Self-Driving Database Management Systems

Paper:

Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin,
Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron,
Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor,
Dana Van Aken, Ziqi Wang, Yingjun Wu*, Ran Xian, Tieying Zhang

Carnegie Mellon University, *National University of Singapore
2017

Presentation:

George Wort
University of Cambridge
2017

Contributions

- Presents an architectural model for a “self-driving” database management system (DBMS).
  - Should allow DBMS to adapt without any human intervention.
  - Optimizes system for the predicted future workloads.
  - Measures effects of actions to better schedule deployment.
- Presents Peloton.
  - A skeleton implementation of the theoretical architecture.

Previous Self-Tuners

- Must prepare workload samples.
- Requires spare hardware to test on.
- Requires intuition into the DBMS’s internals.
- External to the DBMS.
  - Limited actions that can be taken.
- Reactionary as they cannot predict future workloads.
- Cannot take a holistic view that considers more than one problem at a time.
- Often require restarting on change.
- Many actions too slow.

Application Workloads

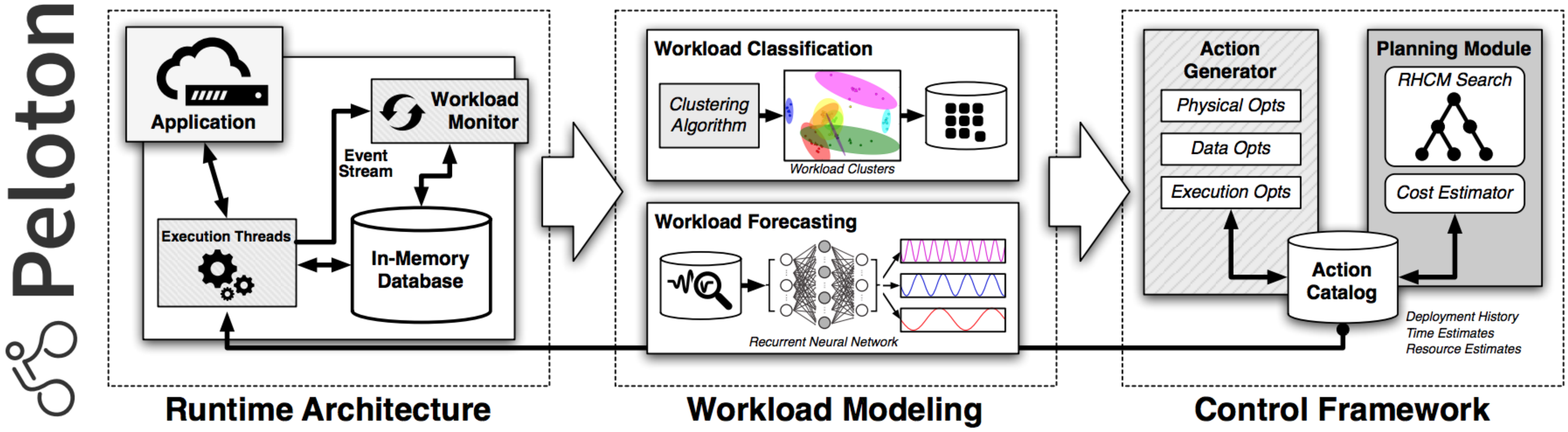
- Online Transaction Processing (OLTP)
  - Row-oriented.
  - Optimizes writes.
- Online Analytical Processing (OLAP)
  - Column-oriented.
  - Optimizes reads.
- Hybrid Transaction-Analytical Processing (HTAP)
  - Execute OLAP queries as soon as data is written.

