Rainbow: Adaptive Layout Optimization for Wide Tables

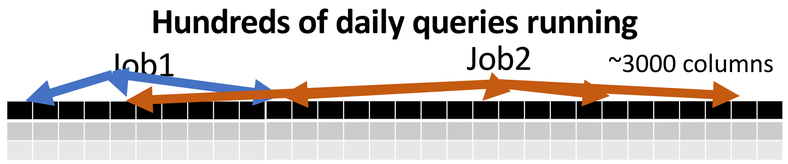
Haoqiong Bian, Youxian Tao, Guodong Jin, Yueguo Chen*, Xiongpai Qin, Xiaoyong Du
DEKE Key Lab, Renmin University of China
chenyueguo@ruc.edu.cn

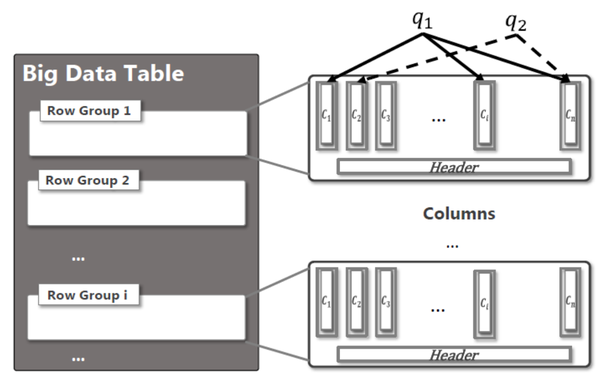
Motivation and Fundamental Techniques

Application of Wide Tables

In many web applications, data analysis jobs may require very wide tables, with thousands of columns that have the following advantages:
1) All the needed columns of a query are likely to be read from one single table to avoid the expensive join;
2) When new analysis requirement comes, new columns can be easily appended to an existing table without affecting the existing data analysis applications.

Wide tables are stored as a set of columnar format files (RCFile, ORC, Parquet…) on HDFS. The files are naturally partitioned into many row groups, and within each row group, data is stored in columnar wise.

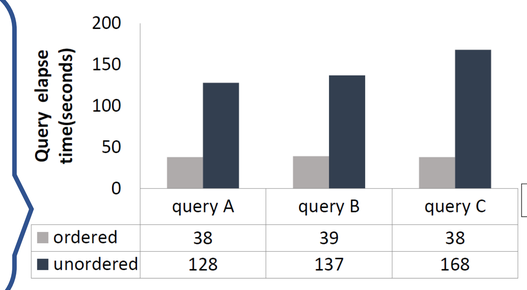




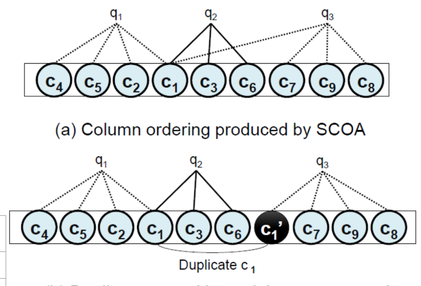
Layout Optimization Problem

An analytical query reads the required columns within each row group.
1000+ columns in a 64MB row group. <64KB/column.
Disk seeks take the majority of the I/O cost.

If required columns of the 3 queries are adjacent, performance can be **improved by up to 70%**



	query A	query B	query C
ordered	38	39	38
unordered	128	137	168



Seek Cost: Given two data objects *i* and *j*, the seek cost from *i* to *j* is denoted as $Cost(i,j)=f(dist(i,j))$, where *f* is the seek cost function which depends on the hardware. Different types of disks might be somewhat different.

