6.824 -2020年春季

6.824实验1：MapReduce

截止日期：2月14日23:59

介绍

在本实验中，您将构建一个MapReduce系统。您将实现一个worker进程，该worker进程将调用Map和Reduce函数并处理读写文件，以及一个master进程，该master进程将任务分发给某一个worker进程，并处理失败的worker进程。您将构建类似于 MapReduce论文的内容。

合作政策

您必须编写并上交该课程(6.824)的所有代码，除了我们作为例程提供给您的代码，不允许您查看其他任何人的解决方案，也不允许您查看前几年的解决方案。您可以与其他学生讨论作业，但不能查看或复制彼此的代码。制定此规则的原因是，我们相信您将自己设计和实施实验室解决方案，从而从中学到最多的知识。

请不要发布您的代码或将其提供给现在或将来的6.824学生。 `github.com`仓库默认是公开的，因此除非您将仓库设为私有，否则请不要在其中放置代码。您可能会发现使用MIT的GitHub方便 ，但是请确保创建一个私有存储库。

软件

您将使用GO实现该实验（以及所有实验） 。Go网站包含许多教程信息。我们将使用Go 1.13版为您的实验室评分；您也应该使用1.13。您可以通过运行`go version`来检查Go 版本。

我们建议您在自己的计算机上进行实验，以便可以使用已经熟悉的工具，文本编辑器等。或者，您可以在Athena上的实验室工作。

苹果系统

您可以使用Homebrew安装Go。安装Homebrew后，运行`brew install go`。

Linux系统

根据您的Linux发行版，您可能能够从软件包存储库中获取最新版本的Go，例如，通过运行`apt install golang`。否则，您可以从Go的网站手动安装二进制文件。首先，确保您正在运行64位内核（`uname -a`应提及“ x86_64 GNU / Linux”），然后运行：

```
$ wget -qO- https://dl.google.com/go/go1.13.6.linux-amd64.tar.gz | sudo tar xz -C / u
```

您需要确保`/usr/local/bin`在您的`PATH`上。

Windows系统

这些实验室可能无法直接在Windows上运行。如果您喜欢冒险，可以尝试使它们在Windows Subsystem for Linux中运行，并遵循上面的Linux说明。否则，您可以使用Athena。

Athena

您可以通过`ssh {your kerberos}@athena.dialup.mit.edu`登录到公共Athena主机。登录后，要获取Go 1.13，请运行：

```
$ setup ggo
```

入门

您将使用git（版本控制系统）获取初始实验室软件 。要了解有关git的更多信息，请参阅 Pro Git Book或 git用户手册。获取6.824实验代码：

```
$ git clone git://g.csail.mit.edu/6.824-golabs-2020 6.824
$ cd 6.824
$ ls
Makefile src
$
```

我们在`src/main/mrsequential.go`中为您提供了一个简单的顺序mapreduce实现 。它可以在一个进程中执行map后执行reduce。我们还为您提供独立的MapReduce应用：word-count`mrapps/wc.go`和文本索引中`mrapps/indexer.go`。您可以按如下顺序进行字数统计：

```
$ cd ~/ 6.824
$ cd src / main
$ go build -buildmode = plugin ../mrapps/wc.go
$ rm mr-out*
$ go run mrsequential.go wc.so pg*.txt
$ more mr-out-0
A 509
ABOUT 2
ACT 8
...
```

`mrsequential.go`将其输出保留在文件`mr-out-0`中。输入来自名为`pg-xxx.txt`的文本文件。

随时从`mrsequential.go`借用代码。您还应该查看`mrapps / wc.go,` 以了解MapReduce应用程序代码的实现细节。

你的工作

您的工作是实现一个分布式MapReduce，它由两个程序（master程序和worker程序）组成。只有一个master进程，一个或多个worker进程并行执行。在真实的系统中，工作人员将在一堆不同的机器上运行，但是对于本实验，您将全部在单个机器上运行它们。worker将通过RPC与master服务器对话。每个工作进程都会向主服务器请求一个任务，从一个或多个文件中读取任务的输入，执行任务，并将任务的输出写入一个或多个文件。master应注意一个工人是否在合理的时间内没有完成任务（在本实验中，使用十秒钟），并将同一任务交给另一个worker。

