AQP实验报告

学院：信息学院 专业：计算机科学与技术
年级/班级：2022级图灵实验班　姓名： 杨东录
学号：2022201826

引言

探索式数据分析是指在数据分析过程中，通过探索和发现数据中的模式、关系和趋势，以获取对数据的深入理解。它在数据科学和业务决策中具有重要意义。

背景与意义：

1. 数据探索性分析可以帮助我们了解数据的基本特征、分布和异常情况，从而为后续的数据预处理和建模提供基础。
2. 通过探索性数据分析，我们可以发现数据中的隐藏模式和趋势，为业务决策提供新的见解和创新点。
3. 数据探索性分析有助于发现数据质量问题，如缺失值、异常值和不一致性，从而帮助数据清洗和数据质量改进　近似查询处理问题是探索式数据分析中的一个关键问题。在处理大规模数据集时，传统的精确查询可能会面临效率和可扩展性的挑战。为了解决这个问题，可以采用近似查询处理的方法。

近似查询处理的思想是在保证查询结果的一定准确性的同时，通过牺牲一部分精确性来提高查询的效率。

常用的方法：

1. 随机抽样：通过对数据集进行随机抽样，可以快速获取数据的近似统计信息。
2. 概率算法：利用概率算法进行数据采样和估计，可以在降低计算复杂度的同时，得到接近精确查询结果的近似解。
3. 数据压缩和摘要（直方图法）：通过对数据进行压缩和摘要，可以减小数据规模，从而提高查询效率。

在整体方案中，针对近似查询处理问题，我会考虑以下解决方案：

1. 概率算法：我采用直方图法，对数据进行压缩和摘要，以便快速估计数据的统计信息。
2. 数据索引和缓存：根据查询的特点，设计合适的索引结构和缓存机制，加速查询的响应时间。

在实施解决方案时，可能会遇到以下技术挑战：

1. 数据规模和性能：处理大规模数据集时，需要考虑存储和计算资源的限制，以及如何优化算法和数据结构以提高查询性能。

2. 数据采样和抽样：Off-line 提供的时间不足以查完全表，因此需要选择合适的采样方法和样本大小，以确保采样结果具有代表性，并能够提供有意义的分析结果。
3. 精确性和准确性的平衡：在近似查询处理中，需要在保证查询效率的同时，尽量减小查询结果的误差，并在可接受范围内控制近似度。

