Data Pipeline Cost Optimizer: A Linear Programming and Risk Modeling Approach for Cloud Compute Allocation Using Real AWS Spot Pricing Data

Johnson Liu
Cornell University
Email: jl3486@cornell.edu

***Abstract*—Cloud compute costs represent a major operational burden for organizations running data-intensive pipelines. This work presents a full-stack optimization system that assigns data-processing jobs to Amazon EC2 instance types using a combination of linear programming, performance-tier constraints, stochastic spot-price modeling, and risk-aware optimization. Using real historical EC2 spot-price traces from the *ap-northeast-1* region and synthetic job workloads, the system minimizes total compute cost while satisfying memory, performance, and auto-scaling capacity constraints. Enhancements including scenario-based volatility analysis, spot-interruption risk modeling, cost–risk Pareto frontier computation, and an interactive Streamlit dashboard provide a robust, extensible framework for analyzing real-world cloud cost trade-offs. Results demonstrate a 47.7% reduction in compute cost relative to a naive on-demand baseline while maintaining strict performance requirements.**

***Index Terms*—Cloud optimization, EC2 spot instances, linear programming, CVXPY, cost minimization, risk modeling, stochastic optimization, Streamlit, AWS.**

I. Introduction

Modern data-driven systems rely heavily on cloud compute resources to execute extract–transform–load (ETL) pipelines, machine learning (ML) training workloads, and batch analytics. Cloud providers such as Amazon Web Services (AWS) offer a diverse set of instance families with varying compute power, memory capacity, and pricing models (e.g., on-demand, reserved, and spot). Spot instances, in particular, provide substantial discounts in exchange for potential interruptions.

Due to the large cost variability introduced by spot markets, performance requirements, and resource constraints, cloud workload placement becomes a non-trivial optimization problem. Motivated by this challenge, we develop a complete cost-optimization engine that ingests real EC2 spot-price traces, models pipeline jobs with resource demands, and solves a constrained linear program to determine the minimum-cost allocation.

This work contributes (1) a performance- and memory-aware linear program for job–instance assignment, (2) scenario-based pricing volatility modeling, (3) a risk-aware cost minimization formulation, (4) comparison against a naive on-demand baseline, and (5) an interactive dashboard for experimentation. We additionally analyze cost robustness and compute a Pareto frontier for cost–risk trade-offs.

II. Related Work

Prior research has explored cloud workload scheduling via heuristics, reinforcement learning, or integer programming. AWS Auto Scaling and EC2 Fleet provide automated provisioning but expose limited optimization capability to end users. Academic literature often uses synthetic price models, whereas our approach directly leverages real spot-price traces. This work extends classical resource allocation formulations with performance tiers, capacity constraints, and probabilistic spot-price variability.

