DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2018 // CHARITY HILTON

LECTURE #11: QUERY-BASED WORKLOAD FORECASTING FOR SELF-DRIVING DATABASE MANAGEMENT SYSTEMS

CREATING THE NEXT®

PAPER

- **Query-based Workload Forecasting for Self-Driving Database Management Systems**
  - *Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, Geoffrey J. Gordon*
  - Carnegie Mellon University

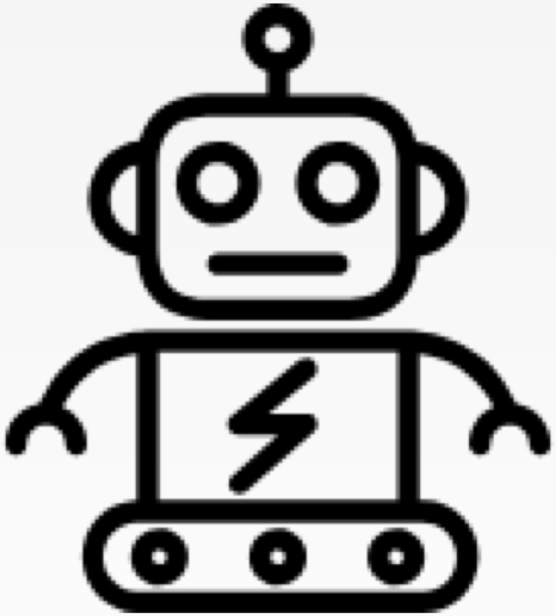
- **Key Topics**
  - Workload Forecasting
  - Self-Driving DBs

LINKS

- **Paper** - http://www.cs.cmu.edu/~malin199/publications/2018.forecasting.sigmod.pdf
- **Slides** - http://www.cs.cmu.edu/~malin199/publications/slides/forecasting-sigmod2018.pdf
- **Poster** - http://www.cs.cmu.edu/~malin199/publications/posters/forecasting-sigmod18-poster.pdf
- **Talk** - https://www.youtube.com/watch?v=ZHAyrsVZfiU
- **Code** - https://github.com/malin1993ml/QueryBot5000

AGENDA

- Problem Overview
- Background
- Key Ideas
- Technical Details
- Experiments
- Discussion



PROBLEM OVERVIEW

INTRODUCTION

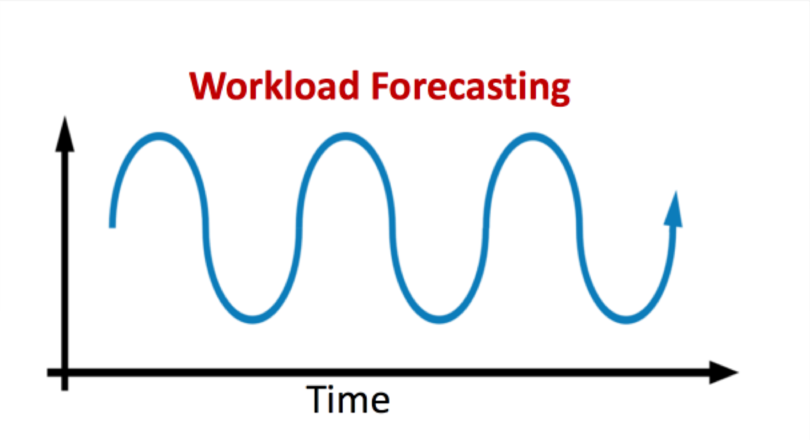
- DBMSs have become more difficult for DBAs to manage
  - Data growth
  - Application usage spikes
  - Hardware issues
- An autonomous DBMS would be able to use machine learning and reduce the need for manual tuning

MOTIVATION

- Workload forecasting is a first step in building self-driving DBMSs
- Optimizations can be applied against future queries to allocate DBMS resources to where they are needed, e.g. indexes, partitioning
- Systems should be hardware and design agnostic

MAIN APPROACH

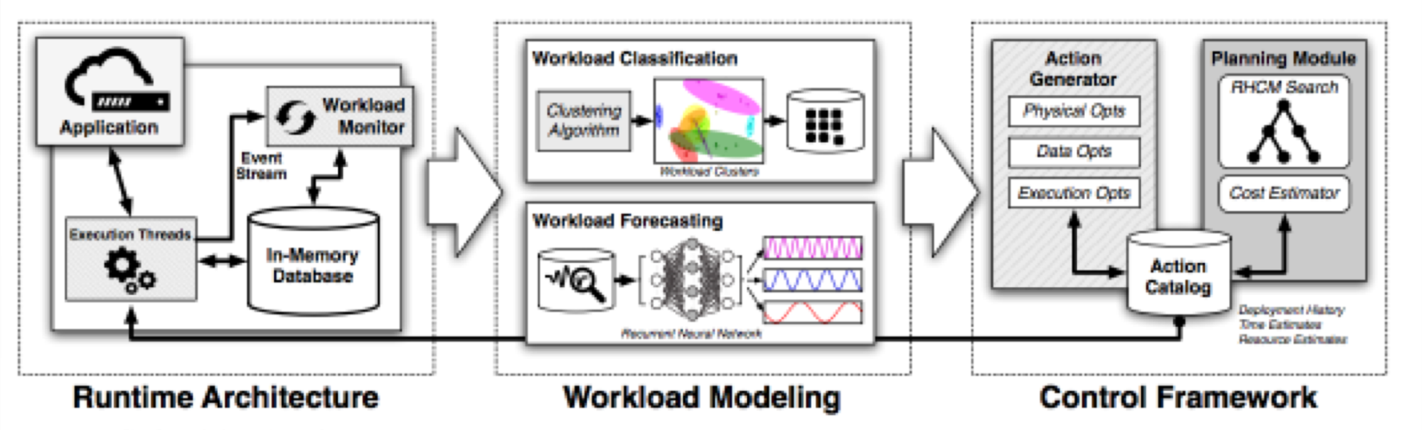
- Introduce *QueryBot 5000* Pipeline!
  1. **Pre-Processor:** Map query to template
  2. **Clusterer:** Cluster templates based on arrival time
  3. **Forecaster:** Use predictive models to predict query patterns
  4. **Evaluate:** Based on automatic index creation



BACKGROUND

AUTOMONOUS DATABASES

1. Monitoring – system status effectiveness of optimizations
2. Workload Forecasting (this paper)
3. Planning – Determine which optimizations to apply



https://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf

WORKLOAD FORECASTING

- Should predict the workload in the future
- Challenges in modern DBMSs:

1. **Application queries have different arrival rates**
   - Arrival rate patterns need to be identified

2. **Composition and volume of queries change over time**
   - Models will need to be recomputed if the patterns change too much

