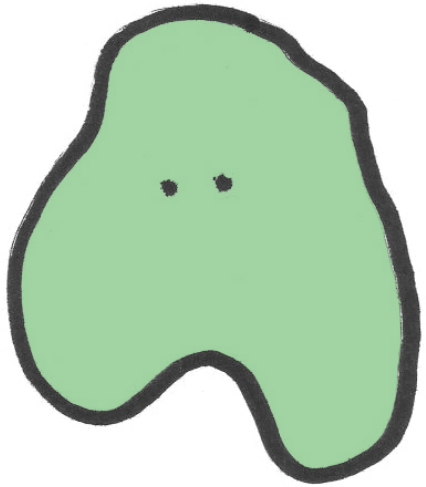


CSAIL

A Robust Partitioning Scheme for Ad-Hoc Query Workloads

ANIL SHANBHAG
MIT

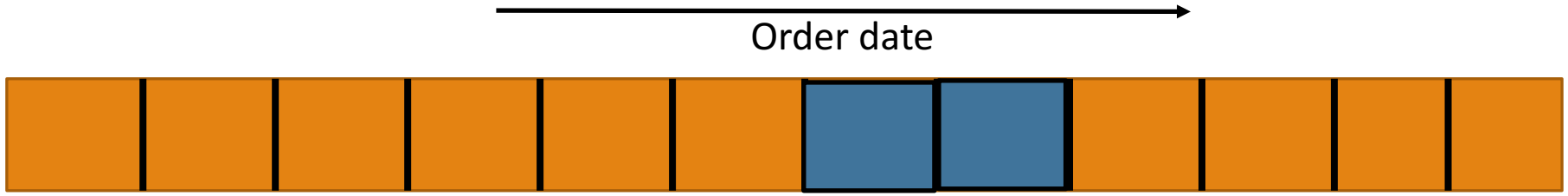
J/W Alekh Jindal, Microsoft
Sam Madden, MIT
Jorge Quiane, QCRI
Aaron J. Elmore, Univ. Chicago

Today

Data collection is cheap => Lots of data !

Data Partitioning

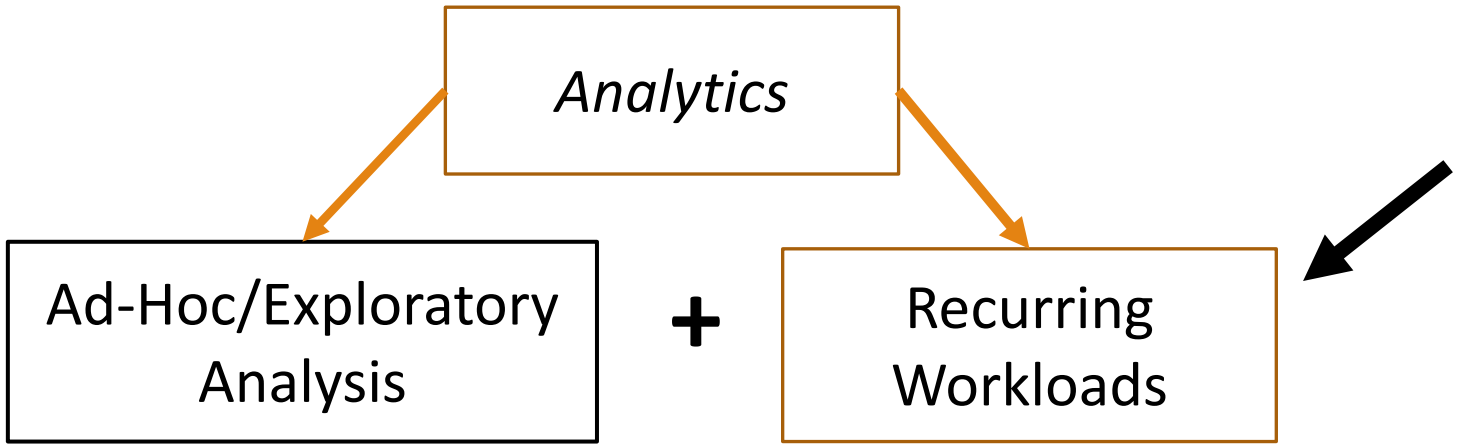
Find average order size for all orders between Sept 10 and Sept 11, 2017



Data Skipping - Skip data blocks not necessary

10% selectivity query => 10x faster if data partitioned on selection predicate

The Problem



Focus of existing work

Give workload =>
Return partitioning layout

Problems:
1. Tedious to collect workload
2. May not be known upfront
3. Changes over time

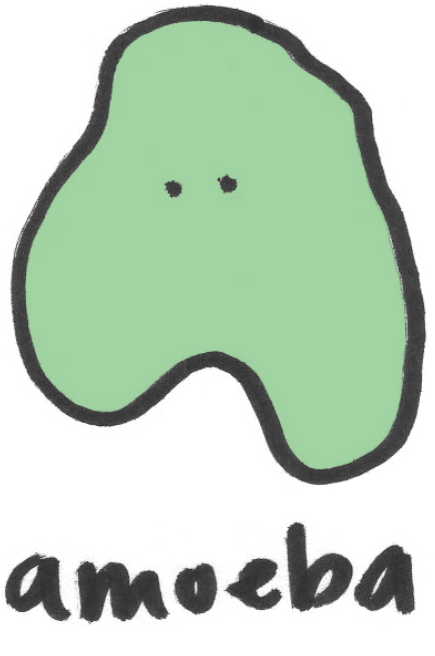
How to get benefits of partitioning in this case ?

