基于 Storm 的 SQL 处理

1. 问题描述

随着数据的体积越来越大、种类越来越多、产生速度越来越快，公司的日常运营经常会产生 TB 级别的数据，数据来源包括网站、社交媒体、交易型商业数据以及其他商业环境中创建的数据等。考虑到数据的生成量，实时处理成为了许多机构需要面对的挑战。Apache Storm[2]作为实时分布式大数据处理系统，具有低延迟、高性能、可扩展和高容错性等优势，保证了大数据进行严格有效的处理。但是 Storm 的开发门槛较高，而 SQL 语言作为使用最广泛的关系数据库操作语言，如果能在 Storm 上支持 SQL，则更加便于实施业务的计算和处理。

因此，此次实验的主要目的是要在 Storm 系统上封装一个 SQL 语句查询接口。所以，首先，需要实现对用户输入 SQL 语句的解析，这里拟采用编译工具 ANTLR[5]来实现。其次，由于 Storm 对数据的计算和处理主要是通过创建一个 Topology 然后提交给 Storm 集群来实现的，因此问题的核心在于如何将用户输入的 SQL 语句动态地映射成 Storm 上相应的 Topology。再次，为了创建该 Topology 需要实现各个算子，如 selection，projection，group by，join，aggregation 等操作。

2. 相关技术

2.1. ANTLR

ANTLR[6]是 Another Tool for Language Recognition 的简称。它与 Linux 下的开源工具 Flex 和 Bison 类似，是一个语言识别工具。开发者只要提供一个描述该语言的上下文无关文法，它就会自动生成对应词法分析器 (Lexer)、语法分析器 (Parser) 和树分析器 (Tree Parser)。其中，文法描述语言是巴科斯范式(BNF)的形式。所以在本次实验中，我们只需要自己设计 SQL 的巴科斯范式，就可以借助 ANTLR 中的 visitor 工具生成访问语法树各个结点的类，然后编程实现各个语法树结点的访问逻辑，就可以获得相应语法树结点的信息。利用这些信息就可以构建该 SQL 语句的 Query Plan。

2.2. JGraphT

JgraphT[7]是一个免费的、开源的 Java 代码库，2003-2007 年，由巴拉克西奈开发。它实现了对数学图论对象和算法的支持。利用它，开发者可以很方便地实现对有向图、无向图、加权图、简单图、复合图、伪图等图论对象的创建。它也支持 Java 泛型，它提供了丰富的图操作的接口，开发者也可以很方便地实现对图的创建、修改、删除、遍历，也可以添加顶点、构造边，进行图的映射、连通性检查、求最短路、求最小顶点覆盖等。在此次实验中，我们需要构建用户输入的 SQL 语句的 Query Plan，其实这里的 Query Plan 就是一个拓扑图，即有向无环图。利用深度遍历 SQL 语法树得到的各种信息，就可以很方便地实现 SQL 语句

Query Plan 的构建。

2.3. Apache Storm

Apache Storm[1]是一个分布式实时大数据处理系统。Storm 设计用于在容错和水平可扩展方法中处理大量数据。它是一个流数据框架，具有最高的摄取率。Storm 是无状态的，它通过 Apache ZooKeeper 管理分布式环境和集群状态，可以并行地对实时数据执行各种操作。Storm 上可以使用各种编程语言，包括 Clojure、java、Ruby、Python 等。它可以保证每个消息至少能得到一次完整处理，当任务失败时，它会负责从消息源重试消息，从而提高系统的可靠性。