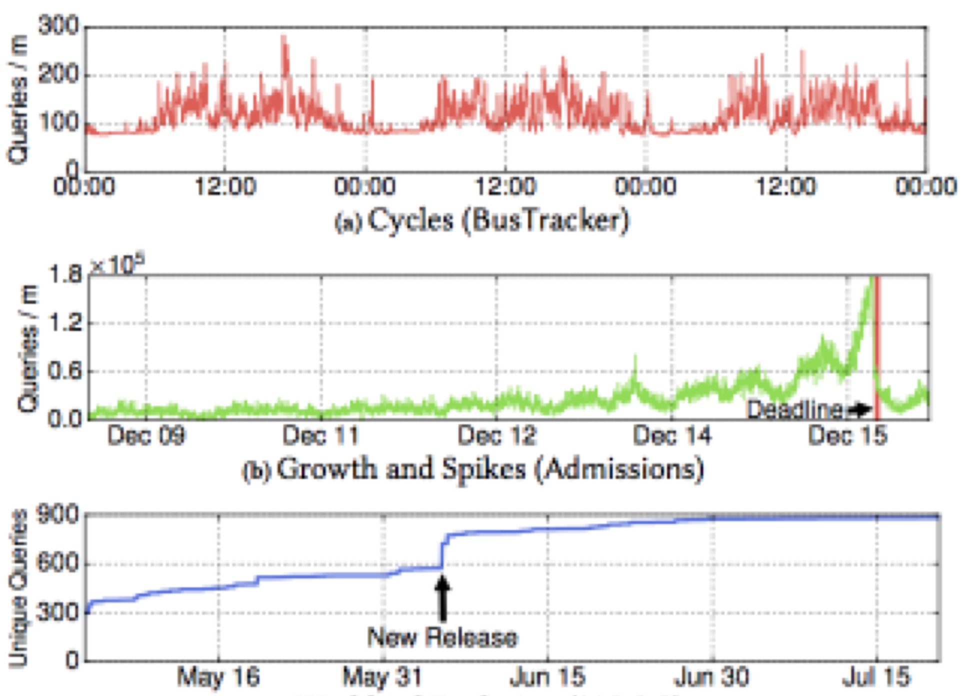
GOALS

- Accurate
- Able to identify patterns
- Able to be performant without interfering with DBMS
- Able to work on a variety of time horizons

http://www.cs.cmu.edu/~malin199/publications/slides/forecasting-sigmod2018.pdf

SAMPLE WORKLOADS

- **Admissions** – university admissions website
- **BusTracker** – mobile app for tracking public transit
- **MOOC** – Web app that offers online courses



(a) Cycles (BusTracker)
(b) Growth and Spikes (Admissions)
(c) Workload Evolution (MOOC)

CYCLES

- Many applications will have more activity in accordance with human behavior, as such modern DBMS workloads are **cyclic**:
  - Applications can have more activity when people are awake during the day time
  - Applications can have more activity during a certain time of year such as when deadlines approach
  - Applications can have more or less activity when new features and/or bugs are released/introduced

GROWTH AND SPIKES

- Query volume generally increases over time
- Applications gain more users, data, etc.
- Spikes occur during popular events or real-life deadlines

WORKLOAD EVOLUTION

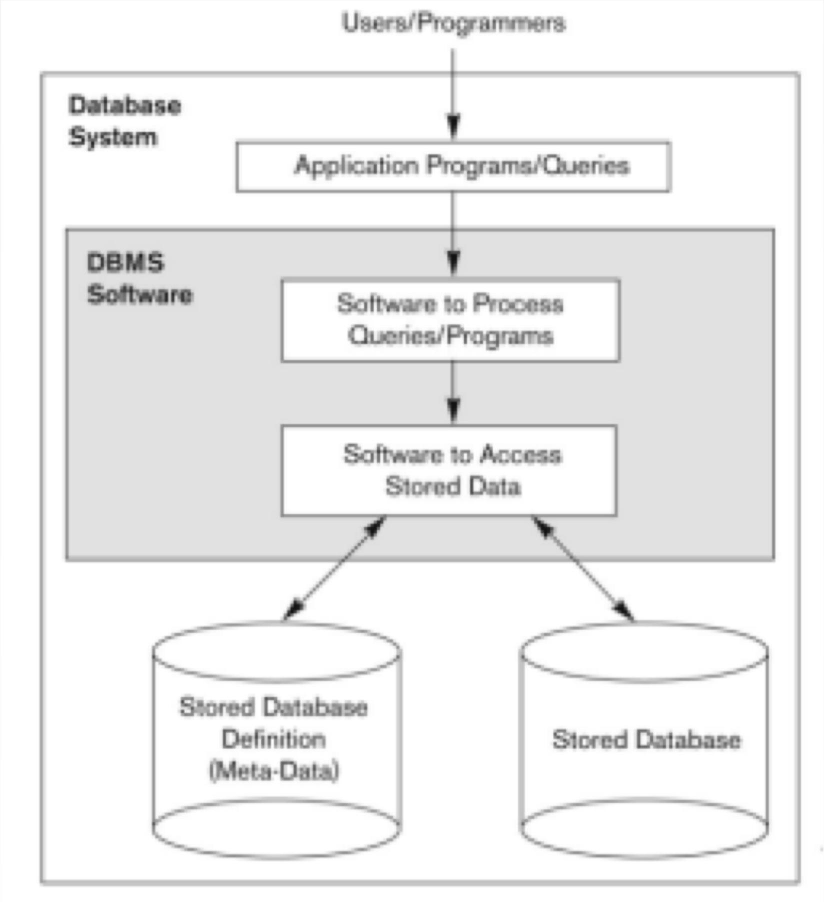
- Database workloads change over time
- This can be related to new users or new features

BACKGROUND DISCUSSION

- There are a variety of workload patterns that a workflow forecasting system must address
- Systems can also have specific sub-groups that must be addressed
- In addition, systems have millions of queries per day, so there is a tradeoff between speed and accuracy of the model

KEY TERMS: DBMS

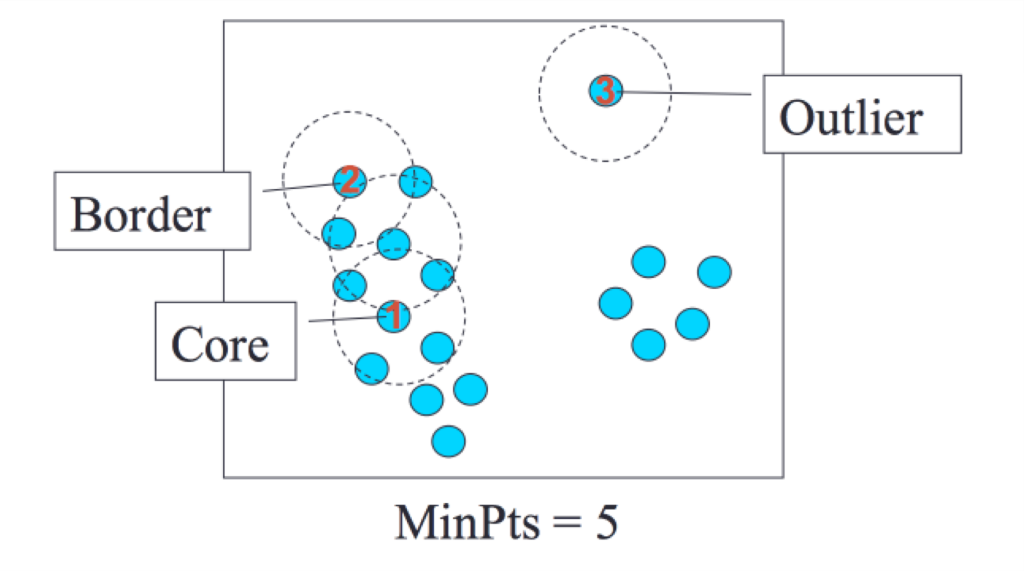
- OLTP or online transaction processing
  - Most software with user interaction is classed as OLTP
- OLAP or online analytical processing
  - Business analytics, reporting and data mining



Elmasri, Ramez, and Shamkant Navathe. *Fundamentals of database systems*. Addison-Wesley Publishing Company, 2010.