我们给了您一些代码，帮助您开始。主机和工作程序的“main”程序位于`main/mrmaster.go`和`main/mrworker.go`中；不要更改这些文件。您应该将实现放在`mr/master.go`， `mr/worker.go`和`mr/rpc.go`中。

这是在单词计数MapReduce应用程序上运行代码的方法。首先，请确保单词计数插件是全新构建的：

```
$ go build -buildmode = plugin ../mrapps/wc.go
```

在main目录中，运行main目录。

```
$ rm mr-out *
$go run mrmaster.go pg-*.txt
```

mrmaster.go 的pg-*.txt参数是输入文件；每个文件对应一个“拆分”，是一个Map任务的输入。

在一个或多个其他窗口中，运行一些工作程序：

```
$go run mrworker.go wc.so
```

当woeker和master完成后，请查看mr-out- *中的输出。完成实验后，输出文件的排序联合应与顺序输出匹配，如下所示：

```
$ cat mr-out- * | sort | more
A 509
ABOUT 2
ACT 8
...
```

我们在main/test-mr.sh中为您提供了一个测试脚本。测试在给定pg-xxx.txt文件作为输入时，检查wc和indexer MapReduce应用程序是否产生正确的输出。这些测试还检查您的实现是否并行运行Map和Reduce任务，以及您的实现是否从运行任务时崩溃的工作程序中恢复。

如果您现在运行测试脚本，则它将挂起，因为主脚本永远不会完成：

```
$ cd~/ 6.824 / src / main
$ sh test-mr.sh
*** Starting wc test.
```

您可以在mr/master.go的“完成”功能中将ret := false更改为true， 以便主机立即退出。然后：

```
$ sh ./test-mr.sh
*** Starting wc test.
sort: No such file or directory
cmp: EOF on mr-wc-all
--- wc output is not the same as mr-correct-wc.txt
--- wc test: FAIL
$
```

测试脚本希望在名为mr-out-X的文件中看到输出，每个缩减任务一个。mr/master.go 和mr/worker.go的空实现不会生成这些文件（或执行其他任何操作），因此测试失败。

完成后，测试脚本输出应如下所示：

```
$ sh ./test-mr.sh
*** Starting wc test.
```

```
--- wc test: PASS
*** Starting indexer test.
--- indexer test: PASS
*** Starting map parallelism test.
--- map parallelism test: PASS
*** Starting reduce parallelism test.
--- reduce parallelism test: PASS
*** Starting crash test.
--- crash test: PASS
*** PASSED ALL TESTS
$
```

您还将从Go RPC软件包中看到一些类似于

```
2019/12/16 13:27:09 rpc.Register: method "Done" has 1 input parameters; needs exactly
```

忽略这个提示。

一些规则：

- 映射阶段应将中间键划分为用于`nReduce` reduce任务的存储桶 ，其中`nReduce`是`main/mrmaster.go`传递给`MakeMaster()`的参数。
- 工作程序实现应将第X个reduce任务的输出放入文件`mr-out-X`中。
- 一个`mr-out-X`文件的每个Reduce函数输出应包含一行。该行应以Go `"%v %v"` 格式生成，并使用键和值进行调用。在`main/mrsequential.go`中 查看注释为“这是正确的格式”的行。如果您的实现不按该格式输出，则测试脚本将失败。
- 您可以修改`mr/worker.go`， `mr/master.go`，和`mr/rpc.go`。您可以临时修改其他文件以进行测试，但是请确保您的代码可以与原始版本一起使用；我们将使用原始版本进行测试。
- worker应将中间Map输出放置在当前目录中的文件中，您的worker以后可以在其中读取它们，作为Reduce任务的输入。
- `main/mrmaster.go`期望`mr/master.go`实现 `Done()`方法，该方法在MapReduce作业完全完成时返回true；届时，`mrmaster.go`将退出。
- 全部完成后，工作进程应退出。一种简单的实现方法是使用`call()`的返回值：如果worker程序无法与master服务器联系，则可以假定worker服务器由于作业完成而退出，因此worker程序也可以终止。根据您的设计，您可能还会发现拥有主人可以交给工作人员的“请退出”伪任务会很有帮助。