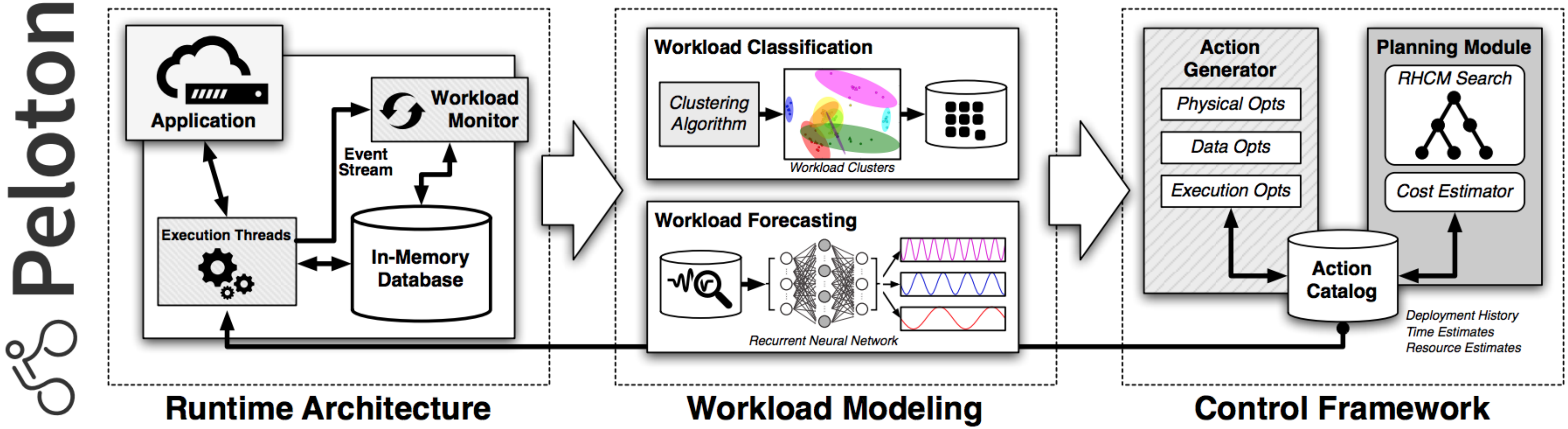
Application Workloads

- Could deploy separate databases.
  - Stream updates.
- Could optimize for different database segments.
- Self-driving DBMS needs to:
  - Forecast resource utilization trends.
  - Choose action to optimize database.
  - Deploy optimization at time of least impact.
- Cannot:
  - Require applications to be rewritten.
  - Rely on program analysis tools that only support certain programming environments.

Actions

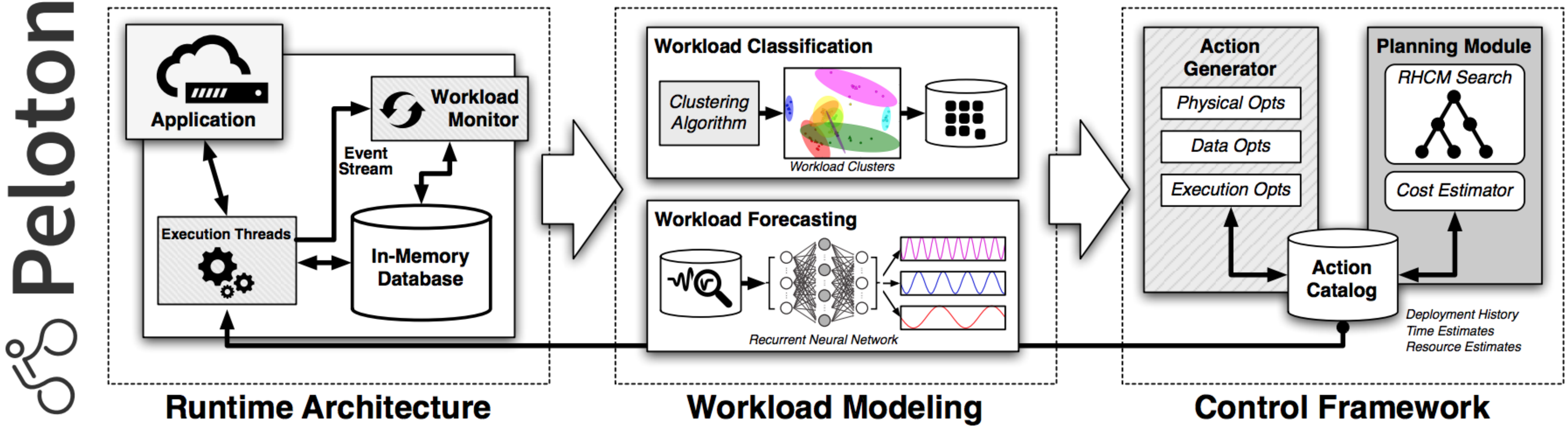
	Types	Actions
PHYSICAL	Indexes	AddIndex, DropIndex, Rebuild, Convert
	Materialized Views	AddMatView, DropMatView
	Storage Layout	Row→Columnar, Columnar→Row, Compress
DATA	Location	MoveUpTier, MoveDownTier, Migrate
	Partitioning	RepartitionTable, ReplicateTable
RUNTIME	Resources	AddNode, RemoveNode
	Configuration Tuning	IncrementKnob, DecrementKnob, SetKnob
	Query Optimizations	CostModelTune, Compilation, Prefetch