KEY TERMS: CLUSTERING

- DBSCAN - Density-based spatial clustering of applications with noise
  - Must define radius and minimum points
  - Core objects have a high density
  - Outliers aren't close to any cluster



http://www.cs.fsu.edu/~ackerman/CIS5930/notes/DBSCAN.pdf

KEY TERMS: ML MODELS

- Types: Linear / Memory / Kernel (non-linear)
- Ensemble models
  – Combine multiple models
- Parametric Models
  – Finite set of parameters
- Non-parametric Models
  – No predefined weights
  – ‘Black box’ model
  – Longer memory, doesn’t generalize

KEY TERMS: INDEXING

- **Primary index** – set of fields that determine uniqueness
- **Foreign key index** – set of fields between two tables to ensure referential integrity
- **AutoAdmin** – Tool for automatically optimizing database indexes

KEY TERMS: FORECASTING

- Prediction Horizon - how long into the future can a model predict (e.g. 1 hour or 1 year)
  - Longer horizons == less accurate
- Prediction Interval – intervals at which queries are calculated and clustered
  - Lower interval == more accurate (but overfitting and larger memory footprint)

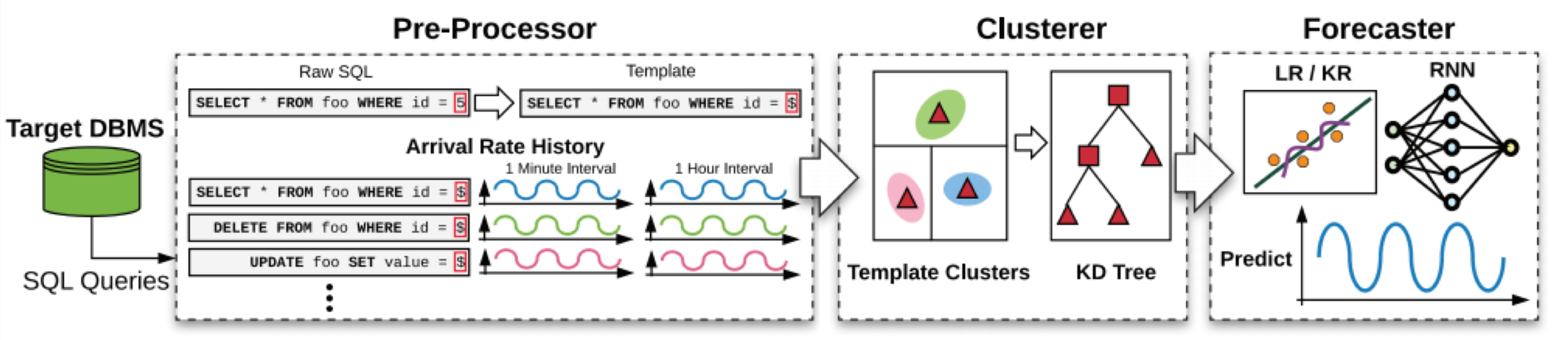
KEY IDEAS

QUERYBOT 5000

- This paper introduces QueryBot 5000 as a workload forecasting module
- Can work externally or embedded in the DBMS
- It is lightweight; has its own internal database and doesn't interfere with transactions
- QB5000 has 3 components: **Pre-Processor**, **Clusterer** and **Forecaster**

TECHNICAL DETAILS

QUERYBOT 5000

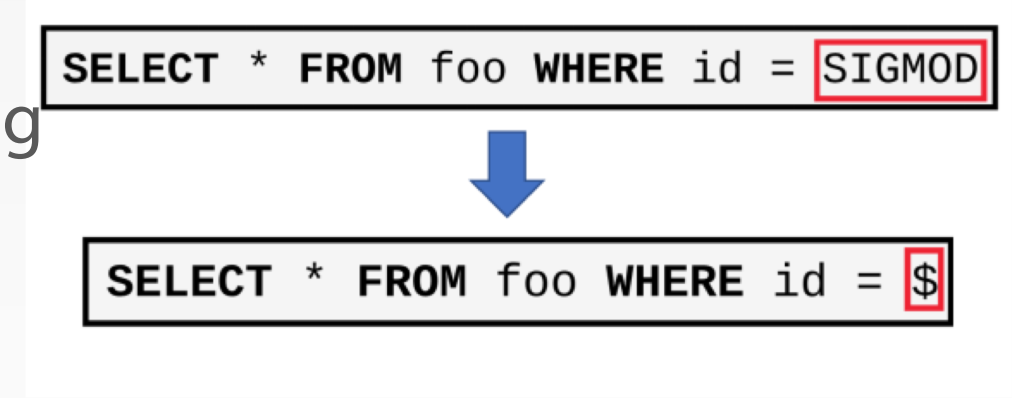


PRE-PROCESSOR

- OLTP - Assumes most queries are ran via software applications using similar constructs
- OLAP – Assumes queries accessed via dashboards and reports
- QB5000 is able to aggregate and characterize queries based on templates to reduce the number of queries
- Reduces query # from **millions** to **thousands**

PRE-PROCESSOR STEPS

- All values are converted to constants
  - Values in WHERE, SET in UPDATE, and INSERT
- Converts to an abstract syntax using DBMS parser
- Cleans up formatting, e.g. parentheses
- Checks for semantic equivalence
- Captures templates along with arrival time



CLUSTERER

- Models built using 1000s of templates still take minutes to train
  - Need to further reduce template count
- Takes templates, clusters them, and further reduces the state space
- Must use features that aren't overly dependent on any one DBMS system

PRE-PROCESSOR EXAMPLES

```
19193,"SELECT distinct a.agency_id FROM m.agency a, m.calendar c, m.trip t WHERE c.agency_id = a.agency_id AND t.agency_id
= a.agency_id AND a.avl_agency_name =  @@@ AND t.trip_id =  @@@ AND ((SELECT extract(epoch FROM c.start_date)*#)) <= # AND
((SELECT extract(epoch FROM c.end_date+#)*#)) >= # "
2017-01-25 05:00:00,11
2017-01-25 05:02:00,12
2017-01-25 05:05:00,12
2017-01-25 05:07:00,13
2017-01-25 05:10:00,13
2017-01-25 05:13:00,14
2017-01-25 05:15:00,14
```