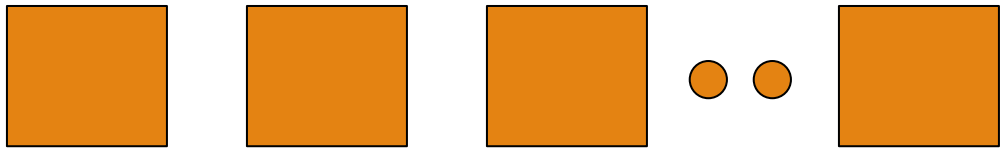
Our Approach

Do everything adaptively !

Two step process:

1. Upfront load the dataset partitioned
2. As users query, *incrementally* improve the partitioning of the data

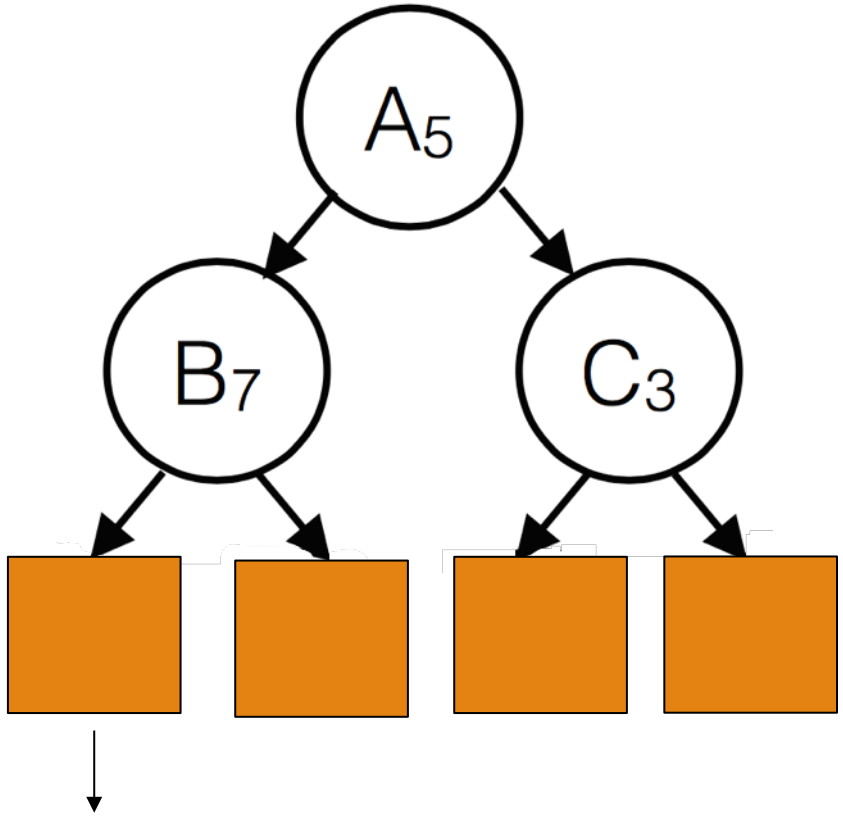


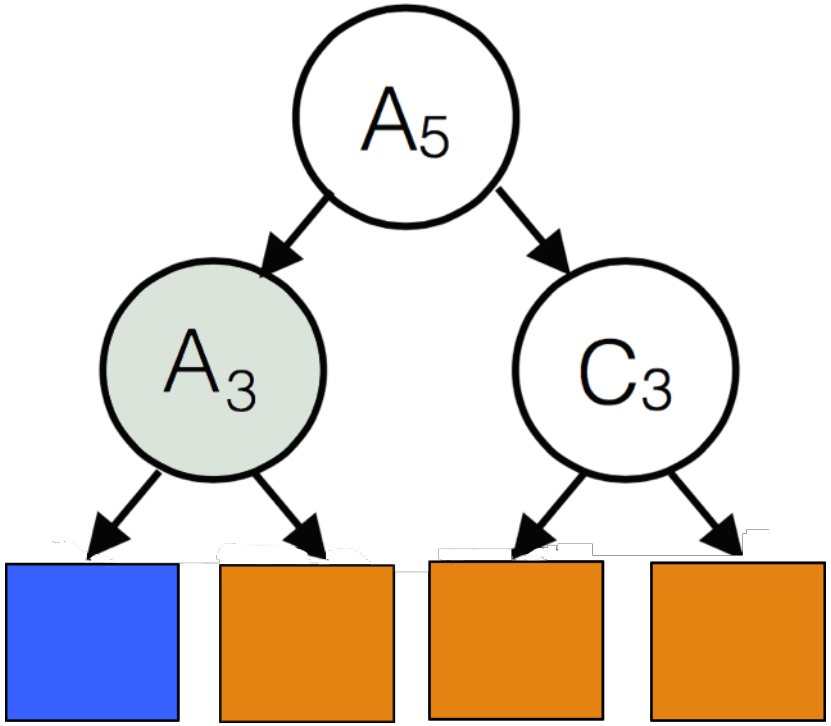


Distributed storage systems like HDFS, files broken into blocks (128 MB chunks)

Upfront Partitioning

