第二次实验报告

姓名	学号	邮箱	院系
姜帅	221220115	js13156223725@163.com	计算机学院

任务一：带词频属性的文档倒排算法

该任务创建了两个Mapreduce job（分别对应java类`InvertedIndex`和`SortByFreuqency`）

InvertedIndex

- **Map阶段的设计思路**
  - 将输入的文本行拆分为单词，并为每个单词生成 `<词语，文档ID:1>` 的键值对，表示该词语在某个文档中出现一次。
  - Key：`Text`类型，表示词语（例如"哈利"）
  - Value：`Text`类型，表示文档ID和词频的组合 例如（“第二部-密室:1”）
  - 伪代码：

```
Function Begin Map(key: Object, value: Text, context: Context)
    // get file name
    docName = get file name from input split

    // 将输入行拆分为单词
    words = tokenize(value)
    wordFreqMap = new HashMap<String, Integer>

    // Output <word, docName:freq> for each word
    for each (word, freq) in wordFreqMap:
        set word = word
        set docInfo = docName + ":" + freq
        context.write(word, docInfo)
End Function
```

- **Reduce阶段的设计思路**
  - 合某个词语在所有文档中的出现信息，计算平均词频，并输出每个文档的词频。
  - Key：`Text` 类型，表示词语（例如 "哈利"）
  - Value：`Iterable<Text>` 类型，表示该词语在不同文档中的出现记录（例如 ["第一部-魔法石:1", "第一部-魔法石:1", "第二部-密室:1"]）
  - 伪代码：

```
Function Reduce(key: Text, values: Iterable<Text>, context:
Context)
    初始化统计数据结构 HashMap docFreqMap and docSet
```

```
    totalFrequency = 0
    for each value in values:
        (docId, freq) = 分割 value 为 [文档ID, 词频]
        totalFrequency = totalFrequency + freq
        docFreqMap[docId] = docFreqMap[docId] + freq // 累加词频
        docSet.add(docId) // 记录出现过的文档
    df = docSet的大小 // 文档频率
    averageFrequency = totalFrequency / df if df > 0 else 0

    output = format("%.2f", averageFrequency) + ", "
    for each (docId, freq) in docFreqMap:
        output = output + docId + ":" + freq + "; "

    outputKey = "[" + key + "]"
    context.write(outputKey, output)
End Function
```

- **输出示例**

```
[举着]    10.33, 第七部-死亡神器:15; 第二部-密室:9; 第六部-混血王子:11; 第三部-阿兹卡班的囚徒:5; 第五部-凤凰社:12; 第四部-火焰杯:10
[举给]    1.00, 第六部-混血王子:1
[举脚]    1.00, 第三部-阿兹卡班的囚徒:1
[举行]    4.43, 第二部-密室:1; 第七部-死亡神器:6; 第六部-混血王子:1; 第一部-魔法石:5; 第三部-阿兹卡班的囚徒:2; 第五部-凤凰社:3; 第四部-火焰杯:13
[举起]    49.14, 第七部-死亡神器:86; 第二部-密室:28; 第六部-混血王子:49; 第一部-魔法石:22; 第三部-阿兹卡班的囚徒:27; 第四部-火焰杯:65; 第五部-凤凰社:67
[举起手来]    2.00, 第三部-阿兹卡班的囚徒:2
[举足轻重]    1.00, 第四部-火焰杯:1
```

完整的输出文件位于`/user/221220115stu/lab2/output/task1/step1`

SortByFrequency

该MapReduce job的目的是对上一个倒排索引任务的输出按平均词频进行**降序排序**，同时应用于任务一和接下来的任务三

- **Map阶段的设计思路**
  - 从输入行中提取平均词频`avgFrequency`，将其取负值作为排序键（因为 MapReduce 默认按键升序排序，取负值可实现降序），并将整行作为值输出
  - Key：`DoubleWritable`类型，表示负的平均词频（例如 -2.50）
  - Value：`Text`类型，表示原始输入行（例如 "[word]　2.50, doc1:2; doc2:1"）
  - 伪代码：

```
Function Map(key: Object, value: Text, context: Context)
    parts = split(line, "\t", 2)
    // 提取词频部分
    secondPart = parts[1]
    commaIndex = find index of "," in secondPart
    frequencyStr = substring(secondPart, 0, commaIndex)

    try:
        // 按照词频，取负，实现降序排序
        frequency = parseDouble(frequencyStr)
        negFrequency = -frequency
        set negAvgFrequency = negFrequency
        set outputValue = line
```

```
            // 输出 <negFrequency, original line>
            context.write(negAvgFrequency, outputValue)
        catch NumberFormatException:
            print error "Failed to parse frequency from: " + line
    End Function
```

