CS848 Paper Presentation
MTCache: Transparent Mid-Tier Database Caching in SQL Server

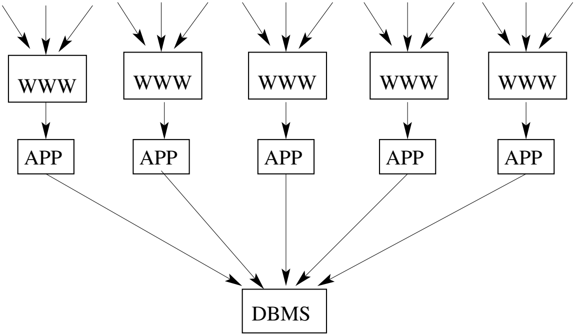
Larson, Goldstein, Zhou
ICDE 2004

Presented by Ken Salem

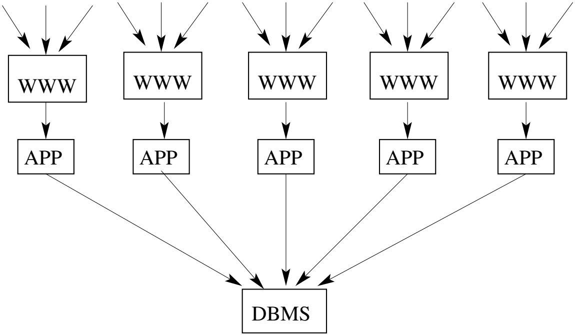
David R. Cheriton School of Computer Science
University of Waterloo

11 January 2010

3-Tier Web Service Architecture



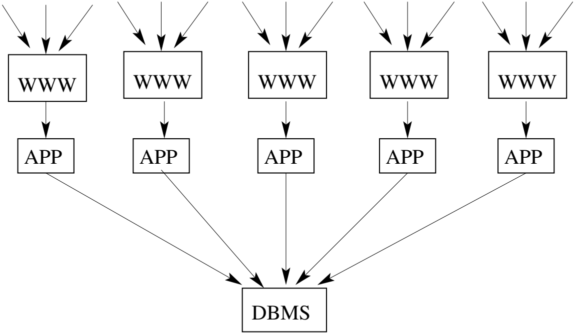
3-Tier Web Service Architecture



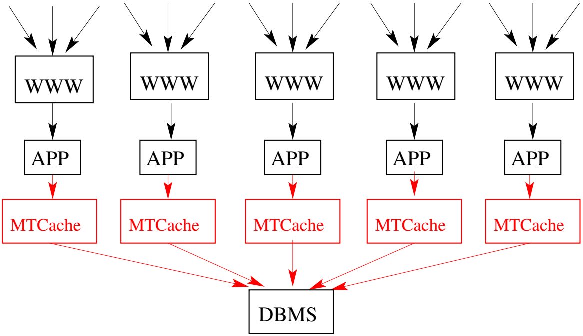
Scalability Problem

- Web and App servers are easy to scale out.
- DBMS can become a bottleneck

MTCache



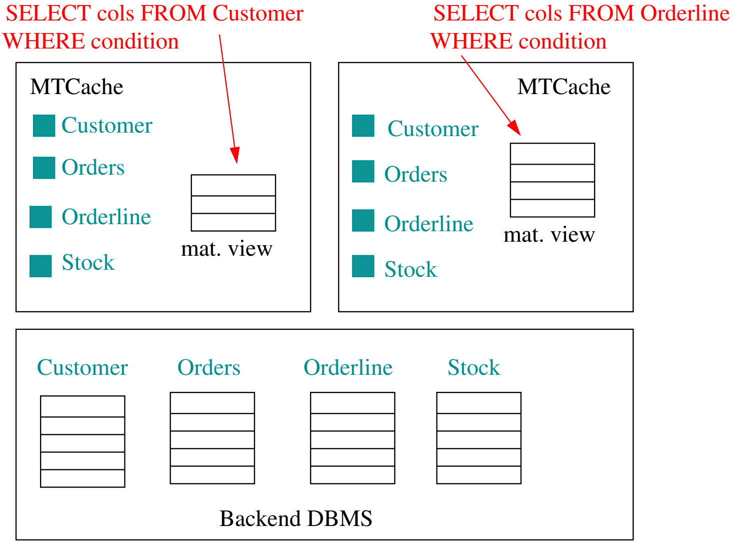
MTCache



Objective

- reduce load on backend DBMS, eliminating bottleneck
- scale out by adding more MTCache nodes

MTCache Databases



MTCache Operation

- each Application Server directs its database requests to an MTCache server, rather than the backend DBMS

MTCache Operation

- each Application Server directs its database requests to an MTCache server, rather than the backend DBMS
- MTCache forwards INSERT, DELETE, UPDATE requests to the backend database and forwards the response to the App Server.