> Instead of partitioning by size, partition by attributes.
> Same number of blocks created as in HDFS. Each block now has additional metadata





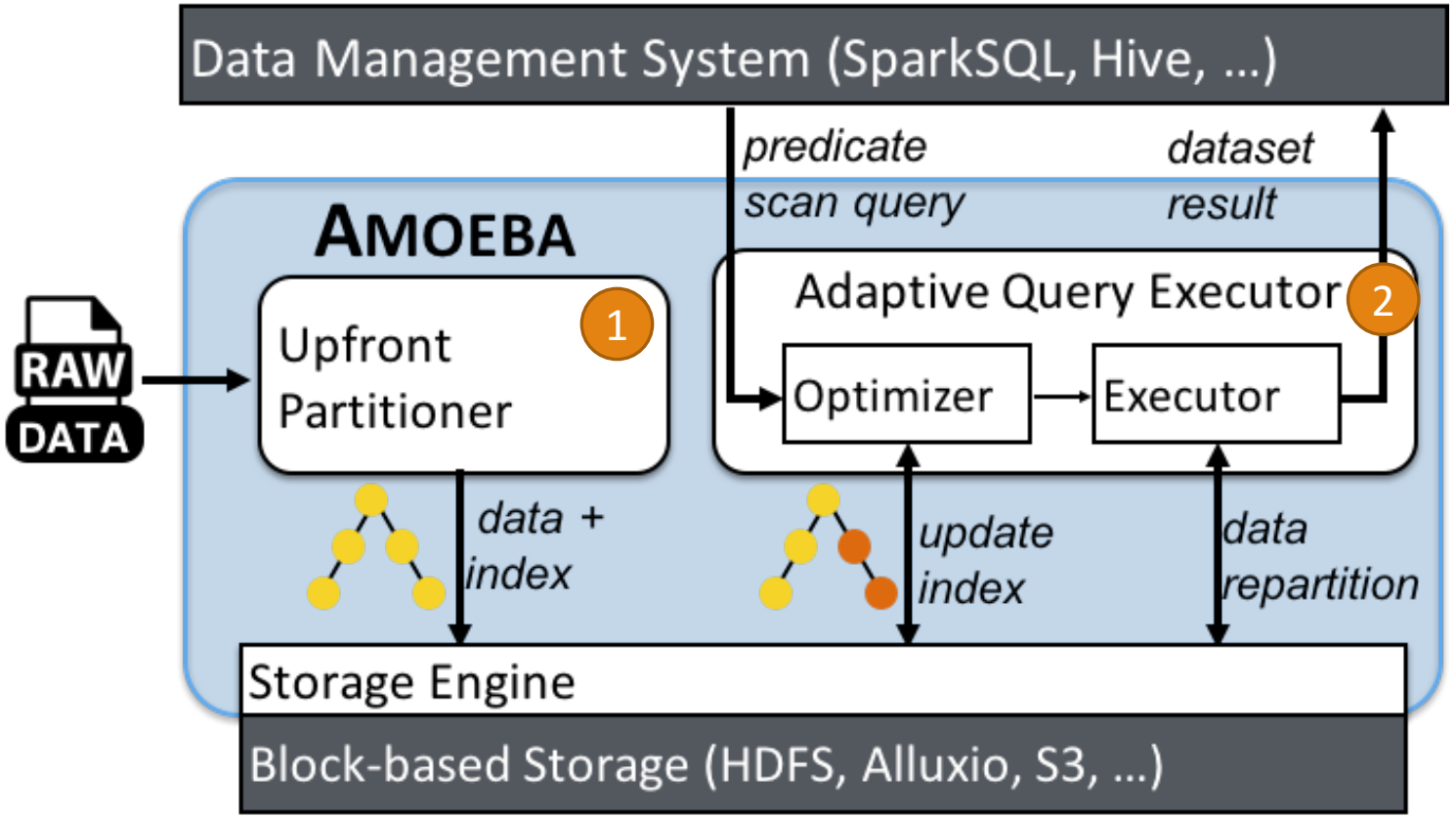
Adaptive Re-Partitioning

When user submits a query, optimizer tries to improve the partitioning by reorganizing the partitioning tree

Here if queries ask A <= 3 many times, replace B_7 by A_3

Done on datasets which are O(1TB) with ~ 8000 node partition trees.

