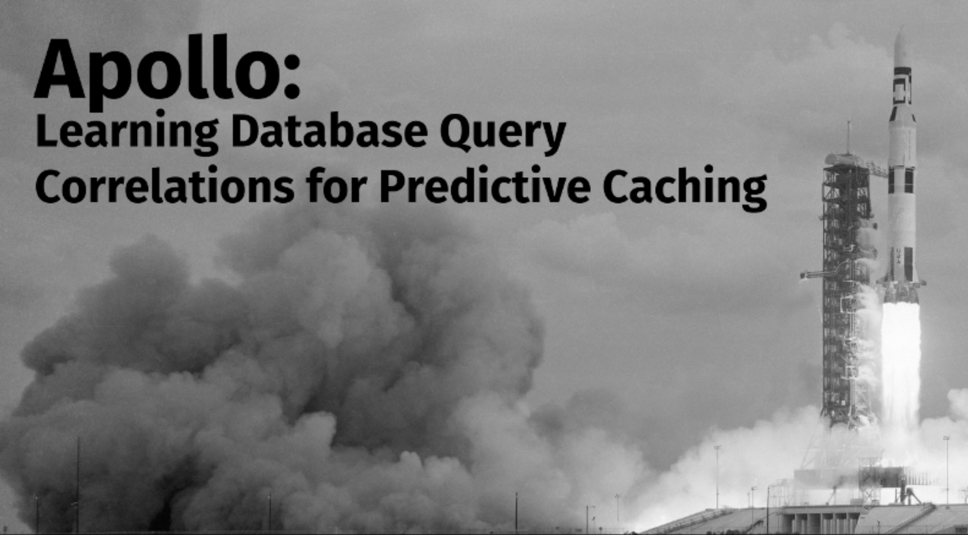


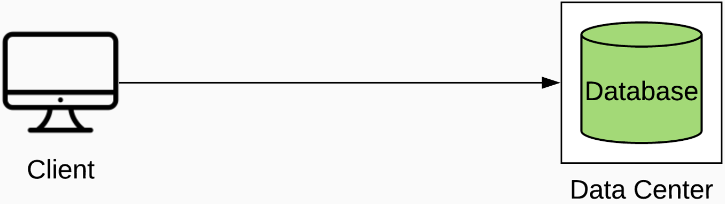
Apollo:
Learning Database Query Correlations for Predictive Caching