2.3.1. Storm 集群架构

Storm 的集群架构如下图 2-1 所示：

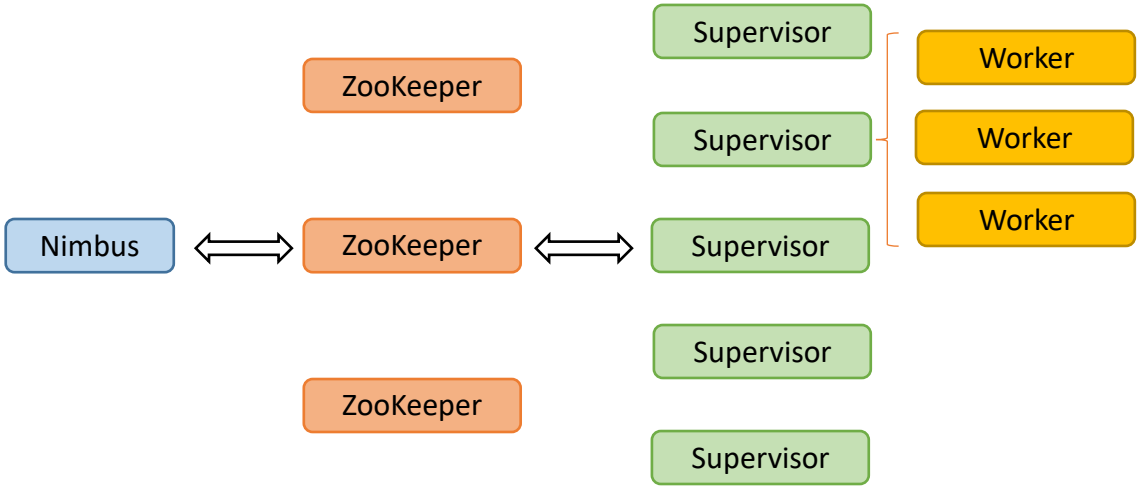


图 2-1 Storm 的集群架构

Storm 整体上是 master/worker 类型的，其中，Nimbus 节点 Storm 集群的主节点。负责在所有工作节点之间分发数据，向工作节点分配任务和监视故障。ZooKeeper 用于集群中的协调。Storm 是无状态的，ZooKeeper 负责公有数据的存放（如心跳信息、集群的状态和配置信息），Nimbus 将分配给 Supervisor 的任务写入 ZooKeeper。Supervisor 是工作节点。接受 Nimbus 分配的任务，管理自己的 Worker 进程。Worker 是具体的处理逻辑组件。不自己运行任务，而是创建执行器并要求他们执行特定的任务。

2.3.2. Storm 工作流程

Apache Storm 从一端读取实时数据的原始流，并将其传递通过一系列小处理单元，并在另一端输出/处理有用信息。如下图 2-2 所示。

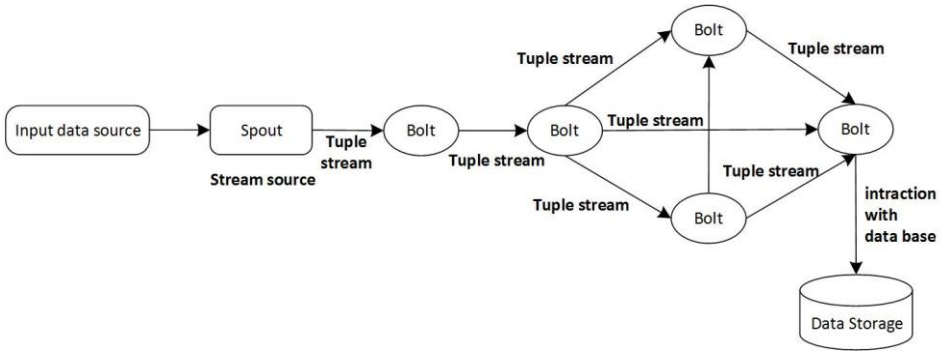


图 2-2 storm 内部工作流程

其中，Spout 为流的源，Storm 可以从原始数据源接受输入数据，或者编写 spouts 来从数据源读取数据；Bolts 是逻辑处理单元，Bolts 接收数据并进行处理后，发射到一个或多个 Bolts，Bolts 可以执行过滤、聚合、连接、与数据源和数据库交互等多种操作。

Spouts 和 Bolts 连接在一起，形成一个拓扑结构(Topology)，拓扑中的每一个节点都是一个处理逻辑节点。Storm 保证拓扑始终运行，知道终止拓扑。

3. 解决方案

在问题描述章节中已经说明了此次实验的目的是要在 Storm 系统上封装一个 SQL 语句查询接口。主要是解决三个问题：首先是对用户输入的 SQL 语句的解析，这里采用编译工具 ANTLR 来实现。其次是将用户输入的 SQL 语句动态地映射成 Storm 上相应的 Topology。再次，再次，为了创建该 Topology 需要实现 selection，projection，group by，join，aggregation 等算子。整个主要包括以下三个阶段：解析 SQL 语句、生成 Query Plan 以及动态创建 Topology。当然事先还要实现 selection，projection，group by，join，aggregation 等这些算子。整个系统的架构图如下图 3-1 所示：

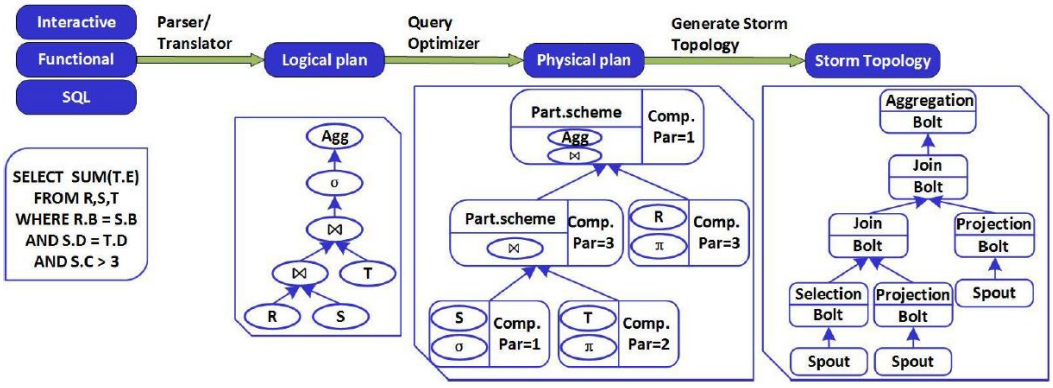


图 3-1 系统架构图

在上图中，Parser 主要用于解析用户输入的 SQL 语句，Query Plan 是解析 SQL 语句创建的拓扑图，Query Optimizer 主要是调整拓扑图中的节点以减少计算的开销。动态生成的 Topology 中的各个 Bolt 负责执行 selection，projection，group by，join，aggregation 等这些算子的操作。

