**BigRLab使用及开发说明**

2017,8

[1 简介 3](#_Toc490748975)

[1.1 系统架构 3](#_Toc490748976)

[1.2 代码目录结构 3](#_Toc490748977)

[2 使用方法 4](#_Toc490748978)

[2.1 ApiServer启动 4](#_Toc490748979)

[2.1.1 前端运行 5](#_Toc490748980)

[2.1.2 后端运行 6](#_Toc490748981)

[2.2 启动算法服务 6](#_Toc490748982)

[2.3 使用线上服务 7](#_Toc490748983)

[2.3.1 Restful API 8](#_Toc490748984)

[2.3.2 命令行批处理 8](#_Toc490748985)

[3 二次开发 10](#_Toc490748986)

[3.1 依赖库 10](#_Toc490748987)

[3.2 添加新算法服务 10](#_Toc490748988)

# 1 简介

BigRLab是一个分布式计算资源分配平台，利用多机并行处理提高业务请求吞吐量和平均响应速度。对外提供两种访问接口：用于在线请求的Restful API和用于批处理请求的命令行终端。

## 1.1 系统架构

整个系统可分为前端请求接收分发模块ApiServer和后端算法服务器集群Algorithm Servers；ApiServer收到外部请求后按照负载均很的原则将事务请求分发到算法集群上，算法集群将处理结果通过ApiServer返回给请求者。详见文档《BigRLab设计文档》。

## 1.2 代码目录结构

本项目github存放地址为：<https://github.com/charles-pku-2013/BF_BigRLab>

代码目录结构如下图所示：



其中common目录存放各模块共用的代码；api\_server目录存放ApiServer模块的实现；algorithms目录存放当前实现的线上服务算法，运行在算法集群上；CppSDK目录存放从外部执行BigRLab命令的实现，不必进入BigRLab自己的shell；services目录存放算法服务的RPC请求客户端实现（详见《BigRLab设计文档》）；others目录存放一些辅助工具，比如文本预处理，不属于BigRLab工程。

# 2 使用方法

## 2.1 ApiServer启动

启动ApiServer命令一般格式为：

[GLOG\_logtostderr=1] ./apiserver.bin [-port] [-alg\_mgr\_port] [-n\_io\_threads] [-n\_work\_threads] [-b]

其中GLOG\_logtostderr=1是控制LOG日志输出的环境变量，这里是将log输出到终端上，BigRLab中所有的模块日志输出都是用glog实现的，日志输出方式有两种，一种是输出到终端屏幕上，像刚刚的例子，另一种是输出到文件中，此时应这样设置：

GLOG\_log\_dir=. ./apiserver.bin …

本例是将日志文件存放到当前目录中，也可以指定其他存放位置。

这种在启动程序命令之前设置环境变量方式只对所启动的程序生效，若要使全局生效可以这样设置：

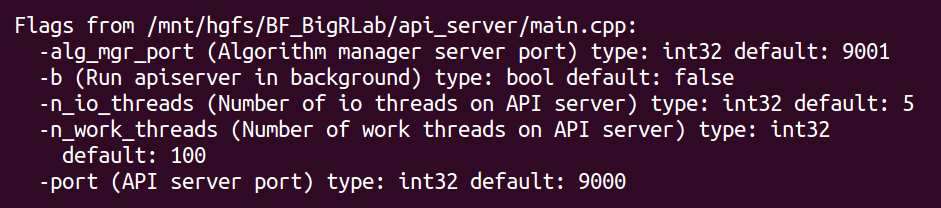
export GLOG\_log\_dir=.

