4.7 分布式数据处理

::: details INFO
译者：Asandstar

来源：4.7 Distributed Data Processing

对应：无
:::

分布式系统通常用于收集、访问和处理大型数据集。例如，本章前面介绍的数据库系统可以对存储在多台机器上的数据集进行操作。任何一台机器都可能不包含响应查询所需的数据，因此需要进行通信来处理请求。

本节将探讨一种典型的大数据处理场景，即数据集过大，无法由单台机器处理，而是分布在多台机器上，每台机器处理数据集的一部分。处理结果通常必须跨机器汇总，以便将一台机器的计算结果与其他机器的计算结果结合起来。为了协调这种分布式数据处理，我们将讨论一种名为MapReduce的编程框架。

使用 MapReduce 创建分布式数据处理应用程序结合了本文中介绍的许多思想。应用程序用纯函数来表示，这些函数用于映射（map）大型数据集，然后将映射的值序列还原（reduce）成最终结果。

函数式编程中的熟悉概念在 MapReduce 程序中得到了最大程度的应用。MapReduce 要求用于映射和还原数据的函数必须是纯函数。一般来说，仅用纯函数表示的程序在执行方式上具有相当大的灵活性。子表达式可以按照任意顺序并行计算，而不会影响最终结果。MapReduce 应用程序会并行评估许多纯函数，重新安排计算顺序，以便在分布式系统中高效执行。

MapReduce 的主要优势在于，它能在分布式数据处理应用程序的两个部分之间实现关注点分离：

1. 处理数据和合并结果的映射和还原函数。
2. 机器之间的通信和协调。

协调机制可以处理分布式计算中出现的许多问题，如机器故障、网络故障和进度监控。虽然管理这些问题会给 MapReduce 应用程序带来一些复杂性，但这些复杂性都不会暴露给应用程序开发人员。相反，构建 MapReduce 应用程序只需指定上述（1）中的映射和还原函数即可；分布式计算的挑战通过抽象被隐藏起来。

4.7.1 MapReduce

MapReduce 框架假定输入是任意类型的大量无序输入值流。例如，每个输入可能是某个庞大语料库中的一行文本。计算分三步进行。

1. 对每个输入应用映射函数，输出零个或多个任意类型的中间键值对。
2. 所有中间键值对都按键分组，因此键值相同的键值对可以被还原在一起。
3. 还原函数合并给定键 `k` 的值；它输出零个或多个值，每个值在最终输出中都与 `k` 相关联。

为了执行这一计算，MapReduce 框架创建了在计算中扮演不同角色的任务（可能在不同的机器上）。映射任务（map task）将映射函数应用于输入数据的某些子集，并输出中间键值对。还原（reduce）任务按键对键值进行排序和分组，然后对每个键的值应用还原函数。映射任务和还原任务之间的所有通信都由框架处理，按键对中间键值对进行分组的任务也是如此。

为了在 MapReduce 应用程序中利用多台机器，多个映射器在映射阶段（map phase）并行运行，多个还原器在还原阶段（reduce phase）并行运行。在这两个阶段之间，排序阶段（sort phase）通过排序将键值对组合在一起，从而使所有具有相同键值的键值对都相邻。

考虑一下计算文本语料库中元音的问题。我们可以使用 MapReduce 框架并适当选择 map 和 reduce 函数来解决这个问题。map 函数将一行文本作为输入，并输出键值对，其中键是 vowel，值是 count。输出中省略了零计数：

```
def count_vowels(line):
    """A map function that counts the vowels in a line."""
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)
```

reduce 函数是 Python 内置的求和函数，它的输入是值的迭代器（给定键的所有值），并返回它们的和。

4.7.2 本地实现

要指定 MapReduce 应用程序，我们需要一个 MapReduce 框架的实现，我们可以在其中插入 map 和 reduce 函数。在下一节中，我们将使用开源的Hadoop实现。在本节中，我们将使用 Unix 操作系统的内置工具开发一个最小的实现。

Unix 操作系统在用户程序和计算机底层硬件之间建立了一个抽象屏障。它为程序之间的通信提供了一种机制，特别是允许一个程序使用另一个程序的输出作为输入。在他们关于 Unix 编程的开创性文章中，Kernigham 和 Pike 断言：""系统的力量更多来自于程序之间的关系，而不是程序本身"。