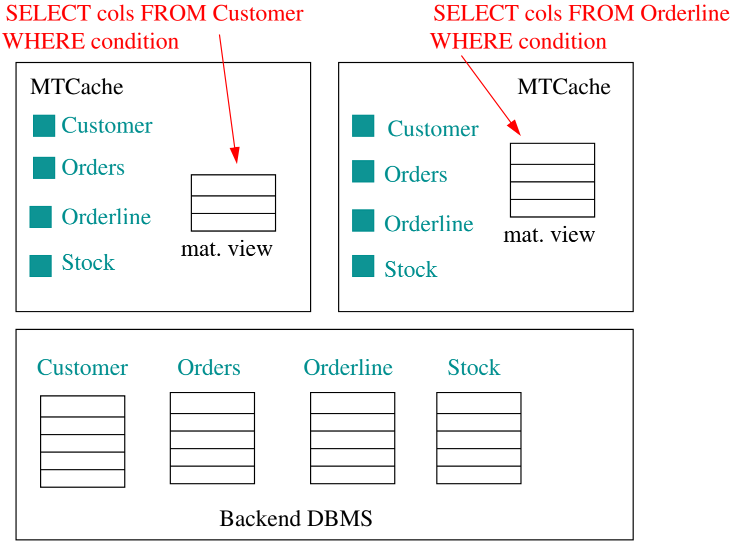
MTCache Operation

- each Application Server directs its database requests to an MTCache server, rather than the backend DBMS
- MTCache forwards INSERT, DELETE, UPDATE requests to the backend database and forwards the response to the App Server.
- Queries (SELECTs) are handled by MTCache, which makes a cost-based decision about whether to:
  - handle the query locally
  - handle the query remotely
  - split the query (and the processing) into local and remote parts

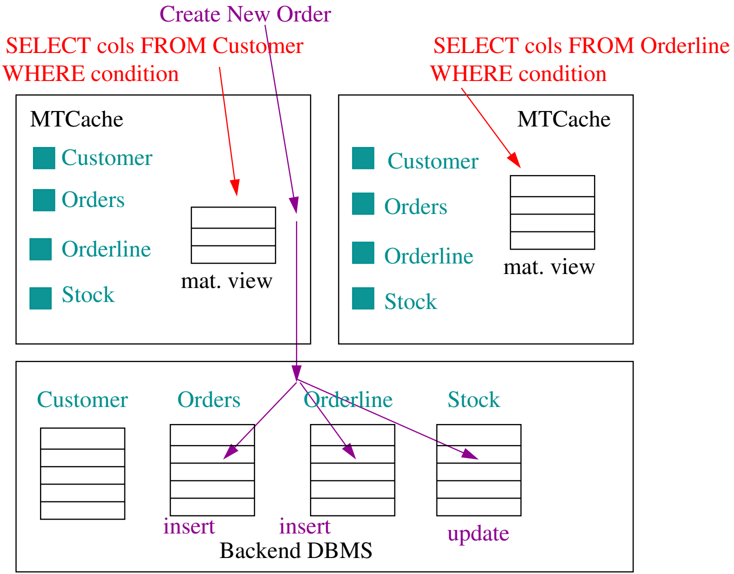
MTCache Operation

- each Application Server directs its database requests to an MTCache server, rather than the backend DBMS
- MTCache forwards INSERT, DELETE, UPDATE requests to the backend database and forwards the response to the App Server.
- Queries (SELECTs) are handled by MTCache, which makes a cost-based decision about whether to:
  - handle the query locally
  - handle the query remotely
  - split the query (and the processing) into local and remote parts
- The backend DBMS lazily propagates updates to MTCache nodes

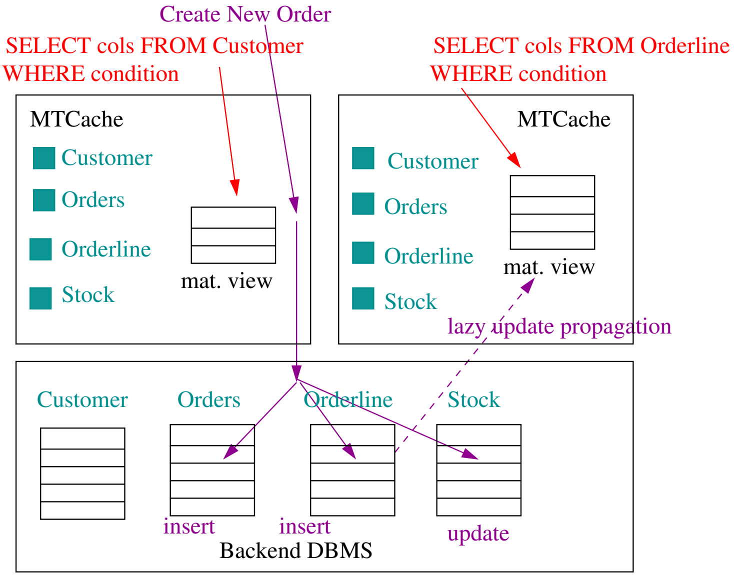
Synchronization



Synchronization



Synchronization



Query Processing 1

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

Query Processing 1

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Suppose query is:

```
SELECT SUM(0L_QUANTITY) FROM OrderLine
WHERE OL_O_ID = 1533
```

