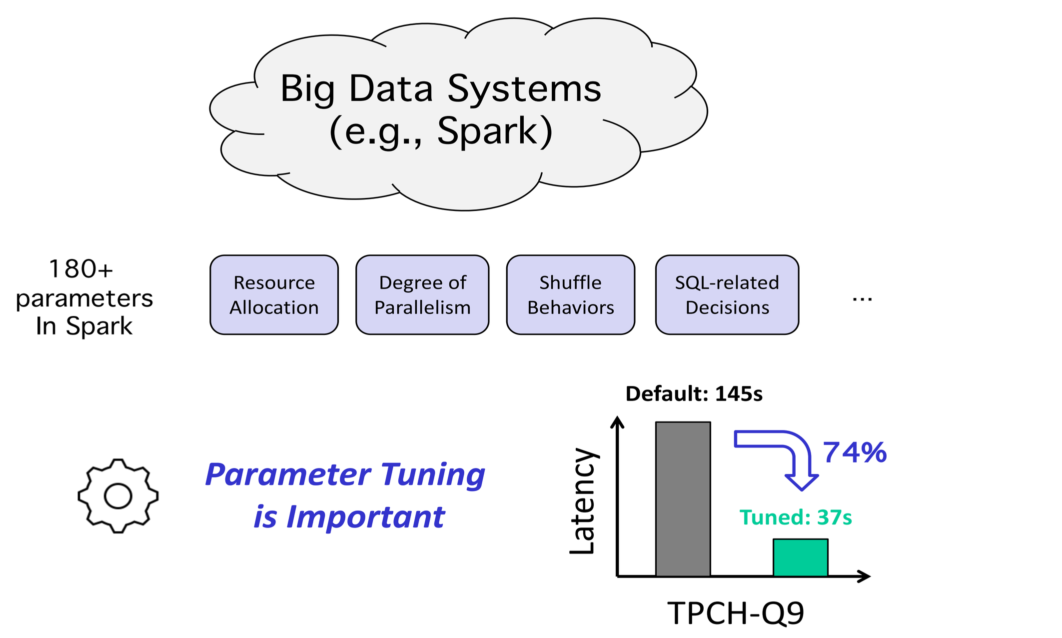
A Spark Optimizer For Adaptive, Fine-Grained Parameter Tuning

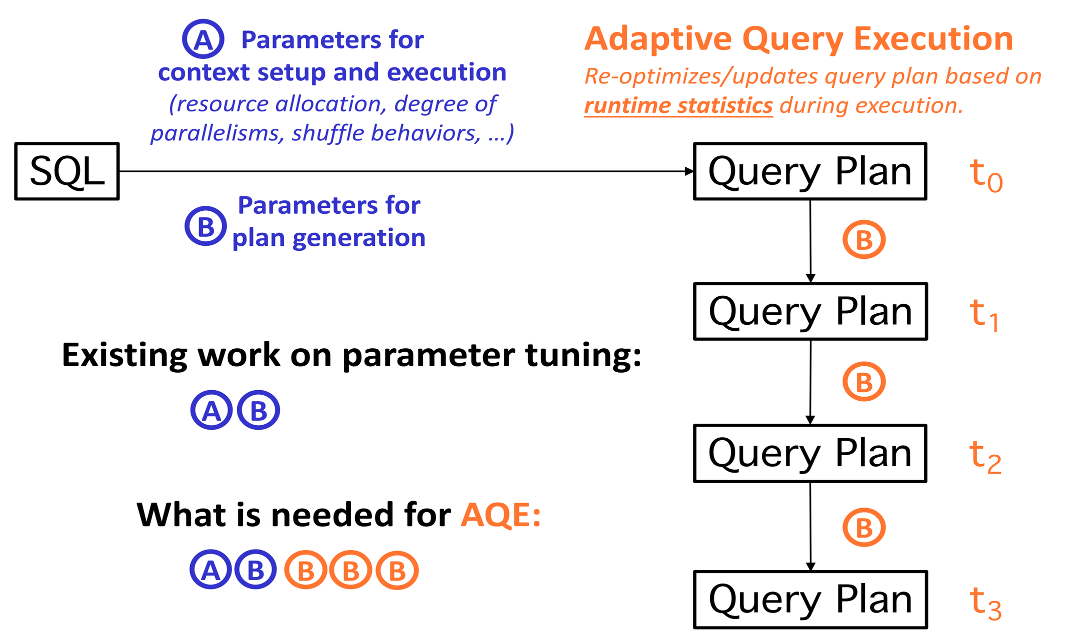
Chenghao Lyu[1], Qi Fan[2], Philipee Guyard[2], Yanlei Diao[1,2],
[1]University of Massachusetts Amherst, [2]Ecole Polytechnique,