Brad Glasbergen, Khuzaima Daudjee, Michael Abebe,
Scott Foggo, Anil Pacaci

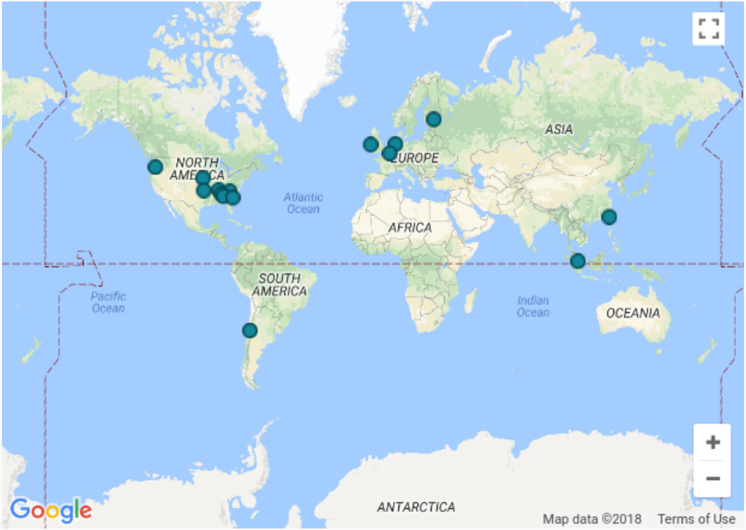
UNIVERSITY OF WATERLOO

@bglasber

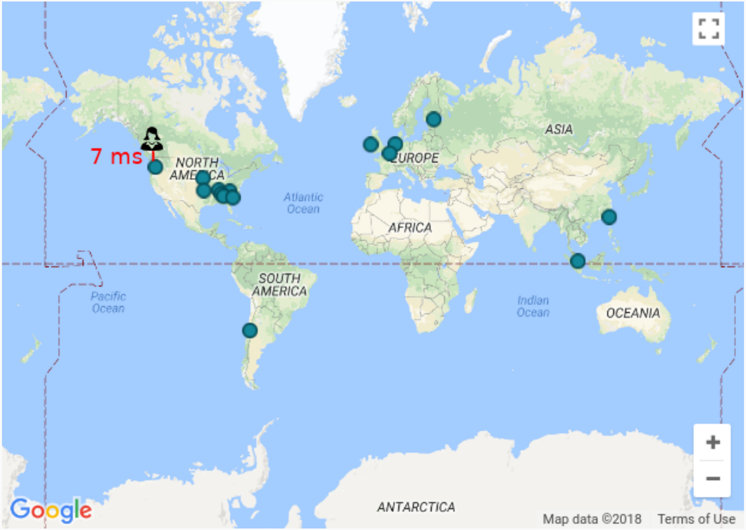
Simple Web Application Architecture



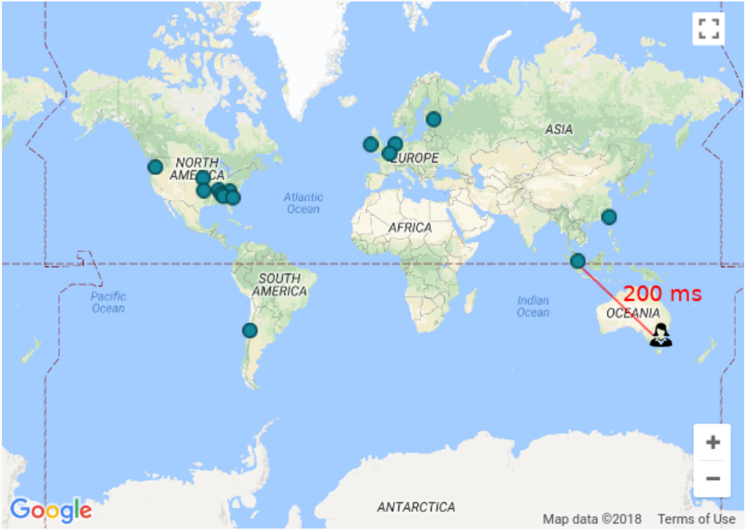
Worldwide Client/Data Center Distribution



Worldwide Client/Data Center Distribution



Worldwide Client/Data Center Distribution

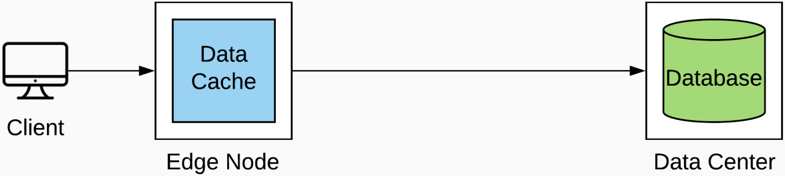


Latency Effects on Clients

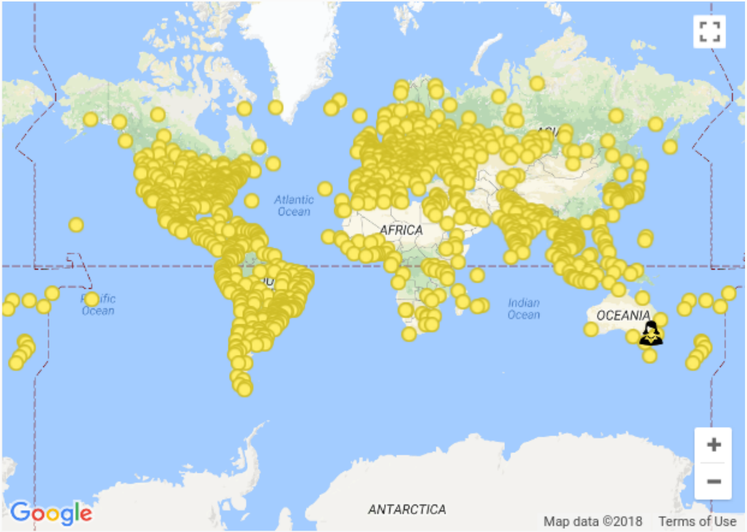
Increased latency reduces user engagement, and consequently revenue!

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

Edge Caching (Content Delivery Networks)



Worldwide Client/Edge Node Distribution

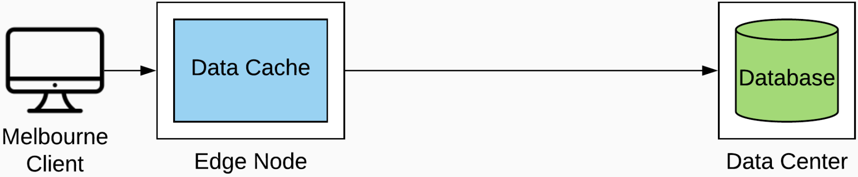


Map of metros where at least one Edge node (GGC) is present.

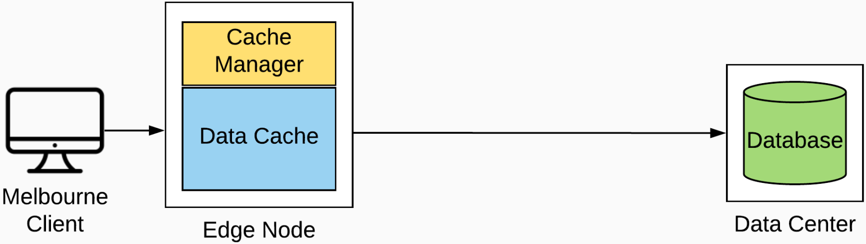
Content Delivery Networks — A Silver Bullet?

- Limited support for non-static data!
- Can we extend support for dynamic data?

