primary |

| │ | spans | ALL |

| └── scan | | |

| | table | rides@primary |

| | spans | ALL |

+---------------------+----------+-------------------+

(11 rows)

根据分析结果可以看到，CockroachDB首先对rides表进行一次全表查询（spans | ALL），过滤出start\_time值满足特定日期的记录，再对users表进行全表检索，匹配出满足JOIN条件的行并进行计数。

考虑rides表很大，数据被切分成多个Range，每个Range进行数据冗余，对应一个租约持有者。这些租约持有者很可能分布在不同的节点上，rides表全表查询这些Range会有很大的网络开销。

使用[SHOW EXPERIMENTAL\_RANGES](https://www.cockroachlabs.com/docs/stable/show-experimental-ranges.html)语句可以查看rides表和users表相关的租约持有者情况：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW EXPERIMENTAL\_RANGES FROM TABLE rides;"

+------------------------------------------------------------------------+------------------------------------------------------------------------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+------------------------------------------------------------------------+------------------------------------------------------------------------+----------+----------+--------------+

| NULL | /"boston"/"\xfe\xdd?\xbb4\xabOV\x84\x00M\x89#-a6"/PrefixEnd | 23 | {1,2,3} | 1 |

| /"boston"/"\xfe\xdd?\xbb4\xabOV\x84\x00M\x89#-a6"/PrefixEnd | /"los angeles"/"\xf1\xe8\x99eǵI\x16\xb9w\a\xd01\xcc\b\xa4"/PrefixEnd | 25 | {1,2,3} | 2 |

| /"los angeles"/"\xf1\xe8\x99eǵI\x16\xb9w\a\xd01\xcc\b\xa4"/PrefixEnd | /"new york"/"\xebV\xf5\xe6P%L$\x92\xd2\xdf&\a\x81\xeeO"/PrefixEnd | 26 | {1,2,3} | 1 |

| /"new york"/"\xebV\xf5\xe6P%L$\x92\xd2\xdf&\a\x81\xeeO"/PrefixEnd | /"san francisco"/"\xda\xc5B\xe0\x0e\fK)\x98:\xe6[@\x05\x91\*"/PrefixEnd | 27 | {1,2,3} | 2 |

| /"san francisco"/"\xda\xc5B\xe0\x0e\fK)\x98:\xe6[@\x05\x91\*"/PrefixEnd | /"seattle"/"\xd4ˆ?\x98\x98FA\xa7m\x84\xba\xac\xf5\xbfI"/PrefixEnd | 28 | {1,2,3} | 3 |

| /"seattle"/"\xd4ˆ?\x98\x98FA\xa7m\x84\xba\xac\xf5\xbfI"/PrefixEnd | /"washington dc"/"Ņ\x06\x9d\xc2LEq\xb8<KG\a(\x18\xf6"/PrefixEnd | 29 | {1,2,3} | 1 |

| /"washington dc"/"Ņ\x06\x9d\xc2LEq\xb8<KG\a(\x18\xf6"/PrefixEnd | NULL | 30 | {1,2,3} | 1 |

+------------------------------------------------------------------------+------------------------------------------------------------------------+----------+----------+--------------+

(7 rows)

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW EXPERIMENTAL\_RANGES FROM TABLE users;"

+-----------+---------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+-----------+---------+----------+----------+--------------+

| NULL | NULL | 51 | {1,2,3} | 2 |

+-----------+---------+----------+----------+--------------+

(1 row)

上述查询结果中：

* rides表被切分成7个Range，其中节点1具有4个租约持有者，节点2具有2个租约持有者，节点3具有1个租约持有者。
* users表只有一个Range，节点2具有该租约持有者。

可以看出，在给定的多表JOIN且带有WHERE子句查询条件的情况下，对rides表的7个range进行全表查询是一个代价很高的操作。为加速查询，可以针对WHERE子句查询条件（rides.start\_time）创建二级索引，同时索引覆盖rider\_id属性列：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="CREATE INDEX ON rides (start\_time) STORING (rider\_id);"

**NOTE:** 若rides表包含1百万行记录，创建该二级索引需要花费几分钟时间。

添加二级索引后，带JOIN操作的查询执行时间从1081.04ms降到71.89ms：

./tuning.py --host=<address of any node> --statement="SELECT count(DISTINCT users.id) FROM users INNER JOIN rides ON rides.rider\_id = users.id WHERE start\_time BETWEEN '2018-07-20 00:00:00' AND '2018-07-21 00:00:00'" --repeat=50 --times

Result:

['count']

['1998']

Times (milliseconds):

[124.19795989990234, 83.74285697937012, 84.76495742797852, 76.9808292388916, 65.74702262878418, 62.478065490722656, 60.26411056518555, 59.99302864074707, 67.10195541381836, 73.45199584960938, 67.09504127502441, 60.45889854431152, 68.6960220336914, 61.94710731506348, 61.53106689453125, 60.44197082519531, 62.22796440124512, 89.34903144836426, 77.64196395874023, 71.43712043762207, 66.09010696411133, 63.668012619018555, 65.31286239624023, 77.1780014038086, 73.52113723754883, 68.84908676147461, 65.11712074279785, 65.34600257873535, 65.8869743347168, 76.90095901489258, 76.9491195678711, 69.39697265625, 64.23306465148926, 75.0880241394043, 69.34094429016113, 57.55496025085449, 65.79995155334473, 83.74285697937012, 75.32310485839844, 74.08809661865234, 77.33798027038574, 73.95505905151367, 71.85482978820801, 77.95405387878418, 74.30601119995117, 72.24106788635254, 75.28901100158691, 78.2630443572998, 74.97286796569824, 79.50282096862793]

Average time (milliseconds):

71.8922615051

可以使用[EXPLAIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/EXPLAIN/)子句查看新查询的执行计划：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT count(DISTINCT users.id) FROM users INNER JOIN rides ON rides.rider\_id = users.id WHERE start\_time BETWEEN '2018-07-20 00:00:00' AND '2018-07-21 00:00:00';"

+---------------------+----------+-------------------------------------------------------+

| Tree | Field | Description |

+---------------------+----------+-------------------------------------------------------+

| group | | |

| └── render | | |

| └── join | | |

| │ | type | inner |

| │ | equality | (id) = (rider\_id) |

| ├── scan | | |

| │ | table | users@primary |

| │ | spans | ALL |

| └── scan | | |

| | table | rides@rides\_start\_time\_idx |

| | spans | /2018-07-20T00:00:00Z-/2018-07-21T00:00:00.000000001Z |

+---------------------+----------+-------------------------------------------------------+

(11 rows)

可以看出CockroachDB此时优先使用二级索引rides@rides\_start\_time\_idx、而不需要全表扫描rides表来检索相关的rides值。

检查新创建的二级索引Range信息：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW EXPERIMENTAL\_RANGES FROM INDEX rides@rides\_start\_time\_idx;"

+-----------------------------------------------------------------------------------------+-----------------------------------------------------------------------------------------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+-----------------------------------------------------------------------------------------+-----------------------------------------------------------------------------------------+----------+----------+--------------+

| NULL | /2018-07-15T02:32:47.564891Z/"seattle"/"r\x8f\xbc\xd4\f\x18E\x9f\x85\xc2\"H\\\xe7k\xf1" | 34 | {1,2,3} | 1 |

| /2018-07-15T02:32:47.564891Z/"seattle"/"r\x8f\xbc\xd4\f\x18E\x9f\x85\xc2\"H\\\xe7k\xf1" | NULL | 35 | {1,2,3} | 1 |

