NetStore

Leveraging Network Optimizations to Improve Distributed Transaction Processing Performance

Xu Cui, Michael Mior, Bernard Wong, Khuzaima Daudjee, Sajjad Rizvi

ACTIVE Workshop @ Middleware 2017
Las Vegas, NV, USA
December 12, 2017
A tremendous amount of data is generated and stored in the cloud.
A distributed query

Server 1
Query:{
Read: Alice.
Read: Bob}

Server 2

Server 3

Server 4

Server 5

Server 6

Server 7
Alice: 1

Server 8
Bob: 2
Part of the network is congested
A flow scheduler can solve this
If the green flows are short-lived flows
A flow scheduler cannot detect the transient congestion
The transaction query may be routed on the congested paths
NetStore

- A transaction processing system co-designed with the network enables two network-aware optimizations

  ➢ **Least bottlenecked path (LBP):** a dynamic flow scheduler that leverages information gathered from a transaction manager

  ➢ **Network-aware caching (NAC):** a database caching optimization that makes caching decisions based on the network topology
Standard database architecture

Transaction Manager

Network

Lock Requests / Lock Responses

DataServer 1

DataServer 2

DataServer 3

Lock management and transaction scheduling
The NetStore controller extends the transaction manager with a network manager.
Least bottlenecked path (LBP)

- The database and network co-design enables NetStore to maintain a global view of the network
- LBP uses this dynamic flow information to approximate the bandwidth allocation for each new flow
- LBP routes the new flow through the best path
LBP can detect the transient network congestion caused by short-lived flows.
LBP selects the best path for each transaction flow
NetStore configures network paths when the system bootstraps.
Benefits of least bottlenecked path

- Makes informed routing decisions based on the dynamic flow information gathered from the transaction manager.
- Balances the network load for short-lived transactional flows when transient network congestion is present.
Network-aware caching (NAC)

- The co-design enables network-aware caching
- NAC leverages cache replicas to reduce the load on the network
- NAC avoids cache invalidations which can increase the network load
DataServer 2 performs a read query on Alice
The NetStore controller maintains a cache index of the cache entries

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The NetStore controller creates a version number for each cache entry.

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
DataServer 2 fetches Alice from DataServer 3
DataServer 2 stores the cache replica and the version number locally.
DataServer 1 performs a read operation on Alice
The NetStore controller determines the best cache replica location for this op.

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
The NetStore controller adds server id 1 to Alice’s cache index

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>2 =&gt; 2, 1</td>
<td>1</td>
</tr>
</tbody>
</table>
DataServer 1 fetches the data from DataServer 2
DataServer 1 stores the result in its local cache
A write operation is performed on Alice
The NetStore controller erases Alice

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>2, 1</td>
<td>1</td>
</tr>
</tbody>
</table>
A new version number is generated when another read operation happens

<table>
<thead>
<tr>
<th>Key</th>
<th>DataServer IDs</th>
<th>Cache Version #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Benefits of network-aware caching

- Augments a database optimization with network-awareness
- Reduces the load on the network
- Avoids cache invalidations
- Performs batch-processing to further improve performance
Experimental setup

- We use Mininet to build a distributed virtual multi-rooted tree network
  - 64 virtual servers
  - Each virtual server runs a transaction client, a transaction server, a background client and a background server
  - 1 Gbps capacity on each link
Experimental setup

- The controller runs on a dedicated machine
- We use a synthetic workload that performs read and write operations
- The key selection process follows a Zipfian distribution with a distribution constant of 0.99
- We use ECMP as a baseline for comparison
Experimental setup: default system parameters

<table>
<thead>
<tr>
<th>NetStore parameter</th>
<th>default values</th>
</tr>
</thead>
<tbody>
<tr>
<td>write transaction percentage</td>
<td>1%</td>
</tr>
<tr>
<td>foreground op data size</td>
<td>6KB</td>
</tr>
<tr>
<td>max background flow data size</td>
<td>8000KB</td>
</tr>
<tr>
<td>average fg interarrival time</td>
<td>50ms</td>
</tr>
<tr>
<td>average bg interarrival time</td>
<td>1000ms</td>
</tr>
<tr>
<td># of key-value pairs in database</td>
<td>2,000,000</td>
</tr>
<tr>
<td># of metadata cache entries in controller</td>
<td>20,000</td>
</tr>
<tr>
<td># of cache entries in each data server</td>
<td>20,000</td>
</tr>
</tbody>
</table>
ECMP vs NetStore: varying the size of background flows
ECMP vs NetStore: varying number of operations in transactions
Conclusion

- We made the case for co-designing cloud applications with network optimizations to improve performance
- NetStore is distributed transaction processing system that offers network-aware optimizations
- NetStore significantly reduces average transaction completion time when parts of the network are saturated
Thank you.

Contact: xcui@uwaterloo.ca