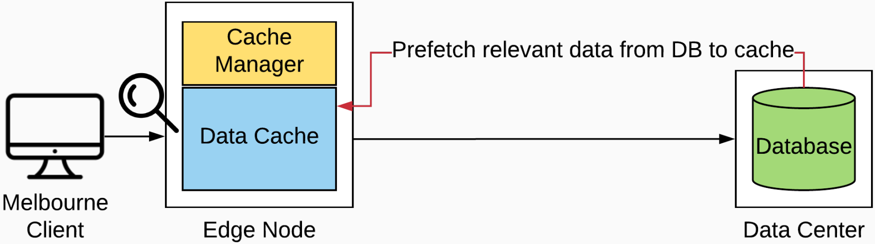
Static Data Edge Cache Architecture



Extending Edge Cache Support



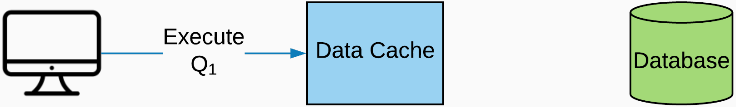
Extending Edge Cache Support



Dynamic Data Requests (TPC-W Benchmark)

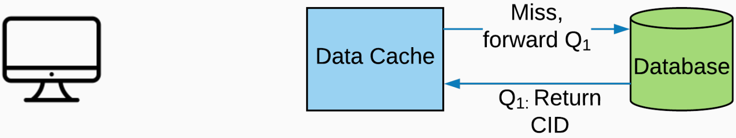
```
1. SELECT  C_ID  FROM CUSTOMER WHERE
C_UNAME = @C_UN and C_PASSWD = @C_PAS

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

Predictive Caching



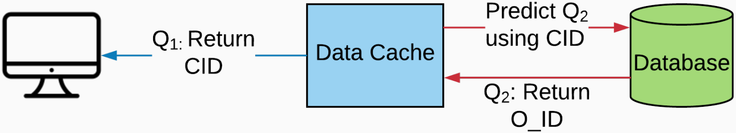
$\mathbf{Q_1}$: Look up customer ID
$\mathbf{Q_2}$: Look up last order for customer ID

Predictive Caching



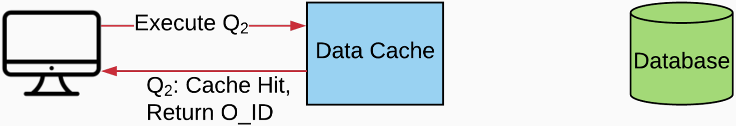
Q_1: Look up customer ID
Q_2: Look up last order for customer ID

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

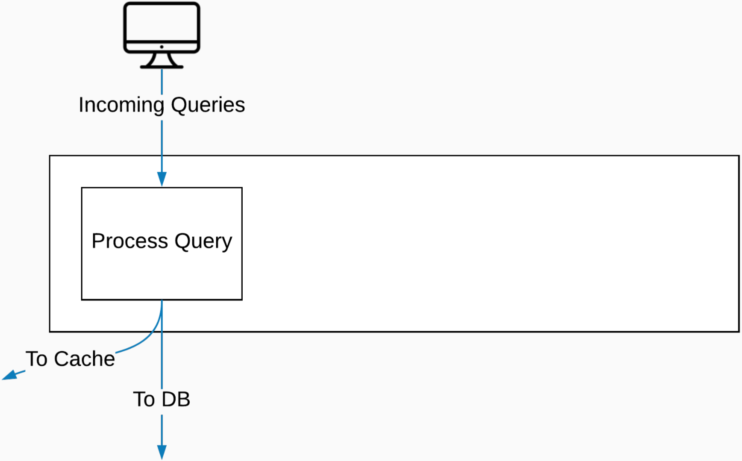
1. Predictive Query Model

2. Cache Management

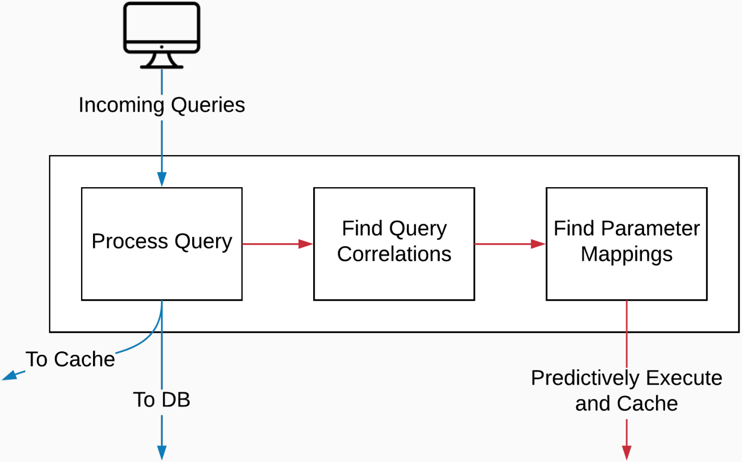
3. Results

Predictive Query Model