Paper Link

Big Data Query Processing

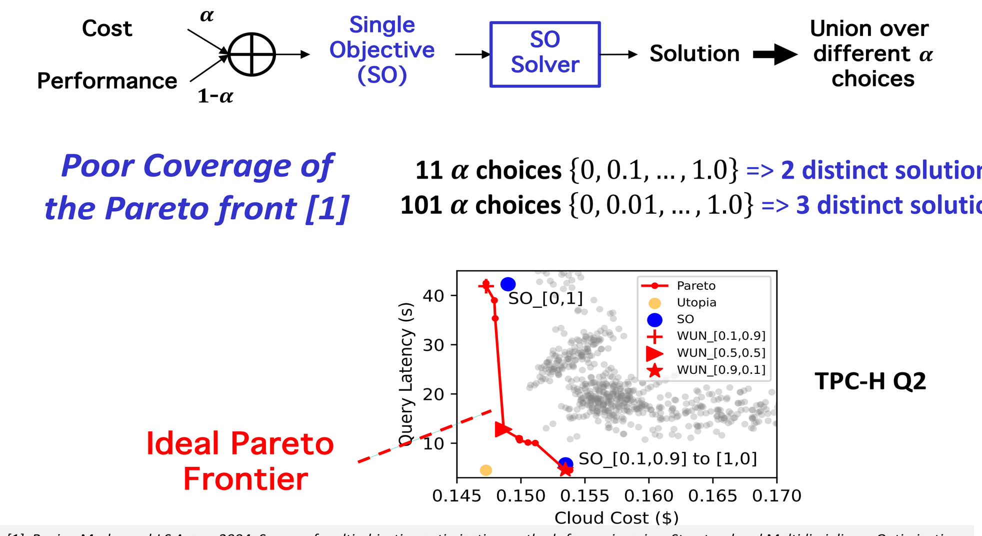


Trend 1: Adaptive Query Execution (AQE)



Trend 2: Cost Performance Reasoning

Prior Work (Multi-Objective Weighted Sum, MO-WS[1])



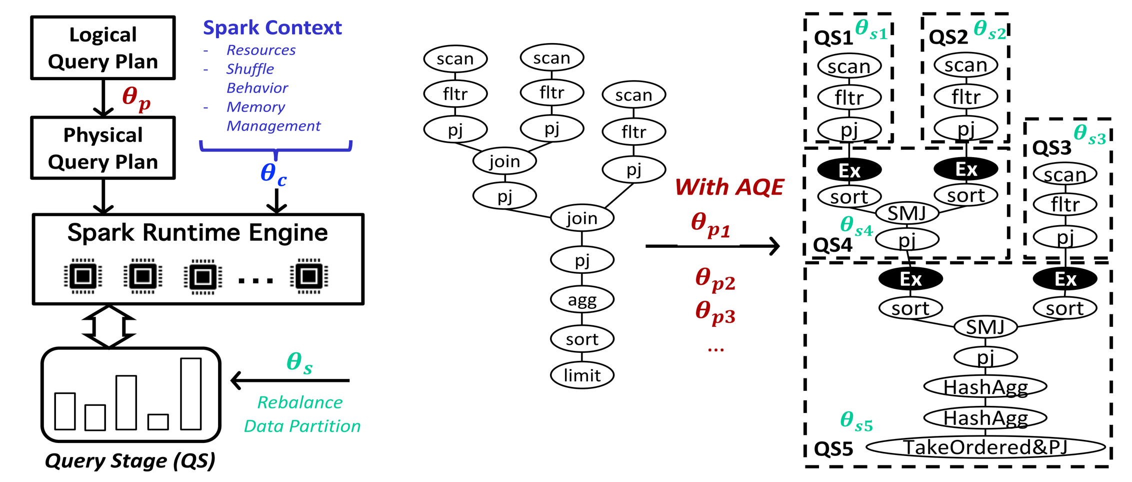
[1]: Regina Marler and J S Arora. 2004. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26, 6 (2004), 369–395.