+-----------------------------------------------------------------------------------------+-----------------------------------------------------------------------------------------+----------+----------+--------------+

(2 rows)

可以看出索引被切分成2个Range，租约持有者均在节点1上。基于SHOW EXPERIMENTAL\_RANGES FROM TABLE users先前的输出，已确认users表的租约持有者在节点2上。

使用IN(子查询)

假设用户需要查询使用最多的前5名车辆各自对应的最近一次行程，则首先需要使用子查询，从rides表中查询出使用最多的前5名车辆ID，将结果作为父查询的IN语句的list，父查询将根据IN语句查询出指定车辆的最近一次行程。

./tuning.py --host=<address of any node> --statement="SELECT vehicle\_id, max(end\_time) FROM rides WHERE vehicle\_id IN ( SELECT vehicle\_id FROM rides GROUP BY vehicle\_id ORDER BY count(\*) DESC LIMIT 5 ) GROUP BY vehicle\_id" --repeat=20 --times

Result:

['vehicle\_id', 'max']

['c6541da5-9858-4e3f-9b49-992e206d2c50', '2018-08-02 02:14:50.543760']

['78fdd6f8-c6a1-42df-a89f-cd65b7bb8be9', '2018-08-02 02:47:43.755989']

['3c950d36-c2b8-48d0-87d3-e0d6f570af62', '2018-08-02 03:06:31.293184']

['35752c4c-b878-4436-8330-8d7246406a55', '2018-08-02 03:08:49.823209']

['0962cdca-9d85-457c-9616-cc2ae2d32008', '2018-08-02 03:01:25.414512']

Times (milliseconds):

[4368.9610958099365, 4373.898029327393, 4396.070957183838, 4382.591962814331, 4274.624824523926, 4369.847059249878, 4373.079061508179, 4287.877082824707, 4307.362079620361, 4368.865966796875, 4363.792896270752, 4310.600996017456, 4378.695011138916, 4340.383052825928, 4338.238000869751, 4373.046875, 4327.131986618042, 4386.303901672363, 4429.6300411224365, 4383.068084716797]

Average time (milliseconds):

4356.7034483

可以看出该查询速度很慢，原因在于子查询结果作为主查询where的判断条件时，即使已创建索引，CockroachDB也需要扫描全表。执行EXPALIN了解详情：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="EXPLAIN SELECT vehicle\_id, max(end\_time) FROM rides WHERE vehicle\_id IN ( SELECT vehicle\_id FROM rides GROUP BY vehicle\_id ORDER BY count(\*) DESC LIMIT 5 ) GROUP BY vehicle\_id;"

+------------------------------------+-----------+--------------------------------------------------------------------------+

| Tree | Field | Description |

+------------------------------------+-----------+--------------------------------------------------------------------------+

| root | | |

| ├── group | | |

| │ │ | group by | @1-@1 |

| │ └── render | | |

| │ └── scan | | |

| │ | table | rides@primary |

| │ | spans | ALL |

| └── subquery | | |

| │ | id | @S1 |

| │ | sql | (SELECT vehicle\_id FROM rides GROUP BY vehicle\_id ORDER BY count(\*) DESC |

| | | LIMIT 5) |

| │ | exec mode | all rows normalized |

| └── limit | | |

| └── sort | | |

| │ | order | -count |

| │ | strategy | top 5 |

| └── group | | |

| │ | group by | @1-@1 |

| └── render | | |

| └── scan | | |

| | table | rides@primary |

| | spans | ALL |

+------------------------------------+-----------+--------------------------------------------------------------------------+

(21 rows)

这是一个复杂的查询计划，值得注意的是基于subquery的查询结果、父查询执行的是rides@primary的全表扫描。子查询返回使用最多的前5名车辆ID，CockroachDB对于每个vehicle\_id会扫描全表主键索引检索出匹配max(end\_time)的结果，而不是使用vehicle\_id具有的二级索引（CockroachDB将会在后续的版本对此进行改进）。

使用IN(给定值列表)

考虑带IN（子查询）的语句执行父查询时，CockroachDB不会使用可用的二级索引，用户可以在应用层先查询出使用最多的前5名车辆：

./tuning.py --host=<address of any node> --statement="SELECT vehicle\_id FROM rides GROUP BY vehicle\_id ORDER BY count(\*) DESC LIMIT 5" --repeat=20 --times

Result:

['vehicle\_id']

['35752c4c-b878-4436-8330-8d7246406a55']

['0962cdca-9d85-457c-9616-cc2ae2d32008']

['c6541da5-9858-4e3f-9b49-992e206d2c50']

['78fdd6f8-c6a1-42df-a89f-cd65b7bb8be9']

['3c950d36-c2b8-48d0-87d3-e0d6f570af62']

Times (milliseconds):

[787.0969772338867, 782.2480201721191, 741.5878772735596, 790.3921604156494, 767.4920558929443, 733.0870628356934, 768.8038349151611, 754.1589736938477, 716.4630889892578, 726.3698577880859, 721.092939376831, 737.1737957000732, 747.978925704956, 736.1149787902832, 727.1649837493896, 725.5918979644775, 746.1550235748291, 752.6230812072754, 728.59787940979, 733.4978580474854]

Average time (milliseconds):

746.184563637

然后根据查询的结果，再使用IN(给定值列表)提交二次查询：

./tuning.py --host=<address of any node> --statement="SELECT vehicle\_id, max(end\_time) FROM rides WHERE vehicle\_id IN ( '35752c4c-b878-4436-8330-8d7246406a55', '0962cdca-9d85-457c-9616-cc2ae2d32008', 'c6541da5-9858-4e3f-9b49-992e206d2c50', '78fdd6f8-c6a1-42df-a89f-cd65b7bb8be9', '3c950d36-c2b8-48d0-87d3-e0d6f570af62' ) GROUP BY vehicle\_id;" --repeat=20 --times

Result:

['vehicle\_id', 'max']

['3c950d36-c2b8-48d0-87d3-e0d6f570af62', '2018-08-02 03:06:31.293184']

['78fdd6f8-c6a1-42df-a89f-cd65b7bb8be9', '2018-08-02 02:47:43.755989']

['35752c4c-b878-4436-8330-8d7246406a55', '2018-08-02 03:08:49.823209']

['0962cdca-9d85-457c-9616-cc2ae2d32008', '2018-08-02 03:01:25.414512']

['c6541da5-9858-4e3f-9b49-992e206d2c50', '2018-08-02 02:14:50.543760']

Times (milliseconds):

[828.5520076751709, 826.6720771789551, 837.0990753173828, 865.441083908081, 870.556116104126, 842.6721096038818, 859.3161106109619, 861.4299297332764, 866.6350841522217, 833.0469131469727, 838.021993637085, 841.0389423370361, 878.7519931793213, 879.6770572662354, 861.1328601837158, 855.1840782165527, 856.5502166748047, 882.9760551452637, 873.0340003967285, 858.4709167480469]

Average time (milliseconds):

855.812931061

此时查询时间由4356.70ms（带子查询）降至1601.99ms（两阶段查询）。

Step 7: 测试/调优写性能

* 批量插入
* 尽可能减少无用索引
* 返回最新插入记录的id

批量插入

假设需要对users表插入100个新用户，常见的方式是执行100条单独的[INSERT](http://doc.cockroachchina.baidu.com/#develop/constraints/insert/)语句。

**NOTE:** 出于演示的目的，以下命令将相同的用户插入100次，每次使用不同的唯一ID。同时需要使用--cumulativeflag，打印100次插入的总时间。

./tuning.py --host=<address of any node> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347')" --repeat=100 --times --cumulative