Apollo Overview



Apollo Overview


Incoming Queries
Process Query
Find Query Correlations
Find Parameter Mappings
To Cache
To DB
Predictively Execute and Cache

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 = **?**

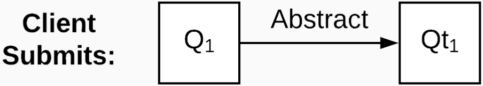
SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = $\boxed{\textbf{?}}$

Client Query Streams

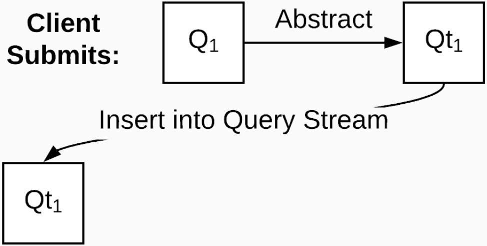
Client Submits:

Q_1

Client Query Streams



Client Query Streams



Client Submits:

Q_1

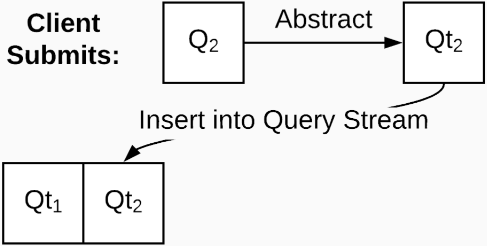
Abstract

Qt_1

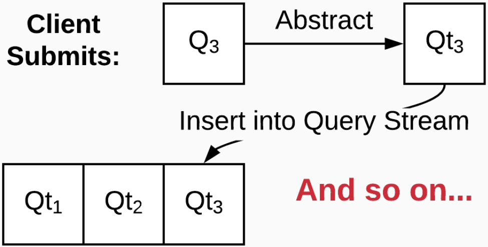
Insert into Query Stream

Qt_1

Client Query Streams



Client Query Streams

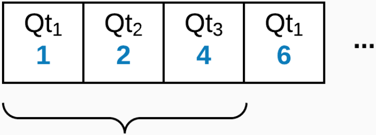


And so on...

Client Query Streams

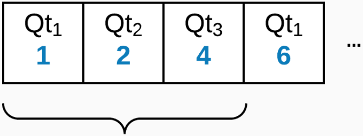
Qt_1	Qt_2	Qt_3	Qt_1	...
1	**2**	**4**	**6**	

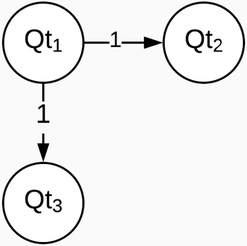