GOAL

Spark Optimizer for Parameter Tuning

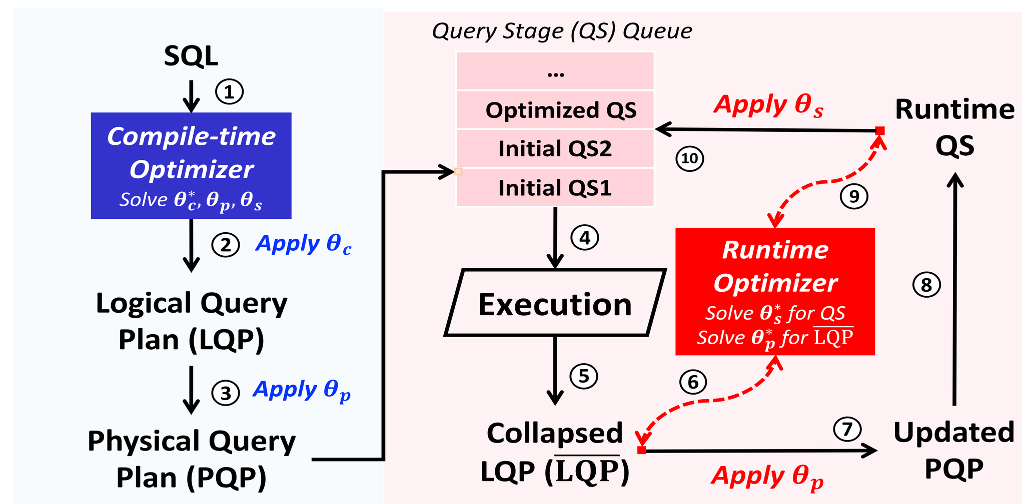
- Controls *ALL* parameters in the new architecture of adaptive query execution
- Casts the tuning problem in the setting of *multi-objective optimization* to better adapt to user cost-preference needs

Challenge 1: Complex Parameter Control



- θ_c Context Parameters —Coarse-grained→ θ_c → Determined at query submission
- θ_p Query Plan Parameters —Fine-grained→ $\theta_{p1}, \theta_{p2}, \theta_{p3}, \ldots$ → Best tuned at runtime
- θ_s Query Stage Parameters —Fine-grained→ $\theta_{s1}, \theta_{s2}, \theta_{s3}, \ldots$

Solution: Hybrid Multi-Granularity Tuning



- **Compile time => Optimal θ_c^*,** with initial $\{\theta_p\}$ and $\{\theta_s\}$, based on **estimated statistics**
- **Runtime => Optimal θ_p^* and θ_s^*** in each AQE step, given θ_c^*, based on **runtime statistics**

Challenge 2: High Overhead of Optimization

Definition 3.3. Multi-Objective Optimization for Spark SQL

$$\underset{\theta_c, \{\theta_p\}, \{\theta_s\}}{\arg\min} \; f(\theta_c, \{\theta_p\}, \{\theta_s\}) = \begin{bmatrix} f_1(\text{LQP}, \theta_c, \{\theta_p\}, \{\theta_s\}, \alpha, \beta, \gamma) \\ \ldots \\ f_k(\text{LQP}, \theta_c, \{\theta_p\}, \{\theta_s\}, \alpha, \beta, \gamma) \end{bmatrix}$$

k objectives

$$\text{s.t.} \quad \begin{aligned} &\theta_c \in \Sigma_c, \\ &\{\theta_p\} = \{\theta_{p1}, \theta_{p2}, \ldots, \theta_{pt}, \ldots\}, \forall \theta_{pt} \in \Sigma_p \\ &\{\theta_s\} = \{\theta_{s1}, \theta_{s2}, \ldots, \theta_{si}, \ldots\}, \forall \theta_{si} \in \Sigma_s \end{aligned}$$