System Architecture



Predicated Scan Query Example:

FIND employees WITH
Age < 30 AND
20k < Salary < 40k

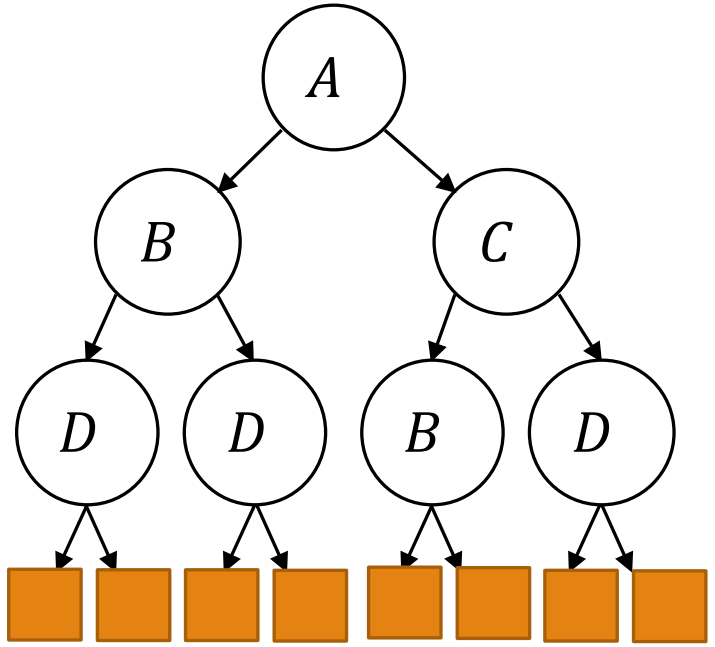
1. Upfront Partitioner

Goal: Generate a partitioning tree

WITHOUT an upfront query workload

> Generates a tree with heterogeneous branching

> Balance the partitioning benefit across all attributes

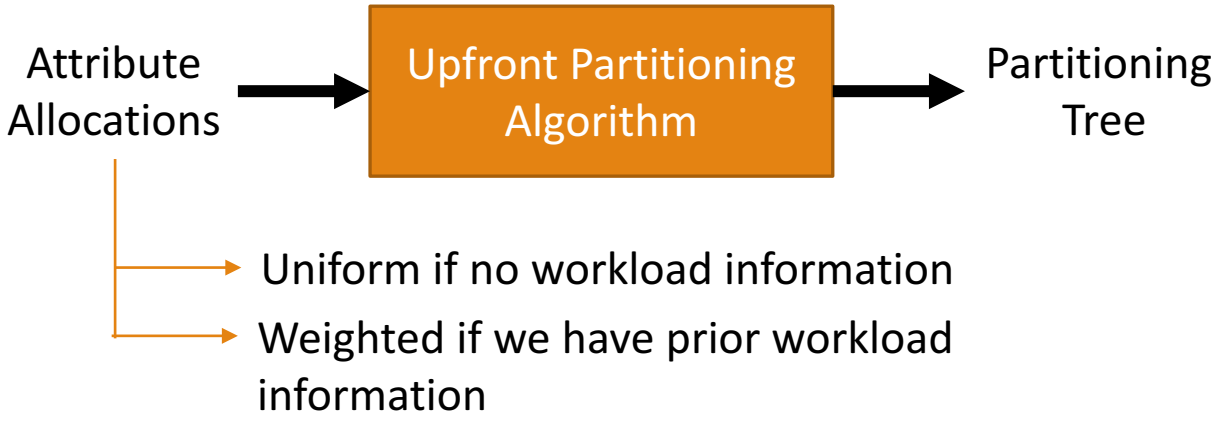


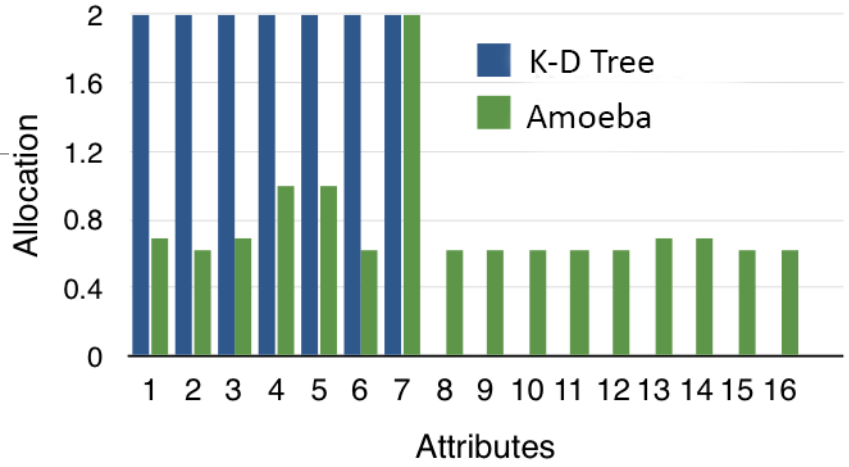
Allocation

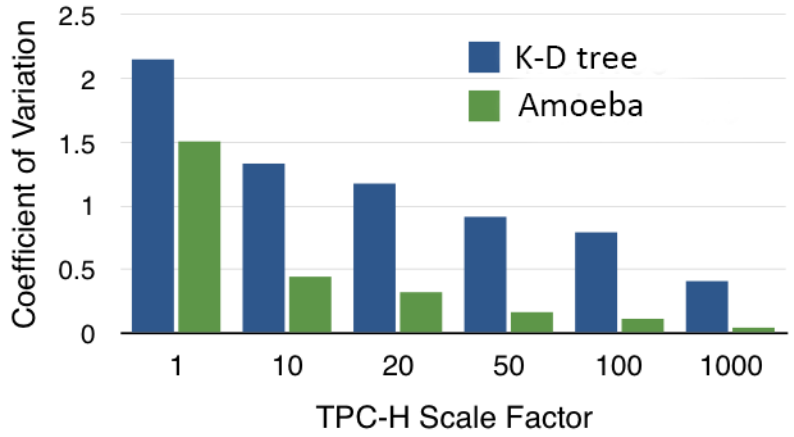
Goal: Balance partitioning benefit across attributes

Allocation of attribute i ~ average partitioning of an attribute j

$$= \Sigma_{\text{all nodes i}} \; n_{ij} \; c_{ij}$$





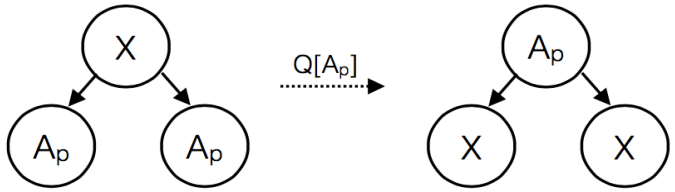


2. Adaptive Query Executor

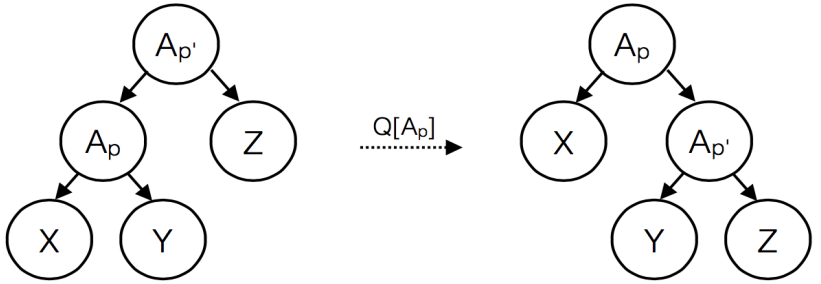
Goal: Return matching tuples + <u>check if partitioning layout can be improved</u>

Alternatives found via transformations on the partitioning tree

1. Swap Rule



2. Pushup Rule



3. Rotate Rule

Getting a plan

Algorithm 2: getSubtreePlan

```
   Input  :Node node, Predicate pred
1  if isLeaf(node) then
2      return P_none(node);
3  else
4      if isLeftAccessed(node) then
5          leftPlan ← getSubtreePlan(node.lChild, pred);
6      if isRightAccessed(node) then
7          rightPlan ← getSubtreePlan(node.rChild, pred)
       /* consider swap                                   */
8      if leftPlan.fullyAccessed and rightPlan.fullyAccessed then
9          currentCost ← ∑_i Cost(node,q_i);
10         whatIfNode ← clone(node);
11         whatIfNode.predicate ← newPred;
12         swapNode(node, whatIfNode);
13         newCost ← ∑_i Cost(whatIfNode,q_i);
14         benefit = currentCost - newCost;
15         if benefit > 0 then
16             updatePlanIfBetter(node, P_swap(node,whatIfNode));
       /* consider pushup                                 */
17     if leftPlan.ptop and rightPlan.ptop then
18         updatePlanIfBetter(node, P_pushup(node, node.lChild, node.rChild));
       /* consider rotate                                 */
19     if node.attribute == predicate.attribute then
20         if leftPlan.ptop then
21             updatePlanIfBetter(node, P_rotate(node, node.lChild));
22         if rightPlan.ptop then
23             updatePlanIfBetter(node, P_rotate(node, node.rChild));
       /* consider doing nothing                          */
24     updatePlanIfBetter(node, P_none(node));
25     return node.plan;
```