Times (milliseconds):

[33.28299522399902, 13.558149337768555, 14.67585563659668, 8.835077285766602, 9.104013442993164, 8.157968521118164, 10.174989700317383, 8.877992630004883, 9.196996688842773, 8.93402099609375, 9.894132614135742, 9.97304916381836, 8.221149444580078, 9.334087371826172, 9.270191192626953, 8.980035781860352, 7.210969924926758, 8.212089538574219, 8.048057556152344, 7.8639984130859375, 7.489204406738281, 9.547948837280273, 9.073972702026367, 9.660005569458008, 9.325981140136719, 9.338140487670898, 9.240865707397461, 7.958889007568359, 8.417844772338867, 8.075952529907227, 7.896184921264648, 9.118080139160156, 8.161067962646484, 9.071111679077148, 8.996963500976562, 7.790803909301758, 7.8220367431640625, 9.695053100585938, 9.470939636230469, 8.415937423706055, 9.287118911743164, 9.29117202758789, 9.618043899536133, 9.107828140258789, 8.491039276123047, 7.998943328857422, 9.282827377319336, 7.735013961791992, 9.161949157714844, 9.70005989074707, 8.910894393920898, 9.124994277954102, 9.028911590576172, 9.568929672241211, 10.931968688964844, 8.813858032226562, 14.040946960449219, 7.773876190185547, 9.801864624023438, 7.989168167114258, 8.188962936401367, 9.398937225341797, 9.705066680908203, 9.213924407958984, 9.569168090820312, 9.19198989868164, 9.664058685302734, 9.52601432800293, 8.01396369934082, 8.30698013305664, 8.03995132446289, 8.166074752807617, 9.335994720458984, 7.915019989013672, 9.584903717041016, 8.049964904785156, 7.803916931152344, 8.125066757202148, 9.367942810058594, 9.21487808227539, 9.630918502807617, 9.505033493041992, 9.830951690673828, 8.285045623779297, 8.095979690551758, 9.876012802124023, 8.067131042480469, 9.438037872314453, 8.147001266479492, 8.9111328125, 9.560108184814453, 8.78596305847168, 9.341955184936523, 10.293006896972656, 9.062051773071289, 14.008045196533203, 9.293079376220703, 9.57798957824707, 14.974832534790039, 8.59689712524414]

Average time (milliseconds):

9.41696166992

Cumulative time (milliseconds):

941.696166992

可以看见100次插入总共使用了941.69ms，然而如果只使用一条INSERT语句，用户之间用逗号隔开，则可以获得更快的速度：

./tuning.py --host=<address of any node> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347'), (gen\_random\_uuid(), 'new york', 'Max Roach', '411 Drum Street', '173635282937347')" --repeat=1 --cumulative

Average time (milliseconds):

18.965959549

Cumulative time (milliseconds):

18.965959549

可以看出，多行批量INSERT相较于多次单行INSERT查询时间上从941.69ms降到了18.96ms。 同理[UPSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPSERT/)和[DELETE](https://www.cockroachlabs.com/docs/stable/delete.html)语句的批量操作也是更高率的。

尽可能减少无用索引

在了解到索引对读性能的提升时，用户也需要了解到索引在写性能上带来的性能下降。 对于users表：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW INDEXES FROM users;"

+-------+----------------+--------+-----+-------------+-----------+---------+----------+

| Table | Name | Unique | Seq | Column | Direction | Storing | Implicit |

+-------+----------------+--------+-----+-------------+-----------+---------+----------+

| users | primary | true | 1 | city | ASC | false | false |

| users | primary | true | 2 | id | ASC | false | false |

| users | users\_name\_idx | false | 1 | name | ASC | false | false |

| users | users\_name\_idx | false | 2 | credit\_card | N/A | true | false |

| users | users\_name\_idx | false | 3 | city | ASC | false | true |

| users | users\_name\_idx | false | 4 | id | ASC | false | true |

+-------+----------------+--------+-----+-------------+-----------+---------+----------+

(6 rows)

表拥有一个主索引和一个建立在name属性上且覆盖credit\_card属性的二级索引，这意味着更新或插入带name、credit\_card、city、id属性的行时除了需要更新主索引，还要更新二级索引。

举个具体的例子，统计name属性以C字母开头的所有行，同时更新这些行的name属性值为相同的名字：

./tuning.py --host=<address of any node> --statement="SELECT count(\*) FROM users WHERE name LIKE 'C%'" --repeat=1

Result:

['count']

['179']

Average time (milliseconds):

2.52413749695

./tuning.py \

--host=<address of any node> \

--statement="UPDATE users \

SET name = 'Carl Kimball' \

WHERE name LIKE 'C%'" \

--repeat=1

Average time (milliseconds):

110.701799393

name属性字段即是主键，也是users\_name\_idx索引，168行记录中2个key都需要更新 现在假设不在需要二级索引users\_name\_idx，删除该二级索引并再次执行更新操作：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="DROP INDEX users\_name\_idx;"

./tuning.py --host=<address of any node> --statement="UPDATE users SET name = 'Peedie Hirata' WHERE name = 'Carl Kimball'" --repeat=1

Average time (milliseconds):

21.7709541321

对比发现，相比于前者需要更新主索引和二级索引，后者只需要更新主索引，执行时间由110.70ms减少至21.77ms。

返回最新插入记录的ID

用户有些时候需要在插入数据后获取到插入记录的ID，以便基于ID进行后续的一些工作。

一种方式是执行完INSERT语句后再执行SELECT操作，获取新插入记录的ID：

./tuning.py --host=<address of any node> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'Toni Brooks', '800 Camden Lane, Brooklyn, NY 11218', '98244843845134960')" --repeat=1

Average time (milliseconds):

9.97304916382

./tuning.py --host=<address of any node> --statement="SELECT id FROM users WHERE name = 'Toni Brooks'" --repeat=1

Result:

['id']

['cc83e0bd-2ea0-4507-a683-a707cfbe0aba']

Average time (milliseconds):

7.32207298279

总执行时间为17.29ms。 用户可以考虑另外一种方式以获得更好的性能，执行INSERT语句时配合使用RETURNING id子句：

./tuning.py --host=<address of any node> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'Brian Brooks', '800 Camden Lane, Brooklyn, NY 11218', '98244843845134960') RETURNING id" --repeat=1

Result:

['id']

['3d16500e-cb2e-462e-9c83-db0965d6deaf']

Average time (milliseconds):

9.48596000671

此时只需要9.48ms，第二种方式由于读写是在一次客户端-服务端路由、而不是第一种方式的两次路由，性能更好。正如前面讨论的一样，表的租约拥有者在不同节点间执行查询，需要花费更多的网络和时间开销。

多区域部署

拓展集群，新增两个其他的zoneus-west1-a和us-west2-a，每个区域部署3台机器与1台额外的运行客户端负载的机器。

Step 8: 创建更多机器

* [创建6台机器](https://cloud.google.com/compute/docs/instances/create-start-instance)，3台在us-west1-azone（Oregon），3台在us-west2-azone（Los Angeles）。注意：
* 使用n1-standard-4机器类型，配置4 vCPUs和15GiB内存
* 使用Ubuntu 16.04系统
* [创建和挂载本地SSD](https://cloud.google.com/compute/docs/disks/local-ssd#create_local_ssd)
* 尽早的应用已配置的防火墙规则，点击**Management**、 **disk**、 **networking**、**SSH keys**，选择**Networking**标签页、并在**Networking tags**配置项填入cockroachdb
* 获得每台n1-standard-4机器的内网IP地址，用于集群节点启动。
* 在us-west1-a和us-west2-azone里，各自创建一台单独的机器用于运行客户端负载。其可以选择配置弱一点的机器，例如n1-standard-1。

