Heracles项目总结

张乾宇

1.项目背景

在当下的网络服务体系中，一个服务通常需要成百上千台机器共同完成（典型的服务包括搜索引擎，电子商务等等），服务完成的时间取决于所有机器中最晚完成的时间。因此，单台机器的服务尾延迟对整体服务的质量具有非常重要的影响。

为了保证服务的尾延迟在可以容忍的范围内，数据中心中的机器通常运行在较低的负载下，典型值约为10%，这造成了极大的资源浪费。

为了提高资源利用率，一种可行的方式是将延迟敏感型服务（LC）与批处理任务（BE）混部在同一台物理机器上，当资源空闲时可以运行批处理任务，以提高资源利用率。但这样带来的问题是，由于对资源（包括CPU core,cache等等）的共享，批处理任务会干扰延迟敏感服务的尾延迟，使之不能提供满足要求的服务。

因此，本项目的目标是：通过采取多种隔离措施（包括硬件和软件）对共享资源进行隔离，在保证延迟敏感程序尾延迟的情况下，尽可能的提高机器的资源利用率。

1. 问题重现

为了重现背景中提到的问题，我们使用Tailbench进行了具体的测试。

Tailbench是由MIT提供的一种尾延迟测量工具，包括8个测试程序和3种测试模式。测试程序包括xapian（搜索引擎）、silo（内存事务数据库）、img-dnn（神经网络）等等。3种测试模式分别为：集成模式，本地回环模式和网络模式。集成模式使用共享内存通信，不经过网络协议栈。本地回环模式使用网络协议栈，但使用本地回环端口，不经过具体的网络环境。网络模式采用完整的网络环境进行测试。

项目关心的共享资源包括：CPU core（包括hyperthread), cache, memory bandwidth, power, network bandwidth。其中power影响CPU的频率，由CPU自己控制，会使程序出现不稳定的行为，因此实验采取固定CPU频率的做法，暂时忽略power对实验的影响。因此测试主要针对其他4种共享资源进行。

实验针对4种共享资源的测试采取如下方法：

（1）hyperthread: 将LC程序固定在一个超线程上，将CPU干扰程序固定在与其具有相同物理核的超线程上，观察LC的延迟情况。这里的CPU干扰程序主要是类似于a+b的死循环。

（2）cache: 将LC程序固定在一个物理核上，将cache干扰程序固定在另一个物理核上，观察LC的延迟情况。这里的cache干扰程序是一个频繁读取数组的程序，步长设置为LLC(last level cache)的大小，使得每次访问均造成缓存缺失，干扰LC的缓存使用。

（3）memory bandwidth: 将LC程序固定在一个物理核上，将memory干扰程序固定在另一个物理核上，观察LC的延迟情况。这里的memory干扰程序与上面的cache干扰程序类似，但增大数组的容量，以便更好的干扰memory bandwidth。

（4）network bandwidth: 将LC程序固定在一个物理核上，将network干扰程序固定在另一个物理核上，观察LC的延迟情况。这里network干扰程序使用iperf，并且主要干扰服务器的出口带宽，因为入口带宽不易控制，因此暂未考虑。

