BigRLab设计文档

2017 年 8 月

# 目录

[目录 1](#_Toc490039744)

[1. 系统需求分析 2](#_Toc490039745)

[1.1 性能需求 2](#_Toc490039746)

[1.2 功能需求 3](#_Toc490039747)

[1.3 非功能需求 3](#_Toc490039748)

[2. 系统设计与实现 5](#_Toc490039749)

[2.1 系统整体架构 5](#_Toc490039750)

[2.2 Service定义 5](#_Toc490039751)

[2.3 ApiServer的设计与实现 7](#_Toc490039752)

[2.3.1 Service管理模块 7](#_Toc490039753)

[2.3.2 请求处理模块 8](#_Toc490039754)

[2.3.3 算法服务管理模块 9](#_Toc490039755)

[2.3.4 负载均衡和异常处理 11](#_Toc490039756)

[2.4 算法服务的设计与实现 11](#_Toc490039757)

# 系统需求分析

在线广告点击行为预测作为广告投放的一个重要环节，其流程涉及一系列复杂的数据处理与计算，根据业务类型和实现方法的不同，自然有其特殊的需求。

## 1.1 性能需求

作为一个可商用的在线广告点击预测系统，在性能上不仅要求有很高的响应速度，而且还要能够及时地更新在线预测所使用的模型，保证使用的是最新最近的数据。此外对预测结果的准确度也是越高越好，因为这将直接影响广告收益。

在响应速度方面，为了能够达到毫秒级的响应速度，除了采取线下训练，线上预测并更新模型的方法之外，本文专门为这个在线广告点击预测系统实现了一个基于MPI的分布式计算集群，将收到的请求按照负载均衡的原则平均分配到集群中的每台主机上，借此提高并发度，加快响应速度。分布式集群架构如图3.1所示：

图 3.1系统整体架构图

其中APIServer是一个http服务器，负责接收请求数据，然后把收到的请求平均分配到计算集群上，集群中的每台主机都有一个统一的算法（LR）模型，计算主机要做的工作是：1）请求数据预处理，主要是特征提取和转换；2）用现有的LR模型结合请求数据特征做出预测，即用户是否会点击该广告；3）将特征数据缓存到Redis服务器上，并待用户实际行为（点击动作是否发生）后更新特征缓存中的记录。模型在线训练更新服务会定期用Redis特征缓存里积累的数据对线上模型做增量训练更新，然后分发给每台计算主机用作后续预测。

根据实际业务需求，本系统设计目标要求线上请求响应时间10ms以内，每秒能处理至少4000个线上预测请求。

## 1.2 功能需求

本系统的主要功能是利用用户的微博文章数据做在线广告点击率预测。其中特征工程，即将用户的微博文章这一非结构化特征转换成结构化特征，是关键环节。业务流程如图3.2所示：

通过文本处理工具转换为数值向量

通过GBDT特征工程转换为结构化特征

和已有的结构化特征结合成新的特征

用逻辑回归(LR)模型进行训练或预测

图 3.2 系统业务流程

依据此业务流程，系统需要实现的功能主要有：a)文本处理，将用户的微博文章转换成数值向量，主要通过jieba工具实现，将在下一章中详细介绍；b)特征工程，用GBDT方法从用数值向量表示的文章集合中发现新特征；c)逻辑回归(LR)模型算法实现，在第二章已详细介绍。

## 1.3 非功能需求

除了支持业务的功能需求和性能需求之外，非功能需求主要包括易用性，可扩展性和准确度。

易用性方面：要求分布式集群上的所有算法通过APIServer对外提供两种服务接口，一是基于WebService的实时线上请求，主要用于提供在线服务的线上算法；二是命令行请求，用于批处理数据文件，主要用于线下算法如文本处理、分词、特征工程。

可扩展性方面：要求能在系统上做二次开发，要在系统上添加一个新的算法服务需要做的工作有：a)将算法实现按照系统规范进行改造封装，部署于算法集群上；b)在ApiServer上实现该算法的请求处理和任务分发功能，称其为算法的proxy。其架构如图3.3所示：



ApiServer

计算集群

任务分发

任务分发

任务分发



外部请求

图 3.3 系统算法服务架构

一个算法服务对应一个Proxy，它主要负责对收到的请求进行预处理如格式转换、判断合法，并将请求分发到算法集群上，要求proxy相对于ApiServer是独立的，添加新的Proxy无需修改或重新编译ApiServer的代码。

