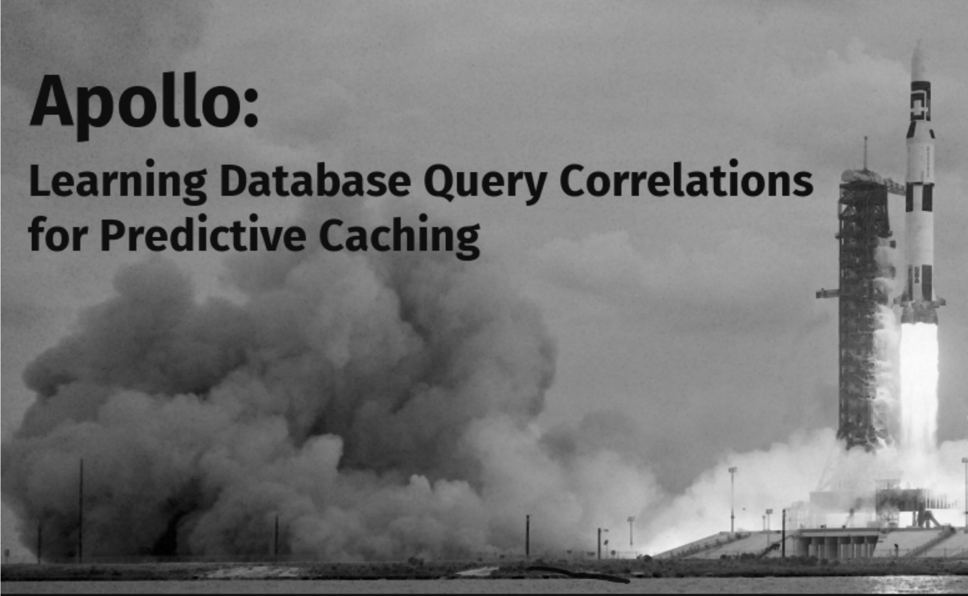


Apollo:

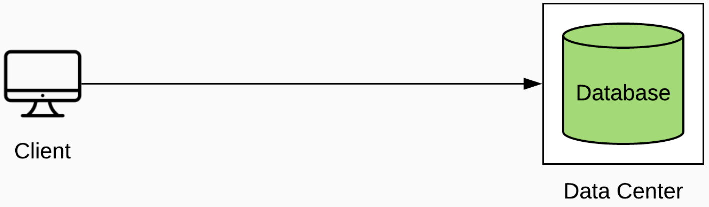
Learning Database Query Correlations for Predictive Caching

Brad Glasbergen

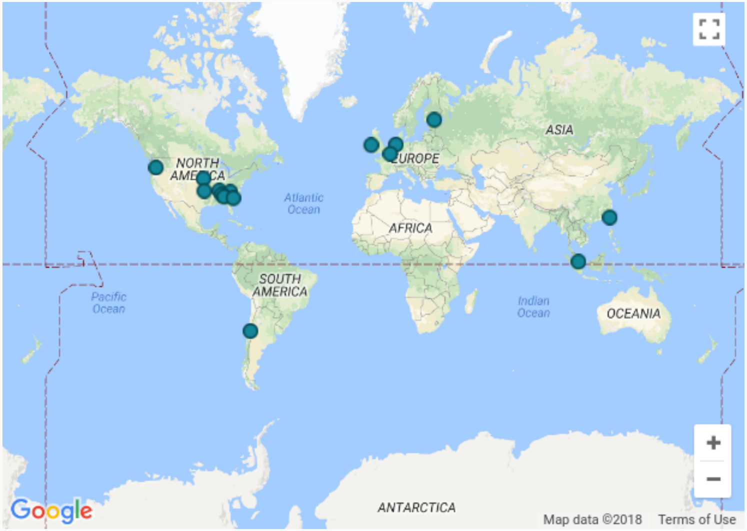
brad.glasbergen@uwaterloo.ca

@bglasber

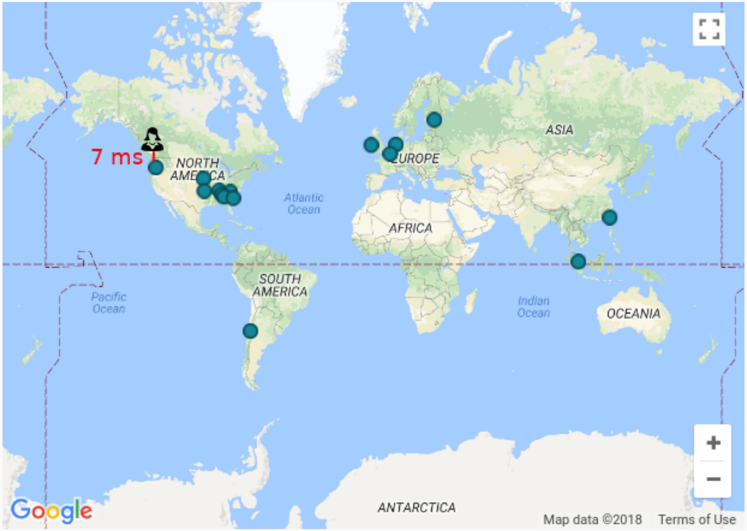
Simple Web Application Architecture



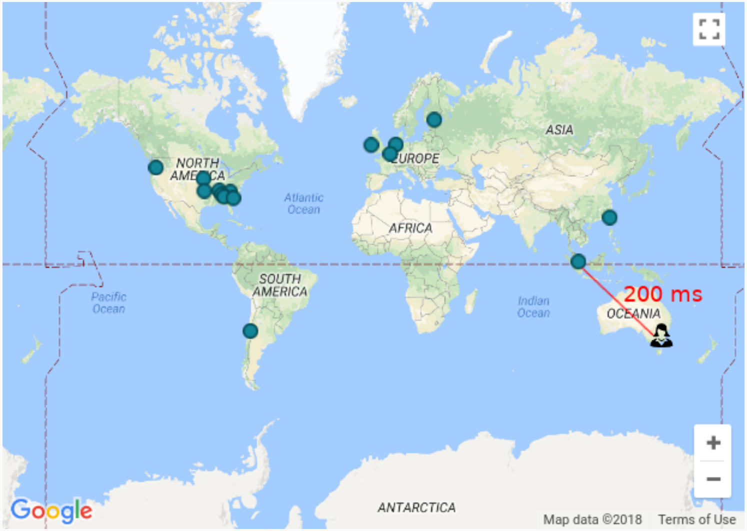
Worldwide Client/Data Center Distribution