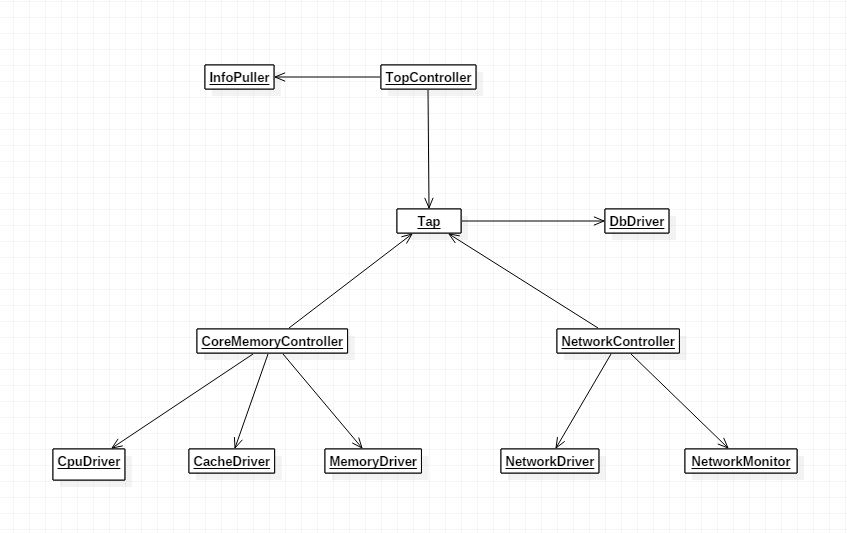
具体操作中张琛昱负责xapian，我负责silo。

根据实验结果（附录1）可以看出，资源的共享确实干扰了LC程序的尾延迟，需要采取一定的措施进行控制。

1. 项目实现

该项目主要参考Google的Herales论文进行实现。

3.1 总体架构



总体架构图

项目具有3个Controller: TopController， CoreMemoryController，NetworkController，三者通过Tap通信。

Tap是一个共享数据结构，它记录系统的整体状态，Controller将状态写入Tap，同时根据Tap中的状态进行操作。Tap同时还根据DbDriver进行BE任务的启动和销毁，DbDriver存储着一定数量的BE任务。

TopController通过InfoPuller获得LC程序的延迟和负载信息，决定当前系统是否开启BE任务，并将决策写入Tap中。

CoreMemoryController控制Cpu core，Cache， Memroy bandwidth三种资源的分配，因为这三种资源具有极强的关联性，因此放到了一起。CoreMemroyController根据Tap中的状态，采用梯度下降算法进行三种资源的分配。CpuDriver，CacheDriver，MemoryDriver执行具体的分配操作。其中MemroyDriver只具有监控内存带宽的功能。

NetworkController控制网络资源的分配。它通过NetworkMonitor获取实时的网络资源使用情况，采用一定的算法，并通过NetworkDriver执行具体的分配操作。

在项目中我主要负责网络模块的设计与实现。

3.2 网络模块

网络模块主要有3个类：NetworkController，NetworkDriver，NetworkMonitor。NetworkController负责分配的算法，NetworkDriver执行分配的具体操作，NetworkMonitor监控程序的实时网络流量。

NetworkDriver主要采用cgroups和TC（Traffic Control）实现。

cgroups是由linux内核提供的资源分配机制，这里主要使用net\_cls模块。net\_cls会给LC程序和BE程序发出的网络数据包打上不同的标签（分类号）。

TC是由Linux内核提供的网络资源分配机制，这里主要使用HTB（分层令牌桶）分配策略。TC根据网络数据包上的标签将数据包分发到不同的分类中，然后使用HTB控制不同分类的带宽上限。

NetworkMonitor主要使用eBPF实现。这里使用eBPF监控net:net\_dev\_xmit这个tracepoint。net\_dev\_xmit位于TC的出口，TC每发送一个网络数据包都会触发该tracepoint。该tracepoint被触发后会执行指定的eBPF程序，eBPF程序根据网络数据包上的分类号统计一定时间内某个分类产生的总流量，并将数据传到用户态供其他程序使用。

NetworkController首先通过NetworkMonitor获得LC和BE分别使用的实时带宽，然后执行下面的算法，最后通过NetworkDriver分配新的带宽。

NetworkController首先预留总带宽的20%，保留给LC使用。然后用总带宽的80%减去LC的实时带宽，将剩余带宽分配给BE。如果相减之后为负值，则BE未获得带宽。如果LC需要更多的带宽，将首先使用保留的20%，NetworkController会在下一次分配中减少BE的带宽，将更多的带宽分配给LC。

4.项目评价

heracles运行在一台xeon E5-2683服务器上。对每个延迟敏感程序（Tailbench中）设计4中测试场景：无干扰，干扰cache，干扰memory bandwidth，干扰network bandwidth。每个测试场景中记录QPS（每秒请求数），CPU load，memory bandwidth，network bandwidth。其中memory bandwidth由于测试工具的原因，测试波动较大。

具体操作中张琛昱负责xapian，我负责silo。

根据测试数据（附录2），heracles基本实现了在保证LC尾延迟的前提下，提高系统资源利用率的目标。

5. 项目改进

因为时间关系，项目实现依然有改进的空间，主要集中在以下方面：

（1）测试中会出现分配给BE6个核之后，LC延迟剧增，然后BE被关闭的周期现象，可以考虑能否使BE稳定在5个核的分配策略。

（2）网络利用率未能达到接近100%，当BE带宽过多时，依然会影响LC的延迟。可以考虑在控制带宽的情况下同时引入优先级的策略。

（3）因为时间问题，当前版本只支持同时运行1个LC和1个BE，可以考虑扩展成同时运行1个LC和多个BE。

（4）power的问题可以继续考虑。

6.总结

项目验证了共享资源会影响LC程序尾延迟的现象，并通过heracles实现了在保证LC尾延迟的前提下，提高系统资源利用率的目标。

附录1 silo干扰数据

silo netwrok test collection

silo network test no interfere

cpu: intel i9-7960x closing hyperthread.

