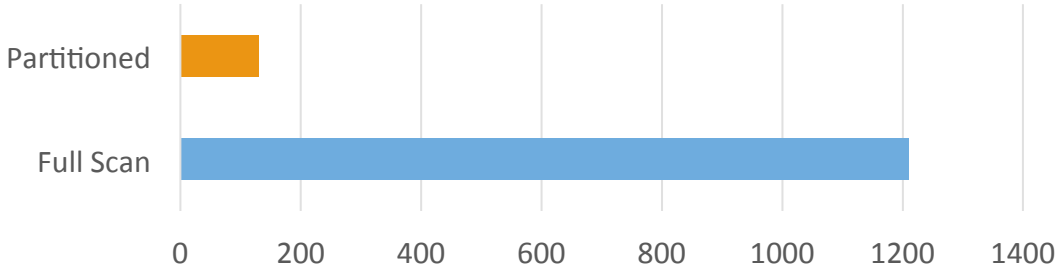
Adaptive Partitioning
For Distributed Query Processing

ANIL SHANBHAG

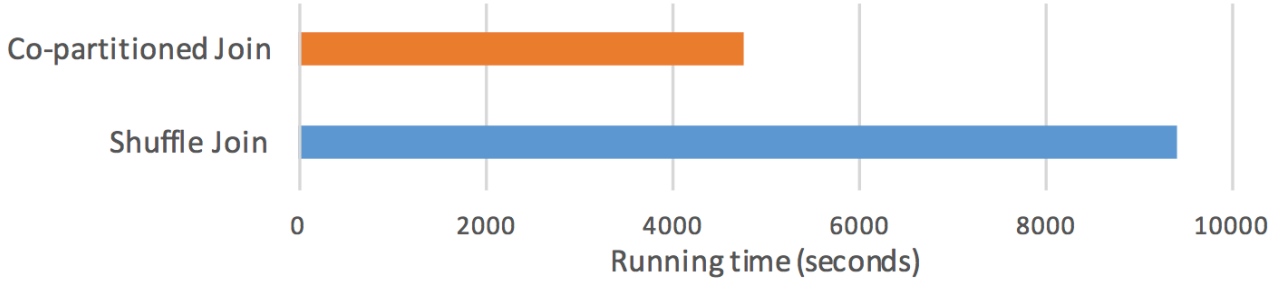
JOINT WORK WITH YI LU, ALEKH JINDAL, SAM MADDEN AND OTHERS

Partitioning is Awesome !



10% selection query is almost **10x** faster when run on fully partitioned data

For lineitem ⋈ order
Co-Partitioned Join almost **2x** faster than shuffle join



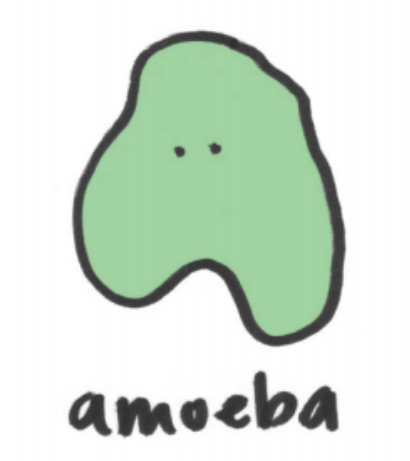
Run on 10 node cluster with Spark 1.6 with TPCH SF 1000

Ad-hoc / Exploratory Analysis

- Set of predicates/tables of interest shift over time
- Hard to get a representative query workload
- Tedious to collect a workload before they can even ask their first query
- Eg: On a real workload, after seeing 80% of the workload – the last 20% contained 57% new queries

Q : How to get benefits of partitioning ?

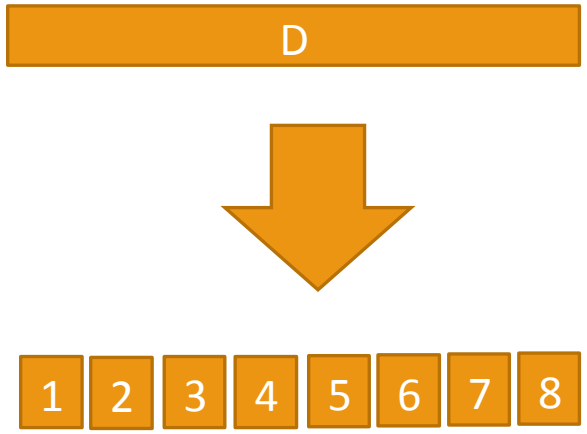
Enter Adaptive Partitioning



+ Adapt.