Worldwide Client/Data Center Distribution



Worldwide Client/Data Center Distribution

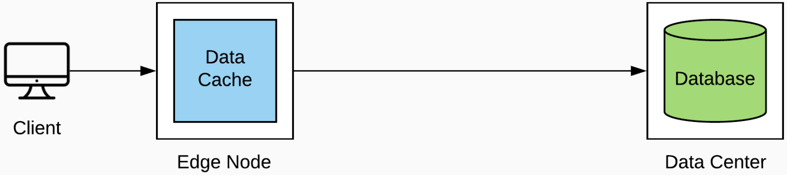


Latency Effects on Clients

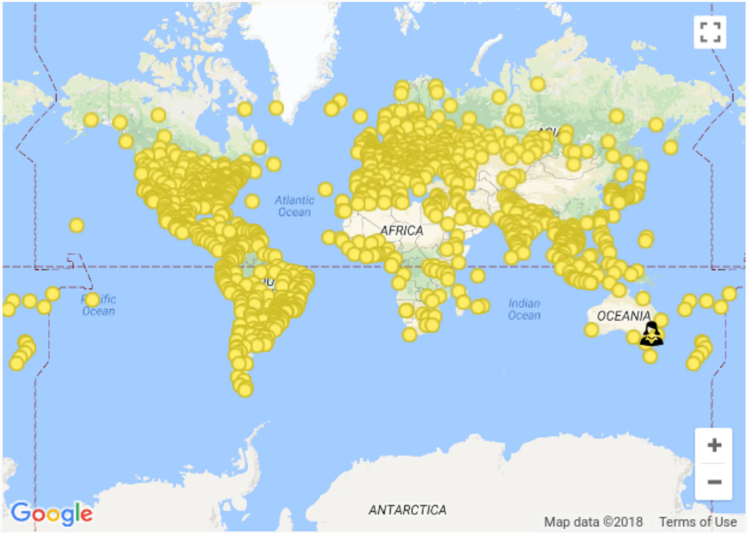
Increased latency reduces user engagement, and consequently revenue!

Schurman et al., “Performance Related Changes and Their User Impact”. *Velocity*, 2009.

Edge Caching (Content Delivery Networks)



Worldwide Client/Edge Node Distribution



Map of metros where at least one Edge node (GGC) is present.

A Problem

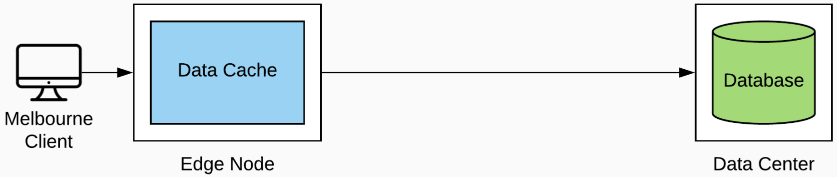
- Limited support for non-static data!

A Problem

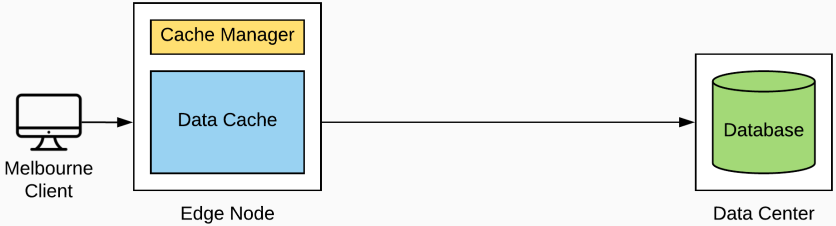
- Limited support for non-static data!
- A majority of webpages rely on personalization and changing data!

Amiri et al., “DBProxy: a dynamic data cache for web applications,” *ICDE*, 2003.

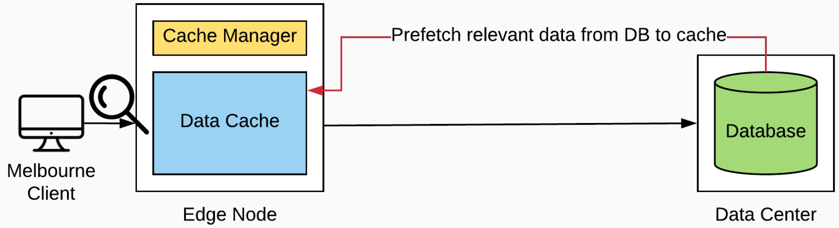
Static Data Edge Cache Architecture



Extending Edge Cache Support



Extending Edge Cache Support



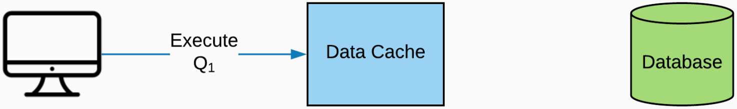
Query Patterns

Execution of a query informs which queries will execute next, and with what parameters.

Dynamic Data Requests (TPC-W Benchmark)

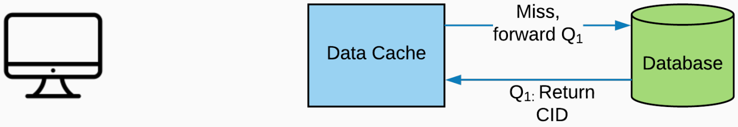
```
1. SELECT  C_ID  FROM CUSTOMER WHERE
C_UNAME = @C_UN and C_PASSWD = @C_PAS

2. SELECT MAX(O_ID) FROM ORDERS WHERE
O_C_ID =  @C_ID
```

Predictive Caching



Q_1: Look up customer ID
Q_2: Look up last order for customer ID

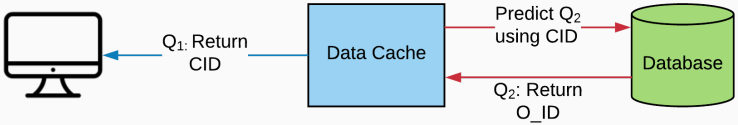
Predictive Caching



Q_1: Look up customer ID

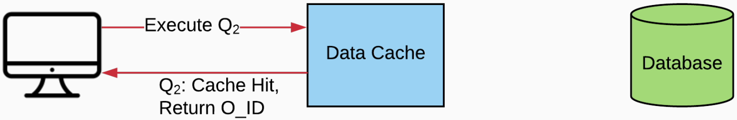
Q_2: Look up last order for customer ID

Predictive Caching



Q_1: Look up customer ID
Q_2: Look up last order for customer ID

Predictive Caching



Q_1: Look up customer ID
Q_2: Look up last order for customer ID

Apollo: Caching Dynamic Data

We developed Apollo, a middleware system that:

Apollo: Caching Dynamic Data

We developed Apollo, a middleware system that:

- Uses online learning to discover client query patterns.