3.1. SQL Parser

SQL Parser 主要用于解析用户输入的 SQL 语句。根据获取到的 SQL 语句，对其进行词法分析并提取其中的 token。再对这个 token 流进行词法分析得到一棵语法树。再采用自底向上的方式访问语法树的各个节点，规约得到需要的信息，这里需要存储的信息主要是投影的 fields，aggregation 操作的名称以及操作的 field，join 的识别以及 join key，group by 的 fields 以及条件，选择操作的过滤条件，时间窗口的宽度等。然后利用这些信息生成一个 Query Plan，即 DAG（拓扑图，有向无环图）。图中的每个起始节点（入度为 0 的节点）都是一个数据源，每个中间节点都对应一个具体的算子。

3.2. 查询优化

通过 SQL Parser 的解析得到了一个拓扑图，但是这个拓扑图还是有待优化的。主要考虑以下优化操作：在有 join 操作的情况下，尽可能地将 selection 操作提前，以降低数据的访问量，从而提升查询性能；能 merge 的节点尽量 merge。最终会构建出一个较好的拓扑图。

3.3. 动态创建 Topology

在 Storm 中，实时的计算是通过 Topology 的各个 Bolt 结点来完成的。Spout 用于产生数据，Bolts 用于对输入其中的数据进行计算并产生结果输出出去。Spouts 和 Bolts 间以及 Bolts 和 Bolts 间的边代表了数据流，即一些连续的元组序列。在 3.2 得到的拓扑图中，图中的每个起始节点对应一个 Spout 节点，即数据源，如表 R、表 S、表 T，这里的表的实际含义是指 Stream 的 id 等；而每个算子操作节点则对应了一个 Bolt 节点。通过按照拓扑序来遍历这个拓扑图就可以动态地生成对应的 Topology。拓扑图中每个节点之间的连接关系与 Topology 中每个节点的连接关系一一对应。

4. 系统设计与实现

4.1. SQL Parser

当收到用户输入的 SQL 语句后，首先需要对改命令进行解析，这里我们利用了开源的语言识别工具 ANTLR 来实现。ANTLR 会根据我们设计的 SQL 上下文无关文法，自动生成对应词法分析器 (Lexer)、语法分析器 (Parser) 和树分析器 (Tree Parser)。借助 ANTLR 中的 visitor 工具生成访问语法树各个结点的类，这样开发者就可以很方便地采用 visitor 模式来访问语法树的各个节点。

这里设计的 SQL 文法描述语言采用巴科斯范式(BNF)的形式。具体实现是针对巴科斯范式中的每一个产生式，深度遍历该语法树分支的每一个节点就可以获取该节点的值。以 SQL 语句“select student.name from student where student.major=cs”为例，在 SQL Parser 的内部会组织成如下图 4-1 所示的语法树：

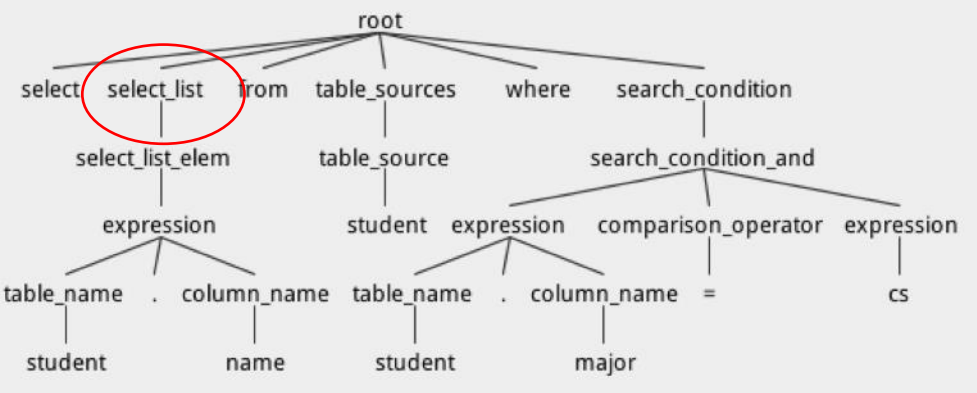


图 4-1 SQL 例子语法树

该语法树的根是”root”结点，对于它的子节点“select_list”，select_list 节点的产生式的实际设计为：

```
select_list
    : select_list_elem (',' select_list_elem)* #printSelectList
    ;
```

