Distributed Data Management

akka-homework, Team null

Marcian Seeger & Justus Gillmann

DependencyMiner & DependencyWorker

- DependencyMiner: Verwaltung der zu bearbeitenden Vergleiche, Arbeitsauftrag an entsprechenden DependencyWorker (DependencyMiner.sendBatch())

- DependencyWorker: Vergleich zweier beliebiger Tabellen (auch zweimal selbe Tabelle möglich) auf INDs hin
  - dafür je möglicher Kombination der vorhandenen Spalten check, ob Werte eine Teilmenge der jeweils anderen Spalte darstellen

DependencyMiner.sendBatch()

```
private void sendBatch(ActorRef<DependencyWorker.Message> dependencyWorker) {
    //If nothing is left to compute we return
    if(firstJoinPartner >= batches.size() || secJoinPartner >= batches.size()){
        // Check if it was the last batch, then we can end our computation
        if(batchDeficit == 0)
            this.end();
        return;
    }
    BatchMessage batch1 = batches.get(firstJoinPartner);
    BatchMessage batch2 = batches.get(secJoinPartner);
    //increase second int
    secJoinPartner++;
    //if second int is already max, we increase first int and set the second to the first+1
    if(secJoinPartner >= batches.size()){
        firstJoinPartner++;
        secJoinPartner = firstJoinPartner+1;
        if(firstJoinPartner >= batches.size() || secJoinPartner >= batches.size())
            return;
    }
    batchDeficit++;
    this.getContext().getLog().info("Start working on task with " + batch1.getId() + " " + batch2.getId());
    //Send working batch to dependency worker, ids are needed for later
    dependencyWorker.tell(new DependencyWorker.TaskMessage(this.largeMessageProxy, batch1.getBatch(), batch2.getBatch(), batch1.getId(), batch2.getId()));
}
```

DependencyWorker.handle(TaskMessage)

```
private Behavior<Message> handle(TaskMessage message) {
    this.getContext().getLog().info("Working!");

    List<List<String>> batch1 = message.getBatchFirst();
    List<List<String>> batch2 = message.getBatchSec();

    int x = 0;
    int y = 0;
    List<TableDependency> dependencies = new ArrayList<>();
    for (List<String> col : batch1){
        y = 0;
        for (List<String> col2: batch2){
            if(col.containsAll(col2)){
                dependencies.add(new TableDependency(message.getTable2ID(), message.getTable1ID(), y, x));
            } else if(col2.containsAll(col)){
                dependencies.add(new TableDependency(message.getTable1ID(), message.getTable2ID(), x, y));
            }
            y++;
        }
        x++;
    }

    LargeMessageProxy.LargeMessage completionMessage = new DependencyMiner.CompletionMessage(this.getContext().getSelf(), dependencies);
    this.largeMessageProxy.tell(new LargeMessageProxy.SendMessage(completionMessage, message.getDependencyMinerLargeMessageProxy()));
    return this;
}
```

Endergebnis

- DependencyWorker: Rückgabe einer (modifizierten) CompletionMessage:

```java
@Getter
@NoArgsConstructor
@AllArgsConstructor
public static class CompletionMessage implements Message {
    private static final long serialVersionUID = -7642425159675583598L;
    ActorRef<DependencyWorker.Message> dependencyWorker;
    List<TableDependency> dependencies;
}
```

```java
@Getter
@AllArgsConstructor
public class TableDependency implements AkkaSerializable {
    int tableID1;
    int tableID2;
    int colID1;
    int colID2;
}
```

→ Verwaltung einer IND (von tableID1.colID1 nach tableID2.colID2)

- DependencyMiner: siehe nächste Folie

DependencyMiner.handle(CompletionMessage)

```
private Behavior<Message> handle(CompletionMessage message) {
    ActorRef<DependencyWorker.Message> dependencyWorker = message.getDependencyWorker();

    batchDeficit--;
    if (!message.getDependencies().isEmpty() && this.headerLines[0] != null) {
        for (TableDependency dep : message.getDependencies()) {
            this.getContext().getLog().info("Dependency found from " + this.inputFiles[dep.getTableID1()].getName() + " to " + this.inputFiles[dep.getTableID2()].getName());
            int dependent = dep.getTableID1();
            int referenced = dep.getTableID2();
            File dependentFile = this.inputFiles[dependent];
            File referencedFile = this.inputFiles[referenced];
            String[] dependentAttributes = {this.headerLines[dependent][dep.getColID1()]};
            String[] referencedAttributes = {this.headerLines[referenced][dep.getColID2()]};

            InclusionDependency ind = new InclusionDependency(dependentFile, dependentAttributes, referencedFile, referencedAttributes);
            List<InclusionDependency> inds = new ArrayList<>( initialCapacity: 1);
            inds.add(ind);

            this.resultCollector.tell(new ResultCollector.ResultMessage(inds));
        }
    }

    //Worker can try to start solving again if there are any more batches left
    sendBatch(dependencyWorker);

    if(isFinished())
        this.end();
    return this;
}
```