Apollo: Caching Dynamic Data

We developed Apollo, a middleware system that:

- Uses online learning to discover client query patterns.
- Predictively executes and caches query results using these patterns to reduce client response time.

Apollo: Caching Dynamic Data

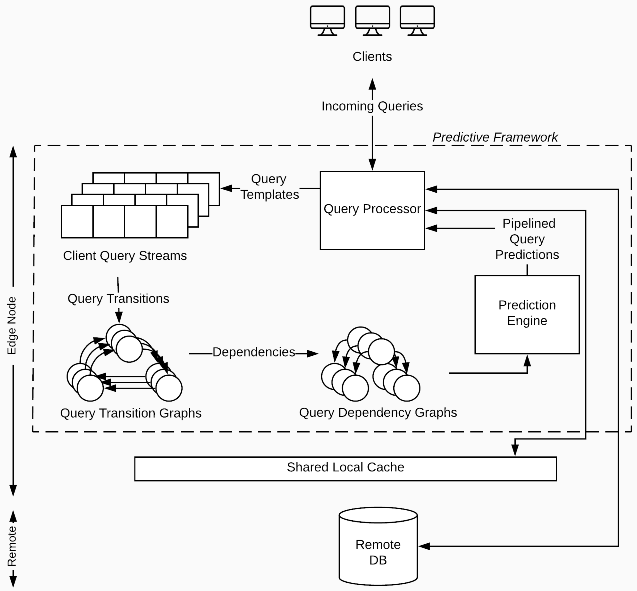
We developed Apollo, a middleware system that:

- Uses online learning to discover client query patterns.
- Predictively executes and caches query results using these patterns to reduce client response time.
- Employs a computationally efficient means of managing updates to cached data.

