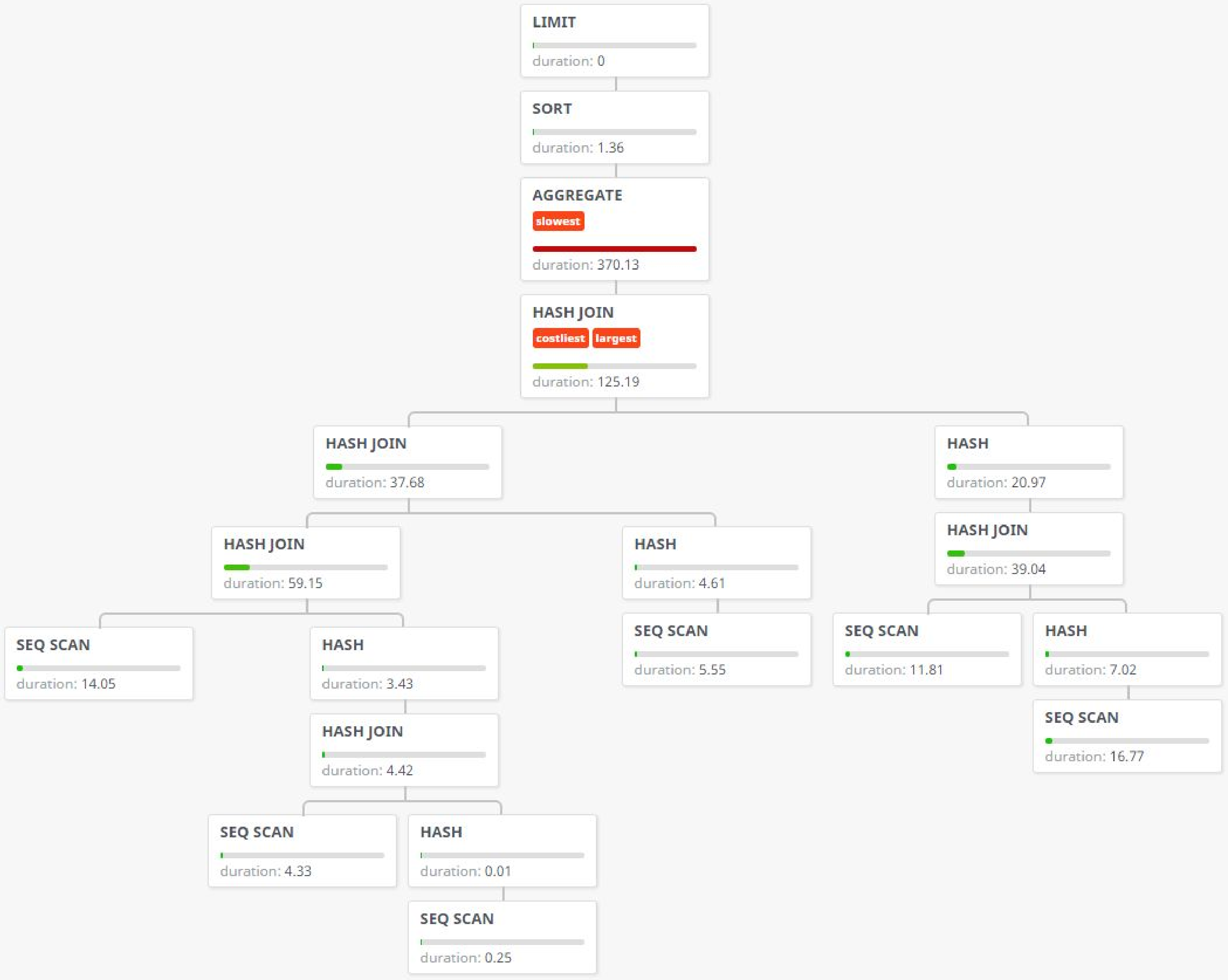
Predicting Query Execution Time

Vamshi,
Advisor : Jayant R. Haritsa,
Database Systems Lab.

Date : 08/03/2015

Background

Example of a Query Tree



Example : Hash Operator attributes

Node Type	Hash
Parent Relationship	Inner
Startup Cost	339.86
Total Cost	339.86
Plan Rows	10000
Plan Width	122
Actual Startup Time	12.446
Actual Total Time	12.446
Actual Rows	10000
Actual Loops	1
Output	p.prod_id,cat.categoryname
Hash Buckets	1024
Hash Batches	1
Original Hash Batches	1
Peak Memory Usage	425

Example : SeqScan Operator attributes

Node Type	Seq Scan
Parent Relationship	Outer
Relation Name	orders
Schema	public
Alias	o
Startup Cost	0
Total Cost	220
Plan Rows	12000
Plan Width	16
Actual Startup Time	0.008
Actual Total Time	5.548
Actual Rows	12000
Actual Loops	1
Output	o.netamount,o.totalamount,o.orderid

Features

- Features of a node are a combination of
  - Own attributes
  - Features of direct child(s) if any.

Introduction

- Problem : Given a Query Q along with its true cardinalities N, predict the execution time.

- Modified problem : Given Q along with complete set of features (available only after executing the query) , predict the execution time.

Big picture

Offline training (One time)

- Benchmark target system. Learn models for each operator at a fine grained level.
- Write the models to disk

Online predictions

- Obtain the query tree.
- For each node in tree, get the prediction from model saved in disk
- Final prediction = Sum of individual predictions.