针对这些挑战，可以采取以下解决方案：

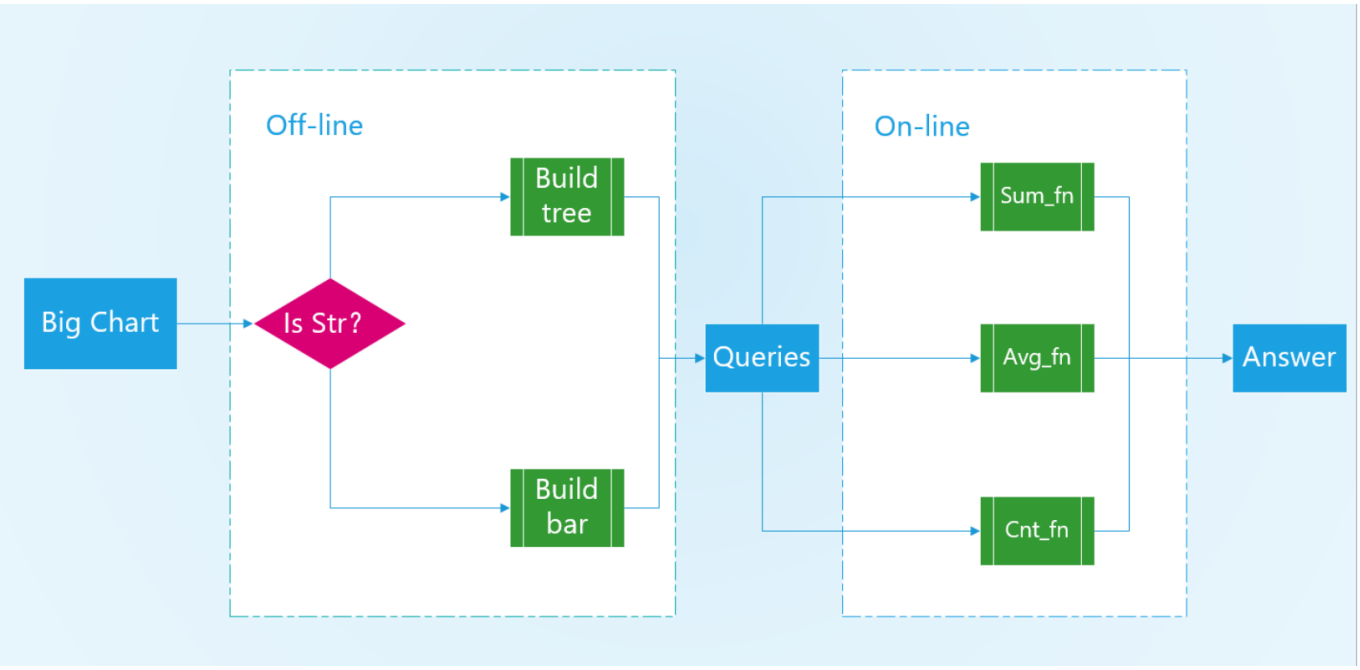
1. 实时监测运行时间：由于水平有限，我很难针对硬件进行相关的优化和处理，只能采用时间监测，当达到了Off-line所要求的上限时便强行终止维护。
2. 算法优化：对现有的近似查询方法进行优化，同时考虑数据的相关性的影响，减小误差并提高查询准确性。
3. 采样规模的自适应调整：最终确定将样本取样0.3做后续分析。 通过采用上述解决方案，可以在保证查询结果准确性的同时，提高探索式数据分析的效率和可扩展性，并从大规模数据集中获取有意义的洞察和发现。

整体方案

问题定义：

当我们面对一个较大的数据集S,此数据集中有若干列数据$S_i, i = 0, 1, 2.. .$, 我们从中选取一个子数据集D ,各列数据为D_i.对于每次请求Q_p,可以被拆分为若干约束Con_{pj},对于每个子类$Class_i$, 计算相关数据的Sum_p,Avg_p,Cnt_p.并使其与实际误差尽可能小.

整体框架：



模块设计：

- Off-line: 指在 aqp.py 中的 Offline 函数,在该函数中完成数字型数据的分桶和字母型数据的建树.
- On-line: 指在 aqp.py 中的 Online 函数,在该函数中我们利用 Offline 传输的桶和树的数据信息,实现O(1)时间的查表工作并根据约束返回结果.

- tree-building: 指对于文本分类数据,我们在树内部架构一个结构体存储所有的数据信息,然后将这个数据信息计算sum, avg, cnt等各方面的信息.
- bar-building: 指对数值型数据进行分桶
- sum-fn: 计算总和结果的函数
- avg-fn: 计算平均值的函数
- cnt-fn: 计算个数的函数

详细设计

定义: Off-line中建立桶定义为B_i ,建立的树定义为T_i .