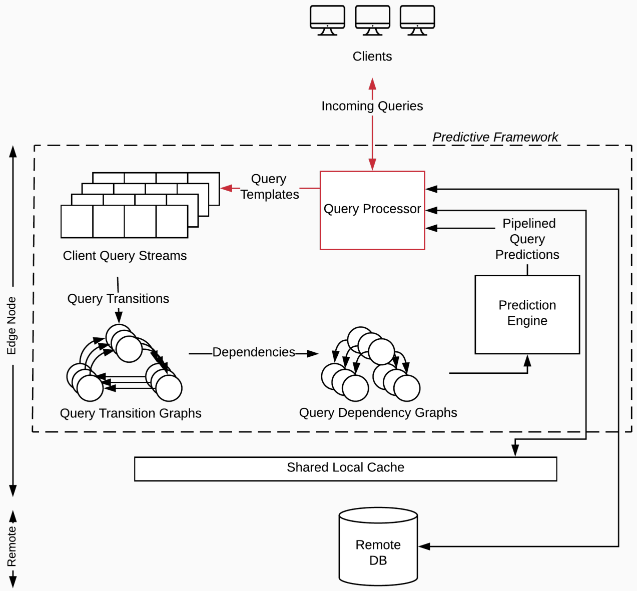
Table of contents

Predictive Query Model

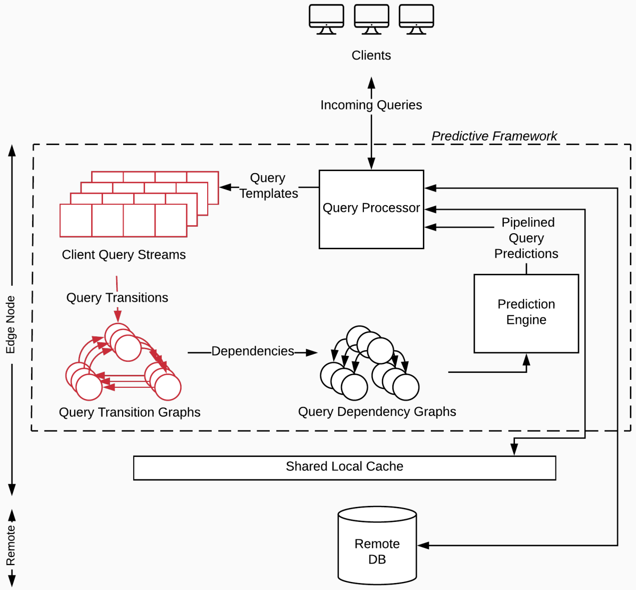
Apollo Overview



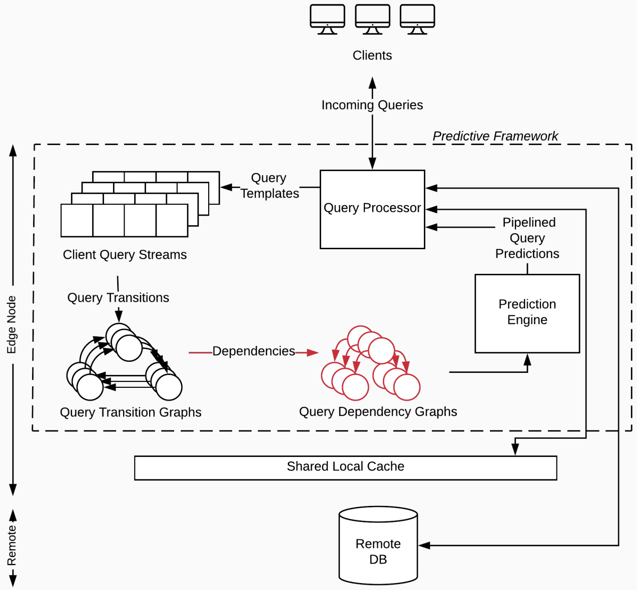
Apollo Overview



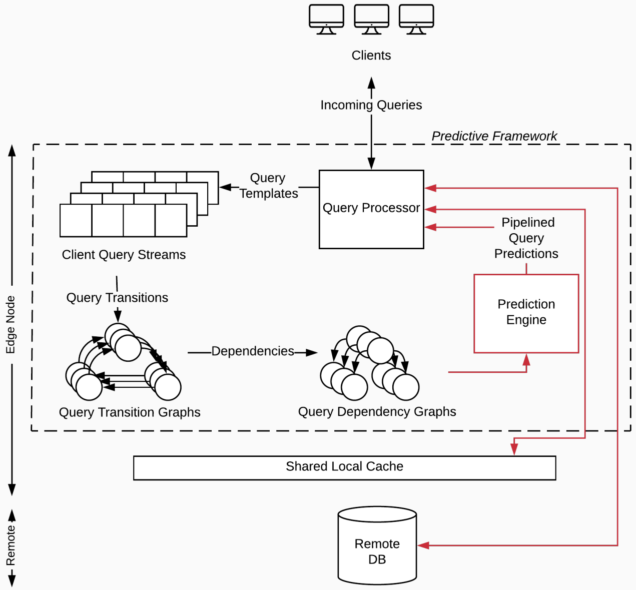
Apollo Overview



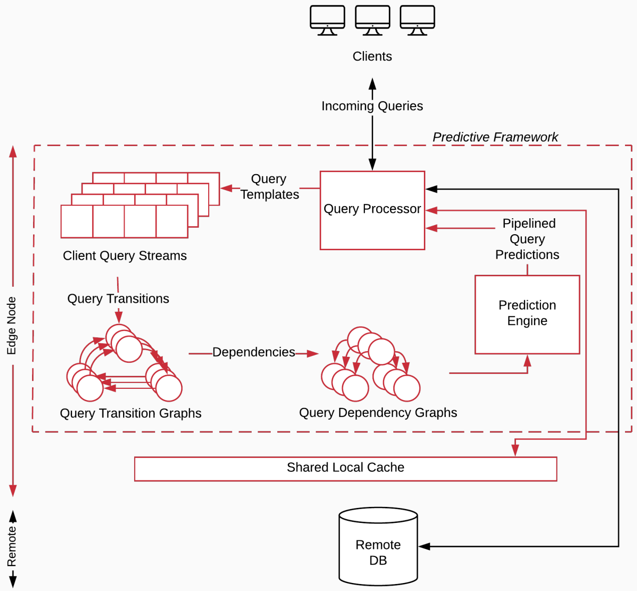
Apollo Overview



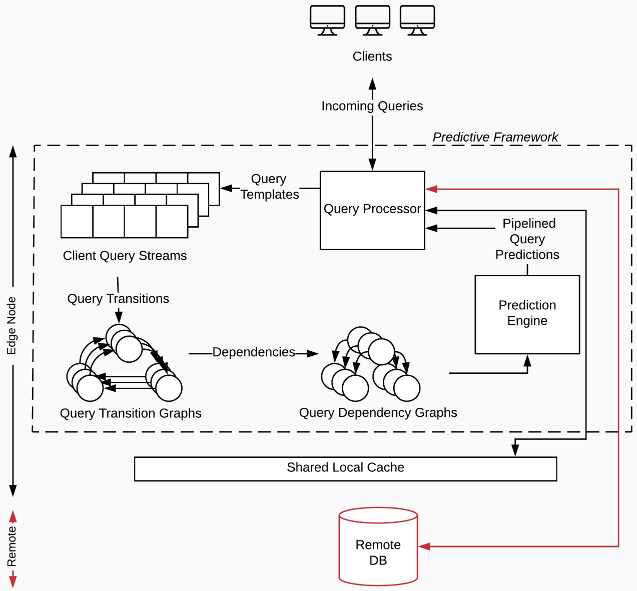
Apollo Overview



Apollo Overview



Apollo Overview



A Query Submission

SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = 'Alice' and C_PASSWD = 'pass'

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{3}$

Query Templates

Two query instances, Q_1, Q_2 share the same query template if they have the same query text modulo *parameterizable constants*.

Abstracting Query Instances to Query Templates

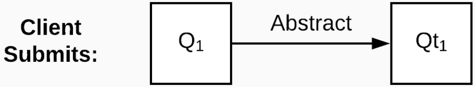
SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{\textbf{?}}$

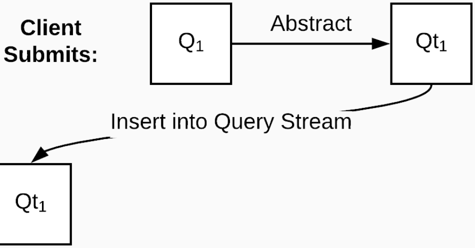
Client Query Streams

Client Submits:
Q_1

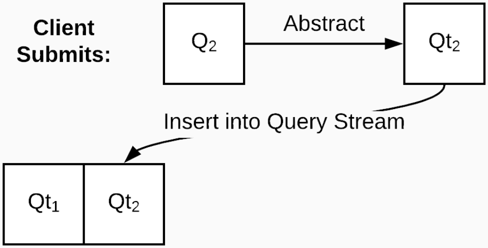
Client Query Streams



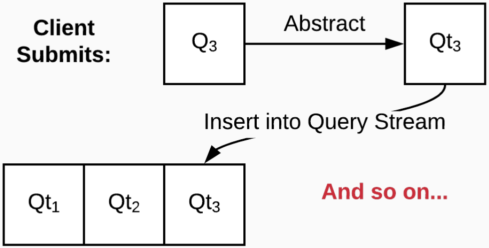
Client Query Streams



Client Query Streams

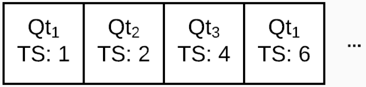


Client Query Streams



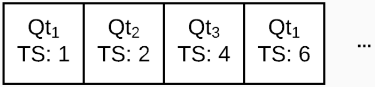
And so on...