- **Reduce阶段的设计思路**
  - 按键（负平均词频）排序后的值（原始行）直接输出，完成降序排序。
  - 输入<key, value>：`<DoubleWrite, Iterable<Text>>`表示平均词频和对应的原式行
  - 输出<key, value>：`<Text, NullWritable>`, 表示原式行和无附加值
  - 伪代码：

```
Function Reduce(key: DoubleWritable, values: Iterable<Text>,
context: Context)
    for each value in values:
        context.write(value, NullWritable)
End Function
```

- **输出示例**

```
[的] 8536.71, 第二部-密室:5047; 第七部-死亡神器:10871; 第一部-魔法石:4209; 第六部-混血王子:8990; 第三部-阿兹卡班的囚徒:5983; 第四部-火焰杯:10863; 第五部-凤凰社:13794
[了] 4706.14, 第七部-死亡神器:5648; 第二部-密室:2683; 第六部-混血王子:5262; 第一部-魔法石:2392; 第三部-阿兹卡班的囚徒:3640; 第五部-凤凰社:7655; 第四部-火焰杯:5663
[他] 4128.71, 第二部-密室:2165; 第七部-死亡神器:4918; 第六部-混血王子:4581; 第一部-魔法石:2259; 第三部-阿兹卡班的囚徒:2963; 第五部-凤凰社:6585; 第四部-火焰杯:5430
[哈利]  2713.43, 第二部-密室:1621; 第七部-死亡神器:3390; 第一部-魔法石:1362; 第六部-混血王子:2873; 第三部-阿兹卡班的囚徒:2136; 第五部-凤凰社:4295; 第四部-火焰杯:3317
[在] 2416.43, 第二部-密室:1318; 第七部-死亡神器:3270; 第一部-魔法石:1157; 第六部-混血王子:2400; 第三部-阿兹卡班的囚徒:1846; 第四部-火焰杯:2847; 第五部-凤凰社:4077
[我] 2270.00, 第二部-密室:1280; 第七部-死亡神器:2772; 第六部-混血王子:2967; 第一部-魔法石:1126; 第三部-阿兹卡班的囚徒:1621; 第四部-火焰杯:2636; 第五部-凤凰社:3488
```

完整的输出文件位于`/user/221220115stu/lab2/output/task1/step2`

任务二：计算TF-IDF

该任务创建了一个MapReduce job，对应java类`CalculateTFIDF`, 目标是计算文本集合中每个词语在每个文档中的 **TF-IDF** 值

- **Map阶段的设计思路**
  - 同任务一，统计每个文档中每个词语的词频（TF），并为每个词语生成`<词语，文档名:词频>`的键值对
  - Key：`Text`类型，表示词语（例如"哈利"）
  - Value：`Text`类型，表示文档名和词频的组合 例如（“第一部-魔法石:1”）
  - 伪代码：同实验一
- **Reduce阶段的设计思路**
  - 根据词频（TF）和IDF计算每个词语在每个文档中的 TF-IDF 值，并输出结果
  - 输入<key, value>：`<Text, Iterable<Text>>`类型， 表示词语及其在文档中的出现记录
  - 输出<key, value>：`<Text, Text>`类型，表示文档名和它其中的词语及其TF-IDF