所以，为了规约得到 select_list 节点的值，需要计算 select_list_elem 节点，所以对于 select_list 结点的处理函数只需要开发者书写深度遍历 select_list_elem 节点的处理逻辑就行了。而为了获取 select_list_elem 节点的值需要实现对它的子节点 expression 的访问逻辑，同样，为了获取 expression 节点的值，需要根据 expression 节点的产生式实现对 table_name 和 column_name 节点的访问逻辑，在这个例子中，最终会获取到 select_list 结点的值是 student.name。

所以通过采用 visitor 模式深度遍历语法树的各个结点就可以获取到开发者想要的信息。在这里我们需要存储的信息有 select_list 的值，它标记了 Projection 操作的信息；table_sources，它标记了流数据的来源；search_condition，它标记了 Selection 操作和 Join 操作的信息；同样还有 group_by_item，它标记了 group by 操作的信息；within_time，它标记了时间窗口的宽度。

获取了以上信息以后就可以生成该 SQL 语句的查询计划，这个查询计划在内存中以拓扑图的方式存储，具体实现调用了开源库 JGraphT。具体实现时，通过遍历 ANTLR 解析出来的信息来生成拓扑图的每一个顶点。由于需要考虑优化操作，所以再额外定义一个优先级队列来设定各个拓扑节点的先后顺序，然后遍历这个优先级队列同时结合各个顶点的属性信息就可以知道各个顶点之间的连接关系，这里的连接关系主要是每个顶点的孩子节点 id 和父亲节点 id。利用这些关系就可以调用 JGraphT 的在各个顶点之间添加边。

4.2. Bolt 算子

在 Storm 中，实时的计算实际上是由 Topology 的中的各个 Bolt 结点来完成的。所以，有必要实现一系列的 Bolt 算子来对应 SQL 中的各种操作。这里的 Bolt 算子主要有 selection、projection、group by、join、aggregation 等，aggregation 主要是针对常规的聚集函数，如 count、sum、max、min、avg 等。

4.2.1. Selection

Selection 算子用于选择行，通常和 where 关键字联系在一起，where 后是过滤条件；也可以和 Projection 及 Aggregation 组合使用。具体实现时，主要是利用 ANTLR 解析 SQL 语句得到的 where 关键字后的过滤条件来对父节点输入的 tuple 进行过滤。获取过滤条件所对应的 field 的值，判断是否满足过滤条件，满足则发送给下一个 Bolt，不满足则丢弃。Selection 算法如下图 4-2 所示：

Algorithm 1 Selection operator

Input: *Tuple* from father
Output: *Tuple* after selection operator(filtering)
1: obtain data attributes to be processed according to semantics
2: **while** *Tuple* is not empty **do**
3: parse the *Tuple* from spout
4: obtain the index of tuple in *where_condition*
5: execute filtering
6: emit resultant *Tuple* to the next bolt
7: **end while**

图 4-2 Selection Bolt 算法

4.2.2. Projection

Projection 算子用于投影数据，即选择具体的某一个或者某几个列。具体实现时，获取需要投影的列信息，传输给 Projection Bolt，这里的 Projection Bolt 主要用于结果的输出，所以打印父节点输入的 tuple 的 fields 值就可以了。Projection 算法如下所示：

Algorithm 2 Projection operator

Input: *Tuple* from father
Output: *Tuple* after projection operator(filtering)
1: obtain data attributes to be processed according to semantics
2: **while** *Tuple* is not empty **do**
3: parse the *Tuple* from spout
4: obtain the index of tuple in *select_condition*
5: print the field value of the tuple
6: **end while**

图 4-2 Projection Bolt 算法

4.2.3. Join

Join 关键字用于多个数据源的连接操作，这里的 Join 操作是基于时间窗的。具体实现时，对在同一个时间窗口内的数据进行连接。Storm 内置了 BaseWindowedBolt 类，可以一次性获取该时间窗口内的所有 Tuple。所以，Join Bolt 需要继承 BaseWindowedBolt 类，并通过 withTumblingWindow 方法设置时间窗口的宽度，即 duration 参数。目前系统仅支持两个流数据的等值连接，这里采用 hash join 的连接方式，用 hashtable 分别存储一个时间窗口内来

源于两个数据流的数据，再将其进行连接和保存，输出结果到下一个 Bolt，一定时间间隔后时间窗口进行更新，再重复上述操作。Join 算法如下所示：

Algorithm 3 Join operator

Input: *Tuple* from Bolt
Output: *Tuple* after join operator

```
for every five seconds do
    parse the format of the data of the slide window
    for each Node t in Tree do
        if Tuple.id equals t.root
            traverse recursively the left children for matching data
            traverse recursively the right children for matching data
            output the resultant Tuple after join operator
        end if
    end for
    emit the Tuple
    update the window
end for
```