Client Query Streams

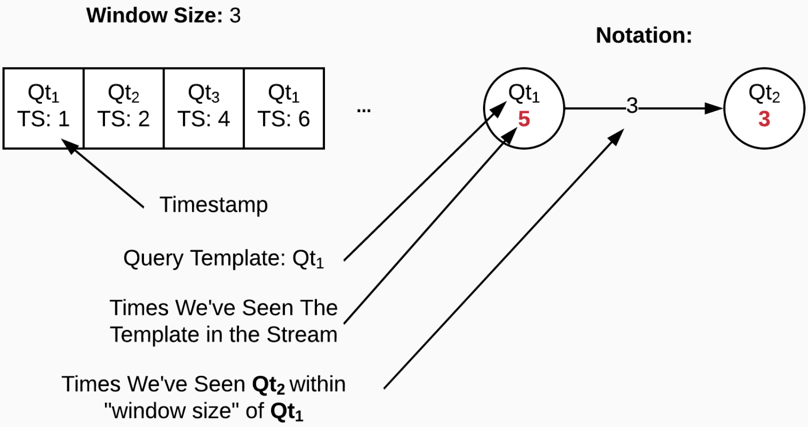


Client Query Streams

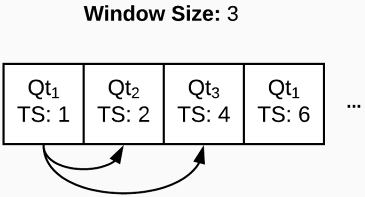
Window Size: 3

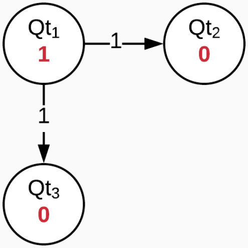


Client Query Streams

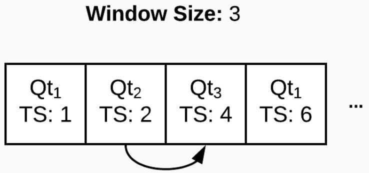


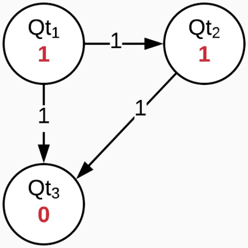
Client Query Streams





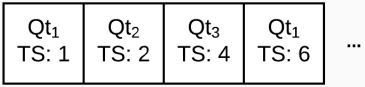
Client Query Streams



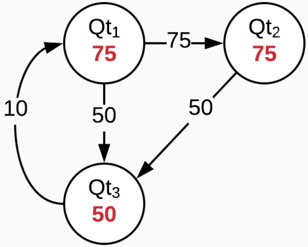


Client Query Streams

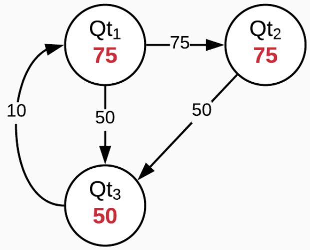
Window Size: 3



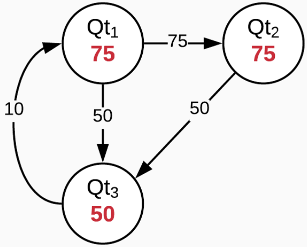
And so on...



Query Transition Graph



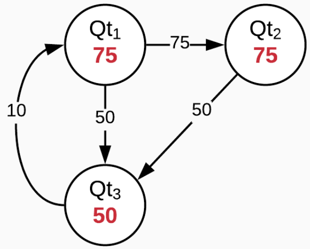
Query Transition Graph



Probability of seeing Qt_2 within sliding window after we've seen Qt_1:

$$P(Qt_2|Qt_1; T \leq \Delta t) = \frac{\text{times } Qt_2 \text{ executed within window of } Qt_1}{\text{times } Qt_1 \text{ executed}} = \frac{75}{75} = 1$$

Query Transition Graph



Probability of seeing Qt_2 within sliding window after we've seen Qt_1:

$P(Qt_2|Qt_1; T \leq \Delta t) = \frac{\text{times } Qt_2 \text{ executed within window of } Qt_1}{\text{times } Qt_1 \text{ executed}} = \frac{75}{75} = 1$

$P(Qt_1|Qt_3; T \leq \Delta t) = \frac{10}{50} = \frac{1}{5}$

Query Transition Graph

The query transition graphs tells us:

- How often a query template is executed

Query Transition Graph

The query transition graphs tells us:

- How often a query template is executed
- Which query templates are correlated with each other. If correlation is sufficiently high ($>$ correlation threshold), then we monitor input and output sets for the queries.

Query Transition Graph

The query transition graphs tells us:

- How often a query template is executed
- Which query templates are correlated with each other. If correlation is sufficiently high ($>$ correlation threshold), then we monitor input and output sets for the queries.

Need parameter mappings for predictive caching!

Client Query Streams

SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{\textbf{?}}$

Client Query Streams

$\overset{3}{\boxed{\text{C\_ID}}}$

SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{\textbf{?}}$

Client Query Streams

$$\overset{3}{\boxed{\text{C\_ID}}}$$

SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

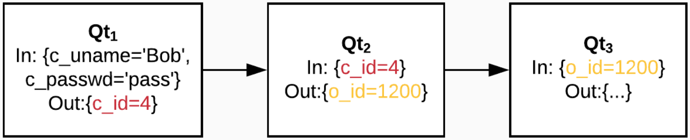
SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\underset{3}{\boxed{\textbf{?}}}$

Client Query Streams

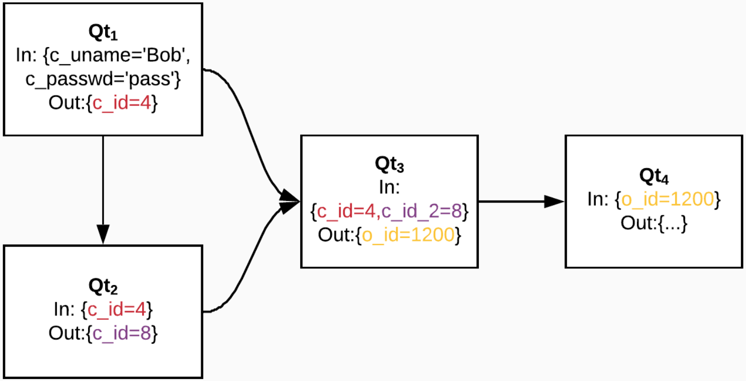
SELECT $\boxed{\text{C\_ID}}$ FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{\textbf{?}}$