这样就不必每次启动程序命令前都加上环境变量设置语句。若要取消刚才的设置，只需要执行：

unset GLOG\_log\_dir

ApiServer其他启动参数包括：-port指定Restful API对外服务端口，默认9000；-alg\_mgr\_port指定与算法集群服务器通信端口；-n\_io\_threads和-n\_work\_threads分别指定http server中io线程的数量和worker线程的数量，一般取默认值即可; -b指定ApiServer在前端运行还是在后端作为一个服务进程运行，前端运行会有一个交互shell，后端运行需要通过CppSDK里的工具bigrcli与之交互。

可以通过执行 ./apiserver.bin –help 找到如下部分查看具体参数含义：

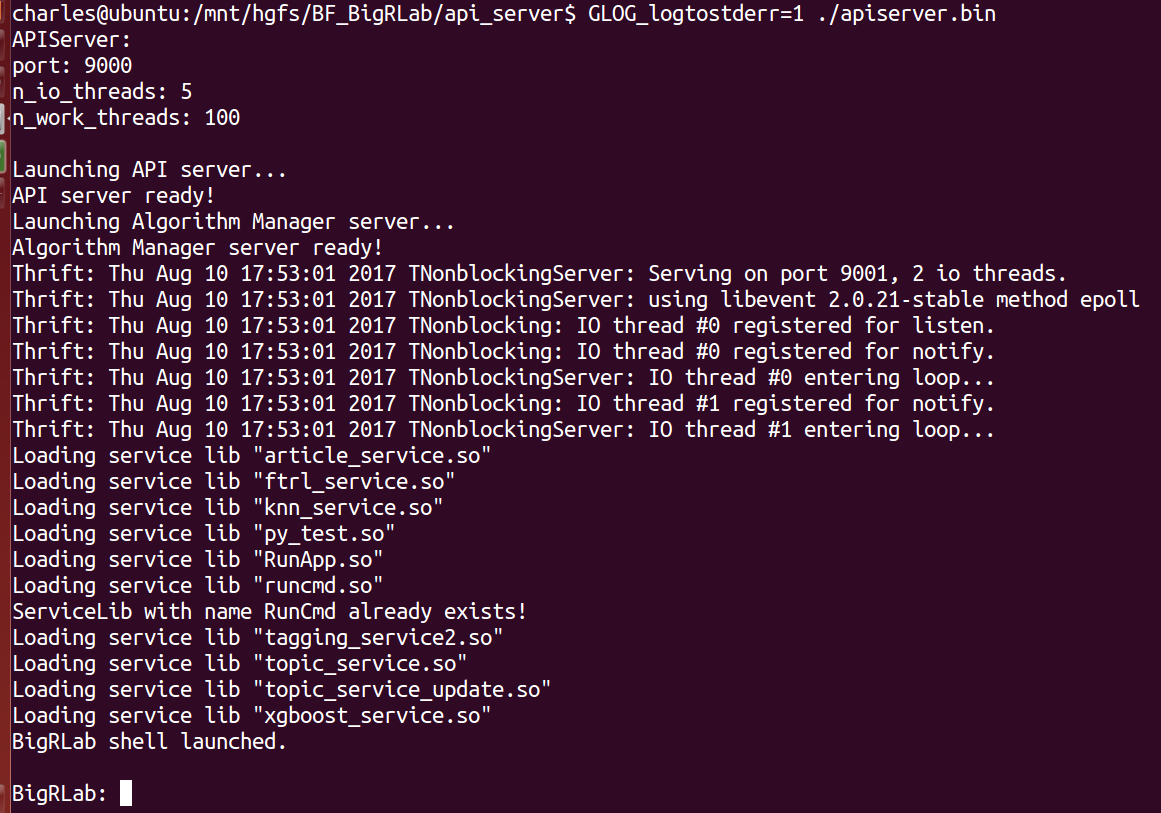


### 2.1.1 前端运行

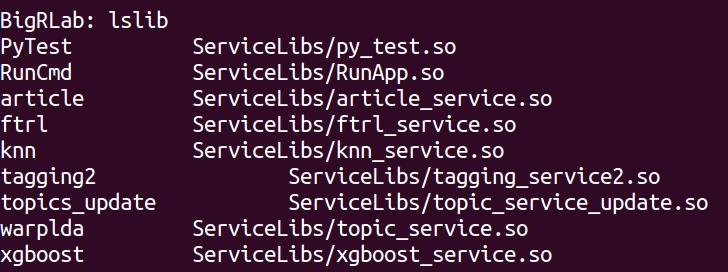
进入工程的api\_server目录运行：

GLOG\_logtostderr=1 ./apiserver.bin

进入ApiServer的交互shell如图：



此时可以执行各种命令如lslib查看系统所支持的算法，如图：



执行lsservice查看线上运行的算法服务如图：



目前没有启动任何算法服务器，所以没有运行的线上服务。后文会介绍如何添加服务。

要退出ApiServer输入命令quit即可。

### 2.1.2 后端运行

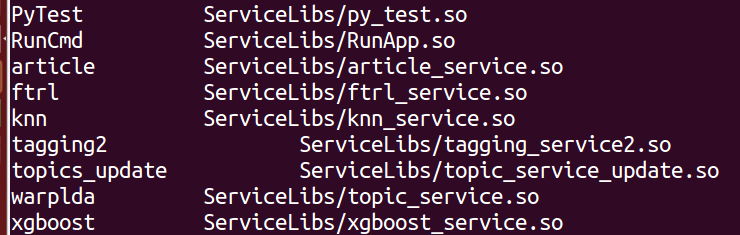
进入api\_server目录运行：

GLOG\_log\_dir=. nohup ./apiserver.bin -port 10010 -alg\_mgr\_port 10011 -b &

会启动ApiServer作为后端服务进程运行(本次没有使用默认端口)，log文件存放在当前目录下。

若要与后端运行的ApiServer进行交互，需要bigrcli工具，进入CppSDK目录，bigrcli.cpp是该工具的实现，编译和使用方法在该文件开头部分。编译完成后运行：

./bigrcli -server http://127.0.0.1:10010 lslib