- 伪代码：

```
Function Reduce(key: Text, values: Iterable<Text>, context:
Context)
    // 初始化数据结构
    docFreqMap = new HashMap<String, Integer>
    docSet = new HashSet<String>
    // 将每个来自Map阶段的单个输入聚合
    for each val in values:
        (docName, freq) = split(val, ":")
        docFreqMap[docName] = parseInt(freq)
        docSet.add(docName)

    // Calculate IDF
    df = size of docSet
    idf = log(totalDocs / (df + 1)) / log(2)

    // 计算TF-IDF并输出
    for each (docName, tf) in docFreqMap:
        tfidf = tf * idf
        outputValue = key + "," + format(tfidf, 3 decimals)
        context write
End Function
```

- **输出示例**

```
第四部-火焰杯, 差不多, -4.238
第三部-阿兹卡班的囚徒, 差事, 3.615
第一部-魔法石, 差使, 1.222
第五部-凤凰社, 差使, 3.667
第五部-凤凰社, 差几, 1.807
第七部-死亡神器, 差别, 2.913
第三部-阿兹卡班的囚徒, 差别, 0.485
第五部-凤凰社, 差别, 0.971
```

完整的输出文件位于`/user/221220115stu/lab2/output/task2`

任务三：去除停用词

该任务的目标是处理文本数据，去除停用词后统计每个词语在每个文档中的词频，按照任务一的格式进行排序后输出，同样创建了两个MapReduce job，第一个对应java类`RemoveStopWords`，第二个java类同任务一，为`SortByFrequency`

RemoveStopWords

- **Map阶段的设计思路**
  - 从输入文档中提取词语，读取停用词表以过滤停用词，并为每个过滤后的词语生成 <词语, 文档ID:词频> 的键值对。
  - <Key, Value>类型：`<Text, Text>`,表示词语，文档ID和词频的组合
  - 伪代码：

```
Function Setup(context: Context)
```

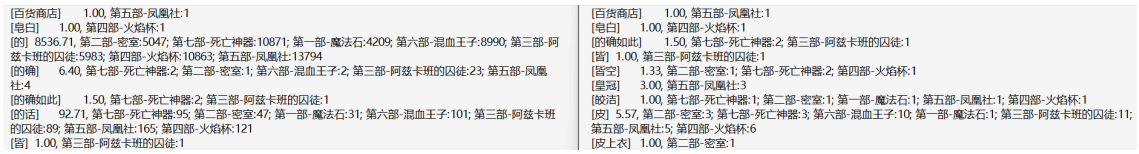
```
    // Load stop words from distributed cache
    cacheFiles = get cache files from context
    for each file in cacheFiles:
        if file contains "cn_stopwords.txt":
            open file and read each item into stopwords
    get document id from input
End Function

Function Map(key: LongWritable, value: Text, context: Context)
    // Split input line into words
    words = split(value, " ")
    for each word in words:
        if word is not empty and not in stopWords:
            wordCounts[word] = wordCounts.getOrDefault(word, 0)
+ 1
End Function

Function Cleanup(context: Context)
    // Output word counts for the document
    for each (word, count) in wordCounts:
        context.write(word, docId + ":" + count)
End Function
```

- **Reduce阶段的设计思路**
  - 聚合某个词语在所有文档中的出现信息，计算平均词频，并输出每个文档的词频。
  - 输入<key, value>：`<Text, Iterable<Text>>` 类型， 表示词语及其在文档中的出现记录
  - 输出<key, value>：`<Text, Text>`类型, 对应[词语]（Key），平均词频和每个文档的词频（Value）
  - 伪代码：

```
Function Reduce(key: Text, values: Iterable<Text>, context:
Context)
    初始化统计数据结构 HashMap: docFreqMap and docSet

    // 统计词频
    for each val in values:
        (docId, freq) = split(val, ":")
        freqValue = parseInt(freq)
        totalFrequency = totalFrequency + freqValue
        docFreqMap[docId] = docFreqMap.getOrDefault(docId, 0) +
freqValue
        docSet.add(docId)

    // 计算平均词频
    df = size of docSet
    averageFrequency = totalFrequency / df if df > 0 else 0

    // 构建输出
    output = format(averageFrequency, 2 decimals) + ", "
```

```
        for each (docId, freq) in docFreqMap:
            output = output + docId + ":" + freq + "; "
        if size of docFreqMap > 0:
            remove last "; " from output

        // 输出结果
        outputKey = "[" + key + "]"
        context.write(outputKey, output)
    End Function
```