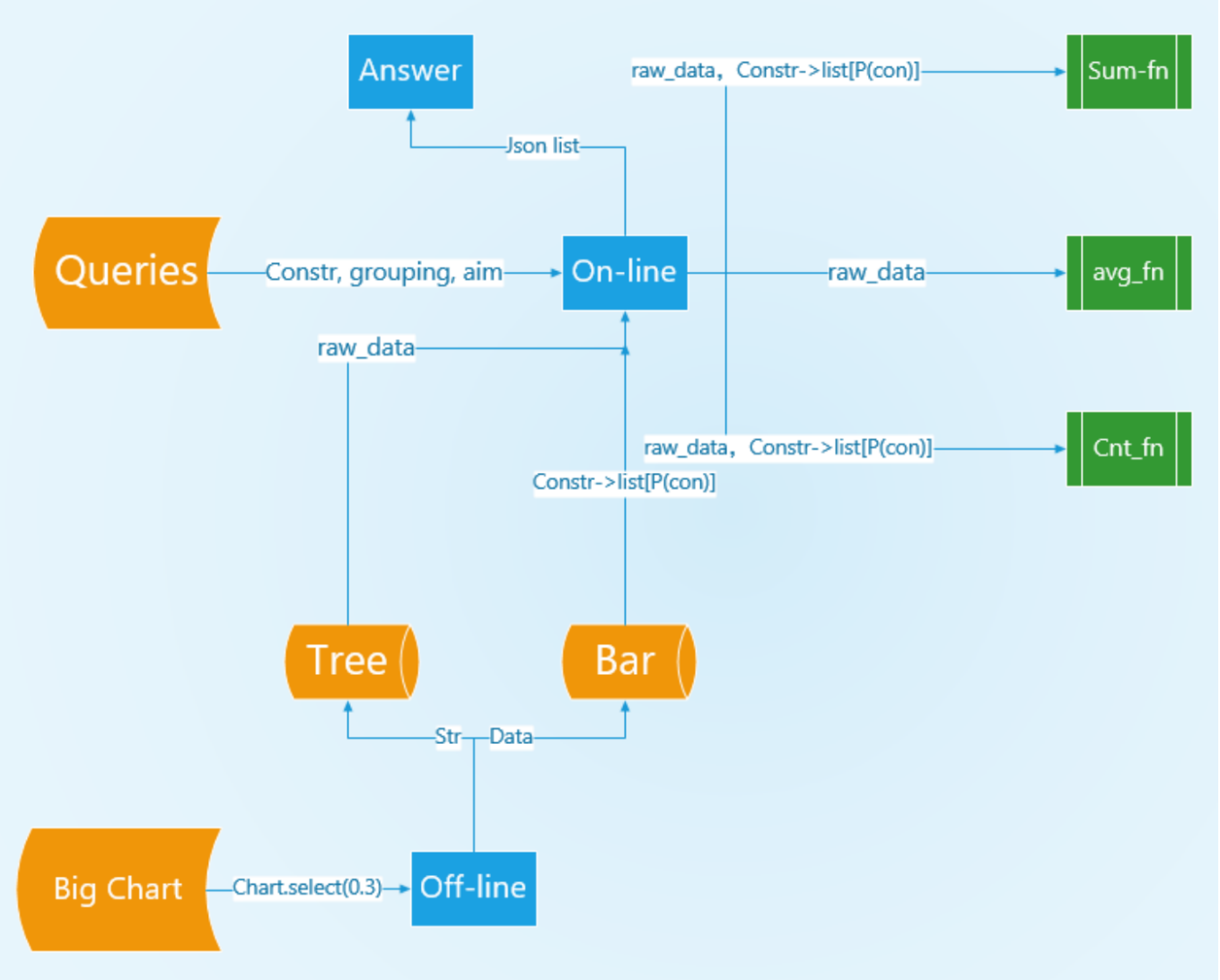
引理:若两列数据没有相关性,则两列数据的约束效果等于分别对结果产生约束结果之积.

证明: 对于请求Q_p, 假设其内部有约束条件$Con_{p_0}, Con_{p_1}, Con_{p_2}, \ldots, Con_{pn}$. 假设对每个数据对结果的约束在随机数据中被满足的概率为P_i

由于两列数据间没有相关性, 由概率的独立性,有
$P(Con_{pi} \bigcap Con_{pj})$=$P(Con_{pi}) \times P(Con_{pj}) = P_i \times P_j$