Query Processing 1

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Suppose query is:

```
SELECT SUM(OL_QUANTITY) FROM OrderLine
WHERE OL_O_ID = 1533
```

- MTCache can execute this query locally, and avoid contacting the backend DBMS

Query Processing 1

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Suppose query is:

```
SELECT SUM(OL_QUANTITY) FROM OrderLine
WHERE OL_O_ID = 1533
```

- MTCache can execute this query locally, and avoid contacting the backend DBMS
- If the query `OL_O_ID` were 3555, then MTCache would have to forward the query to the backend DBMS

Query Processing 2

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

Query Processing 2

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Suppose query is:

```
SELECT SUM(L.QUANTITY)
FROM OrderLine L, Order O, Customer C
WHERE L.OL_O_ID = O.O_ID
AND O.O_C_ID = C.C_ID
AND C.C_LAST = ``Smith''
AND O.O_ID < 2000
```

Query Processing 2

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Suppose query is:

```
SELECT SUM(L.QUANTITY)
FROM OrderLine L, Order O, Customer C
WHERE L.OL_O_ID = O.O_ID
AND O.O_C_ID = C.C_ID
AND C.C_LAST = ``Smith''
AND O.O_ID < 2000
```

- MTCache can execute part of the query locally and part at the backend, or it can send the entire query to the backend. Decision is cost-based

Parameterized Queries

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

Parameterized Queries

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Consider this query:

```
SELECT SUM(OL_QUANTITY) FROM OrderLine
WHERE OL_O_ID = @ID
```

Parameterized Queries

- Suppose MTCache has:

```
SELECT * FROM OrderLine
WHERE OL_O_ID < 3000
```

- Consider this query:

```
SELECT SUM(OL_QUANTITY) FROM OrderLine
WHERE OL_O_ID = @ID
```

- SQL Server may have to optimize this query before the value of the parameter (`@ID`) is known.
- In the case, MTCache will generate a dynamic plan.

Scale-Out Experiment

- workload: TPC-W, which models e-commerce activity
- backend DBMS server: dual CPU
- mid-tier MTCache servers: single CPU
- workload is CPU-bound
- scale-out experiment: increase number of clients and number of MTCache servers to see whether throughput (WIPS) scales
    - how many WIPS per MTCache, and does it scale linearly?
    - by how much does MTCache reduce the load on the backend DBMS?

Baseline Results (No MTCache)

- browsing workload: 50 WIPS
- shopping workload: 82 WIPS
- ordering workload: 283 WIPS

Scale-Out

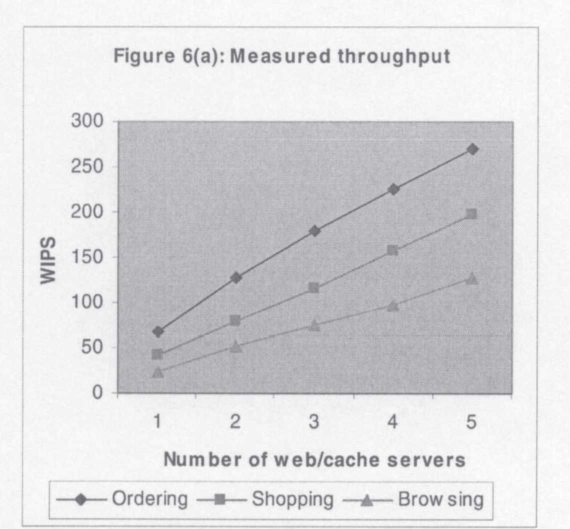


Figure 6(a): Measured throughput
WIPS
300
250
200
150
100
50
0
1
2
3
4
5
Number of web/cache servers
Ordering
Shopping
Browsing

More Scale-Out Results

Workload	No MTCache		5 MTCache		Max
	WIPS	CPU Util.	WIPS	CPU Util.	MTCache
Browsing	50	90%	129	8%	50+
Shopping	82	90%	199	16%	25+
Ordering	283	90%	271	55%	$<$10

Closing Observations

- complexity
  - interaction with many parts of DBMS (query proc, query opt, replication pub/sub, transaction, . . .)
- physical design is manual
- no synchronization guarantees, not even session guarantees (note 2005 VLDB paper [gula05])