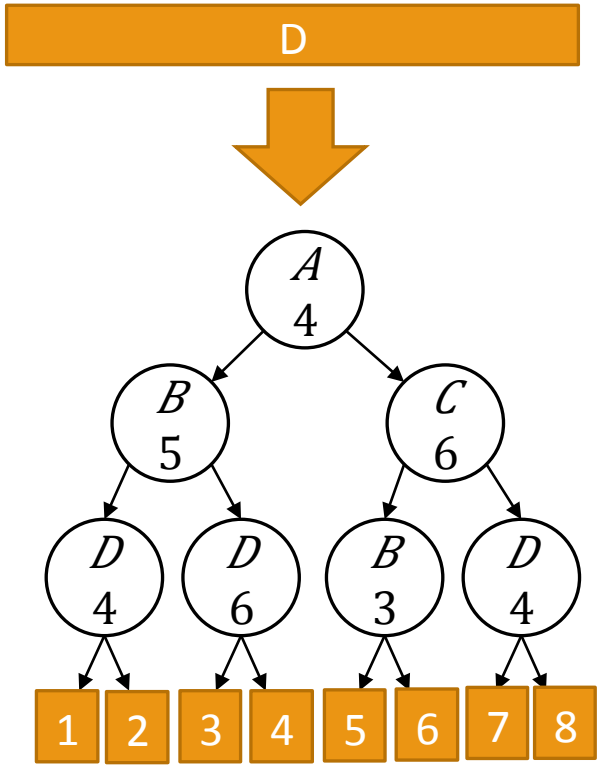
- Starts out with no workload information
- Incrementally improves the partitioning based on upfront queries and a cost model

Key Idea



Default Behavior In HDFS

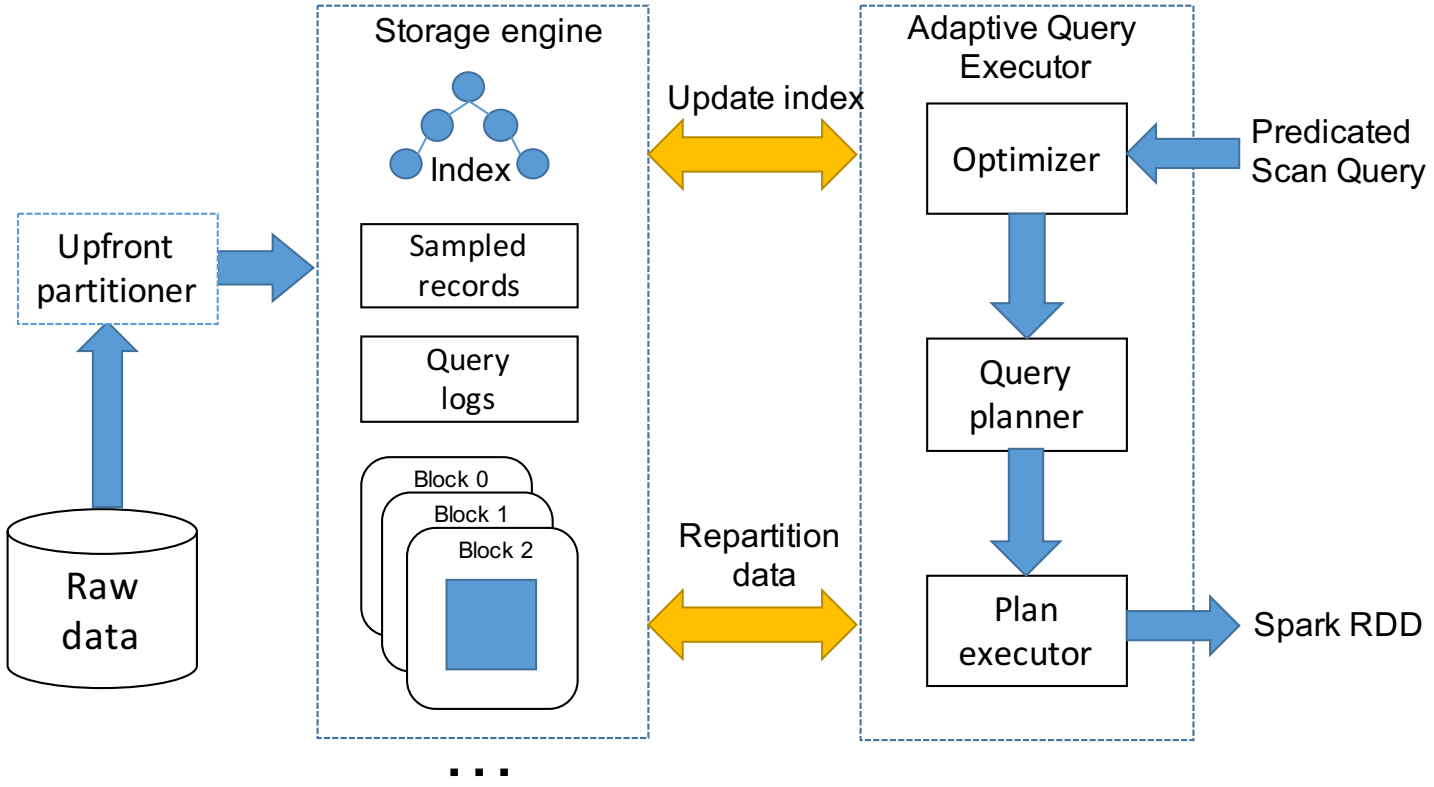
Create blocks of fixed size and replicate it r times across the cluster