Step 9: 拓展集群

对us-west1-a和us-west2-azone里组成CockraochDB的6台n1-standard-4机器，分别进行以下操作： - SSH登陆 - 下载CockraochDB Linux版本 [CockcroachDB压缩包](https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz)，解压并拷贝到PATH

wget -qO- https://binaries.cockroachdb.com/cockroach-v2.0.5.linux-amd64.tgz | tar xvz

sudo cp -i cockroach-v2.0.5.linux-amd64/cockroach /usr/local/bin

* 执行cockroach start命令

*# us-west1-a zone的机器，执行：*

cockroach start --insecure --advertise-host=<node internal address> --join=<same as earlier> --locality=cloud=gce,region=us-west1,zone=us-west1-a --cache=.25 --max-sql-memory=.25 --background

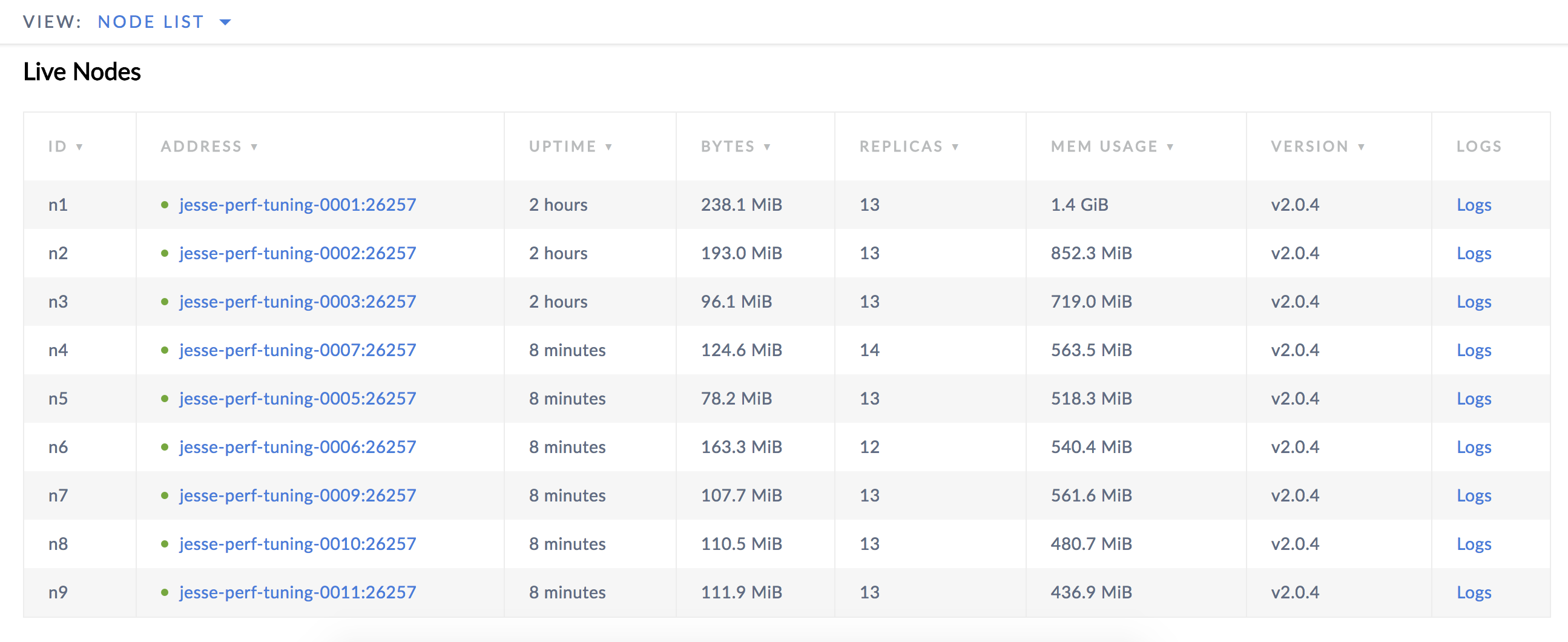
*# us-west2-a zone的机器，执行：*

cockroach start --insecure --advertise-host=<node1 internal address> --join=<same as earlier> --locality=cloud=gce,region=us-west2,zone=us-west2-a --cache=.25 --max-sql-memory=.25 --background

Step 10: 安装Python客户端

参照**Step 5**进行操作。

Step 11: 检查均衡情况

每个节点在启动的时候都指定--locality为对应的GCE zone，CockroachDB则会在多个zone之间均衡分布数据副本。 用户可以通过任意节点登陆Admin界面<node address>:8080，在**Node List**信息中可以看到所有节点的range数目都是差不多的： 

上图中，节点ID与zone映射关系为：

| **节点ID** | **Zone** |
| --- | --- |
| 1-3 | us-east1-b(South Carolina) |
| 4-6 | us-west1-a(Oregon) |
| 7-9 | us-west2-a(Los Angeles) |

选取一台额外创建的未运行CockroachDB的机器，SSH登陆并执行SHOW EXPERIMENTAL\_RANGES语句，查看Range级别的均衡情况。

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW EXPERIMENTAL\_RANGES FROM TABLE vehicles;"

+-----------+---------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+-----------+---------+----------+----------+--------------+

| NULL | NULL | 22 | {1,6,9} | 6 |

+-----------+---------+----------+----------+--------------+

(1 row)

输出结果显示，vehicles数据只有一个Range，每个zone各自有一个Range副本，且租期持有者位于节点6，即us-west1-a zone。

Step 12: 测试性能

针对单区域部署集群的诸多优化，基本上也适用于多区域部署的集群，不同的是大多数情况下，多区域部署的集群数据和租约持有者遍布美国、需要更大的性能开销。

读性能

假设场景：作为纽约的Movr管理员，我们想要获取当前正在使用的所有纽约自行车的ID和描述：

* SSH到带有Python客户端的us-east1-b的机器
* 执行查询：

./tuning.py --host=<address of a node **in** us-east1-b> --statement="SELECT id, ext FROM vehicles WHERE city = 'new york' AND type = 'bike' AND status = 'in\_use'" --repeat=50 --times

Result:

['id', 'ext']

['0068ee24-2dfb-437d-9a5d-22bb742d519e', "{u'color': u'green', u'brand': u'Kona'}"]

['01b80764-283b-4232-8961-a8d6a4121a08', "{u'color': u'green', u'brand': u'Pinarello'}"]

['02a39628-a911-4450-b8c0-237865546f7f', "{u'color': u'black', u'brand': u'Schwinn'}"]

['02eb2a12-f465-4575-85f8-a4b77be14c54', "{u'color': u'black', u'brand': u'Pinarello'}"]

['02f2fcc3-fea6-4849-a3a0-dc60480fa6c2', "{u'color': u'red', u'brand': u'FujiCervelo'}"]

['034d42cf-741f-428c-bbbb-e31820c68588', "{u'color': u'yellow', u'brand': u'Santa Cruz'}"]

...