Algorithm 3: getBestPlan

```
   Input  :Tree tree, Predicate[] predicates
1  while predicates ≠ ϕ do
2      prevPlan ← tree.plan;
3      foreach p in predicates do
4          Plan newPlan ← getSubtreePlan(tree.root, p);
5          updatePlan(tree.root, newPlan, p);
6      if tree.plan ≠ prevPlan then
7          remove from predicates the newly inserted predicate;
8      else
9          break;
10 if tree.plan.benefit > tree.plan.cost then
11     return tree.plan;
12 else
13     return null;
```

Cost Model

The system maintain a window W of past queries

Compute Benefit and Repartitioning Cost for the best plan

Repartitioning **ONLY** happens when reduction in the total cost of the query workload is greater than re-partitioning cost.

Solves constant re-partitioning due to random query sequences and bounds the worse case impact.

$$\text{Cost}(T, q) = \sum_{b \in \text{lookup}(T,q)} n_b$$

$$\text{Benefit}(T') = \sum_{q \in W} Cost(T, q) - \sum_{q \in W} Cost(T', q)$$

$$\text{RepartitioningCost}(T, q) = \sum_{b \in B} c \cdot n_b$$

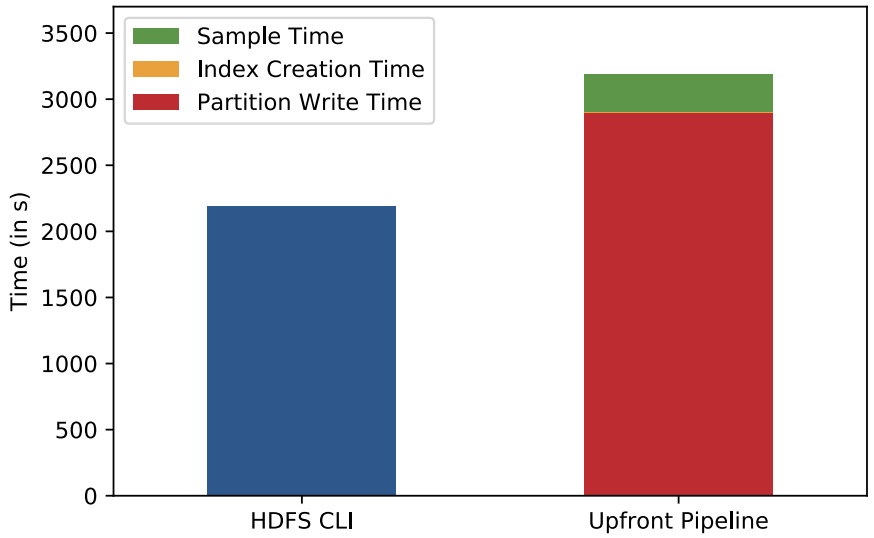
Performance

4 metrics

1) Load time
2) Time taken by first query
3) Aggregate runtime over a workload
4) Incremental improvement with workload hints

Load Time

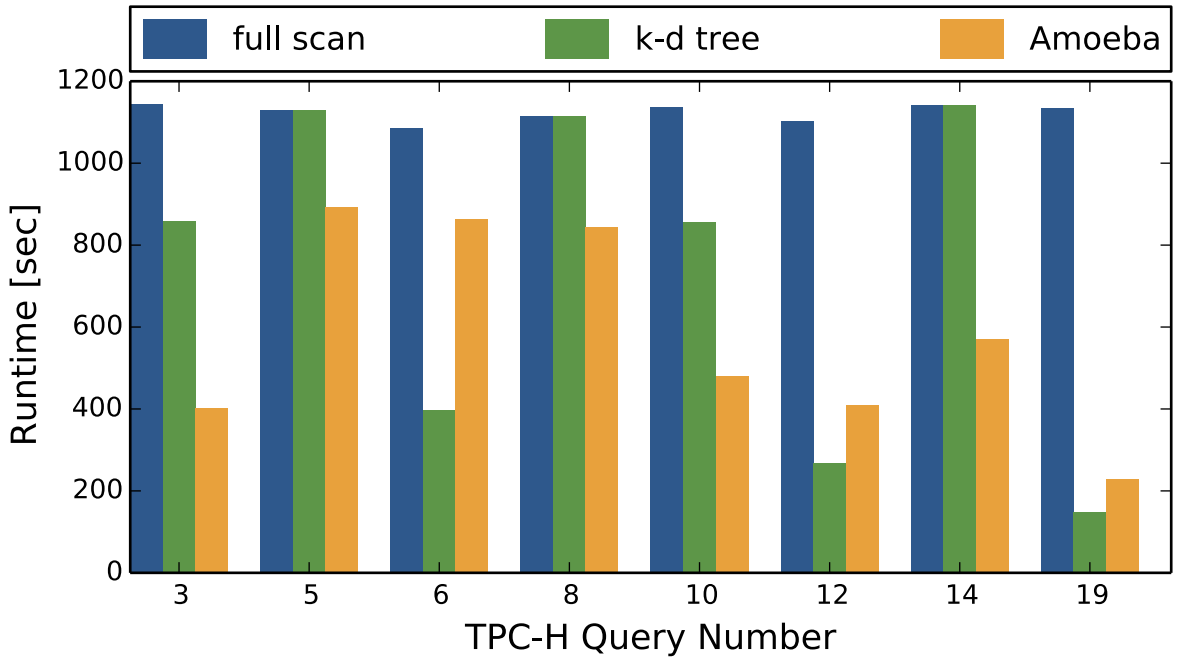
TPC-H: Scale Factor 200 + De-normalized. Data size:**1.4TB**