In Amoeba build a partitioning tree

- Same number of blocks as in HDFS
- Each block has additional metadata
- Can be adapted incrementally

System Architecture



Upfront Partitioner

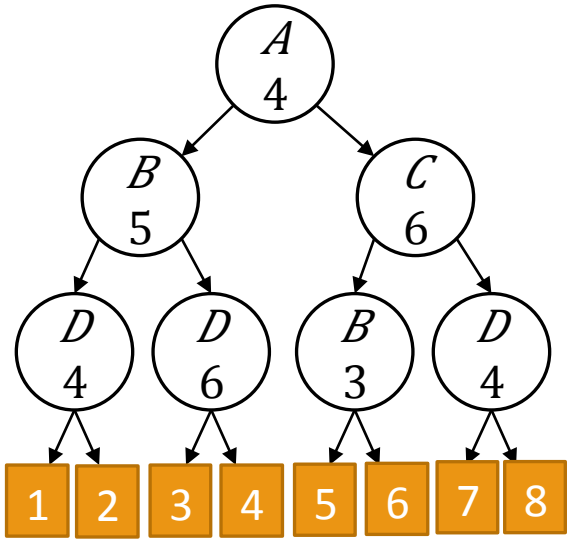
Starts with no workload information and creates partitioning tree

Allocation(j) (average partitioning of an attribute j) = $\Sigma\ n_{ij}\ c_{ij}$

- n_{ij} number of ways node i partitions on attribute j
- c_{ij} fraction of the data this applies to

Heterogeneous branching to ensure each attribute has almost same allocation

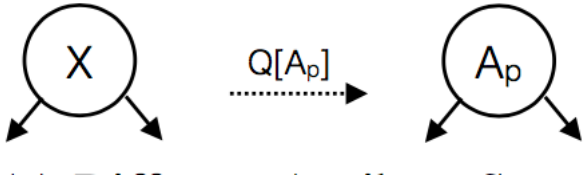
Can incorporate prior workload information to create better partitioning tree



