为 OneFlow 添加新的前端语言

中期报告

项目 ID: 210130141
项目导师: 蔡晟航
申请人: 周泽楷
导师邮箱: caishenghang@oneflow.org
申请人邮箱: 946079208@qq.com

2021 年 8 月 15 日

目 录

1 项目信息和项目进度简述

这一节概述项目信息和项目进度，具体细节请查看后面的小节。

1.1 项目信息

项目名称: 为 OneFlow 添加新的前端语言

方案描述: 在这个项目中，利用 JNI 技术为 OneFlow 实现 Java 前端语言，提供模型部署相关的功能。细节请查看第 2 节。

时间规划: 主办方给出了三个月的开发时间，从 7 月 1 日开始，9 月 30 日结束。之前将开发任务分解为 11 周进行，每周一个小目标，如表 1 所示。剩下两周，根据实际执行情况灵活调配。

1.2 项目进度

已完成工作: 目前已经实现基本的功能，可以完整的走完整个流程，进行模型加载，前向传播，获取输出。即完成了表 1 中 7.1 到 9.1 的工作。此外，为了方便构建动态链接库和 Jar 包，还写了一点 CMake，现在只需要调用 make 命令，就可以构建出动态链接库和 Jar 包，非常的方便 [1]。

遇到的问题及解决方案: 具体请参照第 3 节。

后续工作安排: 具体请参照第 4 节。

本文还提供了一个简单的使用例子，具体请参照第 5 节，给出了代码片段和代码片段运行的输出。

2 方案描述

2.1 方案概述及目标

2.1.1 方案概述

利用 JNI 技术为 OneFlow 实现 Java 前端语言，提供模型部署相关的功能。

2.1.2 目标

为 OneFlow 添加一门新的前端语言，支持模型部署的功能。项目输出一个动态链接库 liboneflow_java.so，用来给 Java 前端链接调用; 一个 jar

[1] 备注：在写这份项目申报书的时候，是 8 月 11 日。写完之后，发现对整体的代码设计有了新的思考，正在进行重构。概括起来就是：java 端不再使用 protobuf，将所有涉及到 protobuf 的操作下沉到 c api 这边。这么做的好处挺多的，减少序列化的开销，c 接口复用程度更高，java 项目不再依赖于 proto-java，java 项目不再需要编译 OneFlow 内部的.proto 文件。重构!

表 1: 计划表

开始时间	结束时间	目标
7.1	7.7	Maven 项目管理和接口设计
7.8	7.14	env, session 初始化
7.15	7.21	实现计算图的加载，加载图结构
7.22	7.28	编译计算图
7.29	8.4	实现 C++ JobInstance
8.5	8.11	加载图权重，启动 session
8.12	8.18	Java 实现一个 NDArray
8.19	8.25	实现模型输入
8.26	9.1	实现模型推理，获取输出，实现关闭 session
9.2	9.8	完善测试用例和重构
9.9	9.15	完善 Java 代码和文档，编写一个调用例子。
9.16	9.22	
9.23	9.30	

包 oneflow-api.jar，给 Java 提供接口；一个调用例子和文档，帮助用户上手使用。

2.2 详细实施方案

按照功能，将代码划分为四个部分。

- 第一部分，调用 OneFlow core，对 core 进行适当封装。提供接口，方便调用，提高代码复用性。
- 第二部分，负责做交互，一方面从 Java 获取数据，向 Java 发送数据，另一方面调用第一部分的接口，推送数据和拉取数据。
- 第三部分，Java 接口，负责做一些相对较高层的处理，调用第二部分暴露的 native 方法，将这些细粒度的方法组合成一个个相对完整的功能，进一步封装。
- 第四部分，CMake 代码，提供项目构建的方法，解决软件包依赖，项目间依赖等问题。最终目标是，用户可以几行命令将整个软件构建起来。

3 问题和解决方案

3.1 数组多次复制的问题

问题描述: Java 将一个数组对象传给 C++，使用 jni.h 中提供的方法从数组对象拿出数组，会发生**一次复制**。OneFlow 启动 Job 是异步的，一旦函数结束，数组就会被销毁掉，此时还需要**再复制一次**。之后还要将数据推送到 Blob 里面，又**复制了一次**。数组复制了多次，如何解决这个问题呢?

- 解决方法：使用 DirectBuffer 即可！在 Java 中，有堆内存和堆外内存之分，堆内存由 JVM 进行管理，会受到 GC 的影响，因此 JNI 调用的时候，为了确保数组不受 GC 的影响，需要进行数据复制。堆外内存，是在 JVM 同个进程中，但是不受 GC 影响，内容是稳定的，在 JNI 调用的时候，不会发生复制，并且传递过去的是一个指针。用了 DirectBuffer 之后，只需要一次数组复制就好了，将数据推送到 Blob 里面。

3.2 Jar 包构建

问题描述: 前期为了方便开发和调试，直接新建了一个 Maven 项目，后来发现，除了我之外别人都不太好构建。此外，前期还将 OneFlow 内部的.proto 文件编译成了.java 文件，放入到了 Maven 项目中，如果 OneFlow 内部的.proto 文件更新了，需要手动更新。这些都不太方便。

- 解决方法（*[2]）：使用 CMake 直接构建 Jar 包，使用 CMake 来编译 proto 文件，并且直接打包提供给 Java 前端用。分为三步走，第一步构建出 proto-java.jar。第二步编译.proto 为.java，并且打包成 oneflow-proto.jar，这一步依赖前一步的 jar 包。第三步，编译 Java 前端的代码，这一步依赖前两步的 Jar 包。