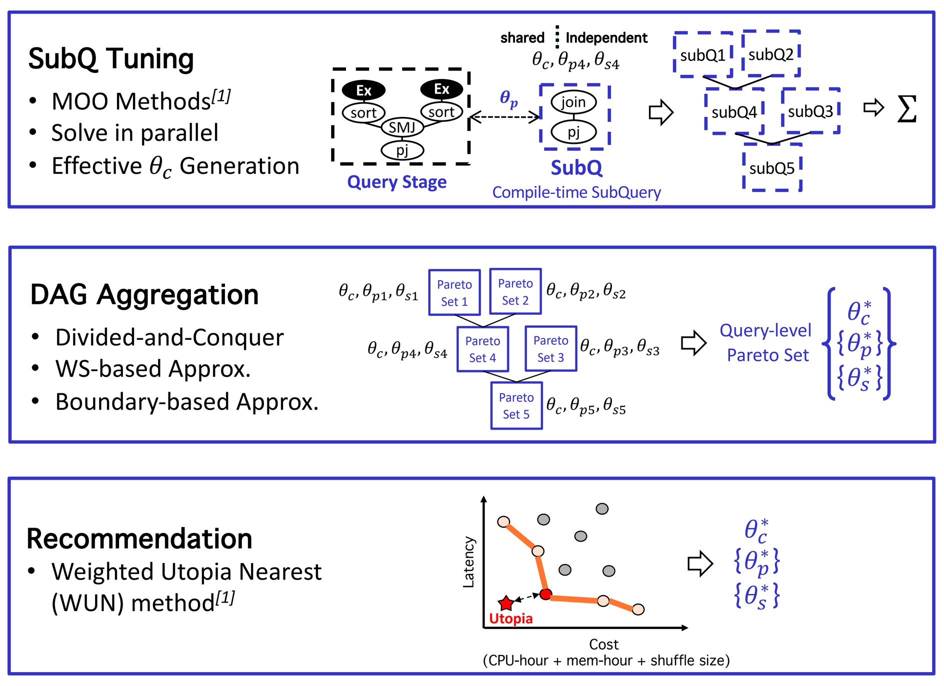
Decision variables

LQP	Logical Query Plan
α	Input characteristics (# rows, size)
β	Distribution of input partitions
γ	Runtime status, encoded by statistics of the running tasks in parallel stages

Non-decision variables

Large Searching Space and High Overhead

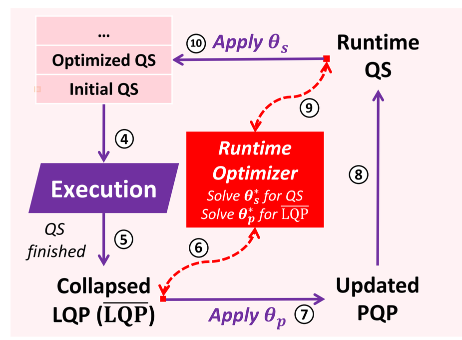
Solution: Compile-Time Optimization



[1]: Song, Fei, et al. "Spark-based Cloud Data Analytics using Multi-Objective Optimization." 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE Computer Society, 2021.