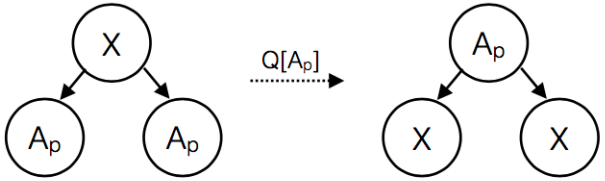
Adaptive Query Executor

For query with predicate A <= p, find the best tree with A_p in it

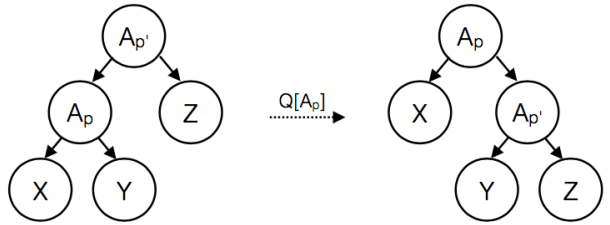
3 Transformation Rules to explore alternative partitioning trees



Swap



Push Up



Rotate

Naively applying rules will generate exponential choices

Explore choices in a bottom-up fashion to get the optimal tree

Cost Model

Given a good alternative tree => Is it worthwhile re-partitioning ?

Maintain a history of queries seen as a Query Window

Cost Model

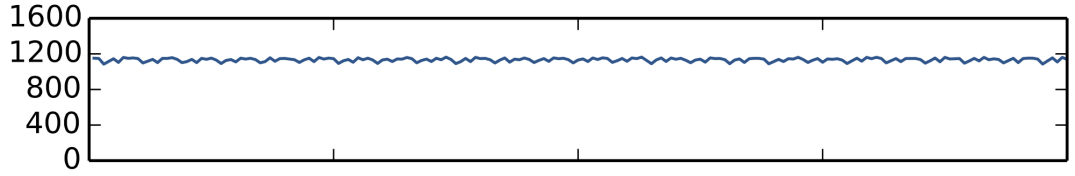
$$\text{Cost}(T, q) = \sum n_b$$

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

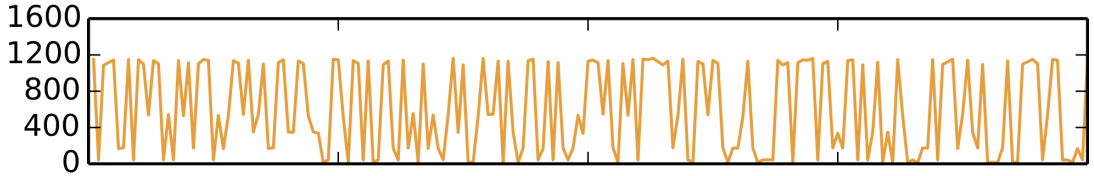
Repartitioning **ONLY** happens when reduction in total cost of query window is greater than repartitioning cost

Performance

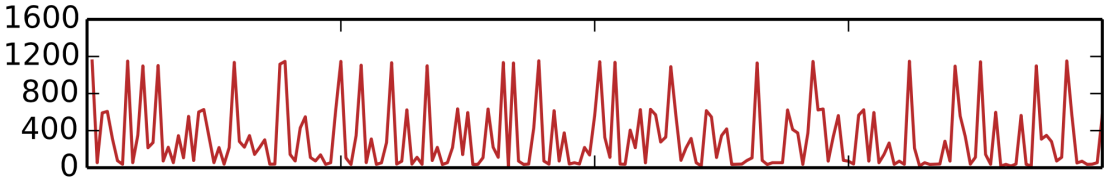
Workload: 200 Queries from 6 TPC-H query templates
Data: TPC-H denormalized 1.4TB (SF 200)