图 4-3 Join Bolt 算法

4.2.4. Group By

Group By 操作一般用于 Aggregation 方法中，Group By 操作也是基于时间窗的，它将数据按照指定的列关键字进行分组，分组之后再对其进行处理。Group By Bolt 也需要继承 Storm 内置的 BaseWindowedBolt 类。提取 group by key 和 Aggregation 需要操作的列。获取当前时间窗内的所有元祖，存到一个 hashtable 里面，key 是 group by key，value 是需要进行聚集操作的 Field 值的链表。Group-by 算法如下图所示：

Algorithm 4 Group by operator

Input: *Tuple* from father
Output: *Tuple* after group by operator

```
for every five seconds do
    parse the format and extract content of the data of the slide window
    for each String str in groupList do
        if key-value does not exist
            insert the key-value into result
        else
            obtain corresponding key-value
        end if
    end for
    execute aggregate operator
    emit the Tuple
    update the window
end for
```

图 4-4 Join Bolt 算法

4.2.5. Aggregation

Aggregation 操作通过对一组数据进行聚合，进而输出一个单一的值。系统中 Aggregation 通常和 Group By 操作结合，将 Group By Bolt 的输出作为 Aggregation Bolt 的输入，通过提取我们需要进行的具体聚合算子，在 Aggregation Bolt 的 execute 方法中实现聚合操作。通过扫描 Group By 操作得到的 hash table 就可以实现对相应的 tuple 的聚集操作。这里的聚集操作主要包括 Sum、Count、Avg、Max 和 Min 等。

4.2.5.1. Sum

对选中的关键字进行加和，再将结果输出，相关算法如下图 4-5-1 所示：

Algorithm 5.1 Aggregation sum operator

Input: *Tuple* from group by bolt
Output: *Tuple* after aggregation sum operator
1: parse the format and extract content of data
2: **for** each the value of tuple **do**
3: sum += value
4: emit the *Tuple*
5: **end for**

图 4-5-1 sum aggregation 算法

4.2.5.2. Count

对选中的关键字进行计数，再将结果输出，相关算法如下图 4-5-2 所示：

Algorithm 5.2 Aggregation count operator

Input: *Tuple* from Bolt
Output: *Tuple* after aggregation count operator
1: parse the format and extract content of data
2: **for** each tuple **do**
3: count++
4: emit the *Tuple*
5: **end for**

图 4-5-2 count aggregation 算法

4.2.5.3. Avg

对选中的关键字进行取平均值。用 sum 存储目前所有数据的总和，count 存储目前所有数据的计数，每收到一个数据则先加到 sum 中，并且 count 值加一，二者相除得到平均值并输出。相关算法如下图 4-5-3 所示：

Algorithm 5.3 Aggregation avg operator

Input: *Tuple* from Bolt

Output: *Tuple* after aggregation avg operator

```
1:  parse the format and extract content of data
2:  for each tuple do
3:      sum =+ value
4:      count++
5:      avg = sum/count
6:      emit the Tuple
7:  end for
```

图 4-5-3 count aggregation 算法

4.2.5.4. Max

输出选中关键字中数据的最大值。每得到一个数据时将其与当前最大值比较，输出其中值较大的数据。相关算法如下图 4-5-4 所示：

Algorithm 5.4 Aggregation max operator

Input: *Tuple* from Bolt

Output: *Tuple* after aggregation max operator

```
1:  parse the format and extract content of data
2:  for each the value of tuple do
3:      if value > max
4:          max = value
5:      end if
6:      emit the Tuple
7:  end for
```

图 4-5-4 max aggregation 算法

4.2.5.5. Min

输出选中关键字中数据的最小值。每得到一个数据时将其与当前最小值比较，输出其中值较小的数据。相关算法如下图 4-5-5 所示：

Algorithm 5.5 Aggregation min operator

Input: *Tuple* from Bolt

Output: *Tuple* after aggregation min operator

```
1:  parse the format and extract content of data
2:  for each the value of tuple do
3:      if value < min
4:          min = value
5:      end if
6:      emit the Tuple
7:  end for
```

图 4-5-5 min aggregation 算法

4.3. 动态生成 Topology

有了以上 Bolt 算子（当然还要有数据源），就可以根据 SQL Parser 生成的 Query Plan(即拓扑图)来动态地生成 Topology 了，然后再把该 Topology 提交给 Storm 集群就可以完成用户的查询任务了。