Client Query Streams



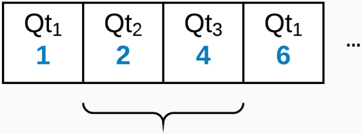
Qt_1	Qt_2	Qt_3	Qt_1	...
1	**2**	**4**	**6**	

Client Query Streams

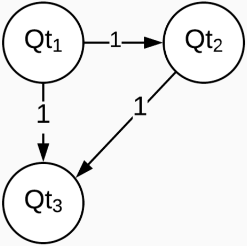




Client Query Streams



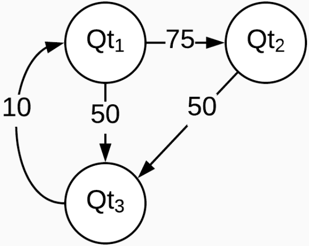
Qt_1	Qt_2	Qt_3	Qt_1	...
1	2	4	6	



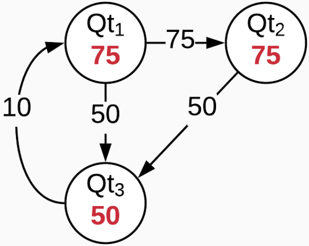
Client Query Streams

Qt_1	Qt_2	Qt_3	Qt_1	...
1	**2**	**4**	**6**	

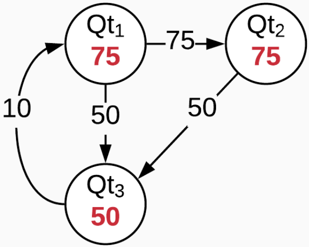
And so on...



Query Transition Graph

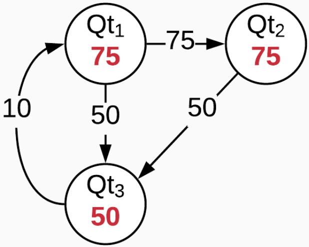


Query Transition Graph



Probability of seeing Qt_2 within sliding window after we've seen Qt_1:
$P(Qt_2|Qt_1; T \leq \Delta t) = \frac{75}{75} = 1$

Query Transition Graph



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

$P(Qt_2|Qt_1; T \leq \Delta t) = \frac{75}{75} = 1$

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

Client Query Streams