Times (milliseconds):

[1123.0790615081787, 190.16599655151367, 127.28595733642578, 72.94511795043945, 72.0360279083252, 70.50704956054688, 70.83487510681152, 73.11201095581055, 72.81899452209473, 71.35510444641113, 71.6249942779541, 70.8611011505127, 72.17597961425781, 71.78997993469238, 70.75691223144531, 76.08985900878906, 72.6480484008789, 71.91896438598633, 70.59216499328613, 71.07686996459961, 71.86722755432129, 71.01583480834961, 71.29812240600586, 71.74086570739746, 72.67093658447266, 71.03395462036133, 71.78306579589844, 71.5029239654541, 70.33801078796387, 72.91483879089355, 71.23708724975586, 72.81684875488281, 71.70701026916504, 71.32506370544434, 71.68197631835938, 70.78695297241211, 72.80707359313965, 73.0600357055664, 71.69818878173828, 71.40707969665527, 70.53804397583008, 71.83694839477539, 70.08099555969238, 71.96617126464844, 71.03586196899414, 72.6020336151123, 71.23398780822754, 71.03800773620605, 72.12519645690918, 71.77996635437012]

Average time (milliseconds):

96.2521076202

先前我们已经查询到，vehicles表的租约持有者在us-west1-a(Oregon)，因此查询需要从东海岸（us-east1-b zone）的网关节点路由到西海岸的机器，处理后返回给网关节点，再将结果返回给客户端。

相比之下，假设作西雅图的Movr管理员，希望获得当前正在使用的所有西雅图自行车的ID和描述：

* SSH到带有Python客户端的us-west1-a的机器
* 执行查询：

./tuning.py --host=<address of a node **in** us-west1-a> --statement="SELECT id, ext FROM vehicles WHERE city = 'seattle' AND type = 'bike' AND status = 'in\_use'" --repeat=50 --times

Result:

['id', 'ext']

['00078349-94d4-43e6-92be-8b0d1ac7ee9f', "{u'color': u'blue', u'brand': u'Merida'}"]

['003f84c4-fa14-47b2-92d4-35a3dddd2d75', "{u'color': u'red', u'brand': u'Kona'}"]

['0107a133-7762-4392-b1d9-496eb30ee5f9', "{u'color': u'yellow', u'brand': u'Kona'}"]

['0144498b-4c4f-4036-8465-93a6bea502a3', "{u'color': u'blue', u'brand': u'Pinarello'}"]

['01476004-fb10-4201-9e56-aadeb427f98a', "{u'color': u'black', u'brand': u'Merida'}"]

Times (milliseconds):

[83.34112167358398, 35.54201126098633, 36.23318672180176, 35.546064376831055, 39.82996940612793, 35.067081451416016, 35.12001037597656, 34.34896469116211, 35.05301475524902, 35.52699089050293, 34.442901611328125, 33.95986557006836, 35.25996208190918, 35.26592254638672, 35.75301170349121, 35.50601005554199, 35.93301773071289, 32.97090530395508, 35.09712219238281, 35.33005714416504, 34.66916084289551, 34.97791290283203, 34.68203544616699, 34.09695625305176, 35.676002502441406, 33.01596641540527, 35.39609909057617, 33.804893493652344, 33.6918830871582, 34.37995910644531, 33.71405601501465, 35.18819808959961, 34.35802459716797, 34.191131591796875, 33.44106674194336, 34.84678268432617, 35.51292419433594, 33.80894660949707, 33.6911678314209, 36.14497184753418, 34.671783447265625, 35.28904914855957, 33.84900093078613, 36.21387481689453, 36.26894950866699, 34.7599983215332, 34.73687171936035, 34.715890884399414, 35.101890563964844, 35.4609489440918]

Average time (milliseconds):

35.9096717834

此时vehicles的租约持有者与客户请求的网关节点位于同一区域，因此该查询仅花费35.90ms，而类似查询在纽约则花费了96.25ms。

写性能

数据在不同区域分布也会影响写入性能。假设纽约有100人、洛杉矶也有100人，希望创建新的Movr账户：

* SSH到带有Python客户端的us-east1-b的机器
* 创建100个纽约用户

./tuning.py --host=<address of a node **in** us-east1-b> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'New Yorker', '111 East Street', '1736352379937347')" --repeat=100 --times

Times (milliseconds):

[710.5610370635986, 75.03294944763184, 76.18403434753418, 76.6599178314209, 75.54292678833008, 77.10099220275879, 76.49803161621094, 76.12395286560059, 75.13093948364258, 76.4460563659668, 74.74899291992188, 76.11799240112305, 74.95307922363281, 75.22797584533691, 75.01792907714844, 76.11393928527832, 75.35195350646973, 76.23100280761719, 75.17099380493164, 76.05600357055664, 76.4470100402832, 76.4310359954834, 75.02388954162598, 76.38192176818848, 78.89008522033691, 76.27677917480469, 75.12402534484863, 74.9521255493164, 75.08397102355957, 76.21502876281738, 75.15192031860352, 77.74996757507324, 73.84800910949707, 85.68978309631348, 75.08993148803711, 77.28886604309082, 76.8439769744873, 76.6448974609375, 75.1500129699707, 76.38287544250488, 75.12092590332031, 76.92408561706543, 76.86591148376465, 76.45702362060547, 76.61795616149902, 75.77109336853027, 81.47501945495605, 83.72306823730469, 76.41983032226562, 75.19102096557617, 74.01609420776367, 77.21996307373047, 76.61914825439453, 75.56986808776855, 76.94005966186523, 75.74892044067383, 76.63488388061523, 76.73311233520508, 75.73890686035156, 75.3028392791748, 76.58910751342773, 76.70807838439941, 76.36213302612305, 75.05607604980469, 76.99084281921387, 79.19192314147949, 75.69003105163574, 76.53594017028809, 75.3641128540039, 76.4620304107666, 75.81305503845215, 76.84993743896484, 75.74915885925293, 77.1799087524414, 76.67183876037598, 75.85597038269043, 77.18396186828613, 78.25303077697754, 76.66516304016113, 75.4399299621582, 76.98297500610352, 75.69122314453125, 77.4688720703125, 81.50601387023926, 76.74908638000488, 76.9951343536377, 75.34193992614746, 76.82991027832031, 76.4460563659668, 75.76298713684082, 76.63083076477051, 75.43802261352539, 76.47705078125, 78.95708084106445, 75.60205459594727, 75.70815086364746, 76.48301124572754, 76.65586471557617, 75.71196556091309, 74.09906387329102]

Average time (milliseconds):

82.7817606926

* SSH到带有Python客户端的us-west2-a的机器
* 创建100个洛杉矶用户

./tuning.py --host=<address of a node **in** us-west2-a> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'los angeles', 'Los Angel', '111 West Street', '9822222379937347')" --repeat=100 --times

Times (milliseconds):

[213.47904205322266, 140.0778293609619, 138.11588287353516, 138.22197914123535, 143.43595504760742, 139.0368938446045, 138.3199691772461, 138.7031078338623, 139.38307762145996, 139.53304290771484, 138.78607749938965, 140.59996604919434, 138.1399631500244, 138.94009590148926, 138.17405700683594, 137.9709243774414, 138.02003860473633, 137.82405853271484, 140.13099670410156, 139.08815383911133, 138.0600929260254, 139.01615142822266, 138.05103302001953, 137.76111602783203, 139.38617706298828, 137.42399215698242, 137.89701461791992, 138.40818405151367, 138.6868953704834, 139.13893699645996, 139.24717903137207, 138.7009620666504, 137.4349594116211, 137.24017143249512, 138.99493217468262, 138.77201080322266, 138.624906539917, 139.19997215270996, 139.4331455230713, 143.18394660949707, 138.0319595336914, 137.6488208770752, 137.27498054504395, 136.3968849182129, 139.0249729156494, 137.9079818725586, 139.37997817993164, 139.32204246520996, 140.045166015625, 137.9718780517578, 139.36805725097656, 139.6927833557129, 139.63794708251953, 138.016939163208, 145.32899856567383, 138.261079788208, 139.56904411315918, 139.6658420562744, 138.02599906921387, 139.7988796234131, 138.24796676635742, 139.9519443511963, 136.5041732788086, 139.43004608154297, 138.16499710083008, 138.2119655609131, 139.69111442565918, 140.30194282531738, 138.14496994018555, 140.00296592712402, 139.44697380065918, 139.35494422912598, 137.9709243774414, 140.78497886657715, 136.4901065826416, 138.44680786132812, 138.69094848632812, 139.2819881439209, 140.45214653015137, 138.3049488067627, 139.4188404083252, 139.9250030517578, 140.40303230285645, 138.7009620666504, 136.9321346282959, 139.20903205871582, 138.14496994018555, 140.14315605163574, 139.30511474609375, 139.58096504211426, 141.16501808166504, 138.66591453552246, 138.3810043334961, 137.39800453186035, 139.9540901184082, 138.4589672088623, 138.72814178466797, 138.3681297302246, 139.1599178314209, 139.29295539855957]