在 Storm 中，实时的计算是通过 Topology 的各个 Bolt 结点来完成的。Spout 用于产生数据，Bolts 用于对输入其中的数据进行计算并产生结果输出出去。Spouts 和 Bolts 间以及 Bolts 和 Bolts 间的边代表了数据流，即一些连续的元组序列。在 3.2 得到的拓扑图(DAG)中，图中的每个起始节点对应一个 Spout 节点，即数据源，如表 R、表 S、表 T，注意这里的表的实际含义是指 Stream 的 id。每个算子操作节点则对应了一个 Bolt 节点。所以通过按照拓扑序来遍历 DAG 中的每个节点，对于 DAG 中的每个节点根据它的类型来创建对应的 Bolt，同时遍历该节点的 father list，创建的 Bolt 实例与它的父节点相连。这样整个 Topology 就被动态地创建出来了，把该 Topology 提交给 Storm 集群就可以执行用户的查询任务了。

5. 测试结果

下面给出几个不同操作组合的测试用例。通过手动创建 student 和 tc 两个 spout 来模拟数据源。

5.1. 测试用例一

当用户输入以下 SQL 查询语句时：

select *tc.id, tc.courseName, tc.courseTeacher, tc.courseTime, tc.location* ***from*** *tc*

生成的拓扑结构图如下所示，其中圆矩形代表 Spout，椭圆形代表 Bolt：

测试结果如下图 5-1 所示：

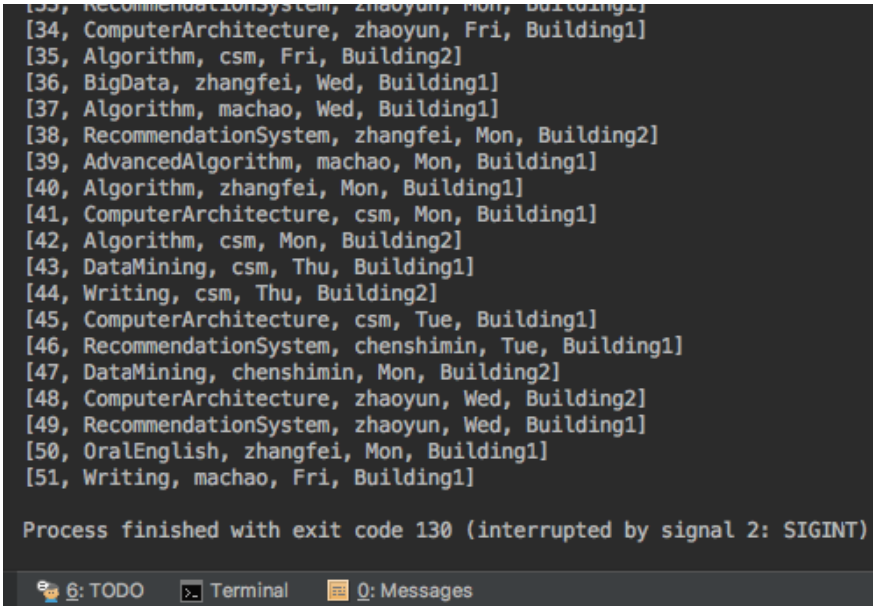


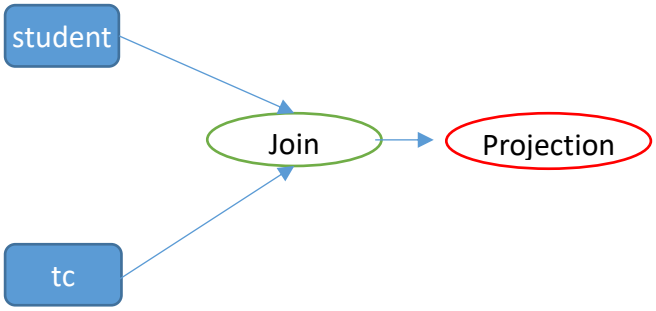
图 5-1 测试用例三输出

5.2. 测试用例二

当用户输入以下 SQL 查询语句时：

select *student.id, student.name, tc.courseName, tc.location* ***from*** *tc,student* ***where*** *student.id=tc.id* ***within*** *5000*