Problem definition

- More formally, Given a set of tuples of the form

$$\langle X_1, X_2, \ldots, X_n, \text{Time}\rangle$$

Where X_i's are positive real numbers.

- Predict target value for a new tuple

$$\langle Y_1, Y_2, \ldots, Y_n\rangle$$

- A classical regression problem.

Modelling

- We need a model that can,
  - Capture the interaction between features.
  - Fit the complex relationship between features and target.
  - Generalize well.

- Models that I have considered,
  - Support Vector Regression (SVR)
  - Ensemble- Gradient Tree Boosting

SVR

- Regression version of SVM.
- A nonlinear function is learned by mapping inputs into high dimensional feature space.
- Projected data is separated using a linear hyperplane.
- It does this through kernel functions.
- Requires considerable tuning for it to work precisely.

Tuning SVR

- Kernel
  - Linear/Polynomial
  - Radial basis function(RBF)
    - Feature space is infinitely dimensional

- Hyper parameters
  - Regularization
  - Gamma

Hyperparameters

- C
  - Key to avoid overfitting.
  - A parameter to *trade* accuracy for the *simplicity* of model.
  - Complex models can fit very well with the training data but might fail with unknown test data.
- Gamma
  - Free parameter of RBF kernel.
  - Limits the radius of support vector i.e., impact of one training example over the model.
- In general we do not know beforehand the optimal (C, gamma). These need to be found through exhaustive search.

Training

- Used skewed version of TPCH to benchmark target system with zipfian factor z=2.

- Database sizes (GB).

  0.1 0.5 1 2 3 4 5

- 440 queries for each database, total = 3080.

- All tables vacuum analyzed, index are created as per TPCH semantics.

Model computation

- For each physical implementation of an operator,
  - (Hyperparameters) = GridSearch(operator)
  - Model = SVR(Hyperparameters)
  - Write Model to disk

- GridSearch(operator)
  - Divide input data into k-folds
  - Use k-1 folds to learn and the other to evaluate.
  - C_range = logspace(-2,8,10)
  - gamma_range = logspace(-7, 3, 10)
  - Best hyperparameters are the ones that produce the lowest error

Contd.

- Cross validation ensures that estimations are unbiased
- Grid Search ensures that we do not overfit.

Error metric

- Mean relative error (MRE)
  - |Actual-Predicted|/Actual
- Does not differentiate between under/over estimation.

Training time

- Total time required = Time to execute training queries on target hardware + Time taken to learn models.

- The actual learning time takes only a couples of sec but this learning need to be done for multiple hyperparameter configurations.

  - Grid search is embarrassingly parallel, each configuration is learnt independently. (Using python's inbuilt support)

  - Current training time ~ 36 + 5 = 41 Hours.

Errors by operator wise, MRE<1

Operator	#Samples	#Features	MRE
IndexOnlyScan	300	11	0.62
NL-Aggregate	190	10	0.02
NL-Materialize	420	10	0.09
Aggregate-Sorted	276	11	0.03
Aggregate-Hashed	2364	11	0.61
Sort-QuickSort	1376	12	0.62
HashJoin	3020	10	0.94
NL-IndexOnlyScan	140	10	0.11
NL-SeqScan	240	10	0.75
MergeJoin	220	10	0.12
Sort-Top N heapsort	1244	12	0.88

Errors by operator wise, MRE 1-3

Operator	#Samples	#Features	MRE
Sort-ExternalMerge	100	12	1.18
BitmapHeapScan	1560	12	2.48

Errors by operator wise, MRE>3

Operator	#Samples	#Features	MRE
Aggregate	920	10	6.18
IndexScan	3098	10	9.82
BitmapIndexScan	1560	10	14.40
NL-BitmapHeapScan	860	10	33.57
Hash	3020	14	61.77
Materialize	400	10	88.40
NL- IndexScan	2900	10	52.58

Takeaway

- 66% of the operator models(=14) have produced MREs less than 2.
- Non linear kernels(RBF) unlike linear, tend to improve when supplied with more data.
- Results validated that operators based on random IO are hardest to predict possibly because of the high variance in training data.

Alternative models

- Have also studied the relevance of alternate models especially the tree based ones.

- Gradient boosting regression trees performed best among them. It's the current state of the art for regression problems.

- Pros
  - Superior accuracy compared to SVR.
  - White box models
  - Can learn the relative importance of features to target.

Contd.

- Cons
  - Poor generalization: The accuracy comes with a cost. GBRT cannot extrapolate i.e., values which lie outside the boundary.
  - Accuracy vs Space tradeoff: GBRT precision is directly proportional to number of estimators it uses underneath..
  - Slow in learning: More hyperparameters to tune (typically 3-5). Implies, grid search takes longer to find the best set of hyperparameters.

What's next?

- As of now, features are completely dynamic. Next step is to consider doing the modelling only with *Q and N(true cardinalities)* as inputs.

- Current features are only numerical. Categorical and string data can add more value but are trickier to consider.

  Example: Complexity of selection predicate etc.

- Model stacking: This can be a workaround for GBRT limitations. Instead of a single model, they can be nested i.e., output of model(s) are supplied as features to another model. [WORK IN PROGRESS!]

Thank you