准确度方面：采用均方根误差(RMSE)测量预测结果的准确度，要求RMSE值较之前只使用用户已有结构化特征的方法降低至少一个数量级。

# 2. 系统设计与实现

## 2.1 系统整体架构

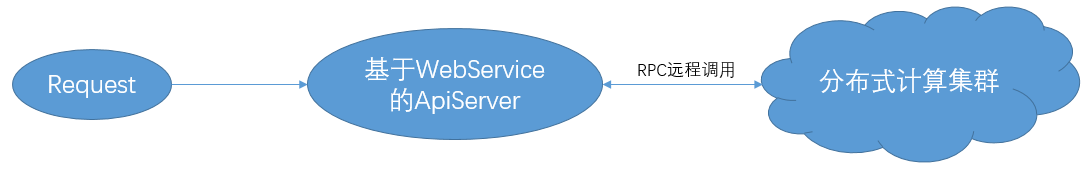
 为了满足性能上高并发、快响应的需求，系统整体采用一个请求处理/分发单元+多个计算单元组成的分布式计算集群的总-分架构。如图4.1所示：

图 4.1 在线请求处理流程

ApiServer是一个基于WebService的在线请求处理服务器，用于接收外部请求，并将收到的请求转换为RPC调用并按照负载均衡的原则分发到计算集群上的主机。计算集群完成请求的计算任务并将结果返回给ApiServer，ApiServer将收到的结果以WebService的形式（http+json）返回给请求者。

分布式计算集群上除了算法运行任务的计算服务器之外，还包括用于特征缓存的数据服务器和用于模型更新的服务器。特征缓存服务用作当收到广告点击预测请求时缓存请求数据中的用户特征和用户实际行为（点击或未点击），当缓存到一定数目时通过增量训练更新线上模型，从而能够实时反映用户兴趣行为的变化。

## 2.2 Service定义

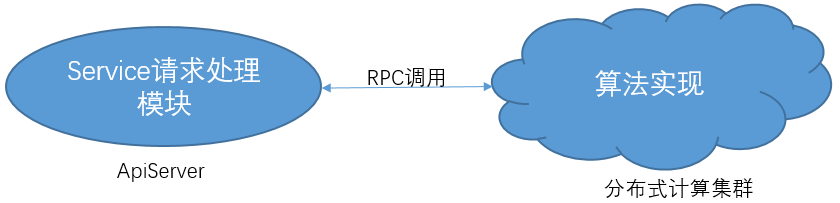
 在本系统中定义一个service为能够对外提供线上服务的算法运行实例，主要包括请求处理模块（ApiServer端运行）和算法实现模块（在分布式集群上运行），如图4.2所示：

图 4.2 Service工作原理示意图

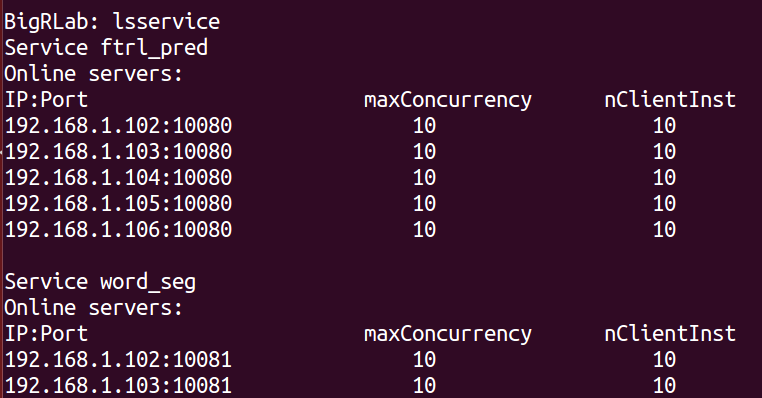
算法实现模块主要由运行算法的计算服务主机构成，根据业务需求可能也会有其他的支持服务如特征缓存数据库等。在系统控制台端可以通过lsservice命令查看当前线上运行的service列表，如图4.3所示：

图 4.3 系统控制台界面查看service列表

可以看到service的服务器列表，每个服务器属性主要有IP地址和端口号、支持的最大并发数（Thrift RPC Server参数）、以及当前的连接数（从ApiServer发起的连接）。

 由于不同的业务处理请求的方式不同，当向系统中添加一个新的service时，除了要实现分布式集群上运行的算法模块之外，还要实现ApiServer上运行的请求处理模块，各个service请求处理模块由service的二次开发人员所编写，因此和ApiServer是松耦合的，支持运行时动态添加而不必重新编译ApiServer。