Average time (milliseconds):

139.702253342

结果显示创建一个纽约用户只需要82.78ms，而创建一个洛杉矶用户需要139.70ms。查看users表的数据分布情况：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SHOW EXPERIMENTAL\_RANGES FROM TABLE users;"

+-----------+---------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+-----------+---------+----------+----------+--------------+

| NULL | NULL | 51 | {2,6,7} | 2 |

+-----------+---------+----------+----------+--------------+

(1 row)

结果显示users表只有一个Range，每个zone一个Range副本，租约持有者在us-east1-b zone，这意味着：

* 在纽约创建用户时，请求访问租约持有者时不需要跨zone。但是由于写入操作需要来自其他区域的Range副本组的一致性确认，即写入必须等待来自us-west1-a(Oregon)或us-west2-a(Los Angeles)中的副本的确认后，才能提交，然后再返回结果给客户端。
* 在洛杉矶创建用户时，会有更多的网络跳跃，从而增加了网络延迟。请求首先需要路由到us-east1-b的租约持有者。租约持有者处理请求，同时必须等待来自us-west1-a(Oregon)或us-west2-a(Los Angeles)的副本的一致性确认后，才能提交，然后再返回结果给西部的客户端。

Step 13: 按城市分区

对于上述场景，改进读写延迟的最有效方法是按城市对数据进行[地理分区](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/)，这意味着改变数据映射到range的方式。表中指定城市的所有行及其索引信息将映射到特定区域，而不是整个表及其索引映射到全局范围。 这种方式定义Range，在配合使用[replication zone](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/)功能时，能够根据分区固定到指定的位置，从而确保来自特定城市用户的读写请求在对应区域内执行处理，数据不会跨区域。

应用License

* 地理分区特性为企业版功能，用户可以先[申请30天试用版License](https://www.cockroachlabs.com/pricing/start-trial/)
* 获得试用版License后，在节点上[应用License](http://doc.cockroachchina.baidu.com/#deploy/enterprise-licensing/#set-the-trial-or-enterprise-license-key)：

cockroach sql --insecure --host=<address of any node> --execute="SET CLUSTER SETTING cluster.organization = '<your org name>';"

cockroach sql --insecure --host=<address of any node> --execute="SET CLUSTER SETTING enterprise.license = '<your license>';"

定义分区

定义所有表及其二级索引的分区情况：

* users表

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER TABLE users \

PARTITION BY LIST (city) ( \

PARTITION new\_york VALUES IN ('new york'), \

PARTITION boston VALUES IN ('boston'), \

PARTITION washington\_dc VALUES IN ('washington dc'), \

PARTITION seattle VALUES IN ('seattle'), \

PARTITION san\_francisco VALUES IN ('san francisco'), \

PARTITION los\_angeles VALUES IN ('los angeles') \

);"

* 定义vehicles表及其二级索引分区

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER TABLE vehicles \

PARTITION BY LIST (city) ( \

PARTITION new\_york VALUES IN ('new york'), \

PARTITION boston VALUES IN ('boston'), \

PARTITION washington\_dc VALUES IN ('washington dc'), \

PARTITION seattle VALUES IN ('seattle'), \

PARTITION san\_francisco VALUES IN ('san francisco'), \

PARTITION los\_angeles VALUES IN ('los angeles') \

);"

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER INDEX vehicles\_auto\_index\_fk\_city\_ref\_users \

PARTITION BY LIST (city) ( \

PARTITION new\_york\_idx VALUES IN ('new york'), \

PARTITION boston\_idx VALUES IN ('boston'), \

PARTITION washington\_dc\_idx VALUES IN ('washington dc'), \

PARTITION seattle\_idx VALUES IN ('seattle'), \

PARTITION san\_francisco\_idx VALUES IN ('san francisco'), \

PARTITION los\_angeles\_idx VALUES IN ('los angeles') \

);"

* 定义rides表及其二级索引分区

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER TABLE rides \

PARTITION BY LIST (city) ( \

PARTITION new\_york VALUES IN ('new york'), \

PARTITION boston VALUES IN ('boston'), \

PARTITION washington\_dc VALUES IN ('washington dc'), \

PARTITION seattle VALUES IN ('seattle'), \

PARTITION san\_francisco VALUES IN ('san francisco'), \

PARTITION los\_angeles VALUES IN ('los angeles') \

);"

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER INDEX rides\_auto\_index\_fk\_city\_ref\_users \

PARTITION BY LIST (city) ( \

PARTITION new\_york\_idx1 VALUES IN ('new york'), \

PARTITION boston\_idx1 VALUES IN ('boston'), \

PARTITION washington\_dc\_idx1 VALUES IN ('washington dc'), \

PARTITION seattle\_idx1 VALUES IN ('seattle'), \

PARTITION san\_francisco\_idx1 VALUES IN ('san francisco'), \

PARTITION los\_angeles\_idx1 VALUES IN ('los angeles') \

);"

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="ALTER INDEX rides\_auto\_index\_fk\_vehicle\_city\_ref\_vehicles \

PARTITION BY LIST (vehicle\_city) ( \

PARTITION new\_york\_idx2 VALUES IN ('new york'), \

PARTITION boston\_idx2 VALUES IN ('boston'), \

PARTITION washington\_dc\_idx2 VALUES IN ('washington dc'), \

PARTITION seattle\_idx2 VALUES IN ('seattle'), \

PARTITION san\_francisco\_idx2 VALUES IN ('san francisco'), \

PARTITION los\_angeles\_idx2 VALUES IN ('los angeles') \

);"

最后，尝试删除rides表无用的索引

cockroach sql \

--insecure \

--database=movr \

--host=<address of any node> \

--execute="DROP INDEX rides\_start\_time\_idx;"

**NOTE:** rides包含1百万条记录，删除索引将花费几分钟时间。

[创建replication zones](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/#create-a-replication-zone-for-a-table-partition-new-in-v2-0)

基于节点位置，将不同城市的数据存储到特定节点上：

| **城市** | **locality** |
| --- | --- |
| New York | zone=us-east1-b |
| Boston | zone=us-east1-b |
| Washington DC | zone=us-east1-b |
| Seattle | zone=us-west1-a |
| San Francisco | zone=us-west2-a |
| Los Angelese | zone=us-west2-a |

**NOTE:** 由于节点位于3个特定的GCE区域，用户只需要配置节点zone=对应区域。假设每个区域使用多个zone，用户可以变更使用region=指定节点区域。

* users表分区

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.users.new\_york --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.users.boston --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.users.washington\_dc --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.users.seattle --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.users.san\_francisco --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.users.los\_angeles --insecure --host=<address of any node> -f -