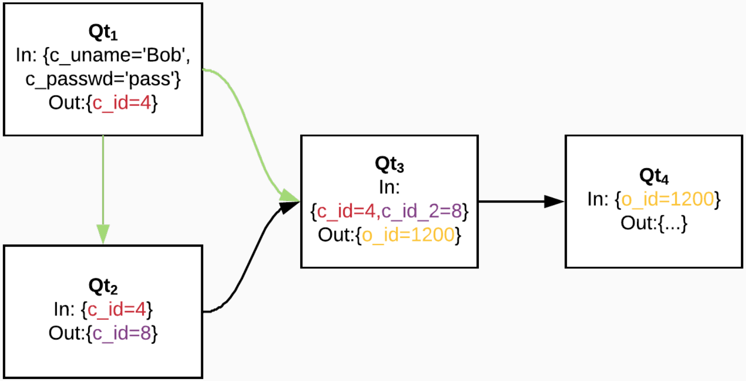
Query Dependency Graph



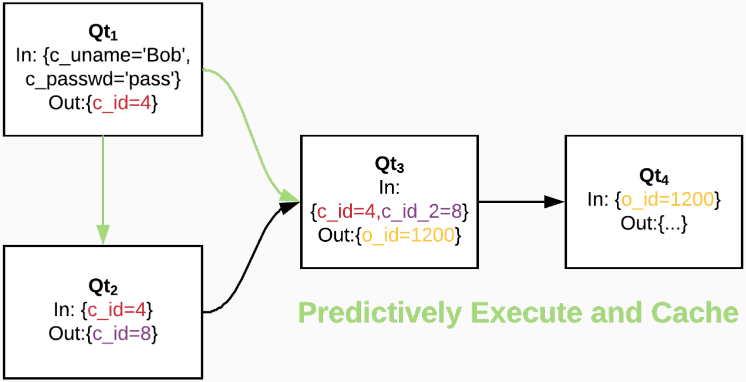
Core Prediction Routine



Core Prediction Routine

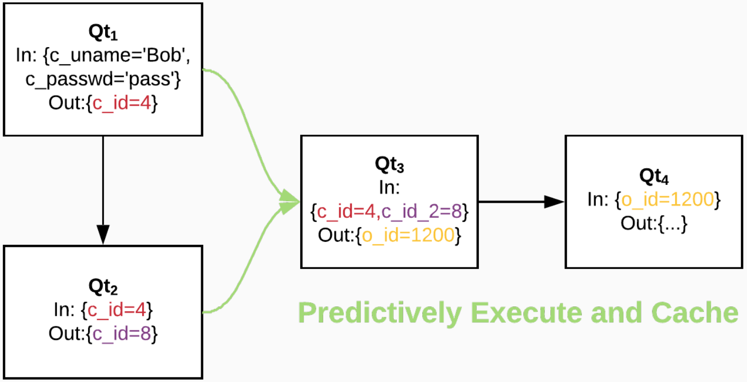


Core Prediction Routine



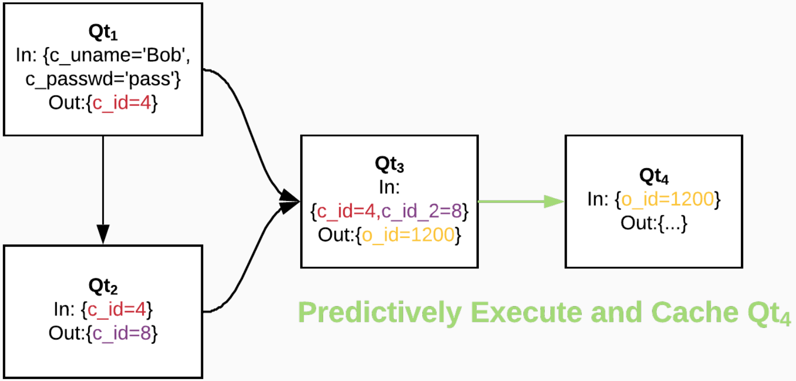
Predictively Execute and Cache Qt_2

Core Prediction Routine

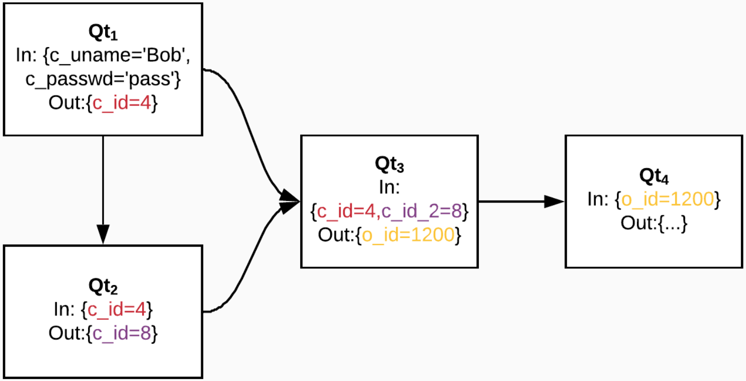


Predictively Execute and Cache Qt_3

Core Prediction Routine



Core Prediction Routine



Predicting Write Queries

Although some write queries (INSERT/UPDATE/DELETE) could be predicted, incorrect predictive executions of such queries would result in modified database state that would need to be undone later.

By predictively executing only read queries, we keep our caching behaviour strictly complementary.

Always Defined Queries

An always defined query is a query whose inputs are always satisfied.

Always Defined Queries

An always defined query is a query whose inputs are always satisfied.

The query can be executed and cached at any time!

Reloading Always Defined Queries

Reload an always defined query after executing a write query if:

Reloading Always Defined Queries

Reload an always defined query after executing a write query if:

- The query was invalidated!

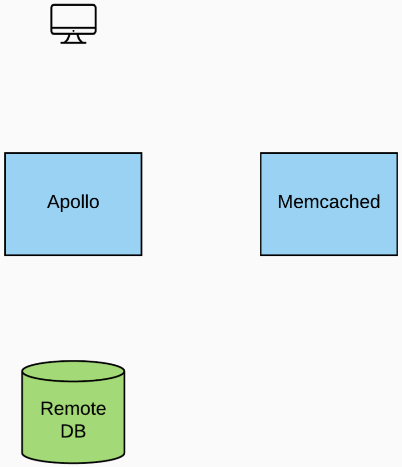
Reloading Always Defined Queries

Reload an always defined query after executing a write query if:

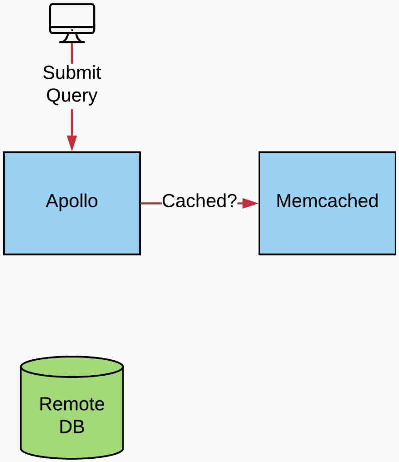
- The query was invalidated!
- The query is considered valuable:
  $\mathit{likelihood\_of\_query}(Qt) \cdot \mathit{avg\_response\_time}(Qt) \geq \text{reload threshold}$