- **输出示例**



> **Note**
>
> *作为对比：左侧为去除停用词前，右侧为去除停用词后，可以看到停用词"的"，"的确"均被去除*

完整输出文件见`/user/221220115stu/lab2/output/task3/step1`

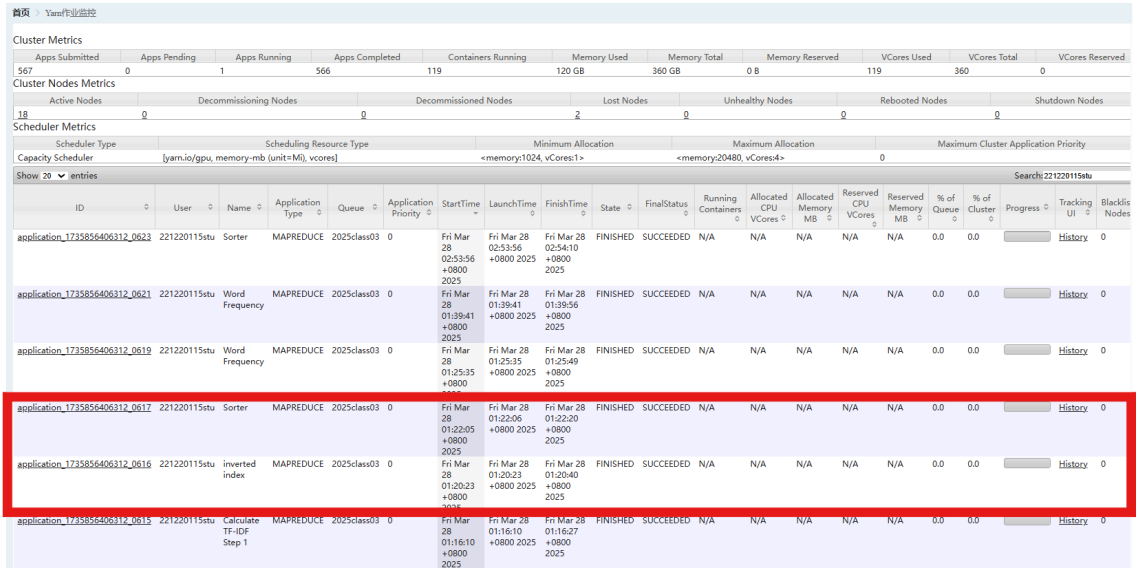
- **排序后的输出示例**

```
[哈利]  2713.43, 第二部-密室:1621; 第七部-死亡神器:3390; 第一部-魔法石:1362; 第六部-混血王子:2873; 第三部-阿兹卡班的囚徒:2136; 第五部-凤凰社:4295; 第四部-火焰杯:3317
[说]  1688.71, 第二部-密室:938; 第七部-死亡神器:1875; 第六部-混血王子:1882; 第一部-魔法石:861; 第三部-阿兹卡班的囚徒:1318; 第四部-火焰杯:1993; 第五部-凤凰社:2954
[赫敏]  742.71, 第二部-密室:299; 第七部-死亡神器:1196; 第六部-混血王子:680; 第一部-魔法石:283; 第三部-阿兹卡班的囚徒:648; 第五部-凤凰社:1251; 第四部-火焰杯:842
[罗恩]  676.00, 第七部-死亡神器:870; 第二部-密室:507; 第六部-混血王子:689; 第一部-魔法石:302; 第三部-阿兹卡班的囚徒:592; 第五部-凤凰社:1011; 第四部-火焰杯:761
[没有]  669.86, 第七部-死亡神器:778; 第二部-密室:338; 第六部-混血王子:742; 第一部-魔法石:388; 第三部-阿兹卡班的囚徒:511; 第五部-凤凰社:1081; 第四部-火焰杯:851
[一个]  664.86, 第二部-密室:418; 第七部-死亡神器:846; 第六部-混血王子:745; 第一部-魔法石:378; 第三部-阿兹卡班的囚徒:316; 第四部-火焰杯:904; 第五部-凤凰社:1047
[里]  608.57, 第七部-死亡神器:754; 第二部-密室:398; 第六部-混血王子:549; 第一部-魔法石:361; 第三部-阿兹卡班的囚徒:405; 第五部-凤凰社:943; 第四部-火焰杯:850
```