* vehicles及其二级索引分区

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.new\_york --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.new\_york\_idx --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.boston --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.boston\_idx --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.washington\_dc --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.vehicles.washington\_dc\_idx --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.vehicles.seattle --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.vehicles.seattle\_idx --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.vehicles.san\_francisco --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.vehicles.san\_francisco\_idx --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.vehicles.los\_angeles --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.vehicles.los\_angeles\_idx --insecure --host=<address of any node> -f -

* rides表及其二级索引分区

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.new\_york --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.new\_york\_idx1 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.new\_york\_idx2 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.boston --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.boston\_idx1 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.boston\_idx2 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.washington\_dc --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.washington\_dc\_idx1 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-east1-b]' | cockroach zone set movr.rides.washington\_dc\_idx2 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.rides.seattle --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.rides.seattle\_idx1 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west1-a]' | cockroach zone set movr.rides.seattle\_idx2 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.san\_francisco --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.san\_francisco\_idx1 --insecure --host=<address of any node> -f -

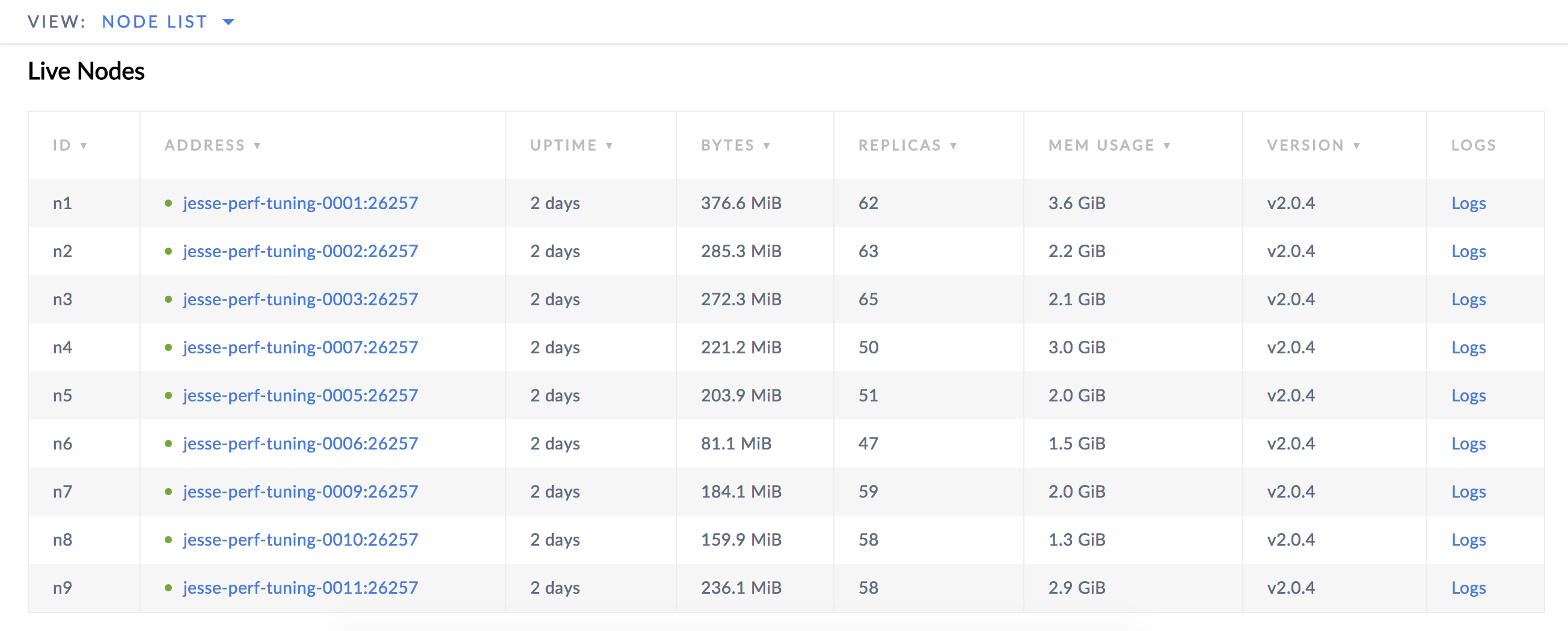
echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.san\_francisco\_idx2 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.los\_angeles --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.los\_angeles\_idx1 --insecure --host=<address of any node> -f -

echo 'constraints: [+zone=us-west2-a]' | cockroach zone set movr.rides.los\_angeles\_idx2 --insecure --host=<address of any node> -f -

Step 14: 检查分区后的均衡情况

经过数分钟的数据分布调整后，CockroachDB将根据配置的约束将数据重新分布到对应的区域。 登陆Admin界面<node address>:8080，点击**Node List**表，可以查看到节点的Range计数十分接近，但比分区前高得多： 

选取一台额外创建的未运行CockroachDB的机器，SSH登陆，针对vehicles表执行SHOW EXPERIMENTAL\_RANGES语句：

cockroach sql --insecure --host=<address of any node> --database=movr --execute="SELECT \* FROM [SHOW EXPERIMENTAL\_RANGES FROM TABLE vehicles] WHERE \"Start Key\" IS NOT NULL AND \"Start Key\" NOT LIKE '%Prefix%';"

+------------------+----------------------------+----------+----------+--------------+

| Start Key | End Key | Range ID | Replicas | Lease Holder |

+------------------+----------------------------+----------+----------+--------------+

| /"boston" | /"boston"/PrefixEnd | 95 | {1,2,3} | 2 |

| /"los angeles" | /"los angeles"/PrefixEnd | 111 | {7,8,9} | 9 |

| /"new york" | /"new york"/PrefixEnd | 91 | {1,2,3} | 1 |

| /"san francisco" | /"san francisco"/PrefixEnd | 107 | {7,8,9} | 7 |

| /"seattle" | /"seattle"/PrefixEnd | 103 | {4,5,6} | 4 |

| /"washington dc" | /"washington dc"/PrefixEnd | 99 | {1,2,3} | 1 |

+------------------+----------------------------+----------+----------+--------------+

(6 rows)

节点ID与zone映射关系为：

| **节点ID** | **Zone** |
| --- | --- |
| 1-3 | us-east1-b(South Carolina) |
| 4-6 | us-west1-a(Oregon) |
| 7-9 | us-west2-a(Los Angeles) |

配合节点ID与zone映射关系，分析输出结果可知：分区后，New York、Boston和Washington DC的数据副本分布在us-east1-b区域的节点1-3上，Seattle的数据副本分布在us-west1-a区域的节点4-6上，San Francisco和Los Angeles的数据副本分布在us-west2-a区域的节点7-9上。

Step 15: 测试分区后性能

分区后，与城市对应的数据存储在最近的节点上，预期针对特定城市的读写性能会有大幅度的提高。

参照**Step 12**再次进行读写性能测试：

读性能

再次假设场景：作为纽约的Movr管理员，我们想要获取当前正在使用的所有纽约自行车的ID和描述：

* SSH到带有Python客户端的us-east1-b的机器
* 执行查询：

./tuning.py --host=<address of a node **in** us-east1-b> --statement="SELECT id, ext FROM vehicles WHERE city = 'new york' AND type = 'bike' AND status = 'in\_use'" --repeat=50 --times

Result:

['id', 'ext']

['0068ee24-2dfb-437d-9a5d-22bb742d519e', "{u'color': u'green', u'brand': u'Kona'}"]

['01b80764-283b-4232-8961-a8d6a4121a08', "{u'color': u'green', u'brand': u'Pinarello'}"]

['02a39628-a911-4450-b8c0-237865546f7f', "{u'color': u'black', u'brand': u'Schwinn'}"]

['02eb2a12-f465-4575-85f8-a4b77be14c54', "{u'color': u'black', u'brand': u'Pinarello'}"]

['02f2fcc3-fea6-4849-a3a0-dc60480fa6c2', "{u'color': u'red', u'brand': u'FujiCervelo'}"]

['034d42cf-741f-428c-bbbb-e31820c68588', "{u'color': u'yellow', u'brand': u'Santa Cruz'}"]

...