Peloton
Application
Workload Monitor
Event Stream
Execution Threads
In-Memory Database
Runtime Architecture
Workload Classification
Clustering Algorithm
Workload Clusters
Workload Forecasting
Recurrent Neural Network
Workload Modeling
Action Generator
Physical Opts
Data Opts
Execution Opts
Planning Module
RHCM Search
Cost Estimator
Action Catalog
Deployment History
Time Estimates
Resource Estimates
Control Framework



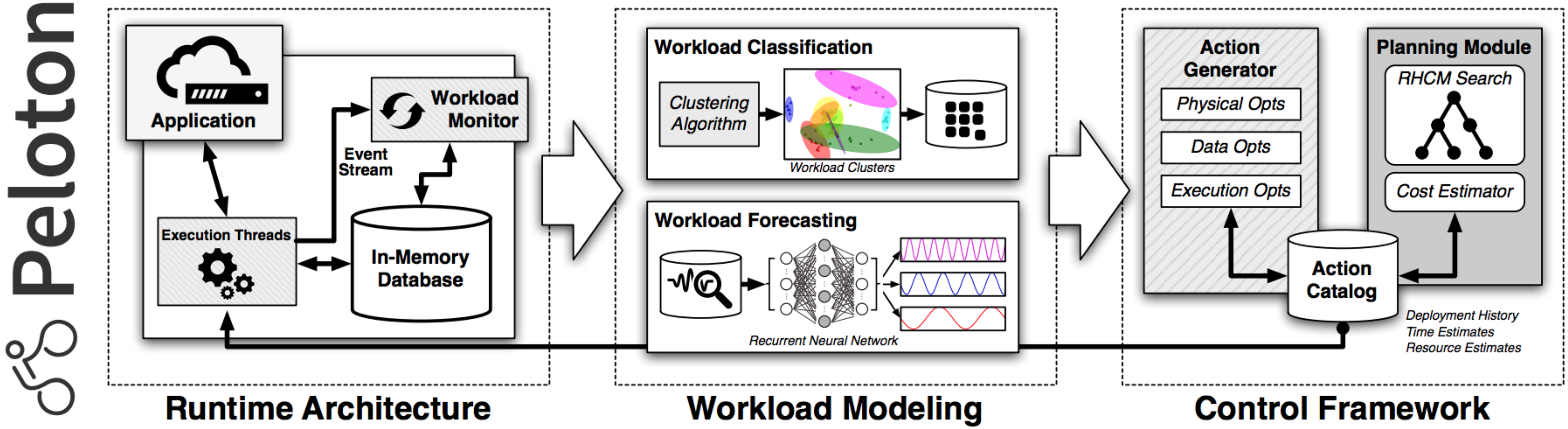
Workload Classification:

- Clusterer uses unsupervised learning. (DBSCAN algorithm)
- Can use runtime metrics or logical semantics.
  - Runtime more sensitive to changes in contents, design or concurrent workloads.
  - Logical semantics isn't as accurate.
- Uses standard cross validation to detect when clusters are no longer correct and require rebuilding.