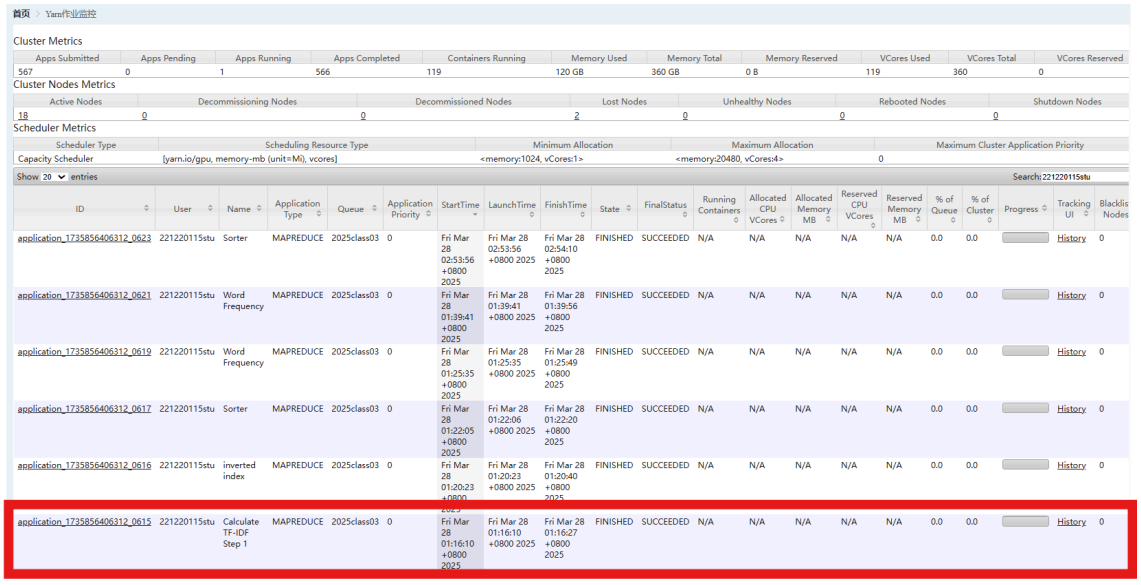
完整输出文件见`/user/221220115stu/lab2/output/task3/step2`

执行报告

- **任务一：InvertedIndex & SortByFrequency**



- **任务二：CalculateTFIDF**



首页 > Yarn作业监控

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
567	0	1	566	119	120 GB	360 GB	0 B	119	360	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
18	0	0	2	0	0	0

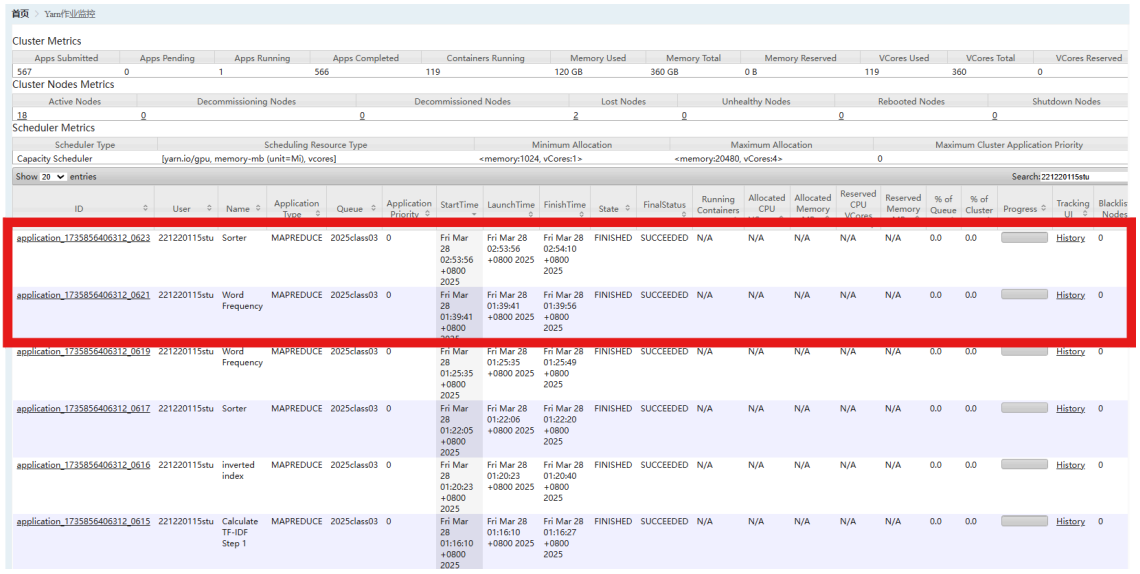
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[yarn.io/gpu, memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:20480, vCores:4>	0