也即约束组合的作用效果等于分别作用后结果之积

流程图



伪代码

```
function name: sum_fn,avg_fn,cnt_fn
input: Constr->list[P(con)], raw_data->double,
output: Sum->double, avg->double, cnt->double
-----------------------------------------------------
sum_fn begin:
for every P in Constr:
        raw_data*=P
return raw_data
end.
avg_fn begin:
return raw_data
end.
cnt_fn begin:
for every P in Constr:
        raw_data*=P
```

```
return raw_data
end.----------------------------------------------------------------------------------------
--
----
function name: Off-line On-line
input: Chart, queries(Constr, grouping, aim)
output: answer
----------------------------------------------------------------------------------------
----
Off-line begin:
Chart = Chart.select(0.3)
for column in Chart.columns:
if column.dtype == str:
for row in column:
Tree[column][row].append
else:
devide column into k bar
end.
On-line begin:
for query in queries:
get Constr->list, grouping->Optional[column.name,None], aim->Optional["sum","avg","cnt"]
for constr in Constr:
translate constr into P[con]
new_Constr.append(P[con])
```

样例展示:

1. {"result_col": "count", "*", "predicate": [{"col": "DISTANCE", "lb": "None", "ub": 2181.644290475887}],"groupby": "None", "ground_truth": 963162}

- 根据Chart 建立树结构和分桶
- 将predicate翻译为概率: 由于目标是全集,就无需再调用树结构,直接在桶里查找该数据,找到该数据落在第十个桶里,在第十个桶里计算该结果为0.63,返回概率为0.963
- 将总数raw_data=1000000和0.963传入cnt_fn函数,得到结果为963000

1. {"result_col": "DEST"], ["sum", "DISTANCE", "predicate": [{"col": "TAXI_OUT", "lb": "None", "ub":595.6832653624532}], "groupby": "DEST"}

- 根据Chart建立树结构分桶.
- 将predicate翻译为概率:由于目标集为DEST,在树中查找在DEST的每个子类下DISTANCE的sum,avg,cnt等信息,此时raw_data=sum. 其余同上
- 分别将各个子类的raw_data和predicate传入sum_fn函数计算结果,由于结果很长,这里不重复了.

实验结果及分析

实验设置:

调用所给的实验数据集,使用评测所需的eval.ipynb函数.查询核心思路为直方图法,环境即A榜环境(数据来自A榜)。 详细数据为4核8G的运行效果。

整体实验结果

表 1 整体实验结果

查询负载	误差（MSLE）	在线时间	离线时间
Q1	0.1287	0.02576	185.5
Q2	0.00004914	0.01257	0.06202
Q3	0.3471	0.03034	0.0636
Q4	0.2533	1.5377	0.06398