图 4.4 ApiServer类结构图

在Service基类中提供两个纯虚函数，分别是handleRequest和handleCommand，前者用来处理http在线请求，对外提供WebService服务，后者提供基于命令行的用户接口。每一个具体的service派生类都要根据自己特定的业务需求和处理方法提供这两个函数的功能实现。

此外，ApiServer端运行的Service请求处理模块还负责计算集群上服务器的管理，包括新服务器加入，已有服务器下线，以及服务器异常处理；还负责将服务请求按照一定的规则分发给计算服务集群上的服务器处理。系统默认的是基于负载均衡的分发方案，随机挑选一个服务器处理当前请求，各具体的Service派生类可根据自身业务需求改写这一方案。对于异常处理：如集群中的某一台服务器崩溃，则RPC调用会抛出异常或请求超时，此时系统会自动重新选择另一台服务器处理请求，并将出现问题的服务器从候选列表中删除。

## 2.3 ApiServer的设计与实现

ApiServer在本系统中主要负责接收外部业务处理请求并且请求分发到计算集群上处理，其主要包括3个模块，Service管理，请求处理，和算法服务集群管理，本节将分别对这3个模块展开介绍。

### 2.3.1 Service管理模块

在本系统中ApiServer主要任务是接收外部请求，并交给请求所指定的service处理。为了能够支持运行时动态加载，所有在ApiServer上运行的service请求处理模块都被要求编译成动态链接库。在ApiServer端通过loadlib命令加载。为了省去每次添加新的service都要手动加载的麻烦，可以将service动态链接库路径记录在autoload.conf文件中，每次ApiServer启动时会先依据此文件记载的内容加载相应的service请求处理动态链接库。由加载的service动态链接库创建service实例。

Service动态链接库和service实例都由ServiceManager类管理，该类采用单例模式，只有一个全局唯一实例。ServiceLib类是对service动态链接库的类抽象，每一个该类的对象拥有一个service动态链接库的handle，此外提供两个函数用于创建该service实例和获取库的名称；每一个service动态链接库都有一个全局唯一的名称，此外还必须提供用于创建实例和获取名称的函数接口以供ServiceLib调用。它们之间的关系如图4.5所示：

图 4.5 Service相关类图

### 2.3.2 请求处理模块

ApiServer接受的外部请求分为两种，一种是WebService请求，通过http协议传输json封装的字符信息，用于处理实时在线请求，每次请求只处理一条数据并立刻返回结果；另一种是终端输入的命令行，用于线下批量处理存放于文件中的请求数据，结果一般输出到指定的文件中。

对于WebService请求，请求发送者将处理该请求的service名称封装在URL中，一般格式为：

<http://apiserver_addr:port/service_name>

当ApiServer的http服务器模块收到该URL请求后，会通过service\_name字段得到请求服务的service名称，然后通过ServiceManager查找当前系统中有没有该service，若没有返回错误信息，若找到则将请求正文内容作为参数传给该service的handleRequest函数处理。处理结果同样以json格式的文本返回给请求者。

对于命令行格式的请求，其一般格式为：

service $service\_name $args

ApiServer的命令行处理模块从终端读入一行用户输入，若第一个单词为service则表示该命令请求要交给具体的service处理，否则都是系统命令；紧接着读入指定处理该请求的service名称service\_name，然后通过ServiceManager查找这个service，若找到就将剩余的args参数列表传给该service的handleCommand处理，若找不到指定的service就在终端上输出错误信息。

图 4.6 系统内部模块交互图

### 2.3.3 算法服务管理模块

 在计算服务集群上，每一个算法实例启动时都要连接ApiServer（由启动参数指定ApiServer的地址和端口）以完成注册，这样集群上的该算法实例会成为对应service的计算集群的一员，成为一个作业分配的候选者。这个过程在ApiServer端是由算法服务管理模块AlgManager实现的。AlgManager本身也是一个Thrift RPC Server，计算集群上的算法实例通过Thrift RPC远程过程调用和ApiServer进行通信。AlgManager提供如下接口：

图 4.7 算法服务管理类图

当计算集群上算法实例启动时会将自己的信息填入到AlgSvrInfo结构中，然后以此为参数远程调用ApiServer上AlgManager模块的addServer完成注册；在该算法实例退出的时候相应地会调用rmServer从ApiServer上删除自己的注册信息。AlgSvrInfo记录了一个算法实例的关键信息，定义如下：