III. System Architecture

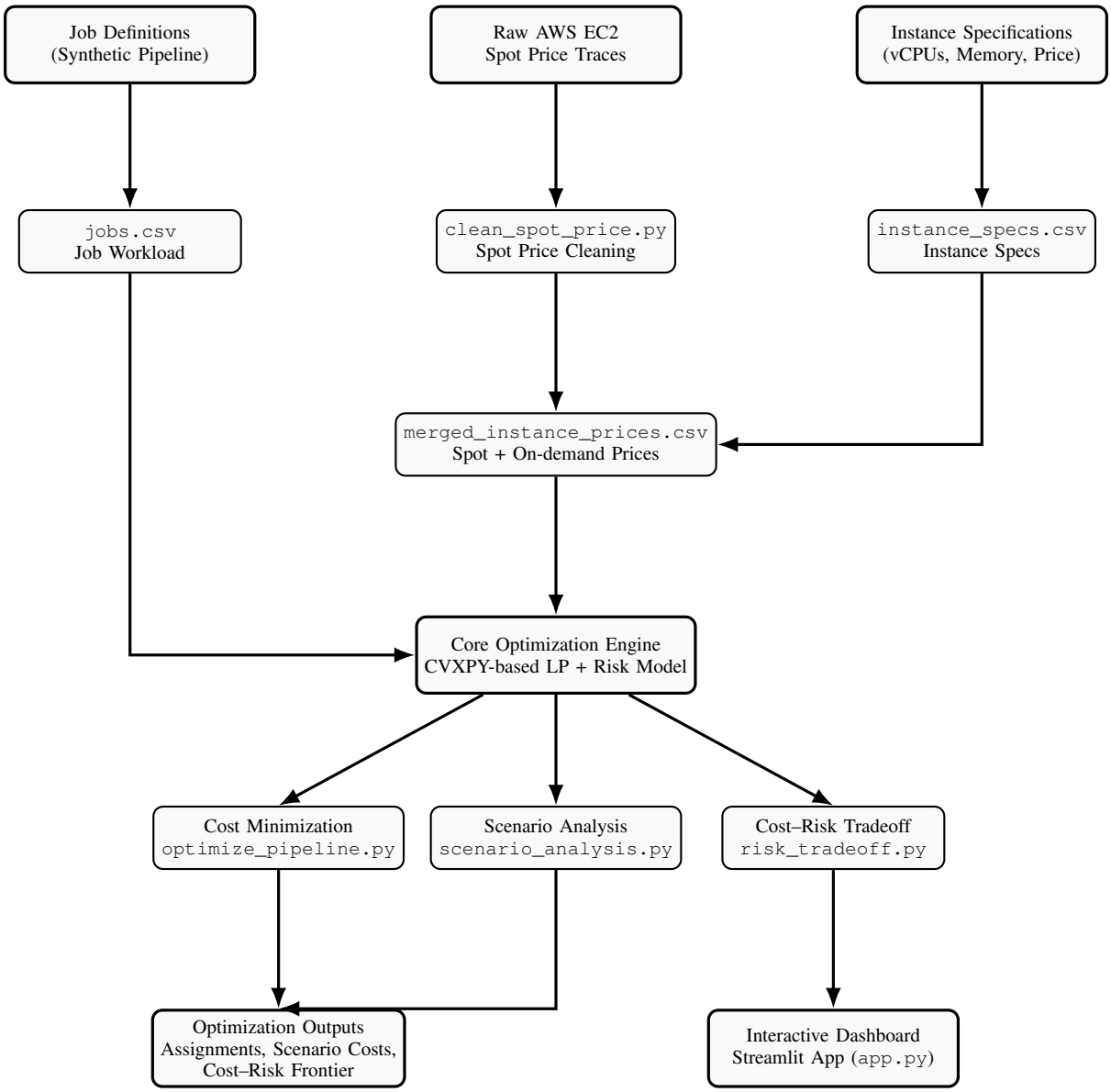

Fig. 1. High-level architecture of the Data Pipeline Cost Optimizer. Job definitions, raw AWS EC2 spot-price traces, and instance specifications are preprocessed into structured workloads and merged price tables. These artifacts feed a CVXPY-based optimization engine that supports cost minimization, scenario analysis under spot-price volatility, and cost–risk tradeoff modeling. The resulting job–instance assignments and scenario outputs are consumed both as offline artifacts and through an interactive Streamlit dashboard.

IV. Optimization Formulation

A. Decision Variables

Let $x_{j,i} \in [0,1]$ denote the fraction of job j assigned to instance type i. The convex relaxation allows tractable LP solving while producing nearly integral solutions.

B. Objective Function

The base objective minimizes monetary cost:

$$\min \sum_{j,i} \text{CPU\_hours}_j \cdot \text{price}_i \cdot x_{j,i}.$$

C. Constraints

1) Job Assignment: Every job must be fully assigned:

$$\sum_i x_{j,i} = 1.$$

2) Memory Feasibility:

$$x_{j,i} = 0 \quad \text{if memory}_j > \text{memory}_i.$$

3) Performance Tiers: Each instance type is assigned a performance tier. Jobs require a minimum tier:

$$x_{j,i} = 0 \quad \text{if tier}_i < \text{tier\_req}_j.$$

4) Capacity Constraints: Autoscaling capacity limits:

$$\sum_j x_{j,i} \leq \text{cap}_i.$$

D. Risk-Aware Objective

Risk is modeled using normalized spot price discount:

$$\text{risk}_i = \max\left(0, \frac{p_i^{od} - p_i^{spot}}{p_i^{od}}\right),$$

where p_i^{od} is the on-demand price and p_i^{spot} is the effective spot price. The combined objective becomes:

$$\min \ \text{Cost} + \lambda \cdot \text{Risk}.$$

V. Implementation Details

A. Codebase Structure

- **clean_spot_price.py** – parses raw spot-price data, splitting tab-delimited records and extracting instance type, timestamp, and price.
- **build_price_table.py** – merges instance specifications with cleaned pricing and computes average spot price per instance type, producing `merged_instance_prices.csv`.
- **optimize_pipeline.py** – core LP solver with memory, performance, and capacity constraints, minimizing cost under current spot prices.
- **scenario_analysis.py** – runs N perturbation scenarios with spot-price noise and records the distribution of optimal total cost.
- **risk_tradeoff.py** – computes a cost–risk Pareto frontier by sweeping the risk aversion parameter λ.
- **baseline_compare.py** – evaluates a naive on-demand baseline allocation and computes relative cost savings of the optimized solution.
- **run_optimizer.py** – unified CLI entrypoint that dispatches to base optimization, scenario analysis, or risk modeling using a YAML configuration file.
- **app.py** – Streamlit web interface enabling interactive adjustment of λ, spot-price noise, and capacity constraints, and visualizing the resulting assignments and cost metrics.

B. Streamlit Dashboard

The dashboard provides sliders for risk aversion λ, spot-price noise, and toggling capacity constraints. Assignments and cost metrics are displayed alongside visualizations such as bar charts of total cost per instance type and scatter plots of job cost versus performance score.

VI. Experimental Results

A. Baseline vs Optimized

The naive baseline assigns each job to the least expensive on-demand instance that satisfies memory and performance constraints. The optimized solver achieves:

$$\text{Baseline cost} = \$52.14,$$

$$\text{Optimized cost} = \$27.27,$$

yielding a **47.7% reduction** in total compute cost.

B. Scenario Analysis

Fifty simulations with $\pm 30\%$ spot price volatility resulted in optimal total costs between approximately \$22.3 and \$27.5, highlighting robustness to price fluctuations.

C. Cost–Risk Frontier

Sweeping $\lambda \in \{0, 0.2, 0.5, 1.0, 2.0\}$ yields a smooth Pareto frontier. Higher risk aversion forces allocation toward higher-priced, more stable instances, increasing cost while reducing exposure to interruption risk.

VII. Conclusion

This work presents a comprehensive framework for cloud cost optimization using linear programming and risk modeling. It spans data ingestion, optimization, uncertainty analysis, and visualization. Future work includes multi-region optimization, ML-based interruption prediction, integer programming for exact selection, and integration with real-time AWS APIs.