提示

- 一种入门方法是修改`mr/worker.go`的 `Worker()`以将RPC发送给主服务器，以请求任务。然后修改母版以使用尚未启动的映射任务的文件名进行响应。然后，修改工作程序以读取该文件并调用应用程序Map函数，如`mrsequential.go`中所示。
- 使用Go插件包在运行时从名称以`.so`结尾的文件中加载应用程序Map和Reduce函数。
- 如果您在`mr /`目录中进行了任何更改，则可能必须重新构建您使用的所有MapReduce插件，例如`go build -buildmode = plugin ../mrapps/wc.go`
- 该实验室依靠工作人员共享文件系统。当所有工作程序都在同一台计算机上运行时，这很简单，但是如果工作程序在不同的计算机上运行，则需要像GFS这样的全局文件系统。
- 中间文件的合理命名约定是`mr-X-Y`，其中X是Map任务号，Y是reduce任务号。
- woker的Map任务代码将需要一种方法以在reduce任务期间可以正确读取的方式在文件中存储中间键/值对。一种可能性是使用Go的`encoding/json`包。要将键/值对写入JSON文件：

```
enc := json.NewEncoder(file)
for _, kv := ... {
  err := enc.Encode(&kv)
```

并读回这样的文件：

```
dec := json.NewDecoder(file)
for {
  var kv KeyValue
  if err := dec.Decode(&kv); err != nil {
    break;
  }
  kva = append(kva, kv)
}
```

- 您的worker的map部分可以使用`ihash(key)`函数（在`worker.go`中）为给定的key选择reduce任务。
- 您可以从`mrsequential.go`借鉴一些代码，以读取Map输入文件，对Map和Reduce之间的中间键/值对进行排序，以及将Reduce输出存储在文件中。
- 主服务器（作为RPC服务器）将是并发的；不要忘记锁定共享数据。
- 使用Go的竞赛检测器，以及`go build -race`和`go run -race`。`test-mr.sh`上有一条注释，向您展示了如何为测试启用种族检测器。
- worker有时需要等待，例如，reduce操作直到最后一个Map完成后才能开始。一种实现是worker定时向master寻求任务，并使用`time.Sleep()`等待。每次请求之间都需要睡眠。另一个实现是，master服务器中的相关RPC处理程序具有一个循环，等待时间为`time.Sleep()`或`sync.Cond`。Go在其自己的线程中为每个RPC运行处理程序，因此一个处理程序正在等待的事实不会阻止主服务器处理其他RPC。
- master无法可靠地区分崩溃的worker，活着的但由于某种原因停工的worker和执行但速度太慢而无法使用的worker。您能做的最好的事情就是让主服务器等待一段时间，然后放弃并将任务重新发布给其他工作人员。在本实验中，让master等待十秒钟；之后，master应假定worker已经死亡（当然，可能没有死亡）。
- 要测试崩溃恢复，可以使用`mrapps/crash.go`应用程序插件。它在Map和Reduce函数中随机退出。
- 为了确保在崩溃时不会有人观察到部分写入的文件，MapReduce论文提到了使用临时文件并在完全写入后自动对其重命名的技巧。您可以使用`ioutil.TempFile`创建一个临时文件，并使用`os.Rename`原子地对其进行重命名。
- `test-mr.sh`运行子目录`mr-tmp`中的所有进程，因此如果出现问题，并且您想查看中间文件或输出文件，请在此处查看。

交接程序

> **Important:**
>
> 提交之前，请最后一次运行`test-mr.sh`。

使用`make lab1`命令打包您的实验作业，并将其上传到班级的提交网站，网址为 https://6824.scripts.mit.edu/2020/handin.py/。

您可以使用MIT证书或通过电子邮件请求API密钥来首次登录。登录后将显示您的API密钥（`XXX`），该密钥可用于从控制台上载lab1，如下所示。

```
$ cd~/ 6.824
$ echo XXX> api.key
$ make lab1
```

Important:

检查提交的网站，以确保它认为您提交了此实验室！

Note：您可以提交多次。我们将使用您上次提交的时间戳来计算迟到时间。

请在Piazza提问。 文档翻译：Reyunn