在 Python 源文件的第一行添加注释，说明程序应使用 Python 3 解释器执行，就可以将 Python 源文件转换为 Unix 程序。Unix 程序的输入是一个可迭代对象，称为标准输入，访问方式为 `sys.stdin`。对这个对象进行迭代会产生字符串值的文本行。Unix 程序的输出称为标准输出（standard output），访问方式为 `sys.stdout`。内置的 `print` 函数将一行文本写入标准输出。下面的 Unix 程序将其输入的每一行反向输出：

```
#!/usr/bin/env python3

import sys

for line in sys.stdin:
    print(line.strip('\n')[::-1])
```

如果我们将该程序保存到名为 `rev.py` 的文件中，就可以将其作为 Unix 程序执行。首先，我们需要告诉操作系统，我们创建了一个可执行程序：

```
$ chmod u+x rev.py
```

接下来，我们可以向这个程序传递输入信息。一个程序的输入可以来自另一个程序。使用 | 符号（称为"管道"）就可以达到这种效果，它将管道前程序的输出导入管道后程序。程序 `nslookup` 输出 IP 地址的主机名（本例中为《纽约时报》）：

```
$ nslookup 170.149.172.130 | ./rev.py
moc.semityn.www
```

cat 程序会输出文件内容。因此，`rev.py` 程序可以用来反转 `rev.py` 文件的内容：

```
$ cat rev.py | ./rev.py
3nohtyp vne/nib/rsu/!#

sys tropmi

:nidts.sys ni enil rof
)]1-::[]'n\'(pirts.enil(tnirp
```

这些工具足以让我们实现一个基本的 MapReduce 框架。这个版本只有单个 map 任务和单个 reduce 任务，它们都是用 Python 实现的 Unix 程序。我们使用以下命令运行整个 MapReduce 应用程序：

```
$ cat input | ./mapper.py | sort | ./reducer.py
```

`mapper.py` 和 `reducer.py` 程序必须实现 map 函数和 reduce 函数，以及一些简单的输入和输出行为。例如，为了实现上述元音计数应用程序，我们将编写以下程序 `count_vowels_mapper.py` 程序：

```
#!/usr/bin/env python3

import sys
from mr import emit

def count_vowels(line):
    """A map function that counts the vowels in a line."""
    for vowel in 'aeiou':
        count = line.count(vowel)
        if count > 0:
            emit(vowel, count)

for line in sys.stdin:
    count_vowels(line)
```

此外，我们还要编写以下 `sum_reducer.py` 程序：

```
#!/usr/bin/env python3

import sys
from mr import values_by_key, emit

for key, value_iterator in values_by_key(sys.stdin):
    emit(key, sum(value_iterator))
```

mr module是本文的配套模块，它提供了 `emit` 函数和 `group_values_by_key` 函数，前者用于发射键值对，后者用于将具有相同键值的值分组。该模块还包含 MapReduce 的 Hadoop 分布式实现的接口。

最后，假设我们有以下名为 `haiku.txt` 的输入文件：

```
Google MapReduce
Is a Big Data framework
For batch processing
```

通过使用 Unix 管道进行本地执行，我们可以计算出 haiku 中每个元音的数量：

```
$ cat haiku.txt | ./count_vowels_mapper.py | sort | ./sum_reducer.py
'a'   6
'e'   5
'i'   2
'o'   5
'u'   1
```

4.7.3 分布式实现

Hadoop 是 MapReduce 框架开源实现的名称，可在机器集群上执行 MapReduce 应用程序，为高效并行处理分配输入数据和计算。它的流接口允许任意的 Unix 程序定义 map 和 reduce 函数。事实上，我们的 `count_vowels_mapper.py` 和 `sum_reducer.py` 可直接用于 Hadoop 安装，以计算大型文本语料库中的元音数量。

与我们简单的本地 MapReduce 实现相比，Hadoop 有几个优势。首先是速度：在不同的机器上同时运行不同的任务，并行应用 map 和 reduce 函数。第二是容错：当一个任务因故失败时，其结果可由另一个任务重新计算，以完成整体计算。第三是监控：该框架提供了一个用户界面，用于跟踪 MapReduce 应用程序的进度。

要使用提供的 `mapreduce.py` 模块运行元音计数应用程序，请安装 Hadoop，将 `HADOOP` 的赋值语句更改为本地安装的根目录，将文本文件集复制到 Hadoop 分布式文件系统中，然后运行：

```
$ python3 mr.py run count_vowels_mapper.py sum_reducer.py [input] [output]
```

其中，`[input]` 和 `[output]` 是 Hadoop 文件系统中的目录。

有关 Hadoop 流接口和系统使用的更多信息，请查阅 Hadoop Streaming Documentation。