AI with ROI

Use ML to Cut Your Snowflake Bill in Half



Presented by Espresso AI

What is a Data Warehouse?



- Large cloud database optimized for analytical queries
- Separate storage and compute, both scalable in the cloud
- You can think of it is putting your data in S3 and autoscaling EC2 boxes on-demand

What is Espresso AI?

We optimize how Snowflake and Databricks process queries, saving our customers millions of dollars.

We take a few minutes to set up, and instantly save up to 80%.



What is Espresso AI?

We optimize how Snowflake and Databricks process queries, saving our customers millions of dollars.

We take a few minutes to set up, and instantly save up to 80%.



Intro



- Ben Lerner, cofounder & CEO of Espresso AI

- I used to work on performance engineering and ML at Google

ML for efficient data warehousing

- We think of this as a constrained optimization problem:
  - Run your workloads as cheaply as possible without increasing the latency of any job

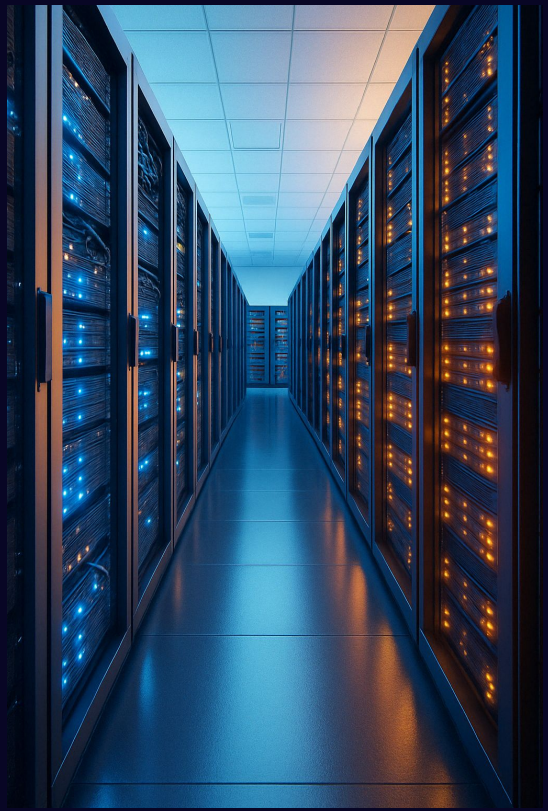
- We do this in two ways:
  - Scheduling
  - Query Optimization



Scheduling

What is Scheduling?

- Scheduling is how you allocate workloads across physical machines in a cluster.
- Data centers handle this with systems likes Borg and Kubernetes
- Good scheduling can dramatically increase efficiency



How does Snowflake do Scheduling?

- Snowflake uses a static config
  - One cluster for Tableau, one for dbt, etc.

- This is terrible for efficiency
  - Often spins up clusters for no reason
  - Most customers have utilization below 50%





Better Scheduling for Snowflake

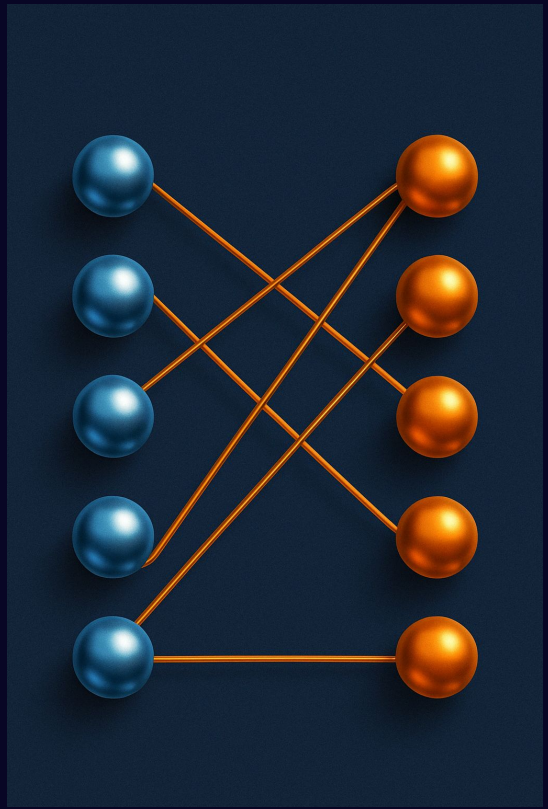
- You can do better scheduling by predicting query runtime and scaling

- This is like the Kubernetes scheduler, but using ML to estimate resource usage instead of having hand-annotated jobs

Large-scale cluster management at Google with Borg - Verma et. al.

How does Scheduling work?

- Scheduling has 3 components:
  - Resource requirements for jobs
  - Resources available on a cluster
  - An algorithm for matching jobs to clusters



Resource Requirements

- With Kubernetes, each job gets annotated with CPU, RAM, disk, etc.

- Snowflake queries don’t include resource constraints, so we interpolate them using LLMs

Available Resources

- With Kubernetes, each machine provides a pool of resources. (Still CPU, RAM, disk, GPUs, etc.)

- Snowflake clusters don’t provide clearly labeled resources, so again, we interpolate them with ML models

Scheduling Algorithm



- Kubernetes algorithms are an ongoing area of research
  - Greedy bin-packing is a reasonable starting point
  - Better algos take into account job timing and resources spikes, predicting future workloads, autoscaling, etc.

- You guessed it: we do this with ML
- Specifically, we use RL to solve our constrained optimization problem of minimizing cost subject to latency

RL for Scheduling

- Tons of interesting problems here
  - Opportunistically upsizing queries to take advantage of capacity
    - This is why things run faster, not just cheaper
  - Understanding job latency vs. query latency
  - Queuing jobs for better packing



What does it look like?

