开源地址：https://github.com/Vincent-Huang-2000/DATA301
感觉不错的话，可以在 Github 上给个 Star

本文档未完结，最新修订日期：2023年6月19日

提供的资料

May 30 Lecture 28:50 讲到考试中可以访问的资料
- 所有的 lecture notes
- PySpark 的文档

题型

- Knowledge [approx. 40 pts]
  - Multiple Choice, choose best answer
- Analysis [approx. 20 pts]
  - Short answer, write 1-4 sentences
- Skill [approx. 40 pts]
  - Apply an algorithm (i.e. compute answer, possibly by calling functions)
  - Implement an algorithm (i.e. write code)

考试主题

Programming
- Map/Reduce functional programming, Message-Passing, Threads, Locks and Atomics, Work Queues, Schedulers, Streaming, MPI, CUDA, SPARK

Algorithms
- Divide and Conquer, Map, Reduce, Group By, Union, Intersection, Difference, Matrix-Vector and Matrix-Matrix Multiplication, Hashing, PageRank, Graphs, Leader Election, Consensus

Systems
- SaaS, PaaS, IaaS, storage and networking architectures, virtual machines and their management, job scheduling, cloud resources, heterogenous processors

Data and Scale
- 5 Vs (Variety, Velocity, Volume, Veracity, Value), Decomposition, Distributed Data Structures, Memory Hierarchy, Shared memory, Shared-nothing, distributed file systems, replication, communication cost, complexity theory

Programming

Map/Reduce functional programming (SPARK)

- lambda
- RDD, parallelize, textFile, map, flatMap, filter, reduce, reduceByKey, groupByKey, sortByKey, join, cogroup, cartesian, collect/take, count, countByKey

Message-Passing, Threads, Locks and Atomics, Work Queues, Schedulers

MPI

- rank, send/recv, broadcast, reduce

CUDA

- Numba (jit / cuda.jit)
- Kernel, block, thread, warp

Algorithms

Divide and Conquer: data and functional parallelism, pipelining

Map, Reduce, Group By

- Frequent Item Sets, Market Basket Analysis

Union, Intersection, Difference

- distance/similarity measures (Jaccard, Cosine, Euclidean/$L_2 Norm$)
- Content recommendation

Matrix-Vector and Matrix-Matrix Multiplication

Hashing: Shingling, Min-Hashing

Graphs

- Flow/PageRank: random walk, recursive/iterative, traps, teleports
- Clustering: hierarchical (Girvan-Newman/betweenness, modularity)

Online: AdWords/balance

- Simulation: simple N-Body (game of life)
- Leader Election, Consensus (RAFT)

Systems

- SaaS, PaaS, IaaS
- storage and networking architectures
- virtual machines and their management
- job scheduling, cloud resources [specifically the Google Cloud Platform you've used], heterogeneous processors
- Hardware/low-level parallelism
    - Functional units
    - Instruction level parallelism
    - Out of order execution
    - Dependencies
- Memory Hierarchy, Shared memory, Distributed memory
- SIMD and GPU vector processing: tradeoffs with CPU – less cache/memory, less control logic, more ALU

Data and Scale

- 5 Vs (Variety, Velocity, Volume, Veracity, Value)
- Decomposition: block, cyclic
- Replication, distributed file systems
- communication cost, complexity theory
    - Amdahl's Law, Gustafson's Law
    - Weak scalability, Strong scalability
    - Elapsed communication cost
    - "roofline performance" and arithmetic intensity

期末样题

multiple choice question

In Market Basket Analysis we might be given the following data:

B1 = {m, c, b} B2 = {m, p, j}
B3 = {m, b} B4 = {b, c, j}
B5 = {m, p, b} B6 = {m, c, x, y}
B7 = {m, c, b, j} B8 = {b, c}
Which relationship has the least confidence?

Select one:

A. p -> m
B. x -> y
C. c -> b
D. m -> x

在进行市场篮子分析时，我们需要计算关联规则的置信度。置信度定义为：

Confidence(A -> B) = support(A U B) / support(A)

在这里，A和B是项集，U代表集合的并集。根据这个定义，我们需要计算每个选项的置信度。这需要我们先找到每个关系中元素A和B的出现频率，以及它们共同出现的频率。

A. p -> m
`Confidence(p -> m) = support({p, m}) / support(p)`
我们可以看到，{p, m}在B2和B5中出现，所以support({p, m}) = 2。p在B2、B5中出现，所以support(p) = 2。因此，Confidence(p -> m) = 2 / 2 = 1。

B. x -> y
`Confidence(x -> y) = support({x, y}) / support(x)`
{x, y}只在B6中出现，所以support({x, y}) = 1。x也只在B6中出现，所以support(x) = 1。因此，Confidence(x -> y) = 1 / 1 = 1。

C. c -> b
`Confidence(c -> b) = support({c, b}) / support(c)`
{c, b}在B1、B4、B7、B8中出现，所以support({c, b}) = 4。c在B1、B4、B6、B7、B8中出现，所以support(c) = 5。因此，Confidence(c -> b) = 4 / 5 = 0.8。

D. m -> x
`Confidence(m -> x) = support({m, x}) / support(m)`
{m, x}只在B6中出现，所以support({m, x}) = 1。m在B1、B2、B3、B5、B6、B7中出现，所以support(m) = 6。因此，Confidence(m -> x) = 1 / 6 ≈ 0.17。