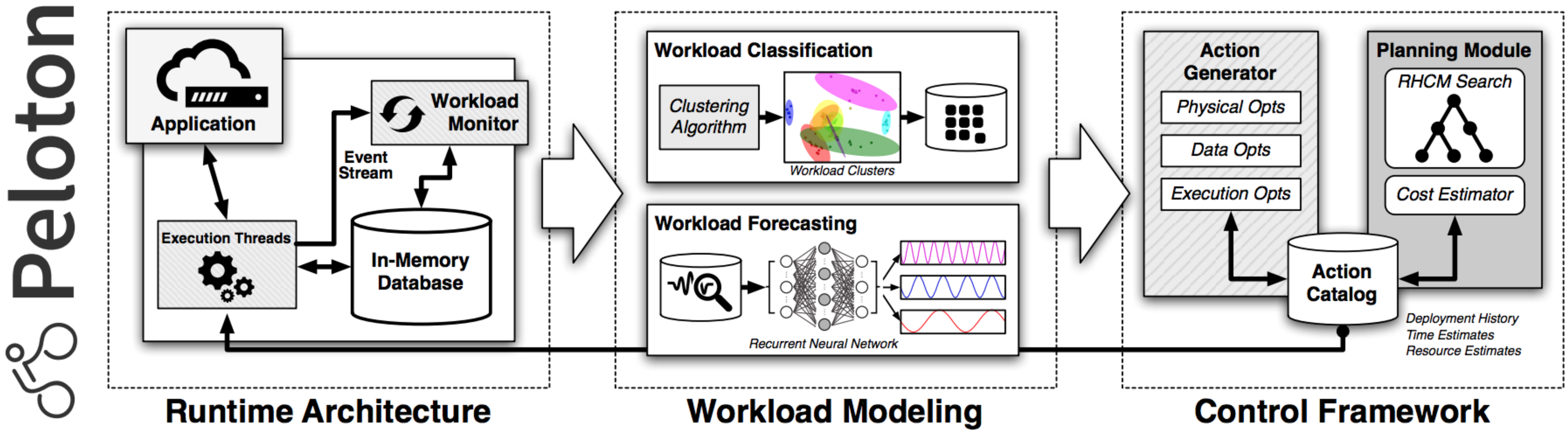
Workload Forecasting:

- Train forecast model that predicts each cluster's arrival rate.
- Identifies periodicity and data growth trends.
- Recurrent Neural Networks (RNNs) are effective at predicting time-series patterns for non-linear systems.
- Specifically uses Long Short-Term Memory (LSTM).
    - Contains special blocks that determine whether to retain old information and when to output it into the network.
- Maintains multiple RNNs that forecast workloads at different time horizons and interval granularities.
- Tracking all queries increases storage and training costs.



Action Generation:

- Searches for actions that might improve performance.
- Guided by forecasting model.
- Stores along with resource requirements and history of effects.



Action Planning:

