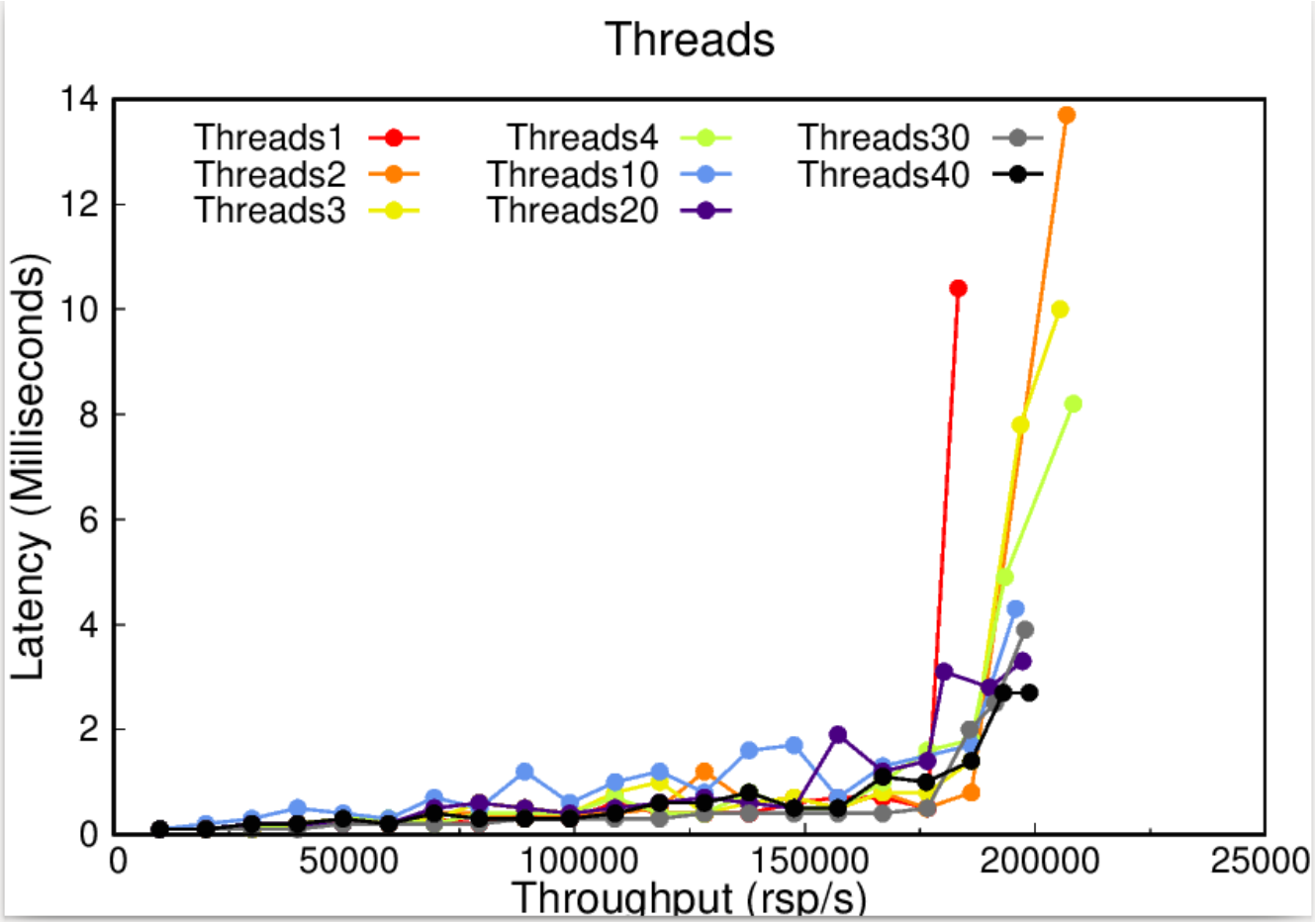
memcached 不同 thread 数量对应 throughput-latency 关系图分析



一，Memcached 网络模型

Memcached 主要是基于 Libevent 的事件库来实现网络线程模型的，主要涉及两个主要文件：memcached.c 和 thread.c 文件。Memcached 首先在主线程中会创建 main_base，Memcached 的主线程的主要工作就是监听和接收 listen 和 accpet 新进入的连接。当 Memcached 启动的时候会初始化 N 个 worker thread 工作线程，每个工作线程都会有自己的 LIBEVENT_THREAD 数据结构来存储线程的信息（线程基本信息、线程队列、pipe 信息）。worker thread 工作线程和 main thread 主线程之间主要通过 pipe 来进行通信。当用户有连接进来的时候，main thread 主线程会通过求余的方式选择一个 worker thread 工作线程。main thread 会将当前用户的连接信息放入一个 CQ_ITEM，并且将 CQ_ITEM 放入这个线程的 conn_queue 处理队列，然后主线程会通过写入 pipe 的方式来通知 worker thread 工作线程。当工作线程得到主线程 main thread 的通知后，就会去自己的 conn_queue 队列中取得一条连接信息进行处理，创建 libevent 的 socket 读写事件。工作线程会监听用户的 socket，如果用户有消息传递过来，则会进行消息解析和处理，返回相应的结果。

二，测试过程

测试环境的物理 CPU 个数为 2,每个物理 CPU 中 core 的个数 (即核数) 为 10, 逻辑 CPU 的个数为 40，CPU 型号为 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz。在测试环境中，我们安装了 Memcached 服务和 mcperf 压力测试工具，并使用 mcperf 压力测试工具调整连接的并发数和请求速度。在开启 memcached 服务器时，使用‘-t’参数可以指定要开启的线程数。线程越多，系统需要的线程调度时间越多。我们将 CPU 线程数设置成逻辑 CPU 的个数时，多余的系统调度的时间开销最少，效率最高。

我们分别将控制 memcached server 线程数的-t 参数设置为 1,2,3,4,10,20,30,40,在不同线程数的条件下创建 100 个并发连接，来连接本机的 11211 端口，即 Memcached 服务的默认端口号。连接创建的速度是每秒

10000 个，每一个连接发送“set”请求 1000 次，这 1000 次请求分别在每秒 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400, 2500 的请求速度下发送以模拟不同的吞吐量，发送的数据大小在 10~1024 个字节中正态分布。从而绘制出 memcached server 在不同线程数下，访问延迟(latency)与请求吞吐量(throughput)之间的关系图。

三，测试结果分析
由性能图可以看出，随着请求吞吐量的增大，当请求吞吐量超过某一阈值时，访问延迟会出现激增。当 Memcached 工作线程数增大时，对应的阈值也随之增大，并且访问延迟在请求吞吐量增大时的增长幅度明显减小。

这是由于 Memcached 使用多线程模式的多路复用 I/O 模型，不同的 worker thread 工作线程负责处理对应连接的读写事件。在 Memcached 工作线程数较小时，可以并发处理的连接数较小，因而在吞吐量增大时无法及时处理请求，会出现大量阻塞请求导致网络流量激增，从而导致访问延迟会出现激增；当线程数增大时，可以并发处理的连接数随之增大，因而访问延迟的增长幅度明显减小。