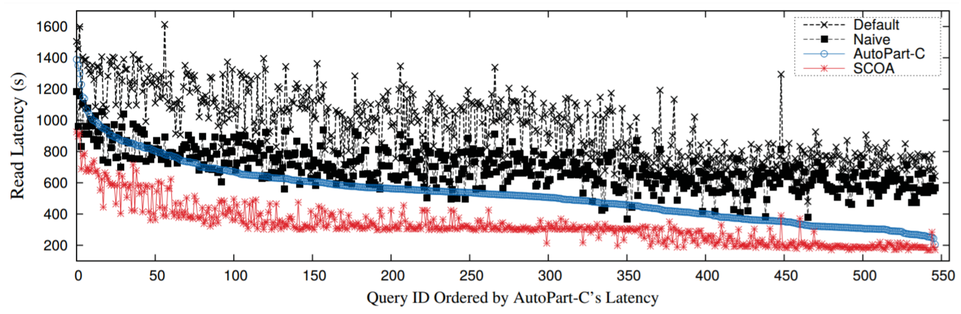
Column Order Strategy: Given a table with *n* columns, a column order strategy $S=\langle c_1,c_2,\ldots,c_n\rangle$ is an ordered sequence of those columns.

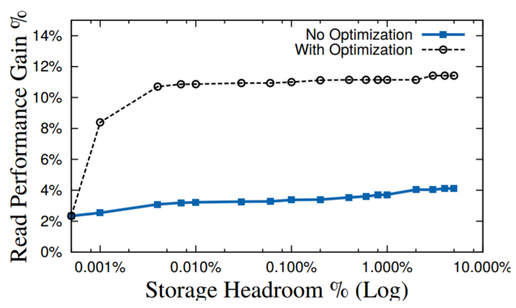
Column Ordering Problem: Given a workload *Q* containing a set of queries, finding an optimal column order strategy $S^*=\langle c_1,c_2,\ldots,c_n\rangle$, such that the overall seek cost of *Q* is minimized. ***NP-Hard → Simulated Annealing based Column Ordering Algorithm (SCOA)***

Column Duplication Problem: Given a workload *Q* and the storage headroom *H*, identify a set of duplicated columns with an ordering strategy S_D such that 1) the total size of duplicated columns is not greater than *H* and 2) the seek cost of *Q* is minimized.

Experimental Study

- SCOA shows up to 50% efficiency gain under real production data and workloads.
- With less than 5% storage headroom, we achieve another 12% gain using column duplication.
- SCOA has been implemented into Microsoft Bing log analysis pipeline.



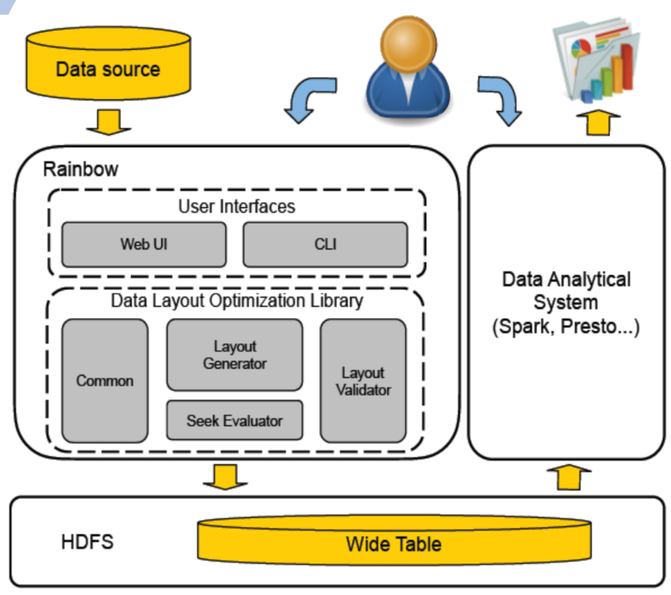


Reference: Wide Table Layout Optimization based on Column Ordering and Duplication. Bian et al. SIGMOD 2017.

How to Optimize Wide Table Layout when Workload Changes?

- Many analytic queries of different types are submitted to the system by many users of different departments.
- The requirements of these users keep changing, so that the whole query workload executed by the system keeps evolving.
- As the workload keeps evolving, we have to dynamically optimize the table layout according to the latest workload.
- It is nontrivial to perform such an optimization manually