Apollo

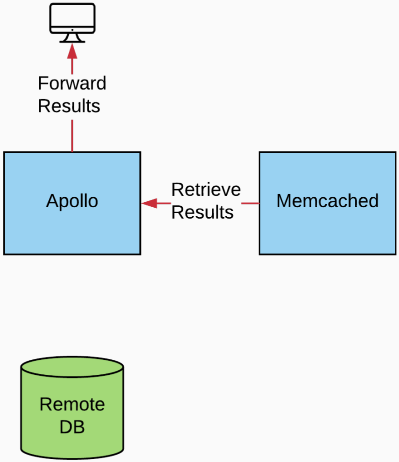
Apollo Architecture



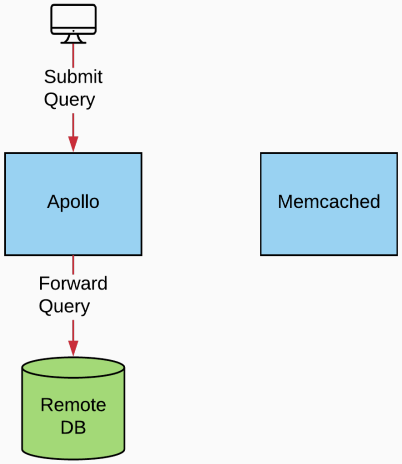
Apollo Architecture



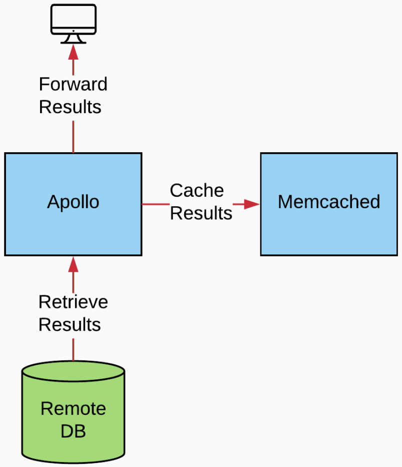
Apollo Architecture



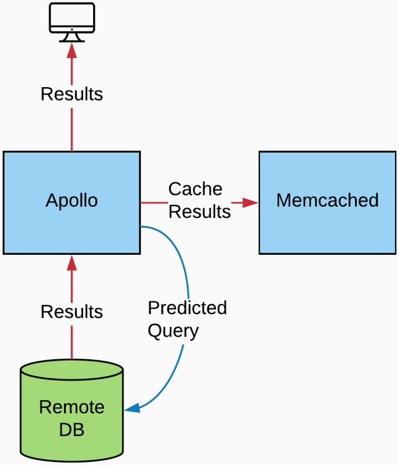
Apollo Architecture



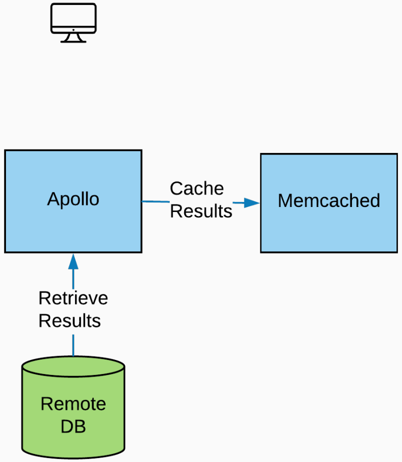
Apollo Architecture



Apollo Architecture



Apollo Architecture



Publish—Subscribe Model

Concurrent requests for the same query will wait until the first query executes and returns its results. That query's result set will be forwarded to the others.

Client Sessions

Clients are guaranteed to see state at least as recent as what they last read/wrote.

Client Sessions

Clients are guaranteed to see state at least as recent as what they last read/wrote.

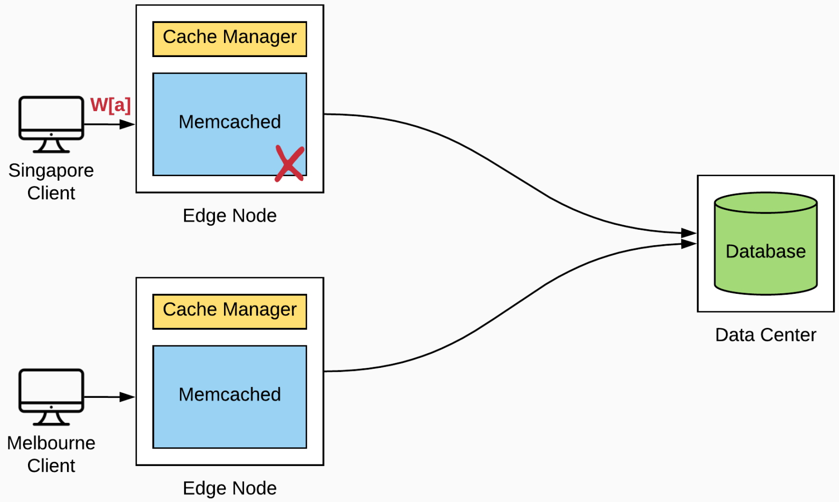
- Client-centric approach to caching!

Client Sessions

Clients are guaranteed to see state at least as recent as what they last read/wrote.

- Client-centric approach to caching!
- Only writes and reads of “fresher” data cause invalidations!

Benefits

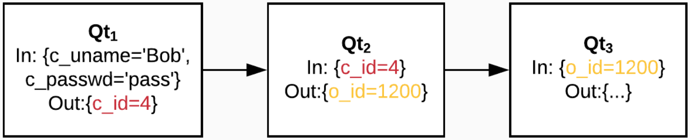


W[a]
Singapore Client
Cache Manager
Memcached
Edge Node
Melbourne Client
Cache Manager
Memcached
Edge Node
Database
Data Center

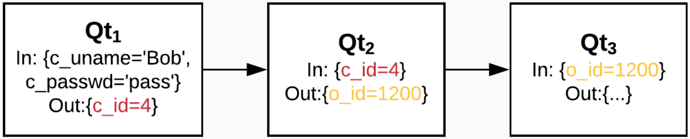
Benefits

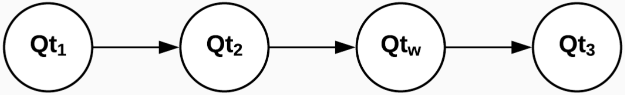
- No global cache invalidations.
- Can predict whether a prefetched query result will be used before invalidation!

