penny-practice2-lac

> 环境

```
OS: Manjaro Linux x86_64
Kernel: 5.15.32-1-MANJARO
CPU: Intel i5-4570 (4) @ 3.600GHz
Memory: 11889MiB
```

一. 安装 LAC

> Java 接口是通过 jni 的形式调用 C++ 的接口, 需要拉取 LAC 源码编译出 liblacjni.so 动态库.
>
> LAC 依赖 Paddle, 因此需要先获取到 Paddle 库, 未找到合适已编译好的库文件, 采取自己编译(LAC Java JNI 编译和使用)

> 本次编译的版本为: Paddle(v1.8.5) 和 LAC(3e10dbe). 按照 LAC 说明文档按步骤执行, 有以下几个问题需要注意

- 编译 Paddle 时**打开 MKL** 选项(LAC 依赖 mkldnn, LAC 参考文档有误), 既去掉选项 `-DWITH_MKL=OFF`
- Paddle make 后, 执行 `make inference_lib_dist` 报错时, 可以尝试使用 `make inference_lib_dist/fast`
- 选择上述标注版本, 否则可能出现编译问题, 或者编译成功后 jni 调用链接错误
- gcc 版本需用 `gcc-8`, **推荐 8.2**, 8.5 也可(*编译过程比较耗时, 跟机器性能相关*). 多版本可以通过修改环境变量的方式切换版本, 如下:

```
export CC=/path/to/gcc-8
export CXX=/path/to/g++-8
```

- LAC 编译时, 上述 gcc 版本设置如果不生效, 可以直接修改 CMakeLists.txt, 增加如下内容

```
SET(CMAKE_C_COMPILER "/usr/bin/cc-8")
SET(CMAKE_CXX_COMPILER "/usr/bin/c++-8")
```

二. 增加新词

- Java 调用方式, 只有 `loadCustomization()`从文件中加载的方式
- 词典文件示例如下, 具体参考 LAC README

```
春天/SEASON
花/n 开/v
秋天的风
落 阳
```

- 示例代码

```
// 可选，装载干预词典
lac.loadCustomization("custom.txt")
```

三. 包装成 web 服务

> 使用 Java 开发, 采用 SpringBoot 框架, 默认 Tomcat Web 容器
> Repo: https://github.com/tinyweet/just-some-practices

- 接口 http://localhost:9090/lac?tc=2
- 参数 **tc**: 每个请求分词线程数

四. 性能测试

- 机器参数见顶部
- JVM

> JDK: Temurin-8.0.322
> 堆: `-server -Xms2048m -Xmx2048m`

0x1 测试准备

语料

> 实验语料: pullword.2022-05-06.log

1. 清理语料, 清理后 **40MB**

```
cut -f2 pullword.2022-05-06.log |grep -av '^$' > yuliao.log
```

2. 文本切割, 按相同文件大小, 行不割断的方式, 比如: 等分 2 份

```
split -n l/2 yuliao.log
```

脚本

```
ls 2/* | xargs ./send.sh
```

> send.sh

```
#!/usr/bin/sh

for f in "$@"
do
    curl -X POST -T "$f" "127.0.0.1:9090/lac?tc=2" &
done
```

0x2 实验结果

> 数据总量为 40MB, 红色标记为最优

并发进程数	分词线程数	分词时间(sec)	分词速度 MB/min
2	1	292	8.22
2	2	**148**	16.22
2	3	151	15.89
2	4	166	14.46
2	5	168	14.29
2	10	174	13.79
4	1	**149**	16.11
5	2	159	15.09
5	3	172	13.95
10	2	191	12.57
20	2	259	9.27
50	2	532	4.51

0x3 结果分析

> Tomcat 并发模型为多线程模型, 因此在处理数据规模一定的情况下, 理论上, 增加并发数和增加分词线程数具有相同效果, 试验数据也可证明. 实验机器为 *4* 核, 分词线程数(分词总线程数)为 4 时执行效率最高. 线程数过多, 线程切换频繁, 执行效率下降严重; 线程数不足 4, CPU 不能充分利用, 效率也不能达到最高.