System Overview

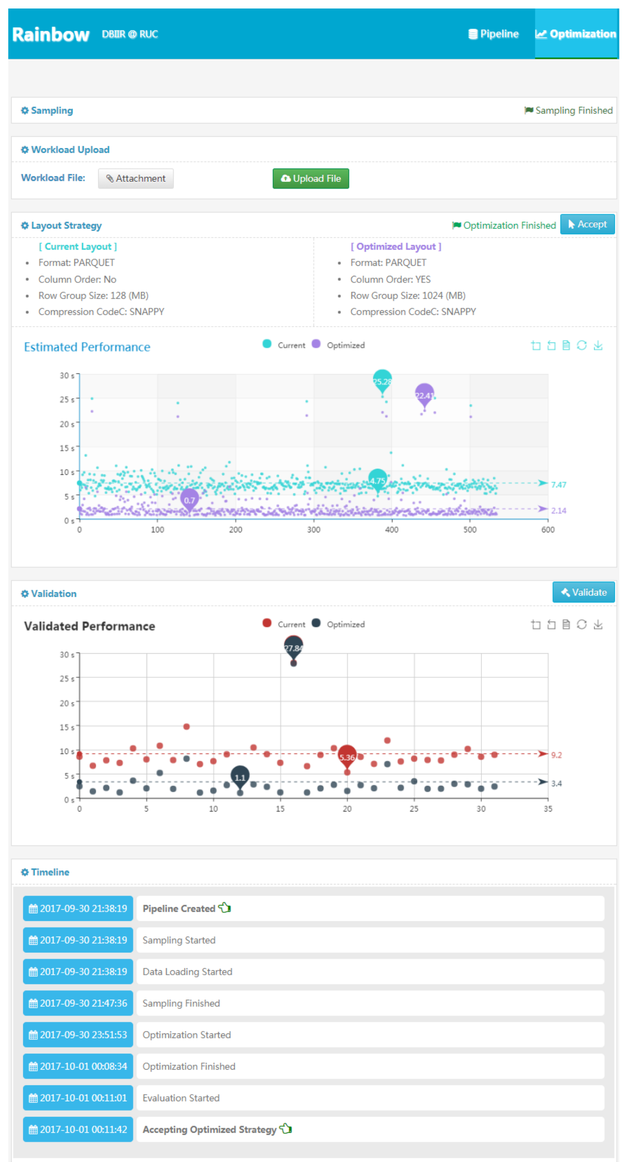


Our adaptive wide table layout optimization system is called Rainbow. It contains six modules (in gray):

- The Web UI module and the CLI (command line interface) module are user interfaces.
- The Common module contains shared functions and interfaces, and maintains system configurations and logs.
- Layout Generator implements layout optimization algorithms.
- Layout Validator evaluates the performance gain of a data layout.
- Seek Evaluator is responsible for deriving the seek cost function according to the underlying hardware and file system.

Rainbow collects the accessed columns of queries without issuing the queries for users. This allows Rainbow to work as an external module without modifying the existing data analysis systems.

Demonstration



Demonstration Video:
https://www.youtube.com/embed/6qaJBPZiHSA

Implementation and Key Techniques

Algorithm 1: AccessPatternCaching

Input: The access pattern $AP=\{c_{q,1},c_{q,2},\ldots,c_{q,m}\}$ of a query q

```
t := current time;
if APC.hits(AP) then
    APC.updateTimestamp(AP, t);
    APC.incrementWeight(AP);
else
    APC.insert(AP, t);
    updateCounter++;
    AP_e := APC.earliestAP();
    if AP_e.timestamp < t − L then
        if prevSize == 0 then
            updateCounter = 0;
            prevSize = APC.size;
            trigger layout optimization and return;
        APC.evict(AP_e);
        updateCounter++;
if prevSize > 0 and updateCounter/prevSize > θ then
    updateCounter = 0;
    prevSize = APC.size;
    trigger layout optimization;
```

Workload evolution

Layout is optimized according to access patterns(APs) of queries. A cache of APs is maintained by a LRU-based policy called *AccessPatternCaching*.

L: lifetime set by user to evict the outdated AP.

θ: user-defined updating threshold to trigger a layout optimization.

Column ordering optimization

Each time a layout optimization is triggered, SCOA is applied to optimize column ordering, according to current APs in the cache.

Row group size optimization

$$Cost(q)=N\times(\theta+SeqRead(q)+Seek(q))$$

Row group size (RGS) affects the query performance in two aspects:

- Larger RGS → larger sequential readings → lower seek cost.
- Larger RGS → less row groups → less constant overheads.

Open Source on GitHub :
https://github.com/dbiir/rainbow