Scheduling

dbt →



Large WH, 15m = 2 credits

Looker →



Medium WH, 7.5m = 0.5 credits

Ad-Hoc Queries →



Small WH, 15m = 0.5 credits

Total = 3 credits

Scheduling

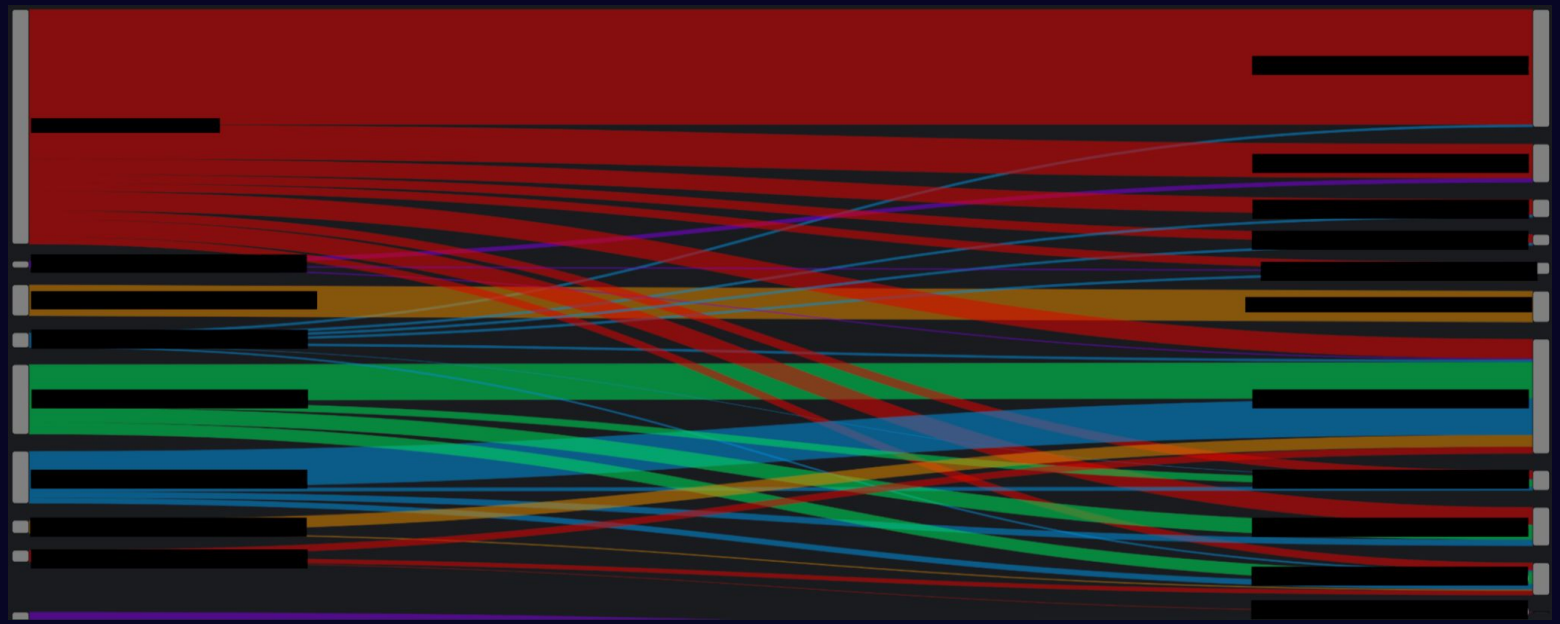


Total = 1.5 credits



Scheduling IRL

Sankey diagram of real-life query routing from an early beta run:





Query Optimization

Query Incrementalization

- You can run ELT jobs massively faster and cheaper by incrementalizing your queries.

- Incrementalizing means taking a query that processes an entire dataset and changing it to reprocess only new data for each run.

Query Incrementalization Example

```
select * from {{ ref('events') }}

{% if is_incremental() %}
where event_time >= (select
max(event_timestamp)::date from {{ this }})

{% endif %}
```



Automatic Query Incrementalization

This gets a lot more complicated.

- Joins
  - Multiple Joins
- Aggregations
- Filters
- Subqueries



Automatic Query Incrementalization

- LogicalTableScan
  - $R_1 + \Delta R_1$
- LogicalProject
  - $\Delta R_x = \pi_{attributes}(\Delta R_{input})$
- LogicalFilter
  - $\Delta R_x = \sigma_{condition}(\Delta R_{input})$
- LogicalJoin
  - $R'_x = R_x \cup \Delta R_x$
  - $R'_x = R'_1 \bowtie R'_2$
  - $\rightarrow R'_x = (R_1 \cup \Delta R_1) \bowtie (R_2 \cup \Delta R_2)$
  - $\rightarrow R'_x = (R_1 \bowtie R_2) \cup (\Delta R_1 \bowtie R_2) \cup (R_1 \bowtie \Delta R_2) \cup (\Delta R_1 \bowtie \Delta R_2)$
  - $\rightarrow R'_x = (R_1 \bowtie R_2) \cup (\Delta R_1 \bowtie (R_2 \bowtie \Delta R_2)) \cup (R_1 \bowtie \Delta R_2)$
  - $\rightarrow R'_x = R_x \cup (R'_2 \bowtie \Delta R_1) \cup (R_1 \bowtie \Delta R_2)$
  - $\rightarrow \Delta R_x = (R'_2 \bowtie \Delta R_1) \cup (R_1 \bowtie \Delta R_2)$

Thank You!

ben@espresso.ai