Prediction Invalidation Detection



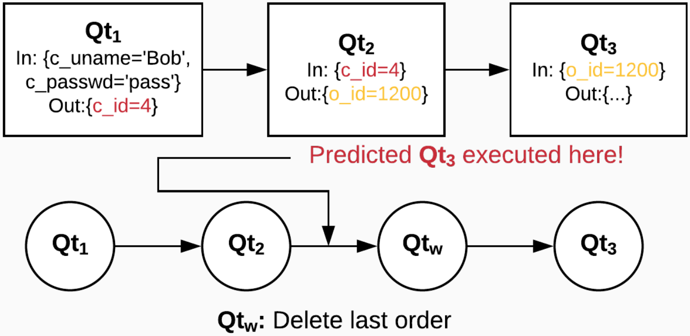
Prediction Invalidation Detection





Qt$_w$: Delete last order

Prediction Invalidation Detection



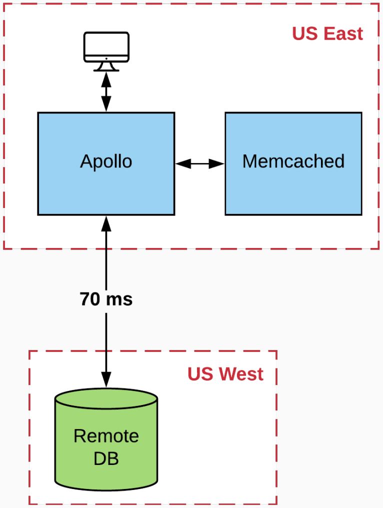
Prediction Invalidation Caching

Prediction Invalidation Caching

- Maintain multiple client transition graphs with different window widths
- Consult client query transition to see if a query is likely to occur that would invalidate the results of our predictively executed query

Experiments

Experiment Configuration



Experiment Configuration

Three configurations:

- Apollo configuration: as described in prior sections.
- Memcached configuration: LRU cache — Apollo with predictive features turned off
- Fido configuration: Use Fido predictive engine instead of Apollo's predictive features!

Fido

$$\underbrace{Q_1, Q_2, Q_3}_{\text{Prefix}}$$

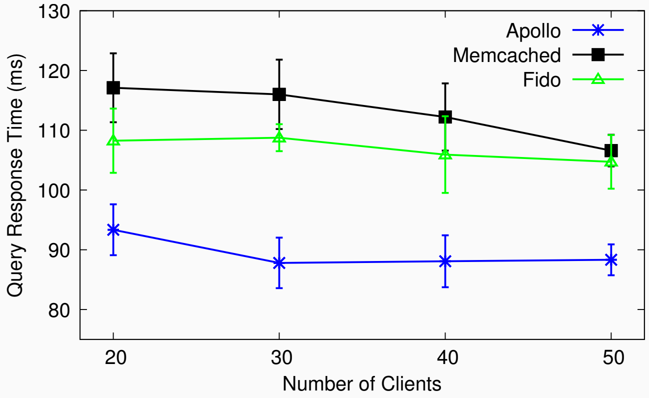
Fido

$$\underbrace{Q_1,Q_2,Q_3}_{\text{Prefix}} \quad \underbrace{Q_4,\ Q_5}_{\text{Suffix}}$$

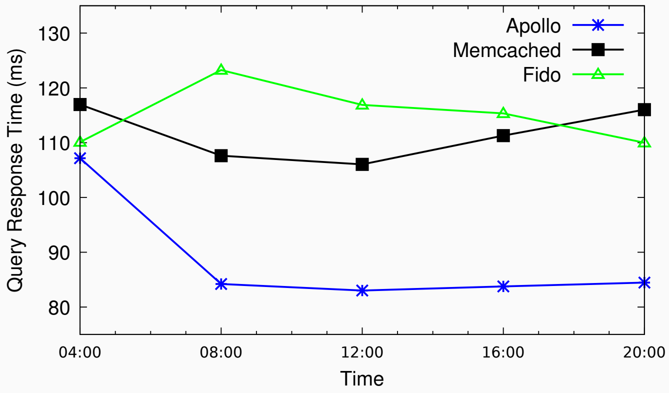
Fido

- Query instance based predictions, instead of query templates.
- Prefix length: 3, Suffix Length: 2
- Requires offline training (Supplied 40 minutes of data).

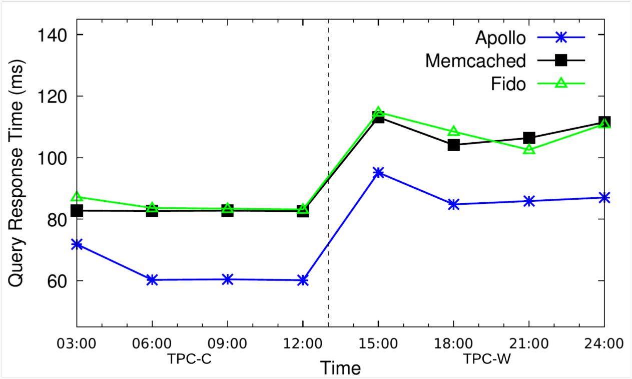
TPC-W Results



Learning Over Time



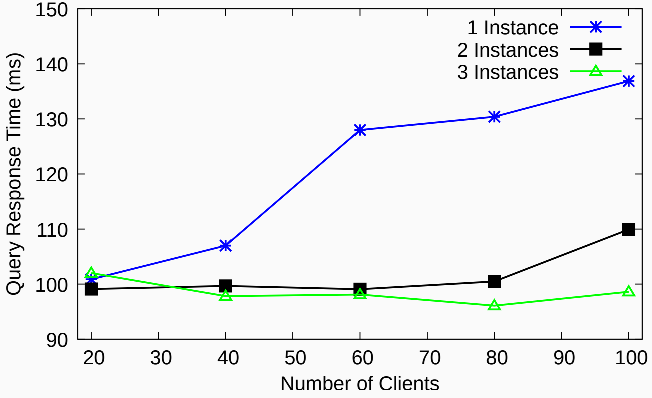
Workload Change



Thanks

Questions?

Multiple Apollo Instances



Parameter Settings (TPC-W)

Parameter	Setting
Window Width	15s
Correlation Threshold	0.99
Reload Threshold	0
Cache Size	5% of DB

Related Work (Systems)

Similarities and Differences Among Related Work

System	Similarities	Differences
Scalpel	• Prefetching via templates	• Offline training • Write Handling • Client-side • Query Rewriting
Fido	• Prefetching • Server-side/middleware	• Offline training • Query Instances • Write Handling

Icon Attribution

Business Woman by Delwar Hossain from the Noun Project
Search by Anusha Narvekar from the Noun Project

CCBY license

K. Amiri, S. Park, R. Tewari, and S. Padmanabhan.
Dbproxy: a dynamic data cache for web applications.
In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, pages 821–831, March 2003.

E. Schurman and J. Brutlag.
Performance related changes and their user impact.
Velocity, 2009.