***Review : H2O: A Hands-free Adaptive Store***

***Problem:***

* Databases are designed off one fixed decision
* But same choice is not suitable for all workloads and requirements
* The way data is stored - defines the maximum performance as it defines how much data must be accessed for query
  + Traditional row database - store data row wise (more suited for transaction (OLTP) based application)
  + Modern column store - stores column at a time(optimal for analytical queries – OLAP)
  + Neither of these are alone a good solution for all use cases.

***Why the problem hard:***

* Defining all possible permutations and combinations of the layout and choosing the most optimal solution is NP hard
* Its not practically feasible to do this for all tables with many attributes.
* Proper heuristics techniques need to be applied to prune/limit the search space without impacting the quality.

***why existing solutions do not work:***

* Database vendors - provide different engines in same software ( MySql has - MyISAM, InnoDB)
* commuincation between different data engines/parts is not possible
* Each engine has knowledge to know how best to access data **only** in its engine

***Main idea:***

* Flexible storage layout - Design dynamically chosen based on the data /queries
* Data layouts combined with proper execution strategies designed to scale up to millions of rows across hundreds to datacenters and trillions of database rows
* H20 is an adaptive Hybrid System - that (re)defines the layout dynamically based on the workload

***Design/*** ***Architecture :***

Main components

* DataLayout Manager: responsible for maintaining and creating different layouts
* Query Processor: evaluates alternate processes plans for the query based on the layouts
* Operator Generator: generates query operators based on the chosen layout and plan
* Adaptation mechanism - periodically invoked to evaluate layouts and propose new ones to Layout manager

*Supported Layouts* - 3 types

* Row-major - tuple sequentially in pages
* Column-major - individual columns
* Grouped-columns - Combined a group of attributes of the original relation.

*Continuous Adaptation:*

* Triggered by evaluating statistics of query derived from query history.
* Evaluates the cost/benefit from a new layout and selects the most fitting solution

*Monitoring:*

* Uses an “adaptive window” of N-queries to monitor the access patterns.
* Window size defines how aggressive/conservative the adaptive strategy will be.
* It is the number of queries in history that will trigger an adaptation mechanism
* Looks at the attributes queried and the frequency of attributes accessed together

***Solution/Algorithm:***

* Starts with the attributes accessed by queries to generate potential layouts
* Initial configuration contains the narrowest possible groups of columns
* The algorithm then progressively improves the proposed solution by considering the new groups of columns
* Generation and selection continued until no further selection is possible
* The cost of transforming to the new layout is weighed against the benefit - to ensure the cost can be amortized.
* Combines data reorganization with query processing in order to reduce the time query has to wait before starting to process.
* Early materialization - helps to generate layout + query -> no cost of scanning twice
* Operators generated dynamically and compiled to machine level on the fly.

***Some Next steps:***

* Evaluate the adaptive technique for a “update” operation – to see if we can design systems that can adapt to various workloads of update – where sometime update is applied to one column, while in other cases update is applied to a few columns.
* Can this technique be applied to the first query without a negative impact on performance