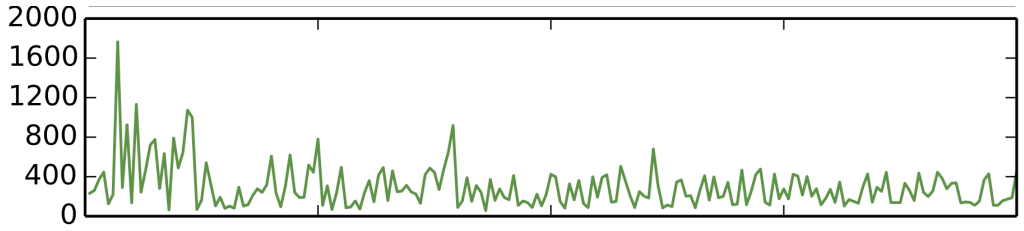
No Partitioning



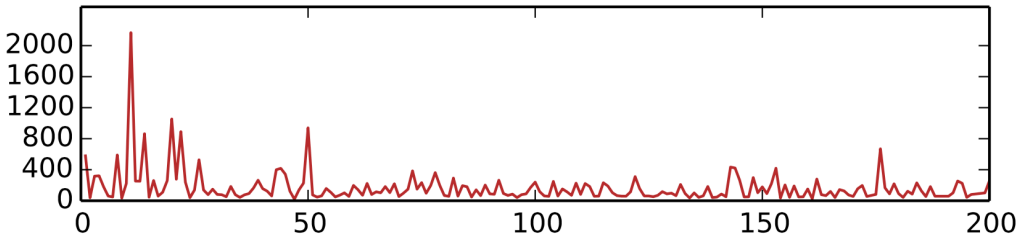
Pre-Partitioned on *orderdate* : 1.88x better



Hybrid Pre-Partition: 3.48x better



Amoeba with no prior info: 3.84x better



Amoeba starting from hybrid pre-partition: 6.67x better

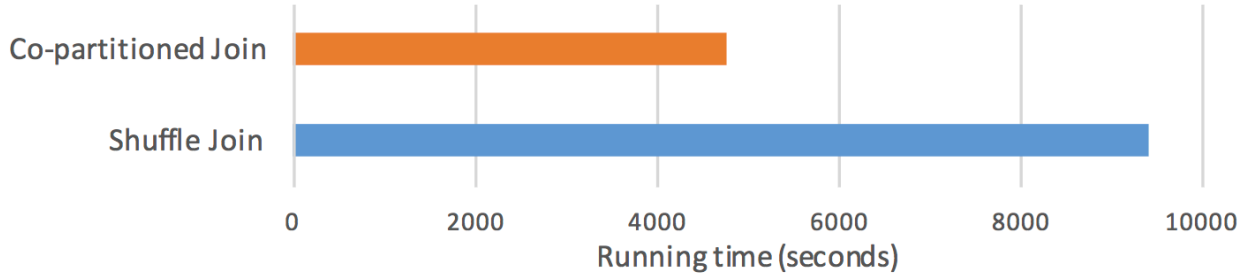
Handling Joins

A single fact table can join with multiple dimension tables

Eg: *lineitem* joins with *customer*, *order* and *part*

Shuffle joins used by default.

Co-Partitioned Join is much better.



But

1. Which tables to co-partition not known upfront
2. This choice might change over time

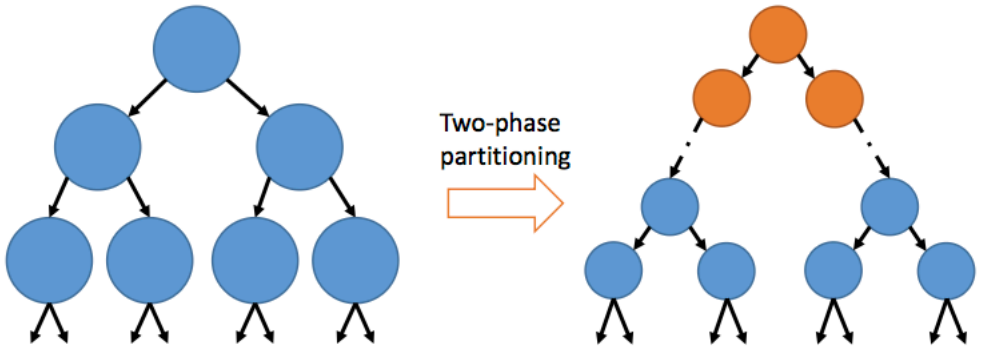
Adapt.