再运行：./bigrcli -server http://127.0.0.1:10010 lsservice 得到“No running service”的输出，可见和前端交互式运行的结果一样。

## 2.2 启动算法服务

下面以knn寻找相似词算法为例介绍如何添加算法服务。

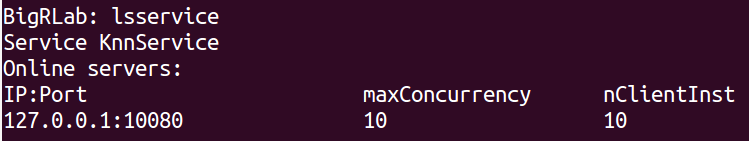
首先启动ApiServer，前端后端均可。

然后进入algorithm/knn目录，在main.cpp文件开始处有介绍使用方法，将命令copy出来，略作修改即可。进入build目录，输入：

GLOG\_logtostderr=1 ./wordknn.bin -idata knn\_test/model.vec -idx knn\_test/index.ann -wt knn\_test/words\_table.txt -algname KnnService -algmgr localhost:9001 -port 10080

其中红色字体部分是每个算法服务必须有的参数，其他参数都是根据具体算法处理的业务设置。-algname是本算法实例进程所属的service名称，同一个service下可能有不止一个的算法实例；-algmgr是向ApiServer的注册地址，与ApiServer启动时-alg\_mgr\_port参数所对应；-port是本算法服务实例在本机提供的访问端口，只供ApiServer访问。

在ApiServer端执行命令lsservice查看在线服务：

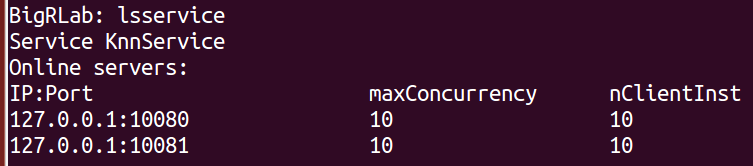


可以看到当前线上有一个Service名为KnnService，就是启动时通过-algname指定的，有一个server实例，在本机运行，服务端口10080。

以同样的方式再启动一个服务实例，因为是在统一台主机上操作，所以端口号必须换一个，实际工作中完全可以在局域网内不同的主机上启动。

GLOG\_logtostderr=1 ./wordknn.bin -idata knn\_test/model.vec -idx knn\_test/index.ann -wt knn\_test/words\_table.txt -algname KnnService -algmgr localhost:9001 -port 10081