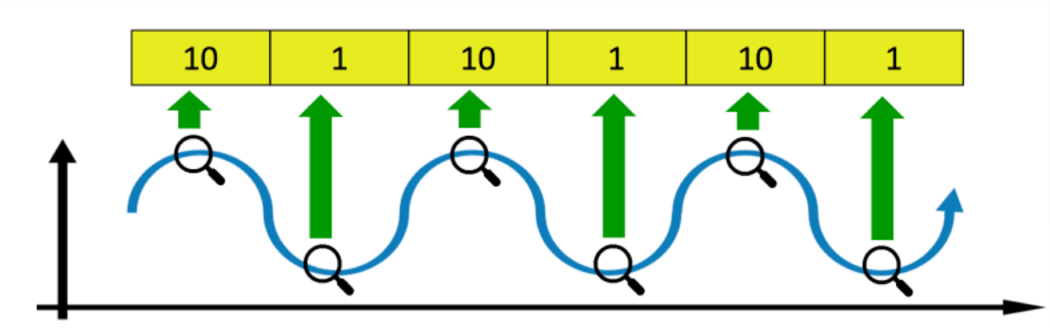
```
299,"select st.trip_id, st.stop_sequence, st.estimate_source, st.fullness, st.departure_time_hour, st.departure_time_minute
, s.stop_lat, s.stop_lon, t.direction_id, t.route_id, r.route_short_name from m.stop AS s RIGHT JOIN m.stop_time AS st  ON
st.agency_id = s.agency_id AND st.stop_id = s.stop_id LEFT JOIN m.trip AS t ON t.agency_id = st.agency_id AND t.trip_id = s
t.trip_id LEFT JOIN m.route AS r ON t.agency_id = r.agency_id AND t.route_id = r.route_id WHERE st.estimate_source in ( @@@
,  @@@) AND st.agency_id = $# AND (((departure_time_hour * # + departure_time_minute) >= ($#-#)  AND (departure_time_hour *
 # + departure_time_minute) <= ($#+#)) OR ((departure_time_hour * # + departure_time_minute) >= ($#-#)  AND (departure_time
_hour * # + departure_time_minute) <= ($#+#)))  order by st.stop_sequence"
2017-01-25 00:01:00,1
2017-01-25 00:11:00,1
2017-01-25 00:16:00,1
2017-01-25 00:18:00,1
2017-01-25 00:20:00,1
2017-01-25 00:26:00,1
```

CLUSTERER FEATURE SELECTION

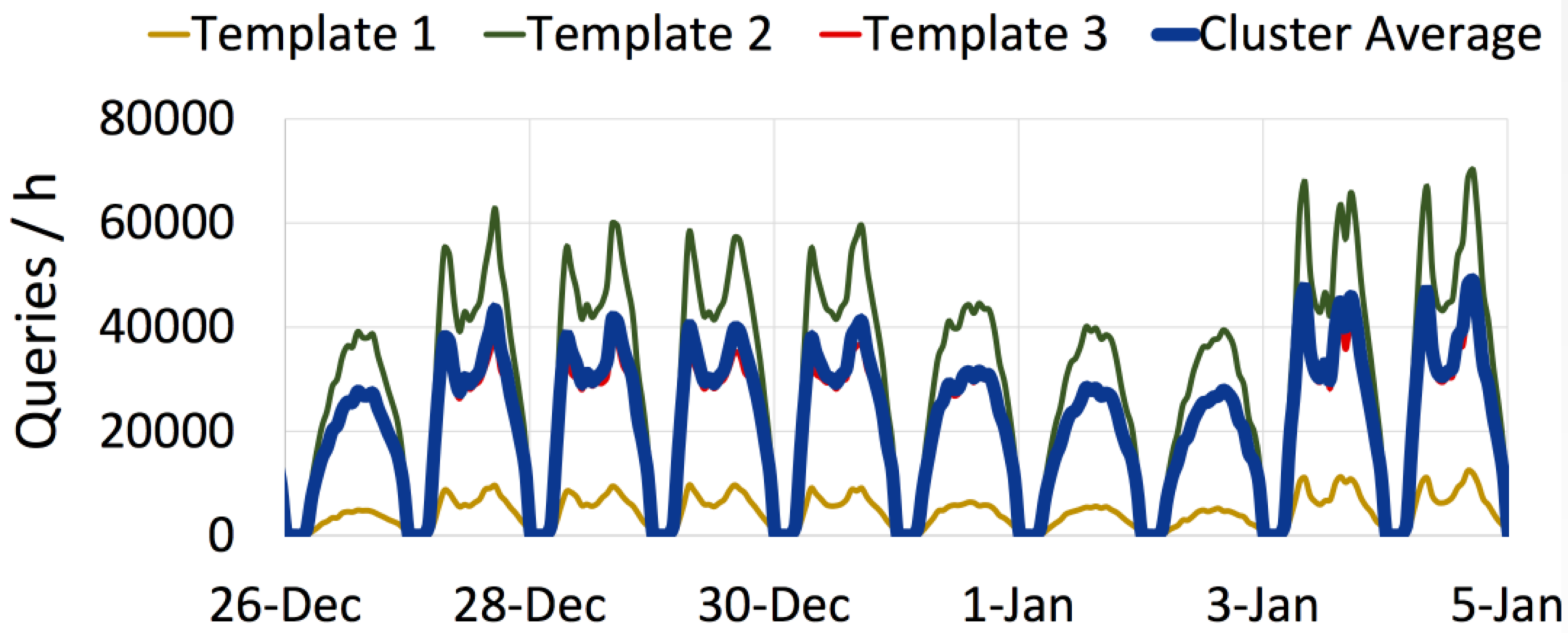
- **Physical** – Runtime metrics, concurrent queries, tuples read, latency, etc.
- **Logical** – Types of queries, columns, joins, etc.
- **Arrival Rate** – Average arrival rate of a template within a cluster

CLUSTERER FEATURE SELECTION

- ~~Physical~~ – Too dependent on DBMS
- ~~Logical~~ – Proven Inefficient
- **Arrival Rate** – Best feature for QB5000!
  - Because we're predicting workload
  - Randomly sampled based on cosine similarity



ARRIVAL RATE HISTORY



Bus Tracking App

ONLINE CLUSTERING

- Modified version of DBSCAN
- QB5000 looks at object centers not just any core object
- Threshold to determine cosine similarity (improves performance)