Loading performance: 1.38 times slower than HDFS

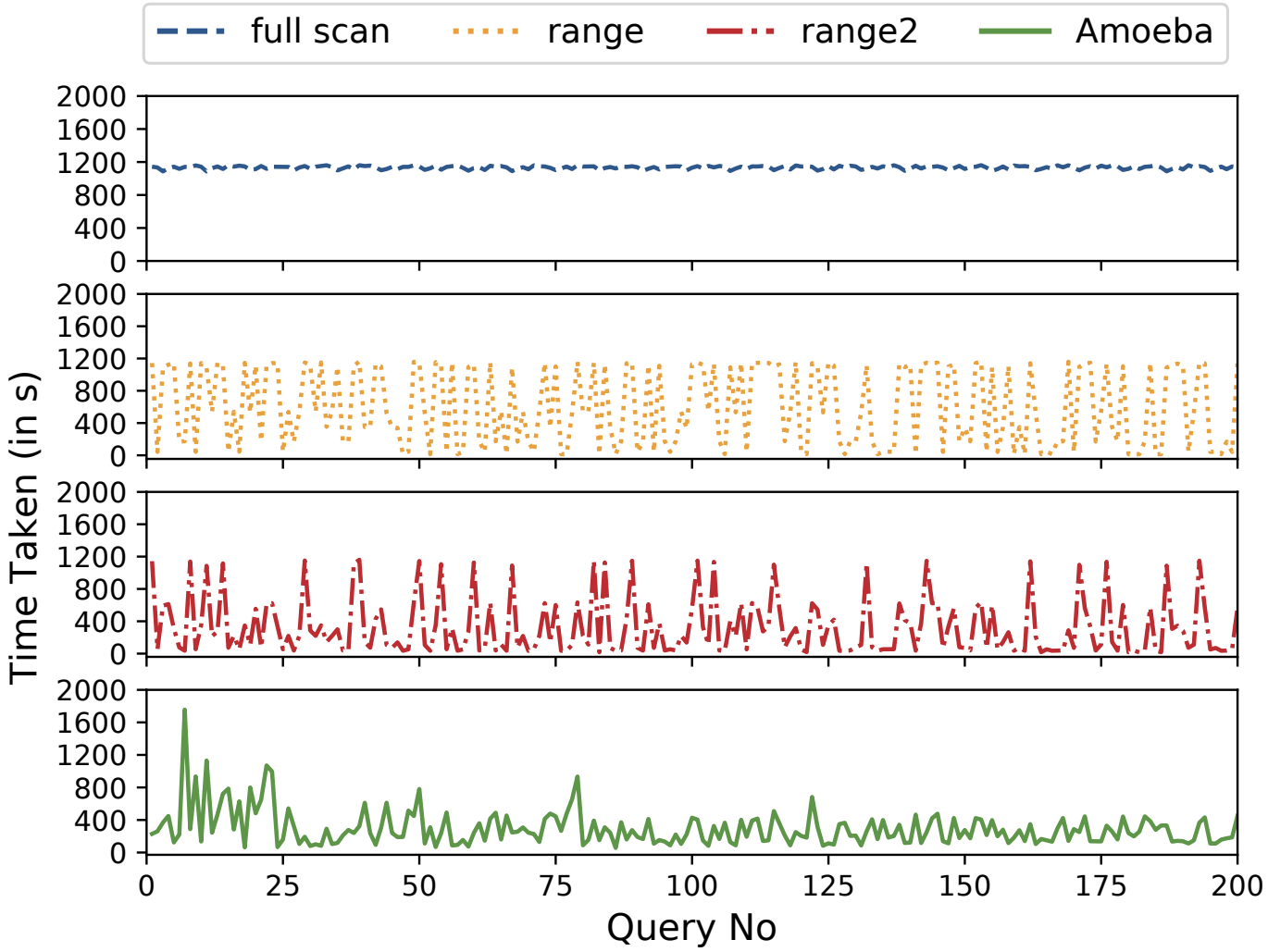
Load time scales almost linearly
with data size and independent of number of columns