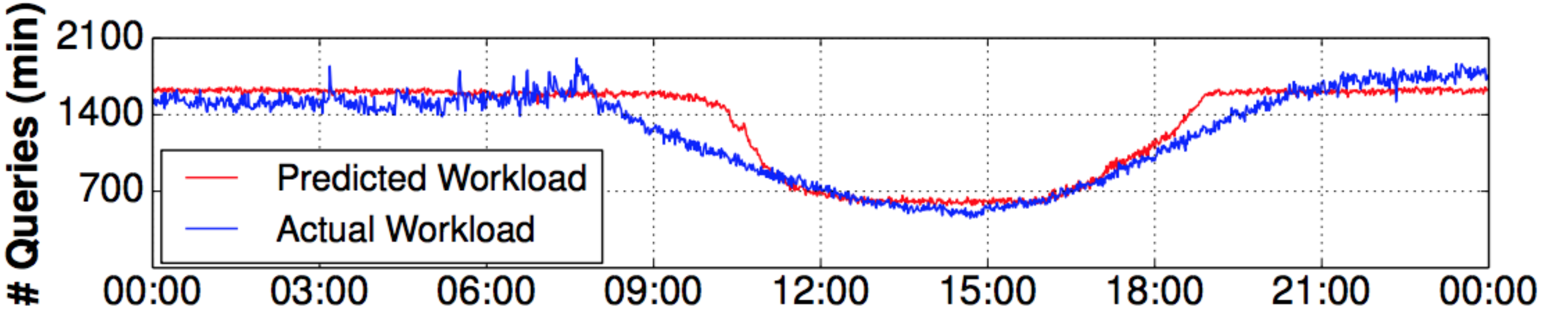
- Uses control theory, Receding-Horizon Control Model (RHCM).
- At each time epoch:
  - Estimates workload for time horizon using forecasts.
  - Searches for a sequence of actions that minimize objective function (latency).
  - Performs first action.
- Avoids recently invoked then reversed actions.
- Uses a cost-benefit model:
  - Cost is estimate of deployment latency and cost on performance.
  - Benefit is change in queries' latencies.
- Deploys actions in a non-blocking manner.

Peloton Implementation

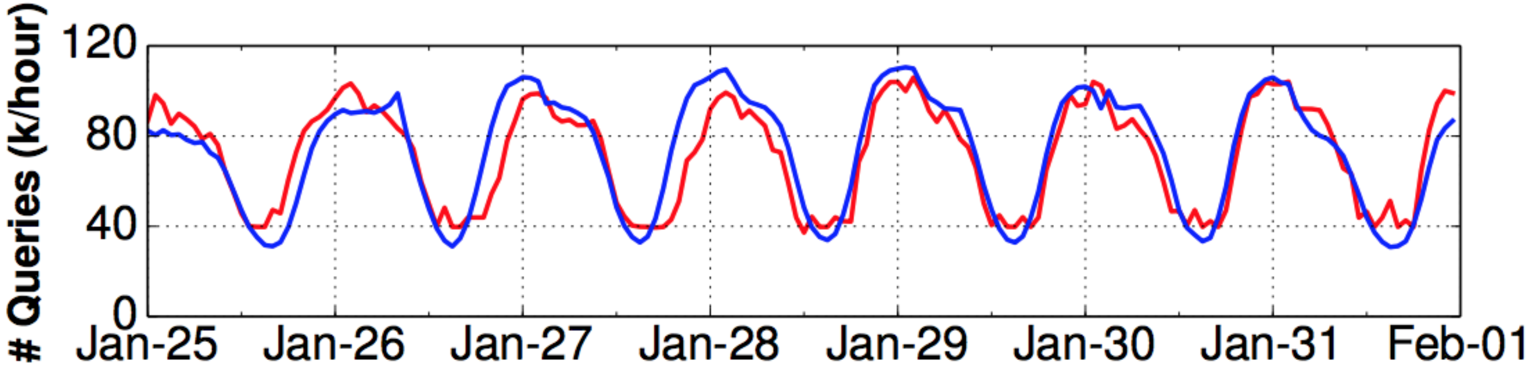
- Assumes queries are already clustered correctly.
  - Clusterer not tested.
- Integrated Google TensorFlow to perform workload forecasting.
  - Uses two stacked LSTM layers on input, connected to a linear regression layer.
  - Uses a 10% dropout rate to avoid over-fitting.
- Uses 1 hour time horizon with 1 minute granularity.
  - Input is per-minute workload over past 2 hours.
- Uses 24 hour time horizon with 1 hour granularity.
  - Input is previous day’s workload.

Peloton Implementation

- Uses 75% of a 4 week data set to train the model.
- Training took 11 and 18 minutes on a Nvidia GeForce GTX 980 GPU.
- Validates using other 25%.
- Predicts with 11.3% for 1-hour and 13.2% for 24-hour.