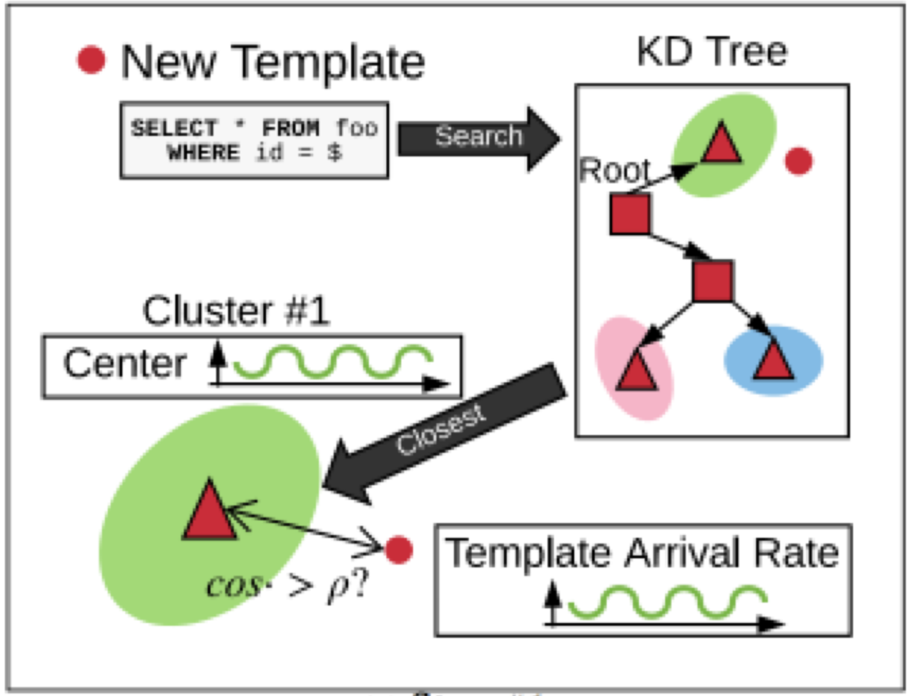
$$\rho \ (0 \le \rho \le 1)$$

- Adjusts clusters without requiring a warmup period

CLUSTERING STEP #1

Check highest similarity score, use kd-tree to find closest center, then updates center.

If no close clusters, create a new cluster.

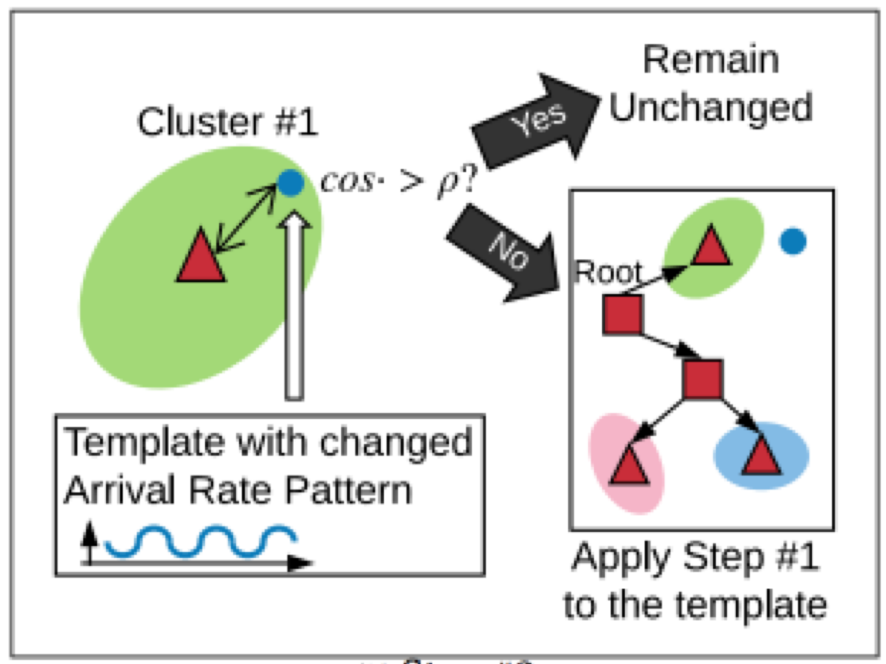


(a) Step #1

CLUSTERING STEP #2

Checks previous points in clusters and make sure they still meet > *p* with cluster center.

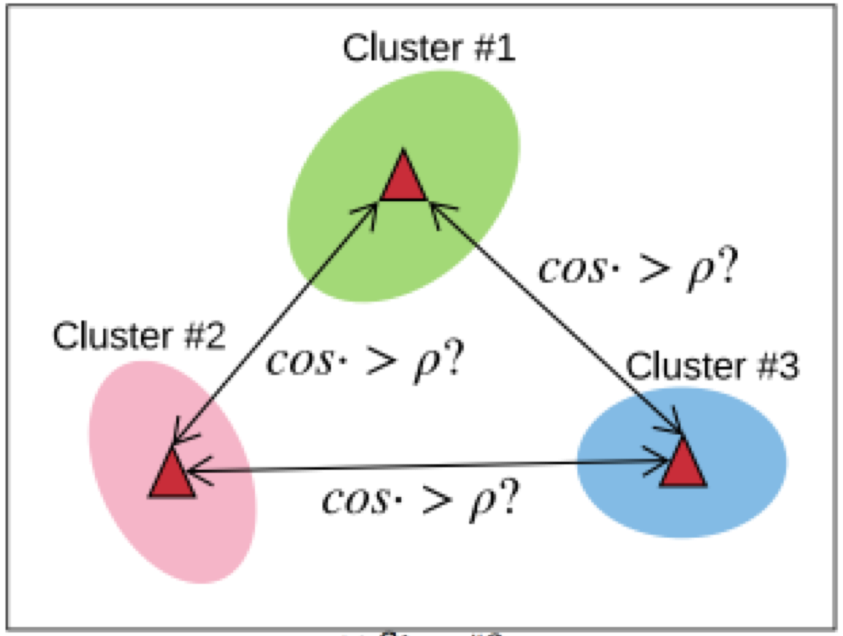
If cluster must be re-centered, that happened in the next execution.



(b) Step #2

CLUSTERING STEP #3

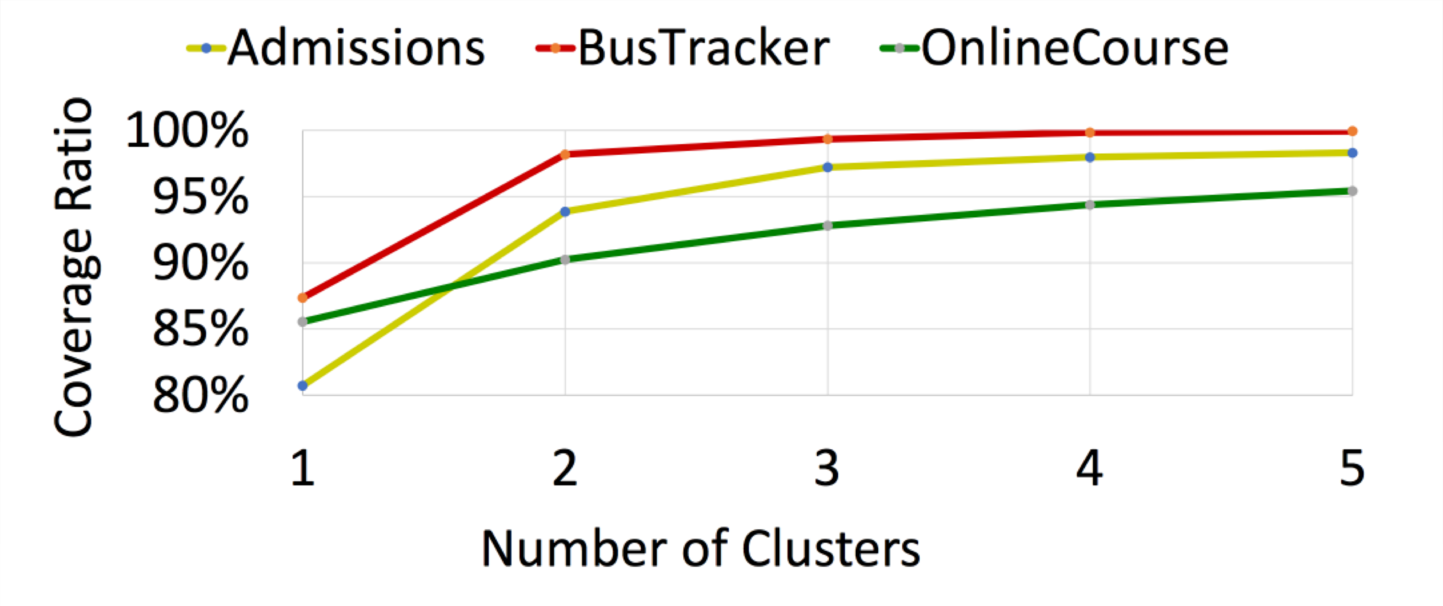
Computes similarity and merge two clusters if centers have cosine similarity > *p*.