生成的拓扑结构图如下所示，其中圆矩形代表 Spout，椭圆形代表 Bolt：



测试结果如下图 5-2 所示：

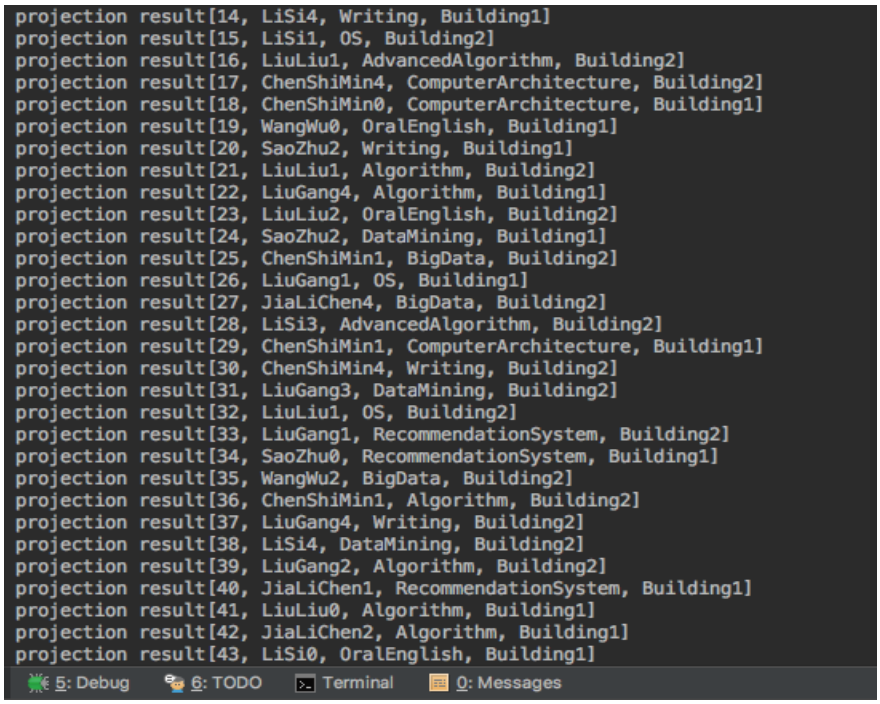


图 5-2 测试用例二输出

5.3. 测试用例三

当用户输入以下 SQL 查询语句时：

select *tc.id, tc.courseName, tc.courseTeacher, tc.courseTime, tc.location* ***from*** *tc* ***where*** *tc.courseName=OS*

测试结果如下图 5-3 所示：

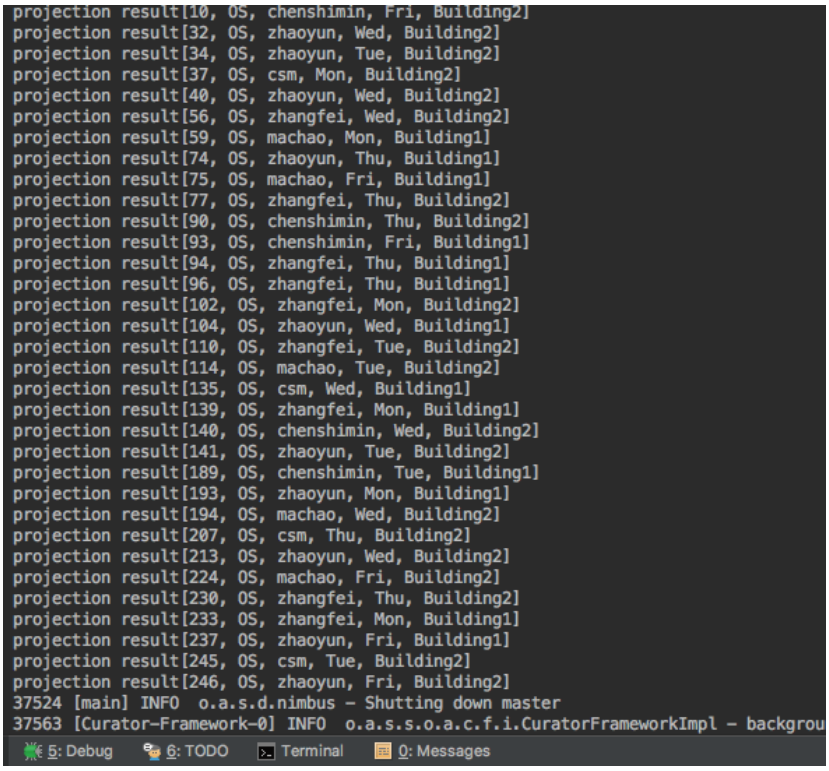


图 5-3 测试用例四输出

5.4. 测试用例四

当用户输入以下 SQL 查询语句时：

select *student.gender, avg(student.gpa)* ***from*** *student* ***group by*** *student.gender* ***within*** *2000*