图 4.8 算法服务信息类图

其中addr和port分别定义了该算法实例所在集群主机的ip地址和所用的服务端口号；maxConcurrency指定了该算法实例所支的最大并发数，即同时能处理的最大请求数；serviceName指定了该算法实例属于哪一个service；libName指定了ApiServer所加载的service请求处理动态链接库的名称标识。

 计算服务集群上一个新的算法实例加入系统过程如图4.9所示：

图 4.9 新算法实例加入流程图

 当计算集群上一个算法实例退出时会远程调用AlgManager的rmServer，进而从对应service服务器列表中删除自己和已经建立的连接，如果删除后该service服务器列表为空，那么ServiceManager会删除该service的信息，后续到来的指定该service处理的请求都会返回错误。过程如图4.10所示：

图 4.10 已有算法实例突出流程图

### 2.3.4 负载均衡和异常处理

为了最大限度提高并发处理能力，对于多台主机构成的计算集群，负载均衡是必须要考虑的。ApiServer在service基类提供了一套默认的负载均衡方案，该方案为每个service创建一个空闲RPC连接池，当新的算法服务器加入的时候，根据其声明的maxConcurrency创建相应数目的空闲连接，然后放入空闲连接池中并执行random permutation的随机乱序操作，确保连续到来的请求尽可能地平均分配到不同的主机上。当一个请求到来的时候，该默认调度方案会从空闲连接池中取出第一个空闲连接，用它完成指定的RPC调用请求后再放回到空闲连接池中以供后续请求使用。每一个继承自service的具体派生类（动态链接库实现的）都可以定义自己的调度方案以覆盖默认方案，只需重写addServer函数并提供相关的数据结构支持即可。

异常处理方面，ApiServer端将异常分为用户异常和系统异常。用户异常是由算法定义的逻辑错误如非法输入，此类异常定义为InvalidRequest异常类，是Thrift Exception的子类。当ApiServer执行远程调用请求时收到这类异常会直接返回给调用者错误信息（终端输出或json文本，取决于请求类型是命令行还是WebService），并按正常流程将用过的RPC连接放回到空闲连接池中；系统异常主要是不可恢复的错误，如远程服务器意外停机，此时若在已经意外断开的连接上执行RPC调用会抛出Thrift Exception异常，对于这类异常系统不会将当前使用的RPC连接放回连接池。为了及时更新各线上算法计算服务器的状态，要求集群上的算法服务器每隔15秒钟向ApiServer发送一次信号，由AlgManager的informAlive远程函数实现，若连续1分钟未收到该主机的信号，系统会从相应service的服务器列表中将其删除。

## 2.4 算法服务的设计与实现

算法服务是对现有算法的改造封装以便和本系统兼容，算法服务实例运行于分布式计算集群上，通过Apache Thrift RPC远程调用和ApiServer端的service lib通信。算法服务本身是一个Thrift Server，按照自身业务逻辑提供远程调用接口供ApiServer端的service lib使用，计算集群上的算法服务实例和ApiServer端该算法对应的service lib实现共同构成了一个完整的service服务。

当算法实例启动时，要填写一个AlgSvrInfo对象，包括自己所在主机的ip地址，和本实例的服务端口号；ApiServer端service lib的名字；service名称；以及所支持的最大并发数，使用这个AlgSvrInfo对象去ApiServer注册，ApiServer会根据service名称找对应的service，若找到就把这个新到的实例加入到其服务器列表中，若找不到就用service lib名称找对应的动态链接库句柄，并用这个链接库创建一个新的service。

为了防止ApiServer意外崩溃，算法实例每隔一段时间（15秒）会启动一次注册流程，ApiServer收到注册请求若发现该实例已经注册，会直接返回实例已注册的错误，算法实例这边会忽略这个错误。这样当ApiServer崩溃重启后，之前所有在线的算法实例都会自动重新上线而不需要人工手动重启。

此外，为了实现模型自动更新，算法实例会自动保存启动时指定的模型数据文件路径，之后若要更新模型文件，只需把要更新的文件放在源文件的同一目录下，并在源文件名之后加上.update后缀，如源文件是model.dat，更新文件就是model.dat.update，算法实例会每隔一段时间检查是否有新的模型更新，若发现更新，就自动下线服务，然后将模型更新文件替换掉源文件，再初始化算法，到ApiServer注册，完成重新上线服务。