实验结果表明：四次查询的时间均比较可观，而离线时间都聚集在Q1的数据结构的建构种。MSLE较大，且Q1出现超180s情况。

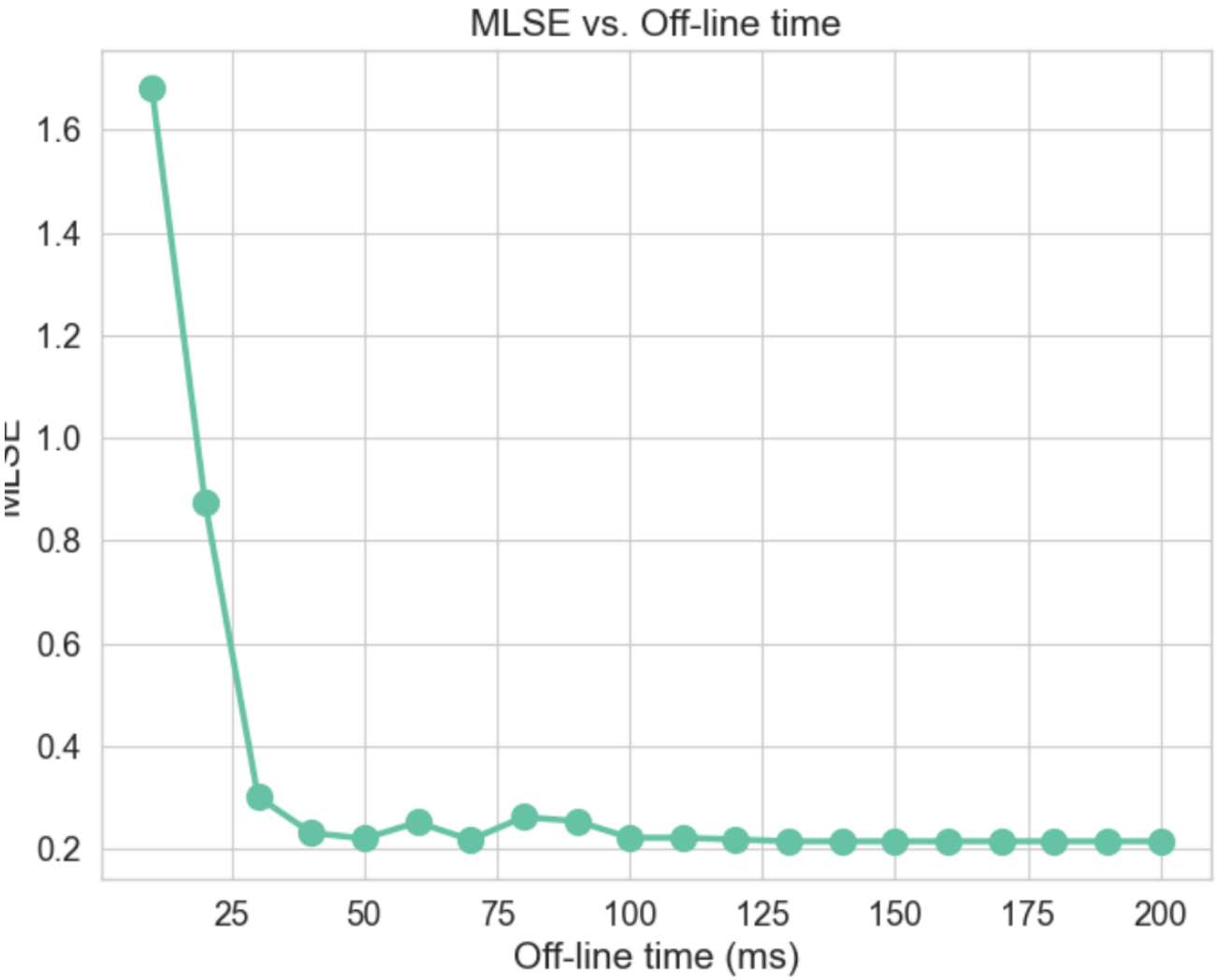
详细分析

Q4 下测得

分桶数量	误差（MSLE）	在线时间	离线时间
10	0.3652	0.08	47
20	0.2412	0.09	46
30	0.1936	0.12	43
40	0.2135	0.47	47
50	0.4146	1.2	46
100	1.2762	1.4	48
5	5.3672	0.08	46

Off-line截断时间	误差（MSLE）	在线时间	离线时间
20	0.8762	0.09	24
45	0.2312	0.09	47
180	0.2147	0.08	183
+∞	0.2147	0.11	192

离线时间-MSLE图线



总结与体会

1. 时间-准确度权衡：AQP技术可以提供在时间和准确度之间的灵活权衡。根据具体的查询需求和应用场景，我们可以选择适当的准确度要求来平衡查询性能和结果质量之间的关系。
2. 实时查询处理：AQP对于需要快速响应的实时查询非常有用。通过牺牲一定的准确度，AQP能够在极短的时间内返回结果，使得实时决策和交互式查询成为可能。

3. 数据量和复杂性：AQP在处理大规模数据和复杂查询时表现出色。对于大型数据集和复杂查询，传统的精确查询可能需要耗费大量的时间和计算资源，而AQP能够在合理的时间内提供近似结果，从而显著减少查询成本。
4. 应用限制：尽管AQP在某些场景下非常有效，但它并不适用于所有查询。某些查询可能对准确性要求非常高，不能容忍任何误差。此外，AQP的性能也取决于估计模型的选择和参数设置，需要仔细调整和优化。

总之，通过这个AQP实验，我深入了解了时间和准确度之间的权衡关系，并且意识到在不同的查询场景中选择适当的AQP方法可以提高查询效率。未来，我将继续探索AQP领域的研究，并寻找更多的优化策略和应用场景，以进一步提高查询性能和结果质量。