(c) Step #3

QB5000 CLUSTER PRUNING

- Focus on large clusters, and ignore outliers.
- Top 5 clusters cover up to 95% of queries



FORECASTER

- Final phase of QB5000
- Predicts arrival time of queries
- DBMSs can use this information to run optimizations

FORECASTER

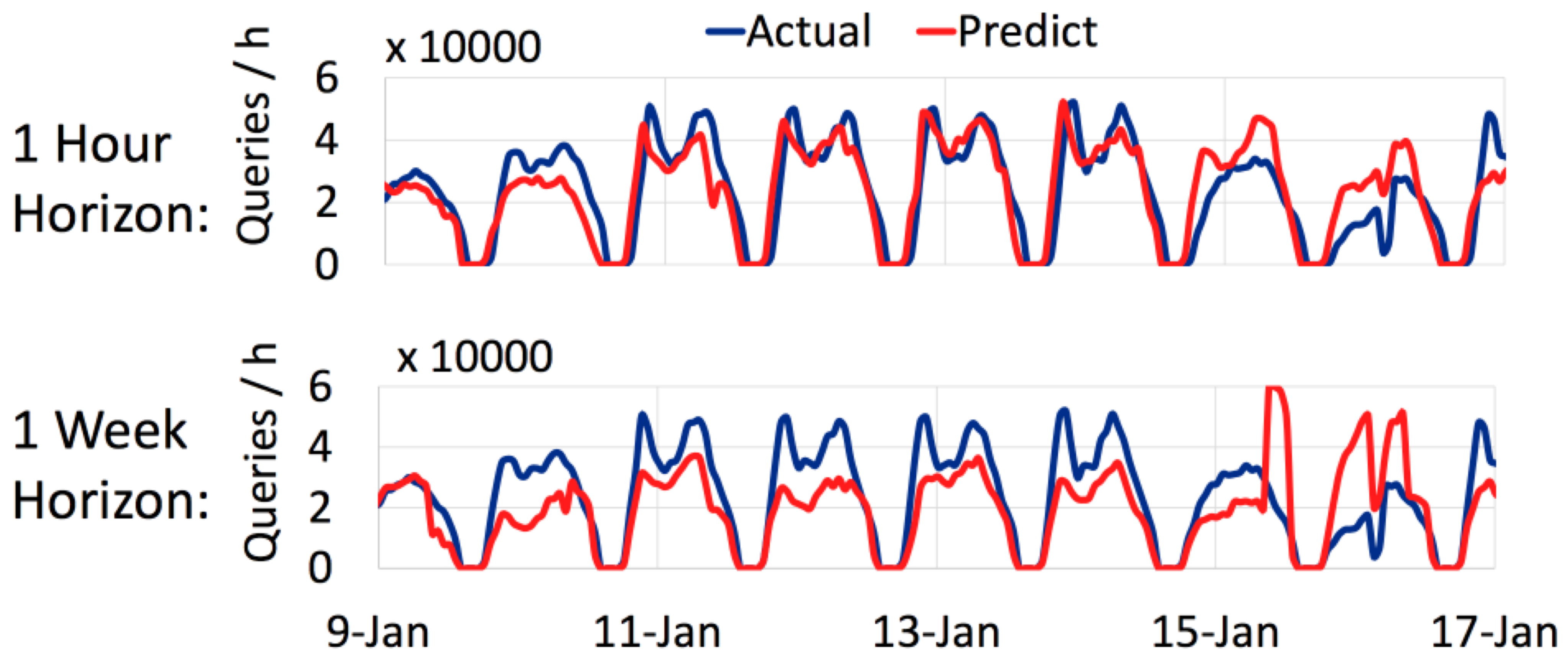
- Linear - good at short term, simpler problems
- Memory – good at complex problems, overfitting
- Kernel – Non-linear, good at predicting spikes
- Ensemble – Combined models

	LR	ARMA	KR	RNN	FNN	PSRNN
Linear	✓	✓	✗	✗	✗	✗
Memory	✗	✓	✗	✓	✗	✓
Kernel	✗	✗	✓	✗	✗	✓

FORECASTER MODELS

- Linear Regression – regresses the arrival rate based on the past
- Recurrent Neural Network – Uses LSTM, good for long term non-linear patterns, has longer memory
- QB5000 used ensemble method to combine LR + RNN for average prediction…except
- Kernel Regression to handle spikes

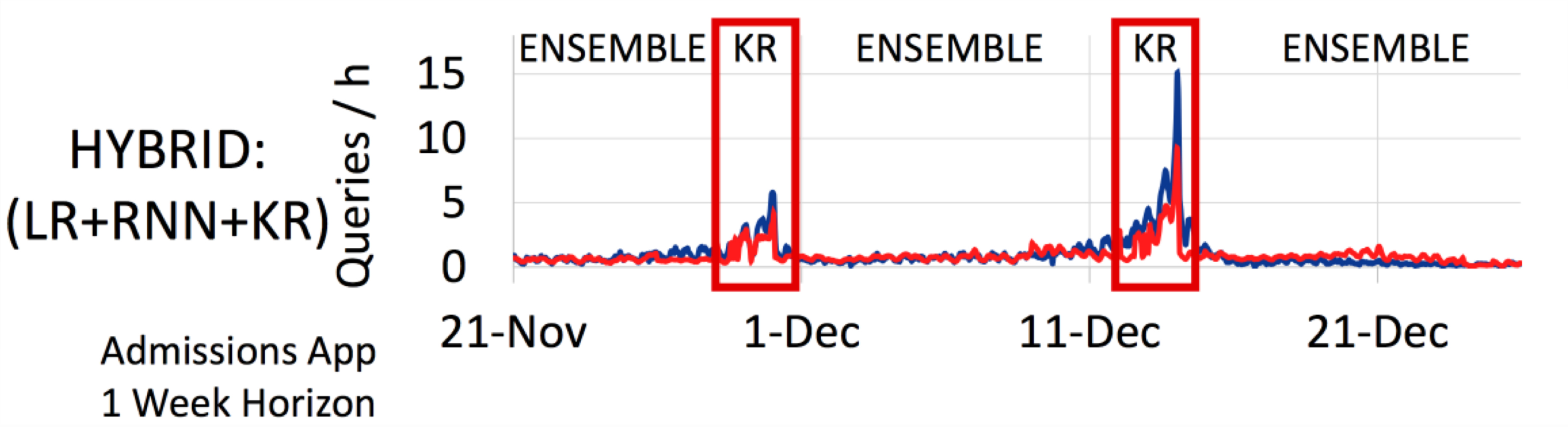
FORECASTER RESULTS



Bus Tracking App

FORECASTER MODELS

- Hybrid – Ensemble (LR + RNN) + KR
- Ensemble - better overall
- KR – better during spikes