SELECT C_ID FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = **?**

Client Query Streams

3

SELECT C_ID FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = **?**

Client Query Streams

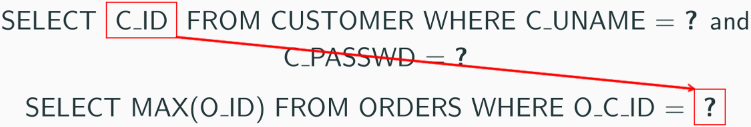
3

SELECT C_ID FROM CUSTOMER WHERE C_UNAME = ? and C_PASSWD = ?

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = ?

3

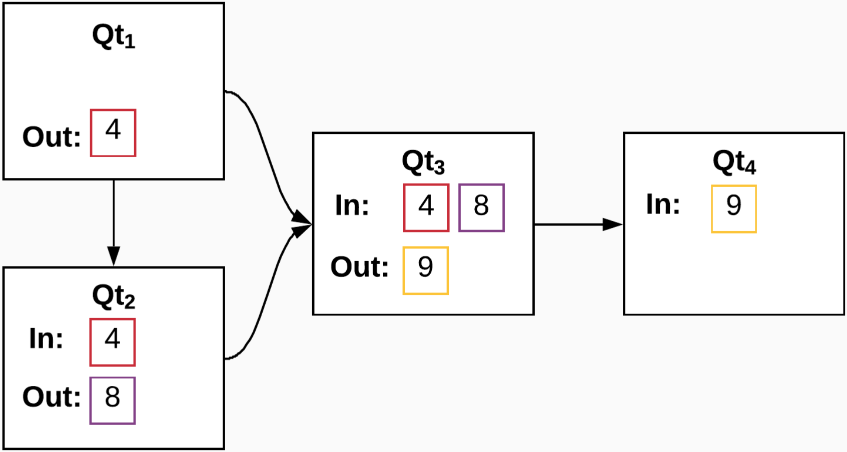
Client Query Streams



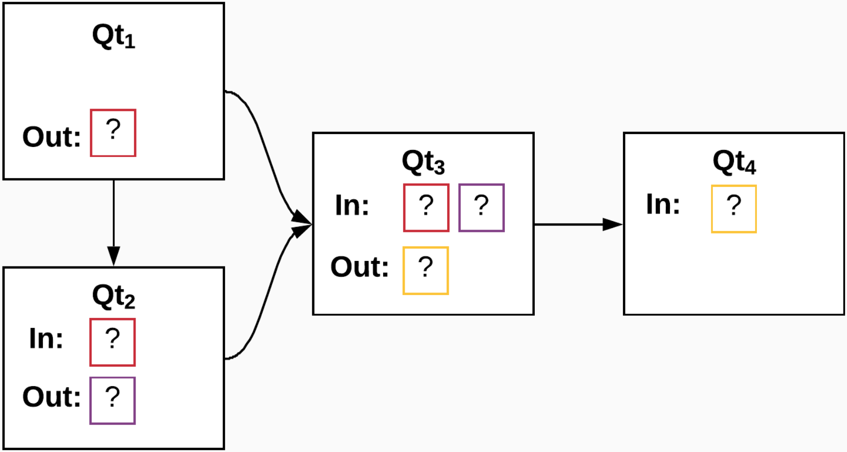
SELECT C_ID FROM CUSTOMER WHERE C_UNAME = **?** and C_PASSWD = **?**

SELECT MAX(O_ID) FROM ORDERS WHERE O_C_ID = **?**

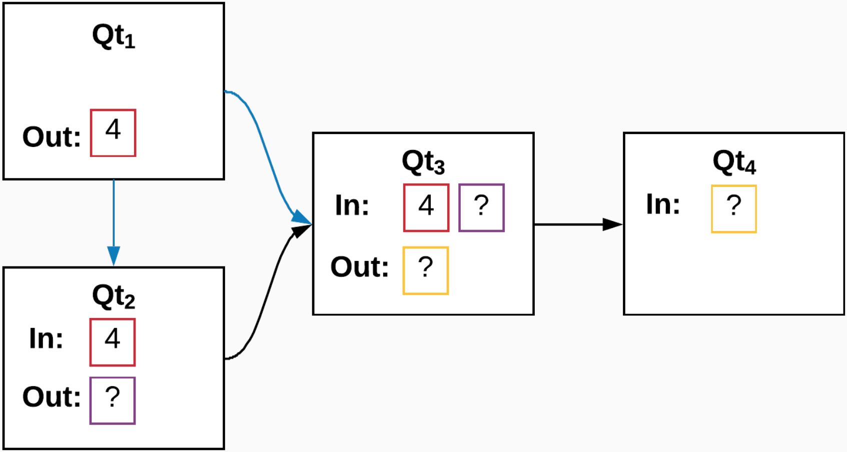
Dependency Graph



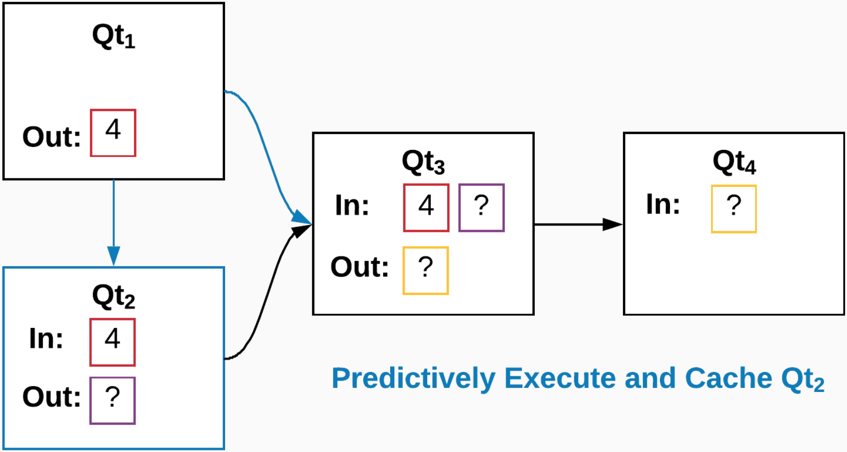
Prediction Routine



Prediction Routine

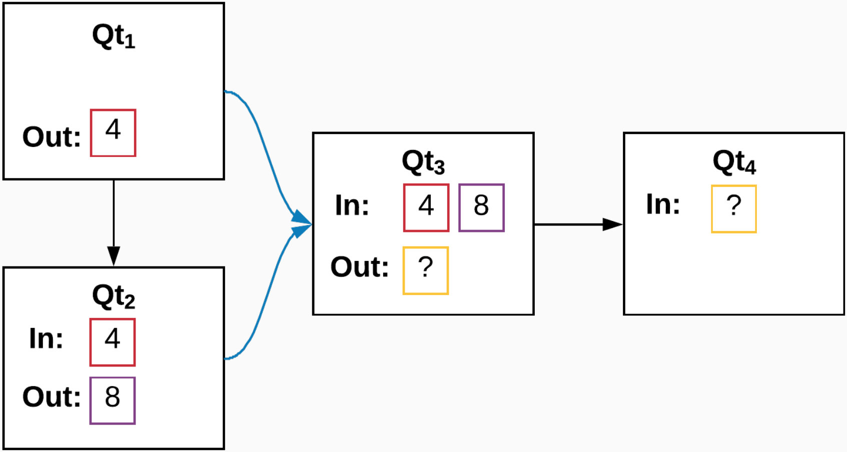


Prediction Routine

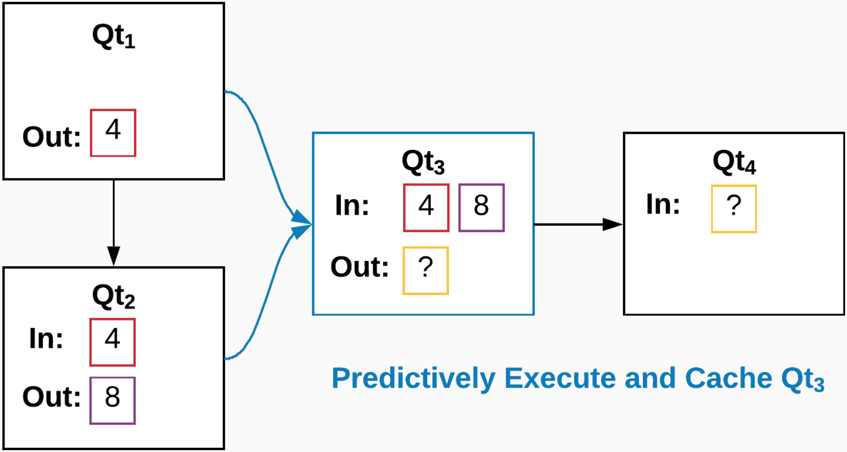


Predictively Execute and Cache Qt_2

Prediction Routine