AdaptDB maintains multiple partitioning trees per dataset

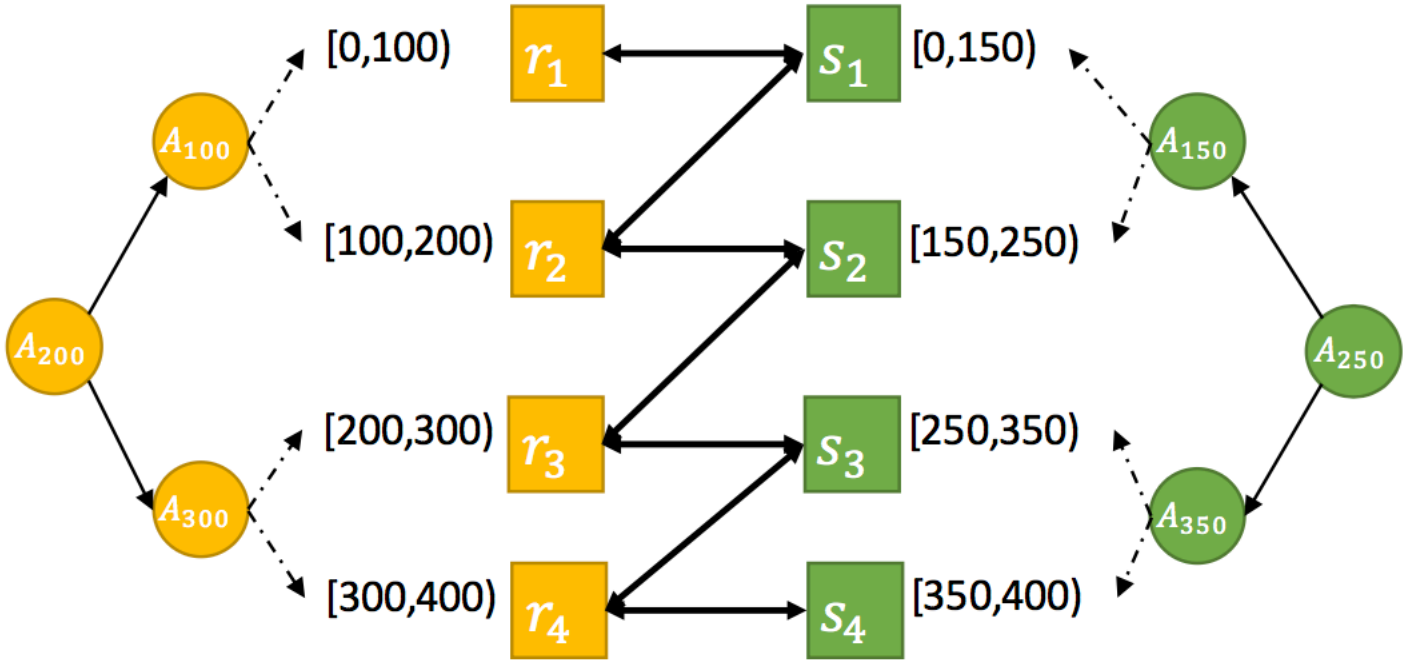
There is one for every frequent join attribute.

Each partitioning tree has two levels:

- Upper Level has the join attribute
- Lower Level has the attributes from predicates



Hyper Join



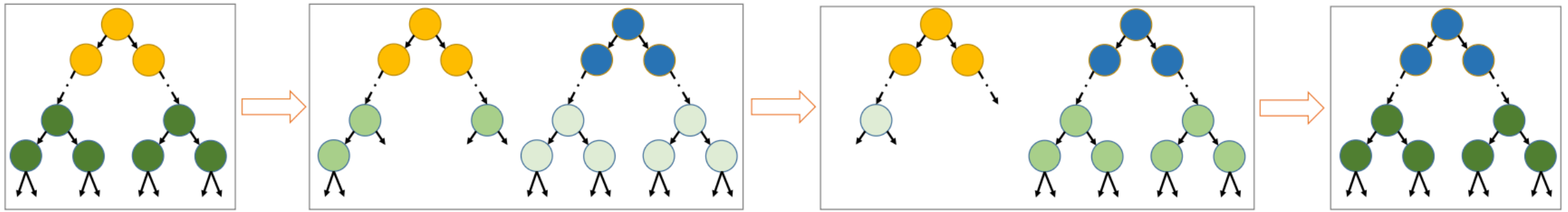
Each task (on spark) has a memory budget. Blocks need to be merged into tasks to run the join.