os: ubuntu16.04 linux4.13.0

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 15 0.317 29.366

4000 400000 30 0.417 25.420

6000 600000 45 5.656 1158.200

silo network test interfere cache

cpu: intel i9-7960x closing hyperthread

os: ubuntu16.04 linux4.13.0

interfere: interfere\_cache\_random \* 7

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 15 0.338 21.066

4000 400000 30 0.581 29.716

6000 600000 50 11.475 1095.448

silo network test interfere cache 1

cpu: intel i9-7960x closing hyperthread

os: ubuntu16.04 linux4.13.0

interfere: interfere\_cache\_random \* 15

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 25 0.529 29.759

4000 400000 40 6.222 29.858

6000 600000 55 15.308 1033.760

silo network test interfere network

cpu: intel i9-7960x closing hyperthread

os: ubuntu16.04 linux4.13.0

interfere: iperf //这里干扰的应该是入口带宽，后来经过讨论认为应该干扰出口带宽

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 15 0.459 29.514

4000 400000 30 0.577 29.618

6000 600000 45 6.821 1053.816

silo network test no interfere

cpu: intel i7-6700

os: ubuntu16.04 linux4.13.0

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 15 0.560 24.245

4000 400000 25 0.575 26.513

6000 600000 35 1.424 104.519

silo network test interfere hyperthread

cpu: intel i7-6700

os: ubuntu16.04 linux4.13.0

interfere: interfere\_cache\_random on the same physical core

QPS MAXREQS cpu load(%) 95th latency(ms) max latency(ms)

2000 200000 20 0.504 20.290

4000 400000 35 2.394 23.297

6000 600000 55 13.416 1826.231

附录2 heracles控制下的silo干扰测试

silo test with heracles

environment

Xeon(R) CPU E5-2683 v4 @ 2.10GHz with performance state

Ubuntu 16.04 Linux 4.13.0

Gigabit Ethernet

program statics

8 threads, max QPS = 10000, slo latency = 6ms (actually 4.761ms @ QPS = 10000)

baseline

QPS latency(ms) CPU utilization(%) mem BW(bytes/s) net bw(bits/s)

2000 3.900 30 20 1844568

4000 3.918 60 30 3819816

6000 4.024 80 50 5450280

8000 4.093 100 70 6705600

10000 4.652 120 90 8361144

iperf

QPS latency(ms) CPU utilization(%) mem BW(bytes/s) net bw(bits/s)

LC / total total LC / BE

2000 5.203 30 / 30 49 1684320 / 688264752

4000 5.465 50 / 50 52 3248240 / 686177520

6000 4.576 70 / 70 48 3899544 / 685655712

8000 4.208 100 / 100 73 6879840 / 0

10000 4.899 120 / 120 102.5 8224920 / 0

latency under iperf test is not stable, the network controller / driver needs to be upgraded, maybe to be attached with priority policy.

streamCache

./stream-llc\_multithread 8 32000 100

QPS latency(ms) CPU utilization(%) mem BW(bytes/s) net bw(bits/s)

LC / total total LC

2000 3.821 30 / 630 12016 1861992

4000 3.841 50 / 650 13927 3651648

6000 3.942 70 / 670 14844 5344944

8000 4.122 100 / 100 71 6818576

10000 4.814 120 / 120 87 8233368

stream Mem

./stream-llc\_multithread 8 320000 1000

QPS latency(ms) CPU utilization(%) mem BW(bytes/s) net bw(bits/s)

LC / total total LC

2000 3.891 30 / 630 25709 1882056

4000 3.825 60 / 650 26007 3726888

6000 4.009 75 / 575 25772 5353920

after BE gets 6 cores, the latency of LC boom, then BE is disabled and a new peroid starts.

8000 4.112 100 / 100 78 6852648

10000 4.911 120 / 120 101 8414208