所以，置信度最低的关系是选项D. m -> x。

short answer question

Web server log files contain records of which page on a server was requested by a user. For example, this is a sample line from a web server log:

piweba3y.prodigy.com - 807301196 GET
/shuttle/missions/missions.html 200 8677

Explain how this is example of Velocity (one of the 5 V's of Big Data)
Explain how this is example of Variety (one of the 5 V's of Big Data)

在大数据的五个V中，Velocity（速度）和Variety（多样性）是非常重要的概念。

1. Velocity（速度）: Velocity 在大数据中主要指的是数据的生成速度或处理速度。在你提供的例子中，web服务器日志文件是一个很好的展示速度的例子。每当用户请求服务器上的页面时，服务器都会立即生成一条记录。这可能会导致每天，甚至每小时，都会生成大量的新数据。这种高速率的数据生成需要高效的处理和分析方法，这就是大数据速度的一个典型示例。

2. Variety（多样性）: Variety 在大数据中是指各种各样的数据类型和来源。在你的例子中，web服务器日志文件也是一个很好的展示多样性的例子。日志文件可能包含各种类型的信息，如请求的IP地址、时间戳、请求的方法（GET或POST）、请求的资源URL、HTTP响应代码，以及响应的大小等。这些数据可能是结构化的（例如，时间戳或响应代码），也可能是半结构化的（例如，URL）。这些不同类型的数据需要不同的工具和技术来处理和分析，这就是大数据多样性的一个典型示例。

apply algorithm question

The Jaccard similarity of sets S and T is |S ∩ T | / |S ∪ T |

Choose the names of two of your friends. Compute the Jaccard similarity between their names where S and T are the sets of letters in each of their names.

假设我有两个朋友，他们的名字是 "Alice" 和 "Elsa"。我们将他们的名字中的字母（不区分大小写）作为集合S和T，然后计算他们的Jaccard相似性。

首先，我们需要确定集合S和T的元素。注意，每个字母只应计入一次，即使在名字中出现多次。

S = {a, l, i, c, e}
T = {e, l, s, a}

接着，我们需要确定S和T的交集（S ∩ T）和并集（S ∪ T）。

S ∩ T = {a, l, e}，所以 |S ∩ T | = 3
S ∪ T = {a, l, i, c, e, s}，所以 |S ∪ T | = 6

最后，我们将这两个数量插入到Jaccard相似性公式中：

Jaccard similarity = |S ∩ T | / |S ∪ T | = 3 / 6 = 0.5

因此，名字 "Alice" 和 "Elsa" 的Jaccard相似性是 0.5。

coding question

Create a python Spark program that does the following:

loads a text file (for example,
http://www.gutenberg.org/cache/epub/16328/pg16328.txt)

finds and prints the length of the longest word (space delimited) in the file that contains the first letter of your first name and does not contain non-letter characters (i.e. the isalpha() method on a string in python should return True)
Full credit for efficient solutions that use Spark functions

下面是一个使用PySpark的程序，该程序加载一个文本文件，并找出包含我名字首字母且不包含非字母字符的最长单词。

```python
from pyspark import SparkConf, SparkContext
from operator import add

def filter_words(word):
    first_letter = 'c' # Assuming the first letter of my name is 'c'
    return word.startswith(first_letter) and word.isalpha()

def longest_word(word1, word2):
    return word1 if len(word1) > len(word2) else word2

# Initialize Spark
conf = SparkConf().setMaster("local").setAppName("LongestWord")
sc = SparkContext(conf = conf)

# Load the text file
rdd = sc.textFile("http://www.gutenberg.org/cache/epub/16328/pg16328.txt")

# Split the text file into words
words = rdd.flatMap(lambda line: line.split(" "))

# Filter words that start with the first letter of my name and contains only alphabets
filtered_words = words.filter(filter_words)

# Find the longest word
longest_word = filtered_words.reduce(longest_word)

print("Longest word: ", longest_word)
print("Length: ", len(longest_word))

sc.stop() # Stop Spark
```

这个程序首先加载一个文本文件，并将其分割为一个单词的RDD。然后，我们使用filter方法来找出所有以特定字母开头且只包含字母的单词。最后，我们使用reduce方法找出最长的单词。

请注意，运行此程序需要在你的环境中安装Spark和PySpark库，而且需要有访问该URL的权限。

必考概念

1. What are the benefits of computing with parallelism?

Speed: Parallel computing can significantly reduce the time it takes to compute complex tasks. By dividing a task among multiple processors that can operate simultaneously, you can complete the task much more quickly than if you used a single processor. This is particularly valuable for tasks that involve large amounts of data or complex calculations.

Efficiency: Parallel computing can be more efficient than sequential computing because it utilizes more of the computer's resources at the same time. While a single processor might spend a lot of time idle, multiple processors can all be used to work on different parts of the task at the same time.

Problem Size: Parallel computing allows for the tackling of larger problems that would be impractical or impossible to handle with a single processor. This is especially important in fields like scientific computing, where simulations and data analysis can involve enormous amounts of data.

Cost-Effectiveness: With the increasing availability and affordability of multi-core and multi-processor systems, parallel computing can be a cost-effective way to achieve high performance.

Reliability: In a distributed system, the failure of one machine does not halt the entire process as the workload can be picked up by other machines. This redundancy can lead to higher reliability.

Concurrency: It can handle many tasks simultaneously. Real-time systems with a large number of inputs and outputs or systems with a large number of users, like a web server, can benefit from this.

Throughput: For tasks that are not time-critical, using parallel computing to run them simultaneously can result in higher overall throughput. This can be a significant advantage in environments where many tasks need to be performed but each individual task is not particularly time-sensitive.