Finding the right set of blocks to form a task NP-Hard

HyperJoin uses a bottom-up algorithm with $O(n^2)$ runtime

Adapting for Joins

Example shows how we adapt as join attribute used shifts from A to B



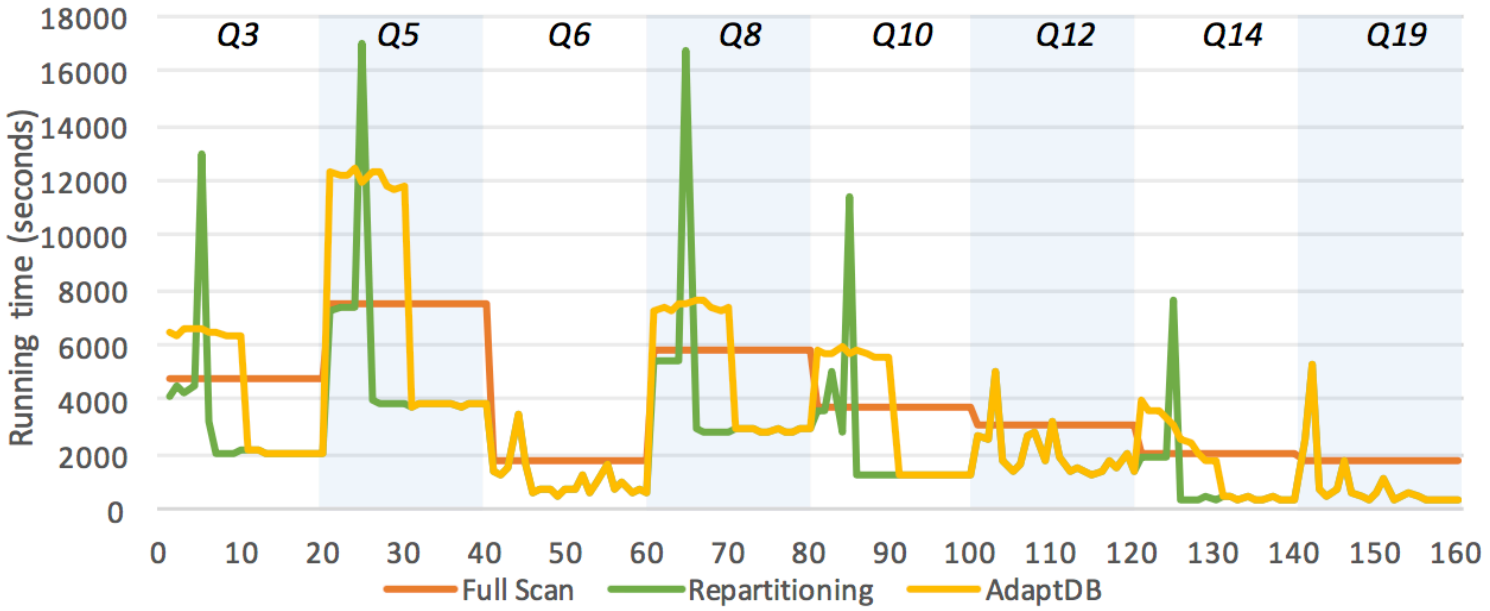
(1) A partitioning tree on A

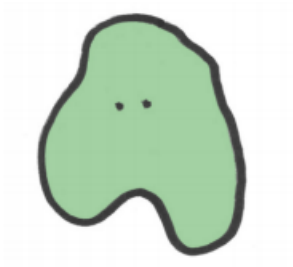
(2) Smooth repartitioning from A to B
A new partitioning tree B is created. Some data blocks under tree A are repartitioned after running each query.

(3) Repartitioning completes

Performance

Switching workload: shift from one query template to next
Data: TPCH SF 1000





+ Adapt.

- Gives you the benefits of partitioning for ad-hoc/exploratory analysis
- Can be created with no upfront query workload
- Allows incremental improvement via smooth re-partitioning
- Can incorporate prior workload information
- Significant speed-up for selections and joins

Anil Shanbhag

anil@csail.mit.edu

anilshanbhag.in

Thank You !

Extra Slides

Example Transformation