Solution: Runtime Optimization

$$\min_{\theta_s} \begin{cases} Cost_{QS_i} \\ Latency_{QS_i} \end{cases}$$



$$\min_{\theta_p} \begin{cases} Cost_{\overline{LQP}_i} \\ Latency_{\overline{LQP}_i} \end{cases}$$

Spark Plugins for Runtime Optimization

- Re-compile Spark to support adding tuning strategies before applying opt. rules
- Server-side: MOO solving
- Client-side: update θ_p and θ_s

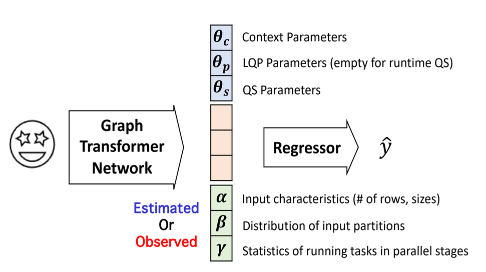
No. of optimization requests = No. of completed QS + No. of runtime QS

Prune unnecessary requests

- Bypass $\overline{\text{LQP}}$ when no join
- Bypass $\overline{\text{LQP}}$ with uncleared dependency
- Skip scan-based QS
- Skip QS with small input size.

Reduce 86-92% requests!
Cost 0.3-0.4s per query

Modeling Techniques and Results



Multi-channel input framework [1]

Modeling Performance over TPC-H and TPC-DS, each with 50K traces

	Target	Ana-Latency/Latency (s) WMAPE	P50	P90	Corr	Shuffle Size (MB) WMAPE	P50	P90	Corr	Xput K/s
TPC-H	subQ	0.131	0.029	0.292	0.99	0.025	0.006	0.045	1.00	70
	QS	0.149	0.027	0.353	0.98	0.002	3e-05	0.004	1.00	86
	$\overline{\text{LQP}}$	0.164	0.060	0.337	0.95	0.010	8e-05	0.002	1.00	146
TPC-DS	subQ	0.249	0.030	0.616	0.95	0.098	0.016	0.134	0.99	60
	QS	0.279	0.060	0.651	0.95	0.028	4e-04	0.023	1.00	79
	$\overline{\text{LQP}}$	0.223	0.095	0.459	0.93	0.107	0.028	0.199	0.99	462

High Accuracy

- Weighted Mean Absolute Percentage Error (WMAPE)
- P50Error, P90Error up to 0.65/0.19 for latency/shuffle size
- Pearson Correlation close to 1.

High Inference Throughput

- 60K-462K per second

[1]: Lyu, Chenghao, et al. "Fine-grained modeling and optimization for intelligent resource management in big data processing." Proceedings of the VLDB Endowment 15.11 (2022): 3098-3111.

Benefits Over Query-Level MOO Against the Default

MO-WS[1]	Multi-Objective Weighted Sum, Coarse-grained Tuning
HMOOC3	Ours, Compile-time, Fine-grained Tuning
HMOOC3+	Ours, Compile-time/Runtime, Fine-grained Tuning

Prioritize improving query speed

	TPC-H MO-WS	TPC-H HMOOC3	TPC-H HMOOC3+	TPC-DS MO-WS	TPC-DS HMOOC3	TPC-DS HMOOC3+
Total Lat Reduction	18%	61%	63%	25%	61%	65%
Avg Solving Time (s)	2.6	0.41	0.70	15	0.41	0.80

1. MO-WS: a total ↓ of 18-25% latency with an average solving time of 2.6-15s
2. Ours: a total ↓ of 63-65% latency with an average solving time of 0.7-0.8s
   - Multi-grained compile-time optimization (↓ 61% with 0.41s)

[1]: Regina Marler and J S Arora. 2004. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 26, 6 (2004), 369–395.

Adaptivity Comparison to SO with Fixed Weights

SO-FW	Reduce to a Single Objectives with Fixed Weights
HMOOC3+*	Hybrid, Multi-granularity tuning (with multi-query plan search)

Performance-cost savings against default configuration over different preferences