Times (milliseconds):

[17.27890968322754, 9.554147720336914, 7.483959197998047, 7.407903671264648, 7.538795471191406, 7.39288330078125, 7.623910903930664, 7.172822952270508, 7.15184211730957, 7.201910018920898, 7.063865661621094, 7.602930068969727, 7.246971130371094, 6.966829299926758, 7.369041442871094, 7.277965545654297, 7.650852203369141, 7.177829742431641, 7.266998291015625, 7.150173187255859, 7.303953170776367, 7.1048736572265625, 7.218122482299805, 7.168054580688477, 7.258176803588867, 7.375955581665039, 7.013797760009766, 7.2078704833984375, 7.277965545654297, 7.352113723754883, 7.0400238037109375, 7.379055023193359, 7.227897644042969, 7.266044616699219, 6.883859634399414, 7.344961166381836, 7.222175598144531, 7.149934768676758, 7.241010665893555, 6.999969482421875, 7.40504264831543, 7.191896438598633, 7.192134857177734, 7.2231292724609375, 7.10296630859375, 7.291078567504883, 6.976127624511719, 7.338047027587891, 6.918191909790039, 7.070064544677734]

Average time (milliseconds):

7.48650074005

对比先前的输出结果，可以发现添加分区后，平均查询时间从96.25ms降至7.48ms。

写性能

假设纽约有100人、洛杉矶也有100人，希望创建新的Movr账户：

* SSH到带有Python客户端的us-east1-b的机器
* 创建100个纽约用户

./tuning.py --host=<address of a node **in** us-east1-b> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'new york', 'New Yorker', '111 East Street', '1736352379937347')" --repeat=100 --times

Times (milliseconds):

[9.378910064697266, 7.173061370849609, 9.769916534423828, 8.235931396484375, 9.124040603637695, 9.358882904052734, 8.581161499023438, 7.482051849365234, 8.441925048828125, 8.306026458740234, 8.775949478149414, 8.685827255249023, 6.851911544799805, 9.104013442993164, 9.664058685302734, 7.126092910766602, 8.738994598388672, 8.75997543334961, 9.040117263793945, 8.374929428100586, 8.384943008422852, 10.58506965637207, 8.538961410522461, 7.405996322631836, 9.508132934570312, 8.268117904663086, 11.46697998046875, 9.343147277832031, 8.31294059753418, 7.085084915161133, 8.779048919677734, 7.356166839599609, 8.732080459594727, 9.31406021118164, 8.460044860839844, 8.933067321777344, 8.610963821411133, 7.01904296875, 9.474039077758789, 8.276939392089844, 9.40704345703125, 9.205818176269531, 8.270025253295898, 7.443904876708984, 8.999824523925781, 8.215904235839844, 8.124828338623047, 8.324861526489258, 8.156061172485352, 8.740901947021484, 8.39996337890625, 7.437944412231445, 8.78000259399414, 8.615970611572266, 8.795022964477539, 8.683919906616211, 7.111072540283203, 7.770061492919922, 8.922100067138672, 9.526968002319336, 7.8411102294921875, 8.287191390991211, 10.084152221679688, 8.744001388549805, 8.032083511352539, 7.095098495483398, 8.343935012817383, 8.038997650146484, 8.939027786254883, 8.714914321899414, 6.999969482421875, 7.087945938110352, 9.23299789428711, 8.90803337097168, 7.808923721313477, 8.558034896850586, 7.122993469238281, 8.755922317504883, 8.379936218261719, 8.464813232421875, 8.405923843383789, 7.163047790527344, 9.139060974121094, 8.706092834472656, 7.130146026611328, 12.811899185180664, 9.733915328979492, 7.981061935424805, 9.001016616821289, 8.28409194946289, 7.188081741333008, 9.055137634277344, 9.569883346557617, 7.223844528198242, 8.78596305847168, 6.941080093383789, 8.934974670410156, 8.980989456176758, 7.564067840576172, 9.202003479003906]

Average time (milliseconds):

8.51003170013

对比先前的输出结果，可以发现创建100个纽约用户操作所花费的时间从82.78ms降至8.51ms。

* SSH到带有Python客户端的us-west2-a的机器
* 创建100个洛杉矶用户

./tuning.py --host=<address of a node **in** us-west2-a> --statement="INSERT INTO users VALUES (gen\_random\_uuid(), 'los angeles', 'Los Angel', '111 West Street', '9822222379937347')" --repeat=100 --times

Times (milliseconds):

[20.322084426879883, 14.09602165222168, 14.353036880493164, 25.568008422851562, 15.157938003540039, 27.19593048095703, 29.092073440551758, 14.515876770019531, 14.114141464233398, 19.414901733398438, 15.073060989379883, 13.965845108032227, 13.913869857788086, 15.218019485473633, 13.844013214111328, 14.110088348388672, 13.943910598754883, 13.73600959777832, 13.262033462524414, 14.648914337158203, 14.066219329833984, 13.91911506652832, 14.122962951660156, 14.724016189575195, 17.747879028320312, 16.537904739379883, 13.921022415161133, 14.027118682861328, 15.810012817382812, 14.811992645263672, 14.551877975463867, 14.912128448486328, 14.078140258789062, 14.576196670532227, 19.381046295166016, 14.536857604980469, 14.664888381958008, 14.539957046508789, 15.054941177368164, 17.20881462097168, 14.64700698852539, 14.211177825927734, 15.089988708496094, 14.193058013916016, 14.544010162353516, 14.680862426757812, 14.32490348815918, 15.841007232666016, 14.069080352783203, 14.59503173828125, 14.837026596069336, 14.315128326416016, 14.558792114257812, 14.645099639892578, 14.82701301574707, 14.699935913085938, 15.035152435302734, 14.724016189575195, 16.10708236694336, 14.612913131713867, 14.641046524047852, 14.706850051879883, 14.29295539855957, 14.779090881347656, 15.485048294067383, 17.444133758544922, 15.172004699707031, 20.865917205810547, 14.388084411621094, 14.241218566894531, 14.343976974487305, 14.602899551391602, 14.64390754699707, 13.908147811889648, 20.69687843322754, 15.130043029785156, 14.754056930541992, 14.123916625976562, 14.760017395019531, 14.25480842590332, 14.446020126342773, 14.229059219360352, 15.10000228881836, 14.275789260864258, 14.42098617553711, 14.935970306396484, 15.175819396972656, 27.69613265991211, 14.856815338134766, 14.902830123901367, 15.029191970825195, 15.143871307373047, 15.524148941040039, 14.510869979858398, 18.740177154541016, 14.97197151184082, 15.30003547668457, 15.158891677856445, 14.423847198486328, 35.25400161743164]

Average time (milliseconds):

15.7462859154

对比先前的输出结果，可以发现创建100个洛杉矶用户操作所花费的时间从139.70ms降至15.74ms。