针对高并发低延迟运行环境低配的程序优化建议

## 语言选择：

高并发场景下主要限制为系统连接数限制，每个连接消耗的系统性能主要表现为内存增加。当需要对大量数据进行处理时CPU会有一个激增。数据缓存以及永久化存储时大量io读写，包括网络io和文件操作io。

选型上主要考虑数据收发性能和连接稳定性。在运行环境配置较低时建议选用静态同时不使用虚拟机的语言，如c,c++,golang。主要原因为虚拟机会损耗一部分性能，同时动态语言没有强类型定义会在运行时消耗大量资源进行消耗，通常计算能力会比静态语言差一个量级，即使进行过解释器优化也会比静态语言差很多。

## 主要痛点：

1. 连接数无法达到
2. 延迟高或抖动
3. 内存激增
4. 服务部署困难

## 针对连接数无法达到情况：

1. linux系统限制：

系统全局允许分配的最大文件句柄数:

sysctl -w fs.file-max=2097152

sysctl -w fs.nr\_open=2097152

echo 2097152 > /proc/sys/fs/nr\_open

持久化 ‘fs.file-max’ 设置到 /etc/sysctl.conf 文件:

fs.file-max = 1048576

/etc/systemd/system.conf 设置服务最大文件句柄数:

DefaultLimitNOFILE=1048576

/etc/security/limits.conf 持久化设置允许用户/进程打开文件句柄数:

\* soft nofile 1048576

\* hard nofile 1048576

TCP 协议栈网络参数

并发连接 backlog 设置:

sysctl -w net.core.somaxconn=32768

sysctl -w net.ipv4.tcp\_max\_syn\_backlog=16384

sysctl -w net.core.netdev\_max\_backlog=16384

可用知名端口范围:

sysctl -w net.ipv4.ip\_local\_port\_range='1000 65535'

TCP Socket 读写 Buffer 设置:

sysctl -w net.core.rmem\_default=262144

sysctl -w net.core.wmem\_default=262144

sysctl -w net.core.rmem\_max=16777216

sysctl -w net.core.wmem\_max=16777216

sysctl -w net.core.optmem\_max=16777216

sysctl -w net.ipv4.tcp\_rmem='1024 4096 16777216'

sysctl -w net.ipv4.tcp\_wmem='1024 4096 16777216'

TCP 连接追踪设置:

sysctl -w net.nf\_conntrack\_max=1000000

sysctl -w net.netfilter.nf\_conntrack\_max=1000000

sysctl -w net.netfilter.nf\_conntrack\_tcp\_timeout\_time\_wait=30

1. 程序本身成为瓶颈：

通常表现为新链接建立成功不久就断开，这种情况需要时如果服务端设计是无状态则可以通过开启新程序通过nginx反向代理来负载均衡（python服务经常做法），如果设计为有状态则需要考虑改成分布式设计或微服务模式。这两种不在讨论范围内。

## 针对高延迟情况：

首先排除网络不稳定问题，如果是网络问题需要进行链路追踪不在考虑范围内。

1. io操作导致阻塞：
2. 所有数据库操作交给新线程或其他进程解决，如果需要返回结果则开启协程来接受返回，使主线程所有操作都为异步，目的为不阻塞其他操作。
3. 文件读写操作尽量避免或只读取一次后缓存到内存中
4. 网络请求全部改成异步请求
5. 数据计算占用大量cpu:
6. 优化算法
7. 采用静态语言重写算法，程序调用静态语言编译好的库
8. 尽量使用多核并行计算
9. 最终确定无法解决那只能换更好的CPU来进行计算（毕竟程序服务质量优先）
10. Cpu性能不稳定：

这种情况出现在云服务器共享cpu的机器上，购买服务器时注意是否有cpu使用基线

1. 大量请求缓存服务：

一般程序内会使用redis,memcache之类的服务来进行缓存，通常是不会造成瓶颈，而在程序设计时可能会需要大量读写缓存服务，会将时间耗费在建立连接和大量指令执行上。

1. 针对热数据读取：当有大量热数据要进行读取最好做一次内存缓存数据只从缓存服务读取一次之后热数据不再从缓存服务读取
2. 建立缓存服务连接池
3. 针对热数据写入缓存：内存缓存同步更新，缓存服务异步更新，同时使用redis的pipeline来批量写入数据
4. 优化缓存的存储结构并尽量避免使用keys等需要全扫描的指令
5. 如果要redis来执行事务类操作建议直接使用redis-lua不推荐使用redis的watch指令来保证一致性

## 针对内存激增情况：

1. 由高连接引发的内存激增：
2. 先测试好单个服务能支持的连接峰值，在启动时建立一定数量连接池来避免瞬间高连接带来的内存激增影响服务。
3. 做好连接的管理，可以使用队列来进行限流避免雪崩
4. 即使对断开的连接进行清理
5. 大量的缓存数据

及时释放非热数据，同时合理设置超时时间

## 针对服务部署困难情况：

如果是linux程序，直接制作docker镜像设置配置文件挂载点即可，如果有其他服务依赖可以使用docker-compose来设置

如果windows应用建议标准配置一次之后在云服务做成镜像。

建议所有用到ip的配置全部使用域名来配置，在部署的时候修改hosts即可

## 结论：

修改系统限制，

采用分布式或微服务架构，

所有阻塞操作都异步进行，

队列管理连接并及时释放损坏的连接，

建立连接池，

数据批量写，

热数据程序内缓存，

计算瓶颈用静态语言重写，

超时时间合理设置，

合理负载均衡，

硬件瓶颈直接加配！