EXPERIMENTS

EXPERIMENTAL ANALYSIS

- Used sklearn, PyTorch and Tensorflow
- Experiments:
  1. Number of Clusters
  2. Prediction Accuracy
  3. Spike Prediction
  4. Prediction Interval
  5. Computation and Storage
  6. Automatic Indexing
  7. Logical vs. Arrival Rate

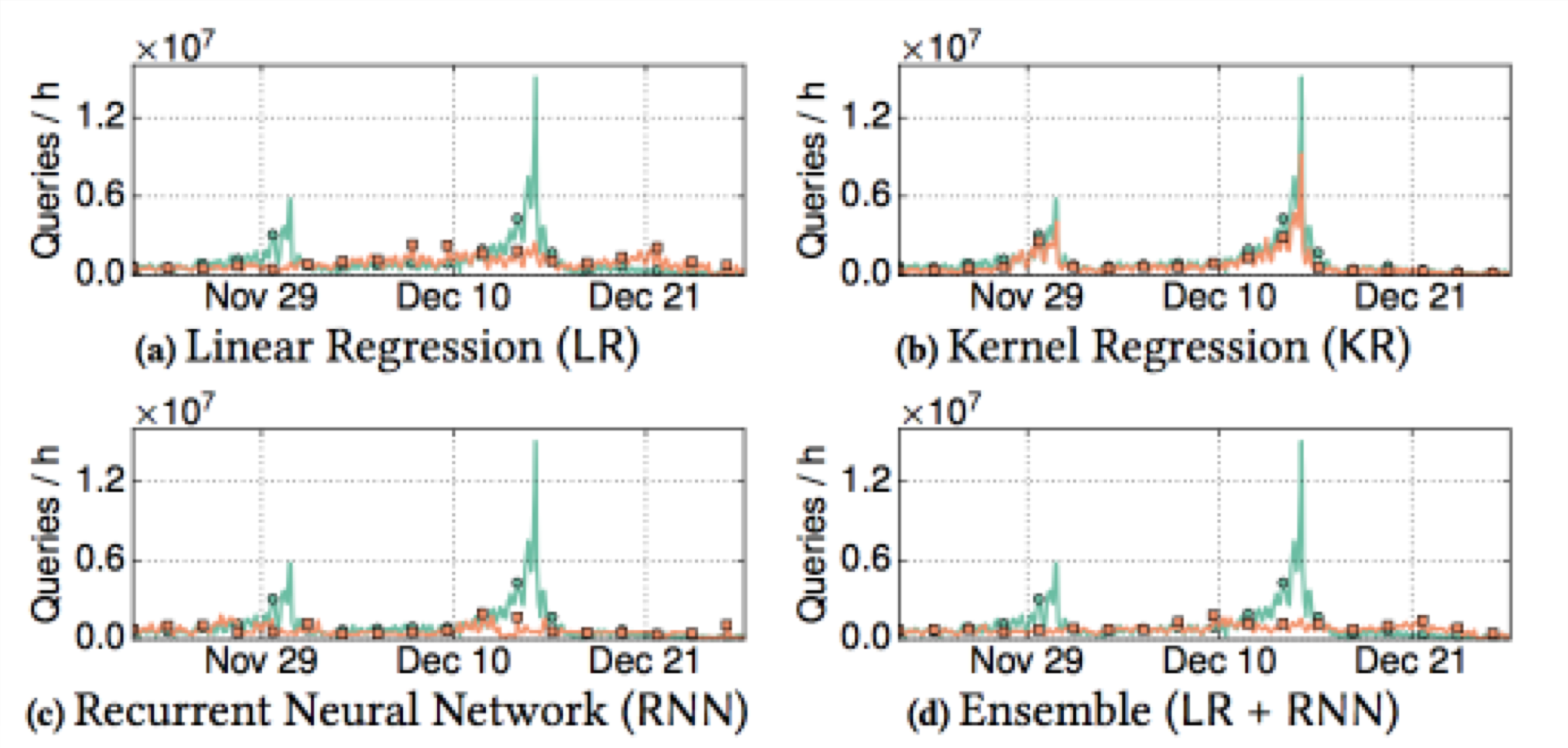
NUMBER OF CLUSTERS

- Goal is to find a the smallest number of high volume clusters
- Set threshold to p=0.8, which does incremental clustering 1x/day
- This covers up to 95% of all queries using less than 5 clusters
- Very few changes in Admissions and BusTracker in subsequent days

PREDICTION ACCURACY

- Use log of MSE (mean-squared error), smaller is better
- Want to avoid models that are overly sensitive to hyperparameters (fixed for QB5000)
- Evaluated ARMA, FNN, PSRNN in addition to previously mentioned models
- Smaller horizons do better with LR
- Horizons >1 day do better with RNN
- Ensemble is the best overall accuracy, but doesn't work on spikes as discussed

PREDICTION ACCURACY

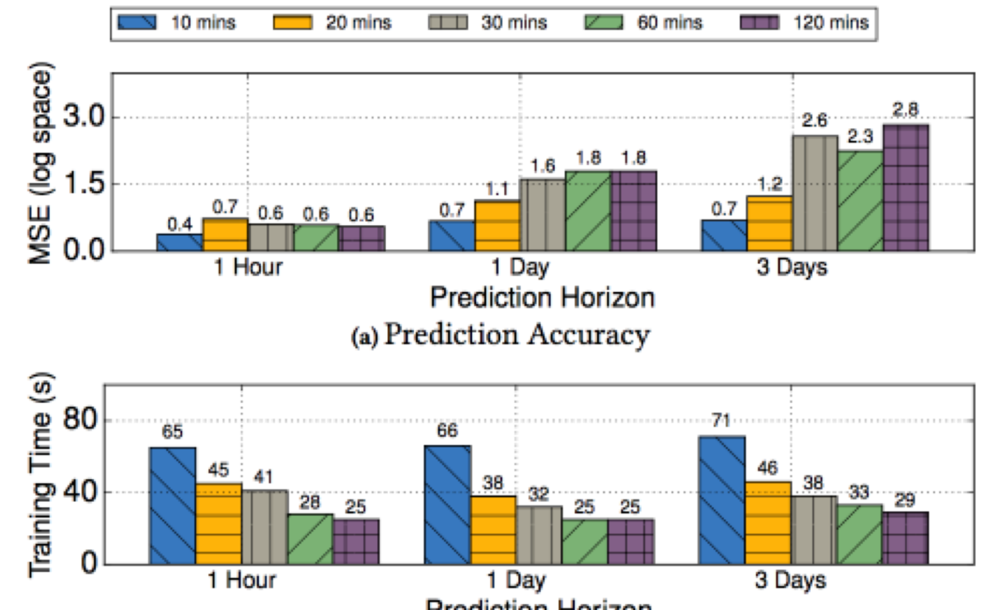


SPIKE PREDICTION

- Ensemble model is unable to predict spikes
- Both LR and RNN likely to get stuck in local optima
- Kernel regression is the only method able to detect spikes
- Used 1-hour intervals and PCA, kernel regression was easily able to identify spikes