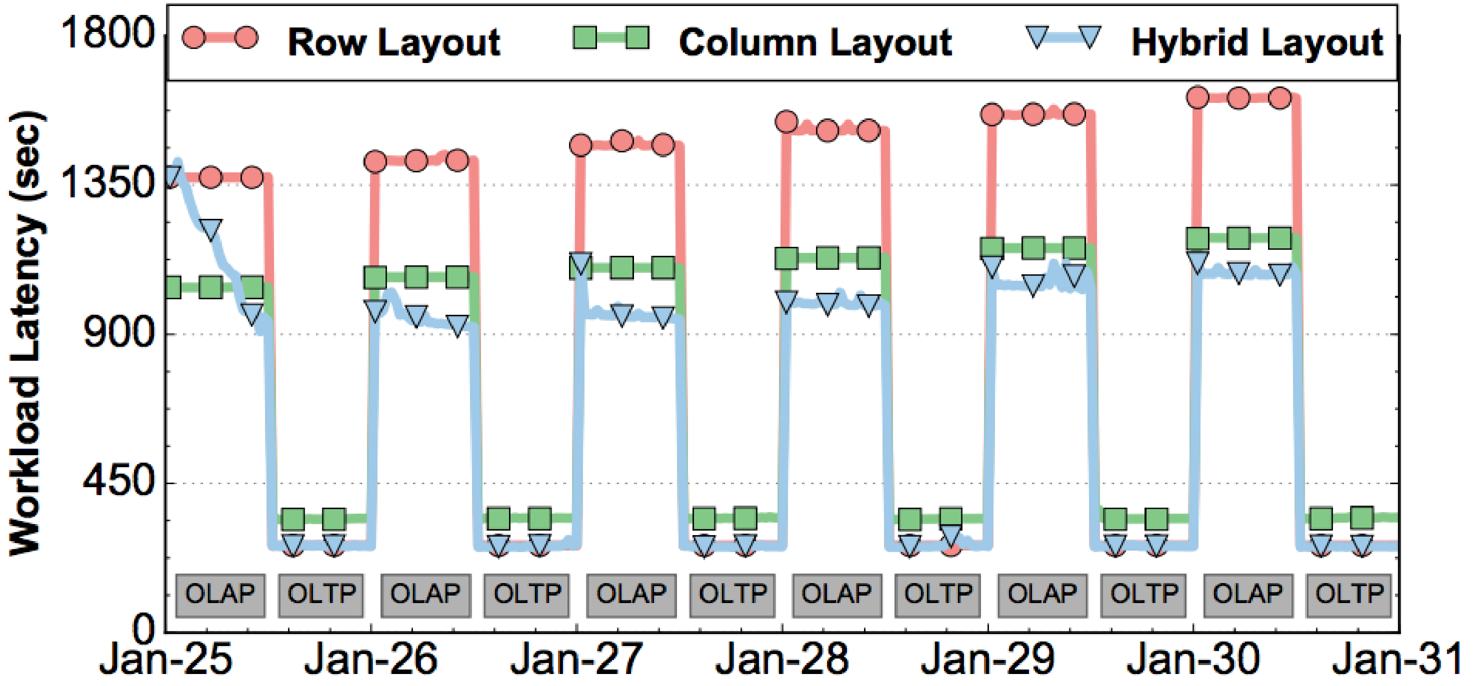
(a) RNN Forecast Model (24-Hour Horizon)



(b) RNN Forecast Model (7-Day Horizon)

Peloton Implementation

- Migrates table to row or column layout based on types of queries.
- Hot tuples are stored in a row-oriented layout.
- Cold tuples are stored in a column-oriented layout.



Criticisms

- Shows only a small gain over simply using the column layout.
- Peleton is only tested on a very predictable workload with simple behaviour patterns.
- Peleton is not tested under any drastic changes such as failures or erratic traffic.
- It claims to be able to do almost everything, yet is only shown to do one very simple change.
  - Could have been scheduled by an administrator.

Criticisms

- The cost of the extra work and resources for such are not properly addressed.
- The action generator is not tested at all as only one action is made available to Peleton.
- Assumes latency is most important metric.
- No kind of possible distribution of the database is mentioned.

Questions?