上述 SQL 语句动态生成的 Topology 如下所示，其中圆矩形代表 Spout，椭圆形代表 Bolt：

测试结果如下图 5-4 所示：

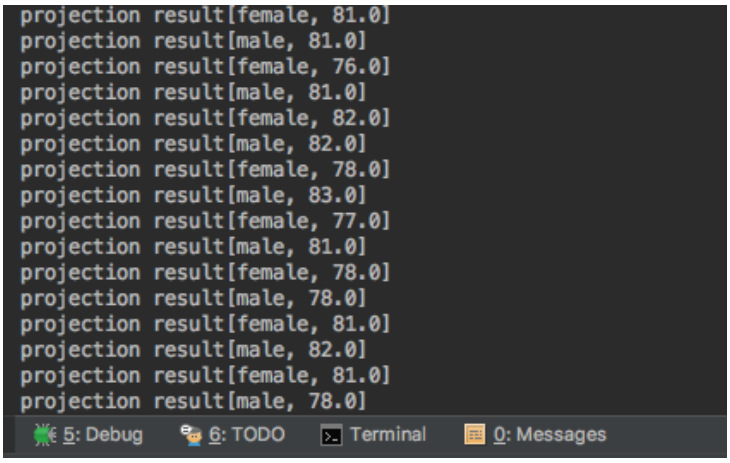
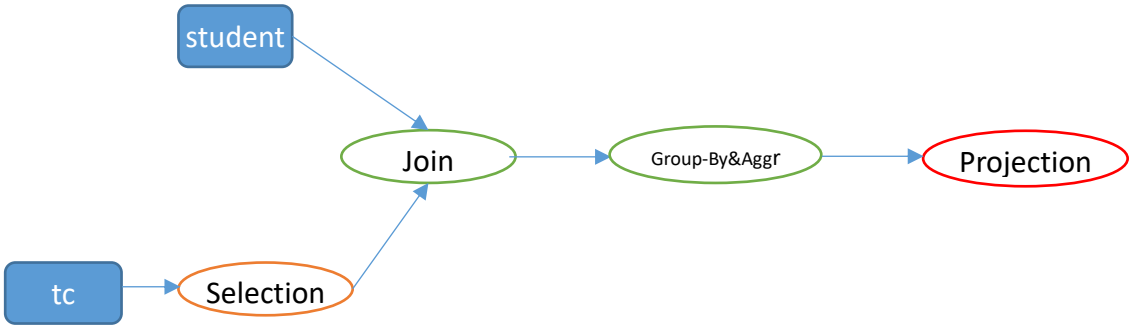


图 5-4 测试用例一输出

5.5. 测试用例五

当用户输入以下 SQL 查询语句时：

select *tc.courseName, avg(student.gpa)* ***from*** *tc,student* ***where*** *tc.id=student.id and tc.location=Building1* ***group by*** *tc.courseName* ***within*** *10000*



测试结果如下图 5-5 所示：

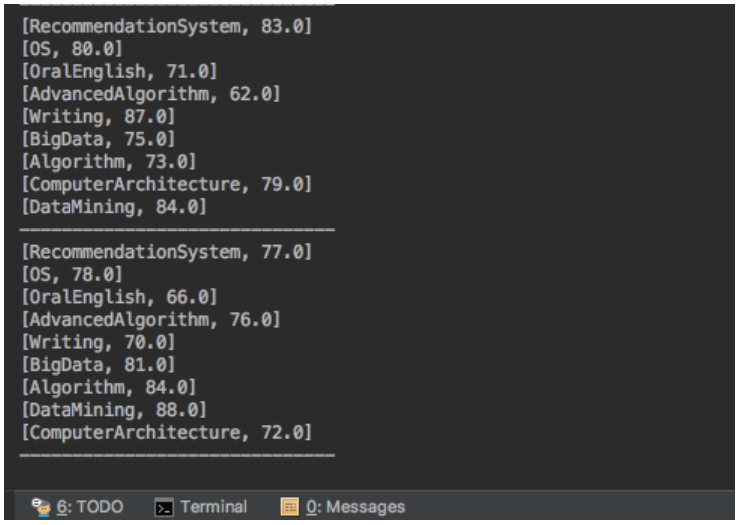


图 5-5 测试用例三输出

6. 小组分工

姓名	学号	分工
江梓涵	201618013229008	Projection、Join 操作、DAG 映射 Topology
李坚松	201618013229011	SQL 解析、DAG 映射 Topology、Selection 操作、修改文档、制作 PPT
刘卉	201618013229016	Group-By 操作、测试、书写文档
张秋平	201618013229025	Aggregation 操作、测试、书写文档

参考文献

[1] Apache Storm. Apache Storm Tutorial[EB/OL]. http://www.tutorialpoint.com,2017,06.

[2] Apache Storm. Storm 官网[EB/OL].http://storm.apache.org,2017,06.

[3] Aleksandar Vitorovic, Mohammed Elseidy, Khayyam Guliyev, et al. Squall: Scalable Realtime Analytics[J]. VLDB, 2016.

[4] JI Yimu, ZHANG Dianchao, SUN Yanfei, et al. SQLS: A Storm-Based Query Language System for Real-Time Stream Data Analysis[J]. Chinese Journal of Electronics, Vol.25, No.6, Nov. 2016.

[5] ANTLR. ANTLR 官网[EB/OL]. http://www.antlr.org/,2017,06.

[6] Antlr. ANTLR4 GitHub 库[EB/OL].https://github.com/antlr/antlr4,2017,06.

[7] JGraphT. JGraphT GitHub 库[EB/OL].https://github.com/jgrapht/jgrapht,2017,06.

[8] JGraphT. JGraphT 官网[EB/OL]. http://jgrapht.org/,2017,06.