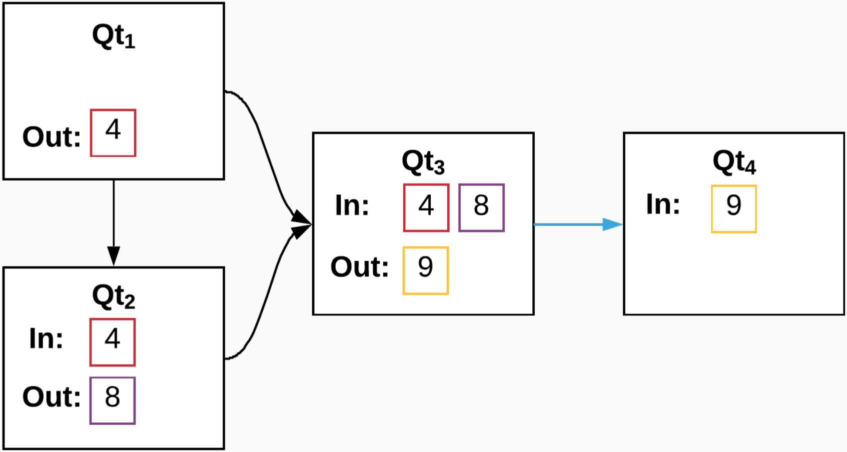


Prediction Routine

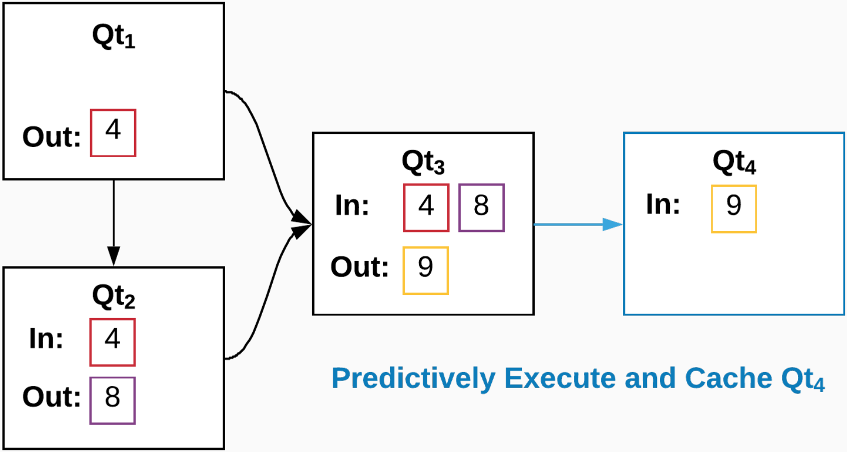


Predictively Execute and Cache Qt_3

Prediction Routine



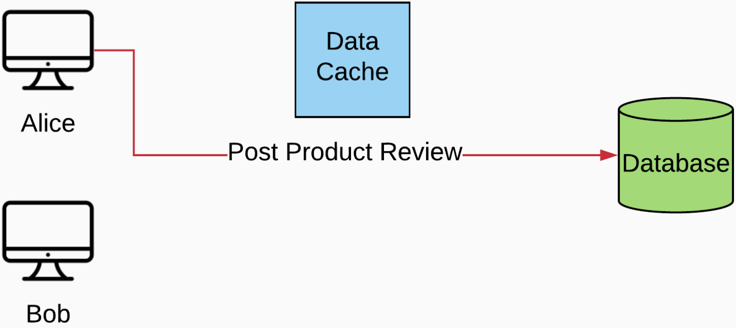
Prediction Routine



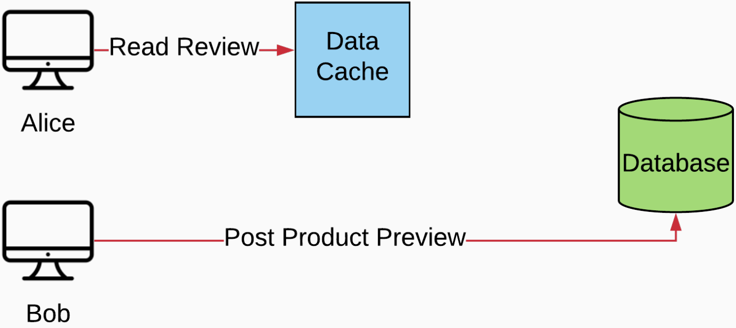
Predictively Execute and Cache Qt_4

Cache Management

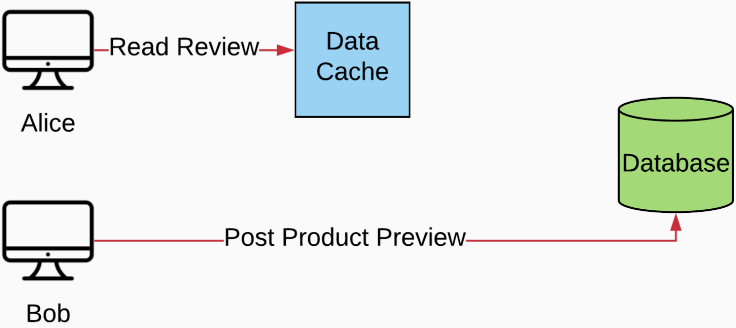
Client-Centric Caching



Client-Centric Caching

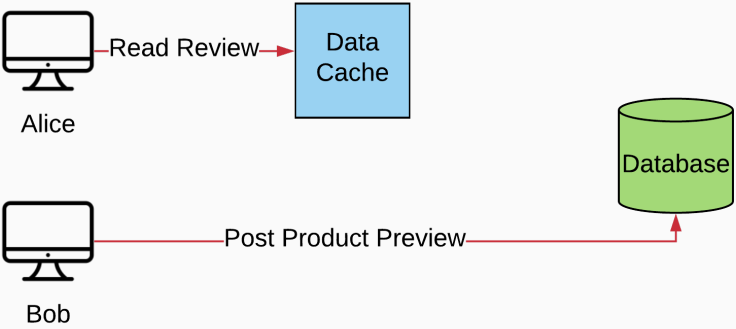


Client-Centric Caching



Alice should see her own order, but does not care about Bob's!

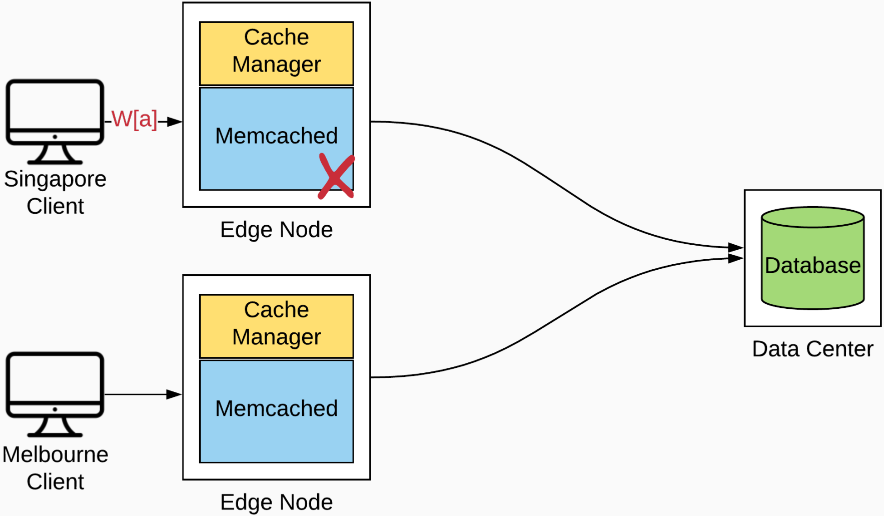
Client-Centric Caching



Alice should see her own order, but does not care about Bob's!

- Improves cache performance, client-centric model
- Support for reading latest data if needed

Benefits of Client-Centric Caching



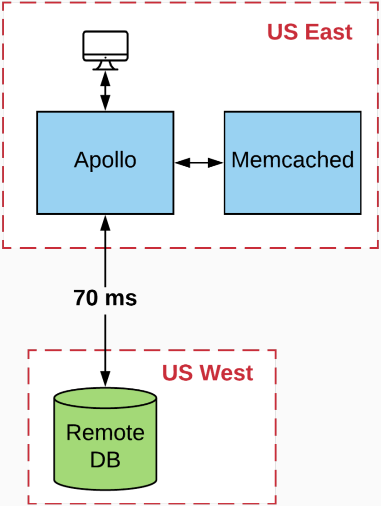
Benefits of Client-Centric Caching