Time taken by first query



On Average: 45% better than full scan
20% better than k-d tree

Aggregate Workload Runtime



Workload: 200 Queries generated from random initialization of 8 query templates of TPC-H benchmark

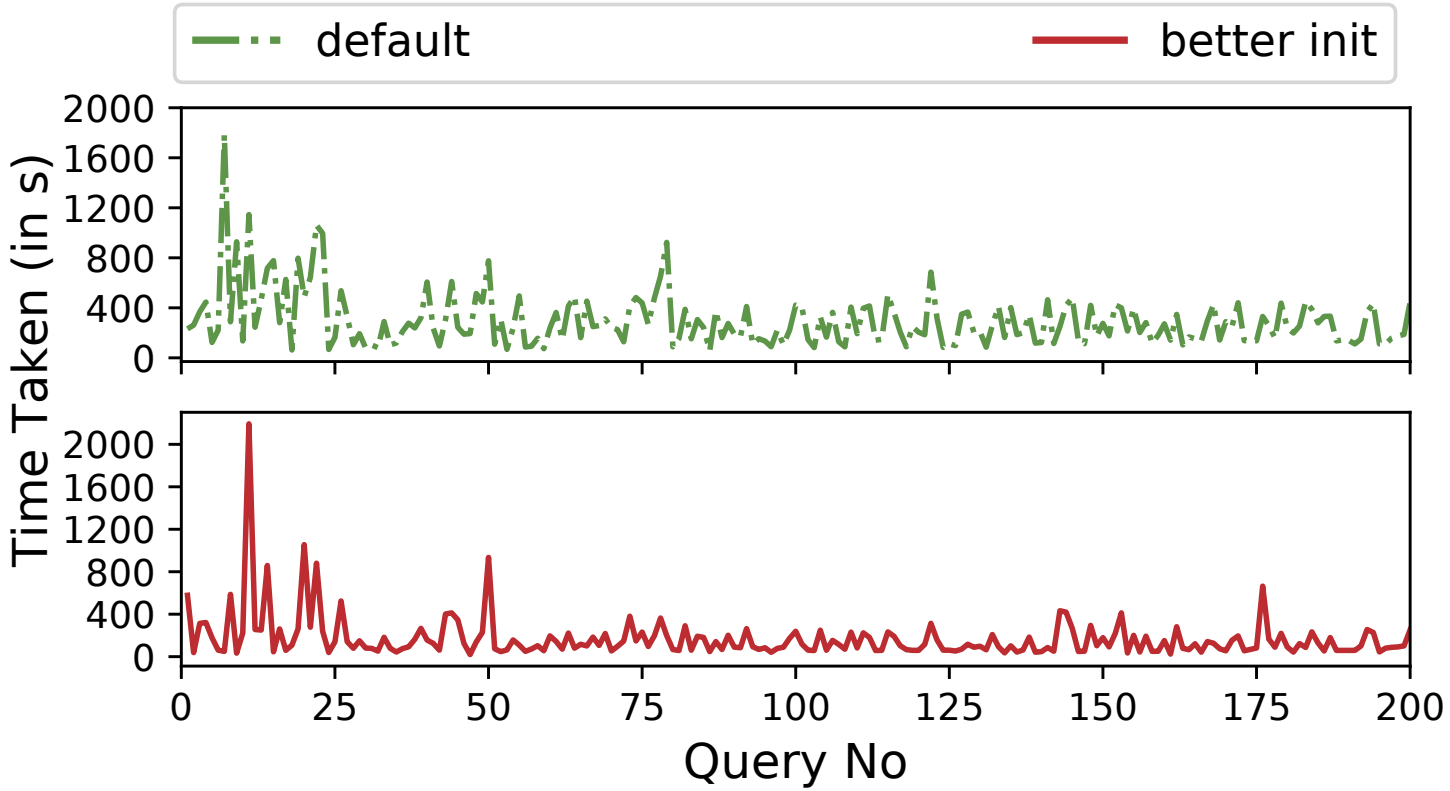
full scan – Baseline

range – partitions on orderdate (1 per date)
1.88x better

range2 – partitions on orderdate(64), r_name(4),c_mktsegment(4),quantity(8)
3.48x better

Amoeba – 3.84x better than baseline

Workload Hints



Better Init:
Starts with custom allocation to mimic *range2*

6.67x better than *fullscan*

Filtering ratio:
default : 0.81
better init : 0.9

Conclusion

- Amoeba is a distributed storage system based on an adaptive data partitioning scheme
  - Low loading overhead
  - Improved first query performance
  - Adapt to changes and significantly improvement to workload runtime
  - Can exploit workload hints
- Allows analysts to get started right away and reap benefits of partitioning without an upfront workload