Show 20 entries Search: 221220115stu

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1735856406312_0623	221220115stu	Sorter	MAPREDUCE	2025class03	0	Fri Mar 28 02:53:56 +0800 2025	Fri Mar 28 02:53:56 +0800 2025	Fri Mar 28 02:54:10 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0621	221220115stu	Word Frequency	MAPREDUCE	2025class03	0	Fri Mar 28 01:39:41 +0800 2025	Fri Mar 28 01:39:41 +0800 2025	Fri Mar 28 01:39:56 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0619	221220115stu	Word Frequency	MAPREDUCE	2025class03	0	Fri Mar 28 01:25:35 +0800 2025	Fri Mar 28 01:25:35 +0800 2025	Fri Mar 28 01:25:49 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0617	221220115stu	Sorter	MAPREDUCE	2025class03	0	Fri Mar 28 01:22:05 +0800 2025	Fri Mar 28 01:22:06 +0800 2025	Fri Mar 28 01:22:20 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0616	221220115stu	inverted index	MAPREDUCE	2025class03	0	Fri Mar 28 01:20:23 +0800 2025	Fri Mar 28 01:20:23 +0800 2025	Fri Mar 28 01:20:40 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0615	221220115stu	Calculate TF-IDF Step 1	MAPREDUCE	2025class03	0	Fri Mar 28 01:16:10 +0800 2025	Fri Mar 28 01:16:10 +0800 2025	Fri Mar 28 01:16:27 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

- **任务三：RemoveStopWord & SortByFrequency**



首页 > Yarn作业监控

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
567	0	1	566	119	120 GB	360 GB	0 B	119	360	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
18	0	0	2	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[yarn.io/gpu, memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:20480, vCores:4>	0

Show 20 entries Search: 221220115stu

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1735856406312_0623	221220115stu	Sorter	MAPREDUCE	2025class03	0	Fri Mar 28 02:53:56 +0800 2025	Fri Mar 28 02:53:56 +0800 2025	Fri Mar 28 02:54:10 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0621	221220115stu	Word Frequency	MAPREDUCE	2025class03	0	Fri Mar 28 01:39:41 +0800 2025	Fri Mar 28 01:39:41 +0800 2025	Fri Mar 28 01:39:56 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0619	221220115stu	Word Frequency	MAPREDUCE	2025class03	0	Fri Mar 28 01:25:35 +0800 2025	Fri Mar 28 01:25:35 +0800 2025	Fri Mar 28 01:25:49 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0617	221220115stu	Sorter	MAPREDUCE	2025class03	0	Fri Mar 28 01:22:05 +0800 2025	Fri Mar 28 01:22:06 +0800 2025	Fri Mar 28 01:22:20 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0616	221220115stu	inverted index	MAPREDUCE	2025class03	0	Fri Mar 28 01:20:23 +0800 2025	Fri Mar 28 01:20:23 +0800 2025	Fri Mar 28 01:20:40 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1735856406312_0615	221220115stu	Calculate TF-IDF Step 1	MAPREDUCE	2025class03	0	Fri Mar 28 01:16:10 +0800 2025	Fri Mar 28 01:16:10 +0800 2025	Fri Mar 28 01:16:27 +0800 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0