- Can predict whether a prefetched query result will be used before invalidation!
- Reloading queries upon invalidation

See paper for details!

Results

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

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

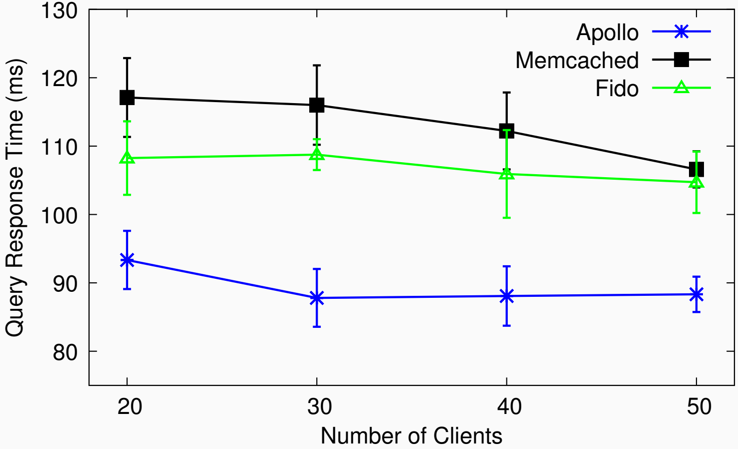
Fido

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

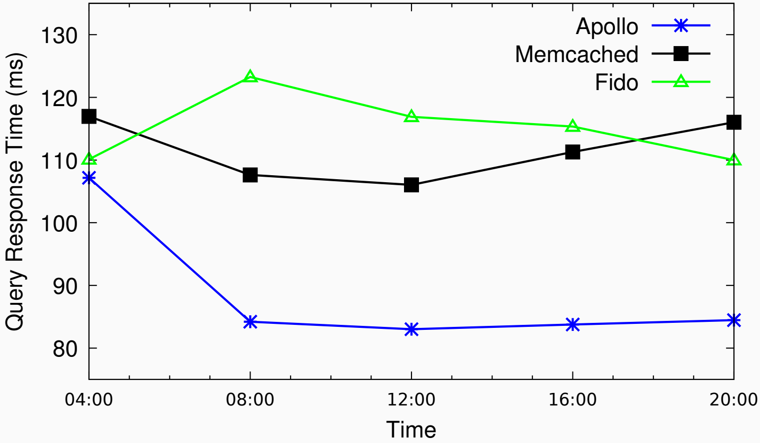
Fido

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

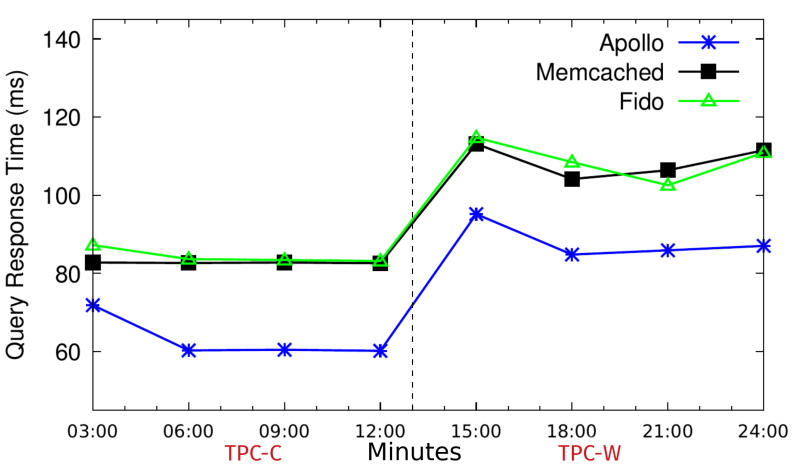
TPC-W Results



Learning Over Time



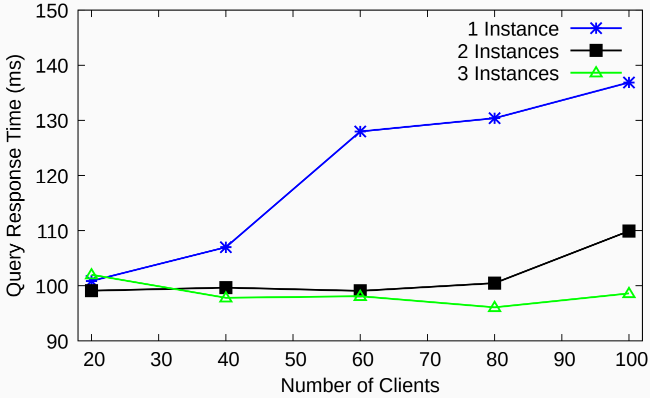
Workload Change



Predictive Caching Works

- Online learning enables workload pattern discovery and adapts to client behaviour changes
- Predictive caching is an effective tool to reduce application latency

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

- 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.