3.3 protobuf 的 bug

问题描述: 在项目申报的时候，发现了 protobuf 的一个 bug。将 proto 文件编译成 java 文件时，proto 中定义的一个 message[3]，它的一个属性命名为 input，生成 java 文件后会和 java 的局部变量产生冲突。

- 解决方法：前期没有找到问题在哪里，所以前期的解决办法时手动修改 message 定义的属性名字，这增大了在其他人那里构建的难度。后

[2]重构的项目里面不再需要编译 OneFlow 内部的.proto 文件了，重构之后只需要第三步即可。
[3]oneflow/core/framework/user_op_conf.proto

来用新版本试了一下，已经解决了。在本机上用 apt 安装的 3.0.0 版本存在这个问题，OneFlow 目前依赖的 3.9.0 没有这个问题。所以更新版本就好了。

4 后续工作安排

4.1 基本要求

基本要求就是实现在 Java 端加载模型，实现前向传播。项目要输出三个东西，动态链接库 liboneflow_java.so，Jar 包 oneflow-api.jar 前端相关的文档和使用例子。

目前基本完成了基本功能，可以加载模型，实现前向传播。**后续工作安排是**，完成重构，写测试，不断改进代码。配合 OneFlow 的员工，进行 Code Review，修改代码，希望能将代码合并进主仓库。

4.2 扩展

在完成基本要求的基础上，如果还有时间，可以进行如下尝试，进行扩展。即使在开源之夏结束之后，出于兴趣我还是很愿意尝试下面的工作。

- 推理加速，OneFlow 内部已经很好的支持了多个计算引擎的运行时加速库，尝试使用 TensorFlow XLA 和 NVIDIA TensorRT 为推理进行加速。
- 支持半精度浮点数 float16。在 Java 端如何存储和传输 float16 类型是一个有意思的问题。将这个问题解决之后，就可以使用混合精度进行模型推理，对推理进行加速。
- 尝试使用 oneflow-api.jar 去部署模型，在使用自己开发的工具包的过程中找到应该改进的地方。我认为技术应该是服务于产品的，上面两点都是考虑性能方面的技术，出发点并非产品。实际上的需求是怎样的呢？用户在使用过程中会遇到什么问题，踩到什么坑呢？只有自己多用用才知道。

5 使用示例

5.1 例子

接口设计本着简单明了的原则出发，甚至抛弃了 checked exception，改成了 runtime exception。

整个系统的输入由三者构成，必须用户提供：第一，job 函数名字，保存计算图的时候函数的名字。第二，模型路径。第三，向量，并且以 Map 的形式提供。Map 的 key 为保存计算图时候，输入的名字，可以使用 signature。

提供了上面三者之后，用户需要做如下操作：1. 创建 InferenceSession 对象; 2. 调用 open，进行初始化; 3. 调用 loadModel，加载模型; 3, 调用 launch，启动 session; 4. 调用 run，进行前向传播，获取结果; 5. 调用 close，关闭 session。

下面给出一个例子。这个模型是经典的 MLP 识别手写体，在这之前已经训练好模型，设置好 signature，并且保存下来。

```
String jobName = "mlp_inference";
String savedModelDir = "./models";
float[] image = readImage("./7.png");
Tensor imageTensor = Tensor.fromBlob(image, new long[]{ 1, 1, 28, 28 });
Map<String, Tensor> tensorMap = new HashMap<>();
tensorMap.put("image", imageTensor);

InferenceSession inferenceSession = new InferenceSession();
inferenceSession.open();
inferenceSession.loadModel(savedModelDir);
inferenceSession.launch();

Map<String, Tensor> resultMap = inferenceSession.run(jobName, tensorMap);
Tensor resultTensor = resultMap.get("output");
float[] resFloatArray = resultTensor.getDataAsFloatArray();
for (float v : resFloatArray) {
    System.out.print(v + " ");
}

inferenceSession.close();
```

5.2 输出

下面输出向量的值，取出最大的数值作为分类，可以看见第“7”个数字是预测的结果。可以看到完成了一次前向传播。

```
-130.93362 -72.18874 -165.4578 -114.83206 -21.068695 -88.18619 -151.45547
    19.964624 -142.91219 -43.09779
```

6 总结

经过这段时间的学习和开发，收获不多不少，下面记录几点思考和收获。

在技术上，学习了一点 Python 的语法，装饰器，事件循环机制；C++ 的智能指针，同步异步；Java 方面学习了 JNI，懂得如何去和本地方法交互；还学习了一点 CMake，以前看 CMake 会让我瑟瑟发抖，现在好多了。

在软件工程实践上，有一些体会。设计很重要，不过像我这样的入门级别的程序员应该先面向功能打代码，面向过程编程。在打代码的过程中，去找到变与不变，发现潜在的需求，对可能的变化做抽象和具体实现的分离。在入门阶段，不知道这些变与不变，是很难做出好设计的。要多思考需求，变与不变，但要先甩开设计。先面向功能编程，在重构的过程中去做设计。等以后经验丰富了，能理解需求，精准找到变化，才先设计再编码。

记得有一次看 OneFlow 的公众号，有篇推送中有一句话让我印象深刻：“在战斗中学习”。铿锵有力，简明精要。在开发的过程中，学习需要的知识，然后马上投入到战斗当中。一方面，加深了对知识的印象和理解，另一方面，在战斗的过程中对知识进行了检验。