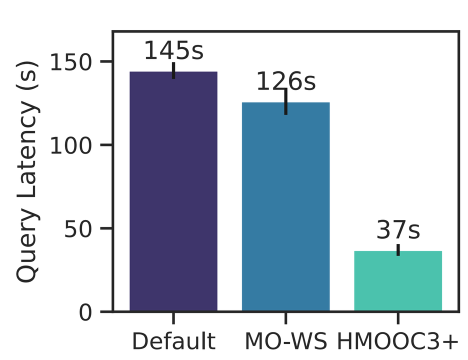
Prefs. Lat/Cost	TPC-H SO-FW	TPC-H HMOOC3+*	TPC-DS SO-FW	TPC-DS HMOOC3+*
(0.0, 1.0)	20% / -11%	-15% / -10%	-6% / 64%	-45% / -22%
(0.1, 0.9)	1% / 1%	-31% / -5%	-28% / 105%	-57% / -7%
(0.5, 0.5)	-1% / 25%	-46% / -3%	-28% / 128%	-59% / 29%
(0.9, 0.1)	-13% / 27%	-51% / 0%	-34% / 139%	-59% / 55%
(1.0, 0.0)	-14% / 44%	-55% / 3%	-26% / 144%	-59% / 64%

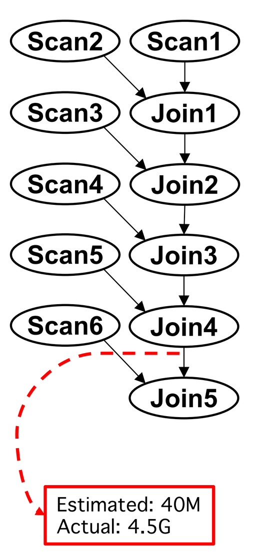
An increasing demand of saving latency

Fails to adapt

1. Ours: Up to 55-59% ↓latency and up to 10-22% ↓cost — Superior adaptivity
2. SO-FW: at most 14-34% ↓latency and rare ↓cost — Not adapting well

Example of Tuning TPC-H Q9

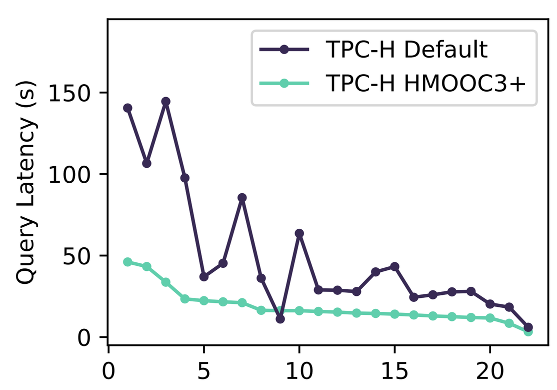




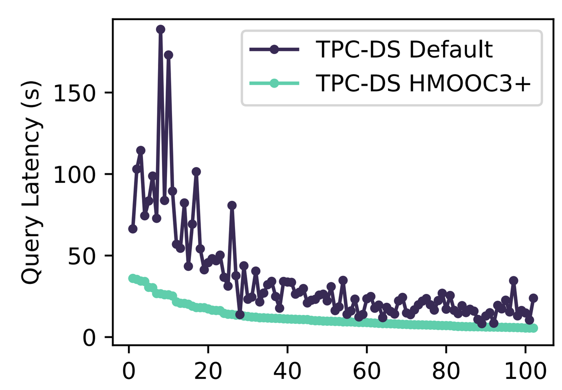
SMJ: Sort-Merge Join
BHJ: Broadcast Hash Join
SHJ: Shuffled Merge Join

Default	MO-WS	HMOOC3+
SMJ	SMJ	SHJ
SMJ	SMJ	BHJ
SMJ	BHJ	BHJ
BHJ	BHJ	BHJ
SMJ	BHJ	SHJ

E2E Performance Comparison for Individual Queries



Latency of Queries in TPC-H



Latency of Queries in TPC-DS

Thanks For Your Attention

- **Code Implementation**
  - [Client-side] Spark plugins to support runtime tuning: https://t.ly/xMTxu
  - [Server-side] Modeling and MOO algorithms: https://t.ly/XqfL1
- **Potential Future Work**
  - Extended to other systems who support runtime adaptivity with observed statistics (e.g., Presto, Greenplum, etc.)
- **Contact Me**
  - Email: chenghao@cs.umass.edu
  - I am expecting a full-time job in 2024/2025

Résumé

WeChat