这次在端口10081上启动，在ApiServer端执行命令lsservice可以看到：



KnnService服务下面比原来多了一个实例。

## 2.3 使用线上服务

线上服务接受两种请求方式，1：Restful API，将请求正文放在http文本里，一般以json封装，返回结果也是http文本；2：命令行批处理，通过命令行指定输入输出文件和其他参数，输入文件按行拆分成请求，输出文件格式为： 行号 请求结果，行号从0开始，但顺序是打乱的，一般不和源文件保持一致，但行号是一一对应的，必要时需要按行号重新排序。

### 2.3.1 Restful API

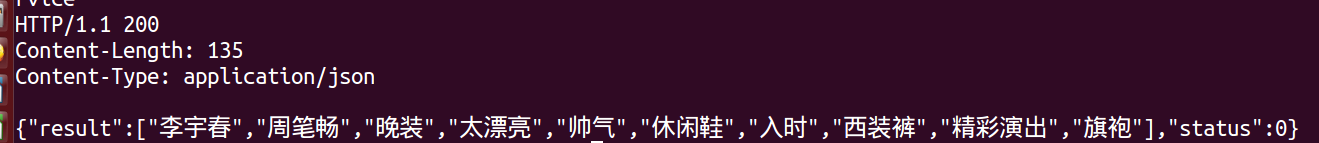
Restful API 请求URL为：

http://ip:port/service

请求正文内容为json封装，具体参数因算法而异，以刚刚启动的knn为例，用curl作为http client工具，在任何一台能直接访问ApiServer机器上运行：

curl -i -X POST -H "Content-Type: BigRLab\_Request" -d '{"item":"李宇春","n":10}' http://$ApiServerIP:9000/KnnService

会得到如下返回结果：



返回结果内容会因算法而异，一般status = 0表示成功，其他值表示失败。

### 2.3.2 命令行批处理

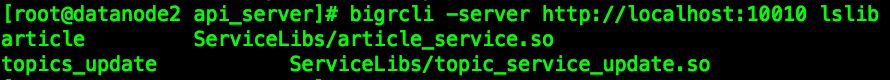
以主题预测topics\_update为例，首先编译service插件，进入目录BF\_BigRLab/services/topics\_update，按照编译cmake工程的方法编译，编译后的结果topic\_service\_update.so会自动存放在api\_server/ServiceLibs目录下[[1]](#footnote-1)。

（重新）启动ApiServer，用lslib命令检查topics\_service\_update.so已经正确加载。进入api\_server目录执行：

GLOG\_log\_dir=. nohup ./apiserver.bin -port 10010 -alg\_mgr\_port 10011 -b &

然后执行（需要将bigrcli工具拷贝到/usr/local/bin目录下）：

bigrcli -server http://localhost:10010 lslib



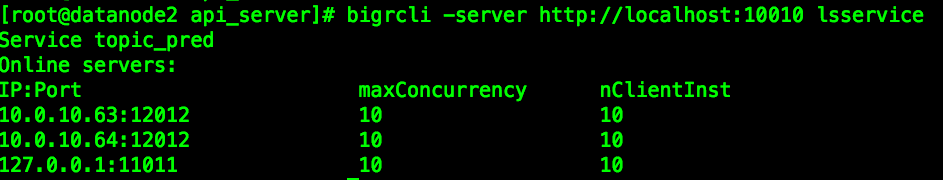
进入algorithm/topic\_update/build目录编译并启动算法服务（可在多台主机上启动）：

GLOG\_log\_dir=. nohup ./demo -algname topic\_pred -algmgr 10.0.10.62:10011 -port 12012 -vec clusterid -vecdict ../data/text\_class\_w2v\_cluster.txt -idx ../data/text\_class\_index.ann -label ../data/text\_class.index.cn &