PREDICTION INTERVAL

- KR uses 1-hr intervals by design
- Accuracy increases on smaller intervals, but longer intervals faster to train
- Tradeoffs
- Settled on 1-hr intervals



(a) Prediction Accuracy

(b) Training Time (GPU)

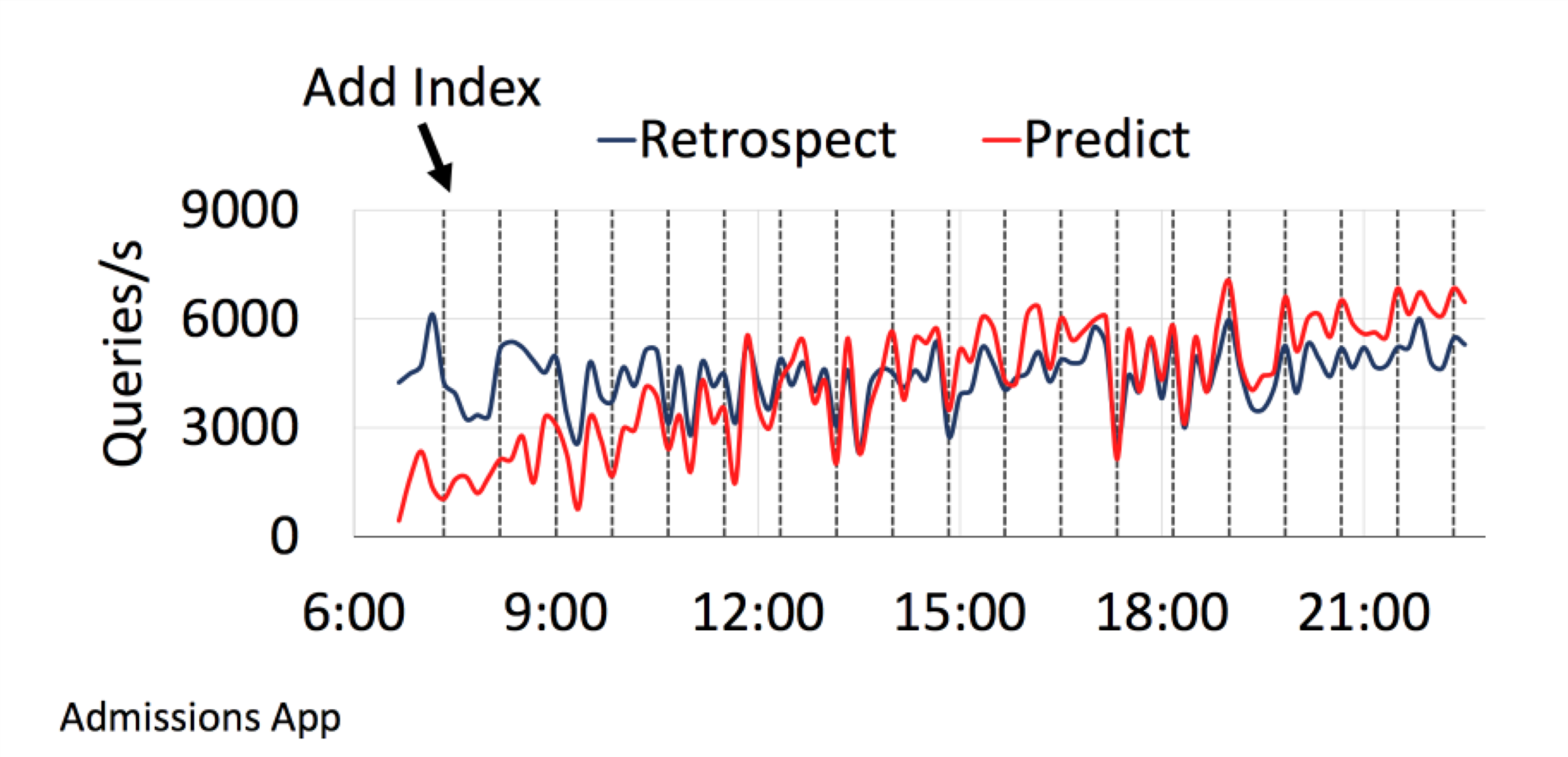
COMPUTATION AND STORAGE

- Pre-processor – time to template and query
- Clusterer – Time to recalculate clusters
- Forecaster
  - LR – smallest and fastest to train
  - RNN – slowest to train
  - KR – largest memory footprint

AUTOMATIC INDEXING

- QB5000 in action!
- Workloads initialized with primary key indexes
- Compared automatic with static indexes, adding them at hourly intervals using AutoAdmin
  - Static performs better initially, but then automatic outperforms

AUTOMATIC INDEXING



LOGICAL VS. ARRIVAL RATE

- Evaluated automatic indexing against logical inputs vs. arrival rate
- ~20% slower for both workloads
- Why?
  - Logical features are poor at determining template similarity
  - Logical features have multiple arrival rate patterns and are hard for models to predict

DISCUSSION

RELATED WORK

- Tools to identify trends for scaling and provisioning
- DBSeer – Offline what-if analysis for workload changes
- DBSherlock – Identify causes of anomalies
- Markov models to predict SQL queries (but don't model workflows)
- Other works look at runtime metrics

STENGTHS

- Lengthy comparison of models
- Lays framework for autonomous DBMS
- Scalable in relation to counterparts
- DBMS Independent
- Hybrid model is able to handle most patterns with good accuracy, works on long and short term horizons

WEAKNESSES

- Will cluster pruning degrade performance over time?
- Is the query pre-processor DBMS agnostic?
- Still has potential to be sensitive to workload changes
- How is the workload interval determined?
- Do you get diminishing returns with auto-indexes, i.e. is it worth the calculation overhead overt time?
- What about overhead time for building indexes? Space constraints?

DISCUSSION QUESTIONS

- Are there any other things you would have evaluated for?
- How can machine learning be used in other ways to optimize DBMSs?
- Could other inputs be considered like semantics?
- What other ways could QB5000 used for optimization?
- Good for understanding how ML can be used to optimize DBMSs
- How does it work with Cloud DBs?
- What is the benefit when using enterprise DBs that already have auto-indexing?

BIBLIOGRAPHY

- Ma, Lin, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. "Query-based Workload Forecasting for Self-Driving Database Management Systems." In *Proceedings of the 2018 International Conference on Management of Data*, pp. 631-645. ACM, 2018.
- http://www.cs.cmu.edu/~malin199/publications/slides/forecasting-sigmod2018.pdf
- Elmasri, Ramez, and Shamkant Navathe. Fundamentals of database systems. Addison-Wesley Publishing Company, 2010.
- http://www.cs.fsu.edu/~ackerman/CIS5930/notes/DBSCAN.pdf
- Pavlo, Andrew, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon et al. "Self-Driving Database Management Systems." In *CIDR*. 2017.