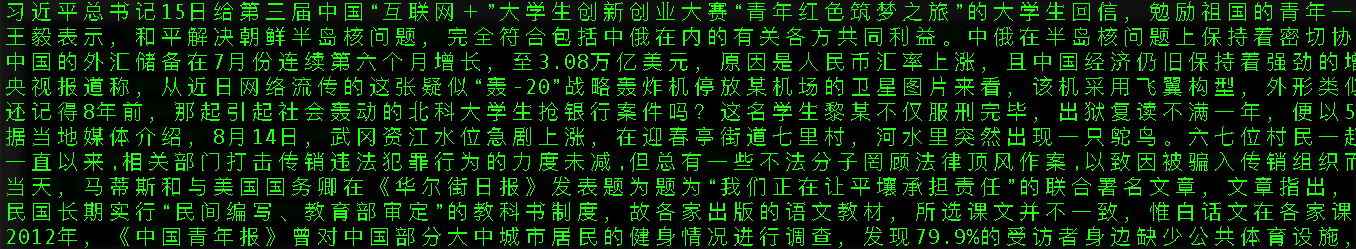
启动完成后到ApiServer所在主机执行：

bigrcli -server http://localhost:10010 lsservice

查看线上服务状态如图：



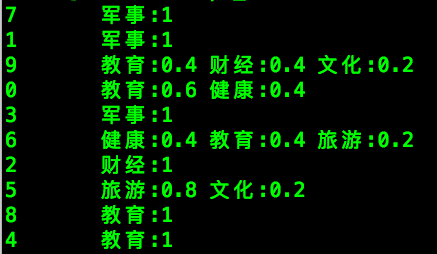
进入api\_server目录，将事先准备好的语料文本corpus.txt放入data目录下，每一行是一篇待预测主题的文章：



执行命令：

bigrcli -server http://localhost:10010 service topic\_pred in:data/corpus.txt out:data/topics.txt topk:5 req:label[[2]](#footnote-2)

这里指定输出文件是data/topics.txt，如果文件很大可能需要等待一段时间，当屏幕上出现”Job Done!”时候表示作业执行完成。查看输出文件中的预测结果：



第一列是对应源文件中的行号，从0开始，是乱序的，后面是预测结果，权值越大表示可能性越高。

# 3 二次开发

## 3.1 依赖库

整个分布式框架是基于thrift RPC通信的，所以Apache Thrift是必须有的库，无论是ApiServer还是算法服务。Thrift库提供了多种服务器server，如simple server， threaded server， Nonblocking server等，本系统中全部使用Nonblocking Server，其编译依赖库也与其他几种server不一样。除了要链接libthrift.so之外，还要链接libthriftnb.so libevent.so。

C++ boost库在本系统中也广泛使用，大多数只使用相应的头文件，需要编译时链接的库主要有libboost\_system.so, libboost\_thread.so, libboost\_iostreams.so，是具体算法业务而定。

此外ApiServer实现Restful API使用cppnetlib库处理http请求，具体详见ApiServer目录下的CMaleLists.txt和各算法服务实现里的CMake文件或Makefile。

## 3.2 添加新算法服务

本系统有很好的兼容性，一个现有的开源算法，只需要按照一定的规范重新封装一下，就可以在本系统上运行。具体而言就是在算法主程序上实现Thrift Server；Thrfit client通过动态链接库实现放在services目录下，ApiServer不需要做任何改动。具体详见算法服务：knn, py\_test, 包括services目录和algorithm目录，代码有详细注释。

1. Service中有些项目不是用cmake而是用Makefile，对这些项目需要手动将编译后的.so文件移动到api\_server/ServiceLibs目录中 [↑](#footnote-ref-1)
2. 其他算法的使用方法一般在响应services目录中的头文件开始注释处说明，命令行中若用到相对路径，当前目录是api\_server目录。 [↑](#footnote-ref-2)