Load Balancing with JET:

Just Enough Tracking for Connection Consistency

Gal Mendelson, Stanford
Shay Vargaftik, VMware Research
Dean H. Lorenz, IBM Research – Haifa
Kathy Barabash, IBM Research – Haifa
Isaac Keslassy, Technion
Ariel Orda, Technion
Load Balancing

Clients

Load Balancer

Servers

Load Balancing with JET: Just Enough Tracking for Connection Consistency
Per-Connection Consistency (PCC)
Per-Connection Consistency – PCC Violation

Load Balancer

Broken connection

Clients

Servers

Load Balancing with JET: Just Enough Tracking for Connection Consistency
Hash-Based Load-Balancing

Server = Hash(Connection ID) % 4

✓ Even distribution
  - Balls in bins
✓ Efficient calculation
✓ “Stateless”
Hash-Based Load-Balancing

Upon adding a new server:
- Most destinations change
- Most exiting connections would break

Server = Hash(Connection ID) % 5
Load-Balancing with a **Consistent Hash**

Server = ConsistentHash(Conn. ID)

- Even distribution, efficient calculation, no state
- Several available algorithms
  - E.g., Ring hash, Highest Random Weight, Maglev hash, AnchorHash
- Most destinations don’t change upon adding a new server
  - A few connections still break
Stateful Load-Balancing

- Remember per-connection state
  - ✓ Never violate PCC
    - » For the tracked connection
  - ✗ Need enough space for Connection Tracking
    - » More state to sync for distributed LBs
  - ✗ Need line-rate key lookups and updates
    - » Many optimizations (Bloom filters, HW-assisted, etc.)

- Used in practice
  - ▪ Maglev, Katran, NGINX, HAProxy
Stateful LB Flow

Receive Packet

Already Tracked?

yes

no

Forward to Destination

Compute Destination

Consistent Hash

Connection Tracking

Extract 5-Tuple

yes

no

Track
“Stateless” Load-Balancing

- Stateful, but no state at load-balancer

For example:

- State may be saved at back-end servers
  - Redirect to correct server if needed
    » E.g., Faild (NSDI ‘18), Beamer (NSDI ‘18)

- State may be saved at user
  - Cookies
    » E.g., Cheetah (NSDI ‘20)
“Stateless” Load-Balancing

- Stateful, but no state at load-balancer
- Stateful, but state at back-end servers
  - “Does this connection belong to me?”
  - If not, daisy-chain to correct server
    » E.g., Failed, Beamer
- State may be saved at user
  - Widely used above L4
    » DNS resolution
    » Cookies
  - L4 cookies
    » E.g., Cheetah

This work is about stateful load-balancers
Stateful Load-Balancing with JET

Remember per-connection state, but **Just Enough Tracking** for maintaining PCC.
How Much is “Just Enough Tracking”? 

- Answer: very little! (if you are careful)
  - Only track connections that would otherwise break

- Consistent-hashing:
  - **Server addition**
    - Only $\approx \frac{1}{N}$ connections are remapped
    - These must be tracked to preserve PCC
  - **Server removal**
    - Only connections on removed server are remapped
    - These connections would break $\Rightarrow$ no need to track

- Tracking $\approx \frac{1}{N}$ of connections is “just enough” to preserve PCC!
  - Naturally extends to multiple additions/removals
    - Tracking $\sim 10\%$ of connections can be “just enough” (see paper for details)
Preparing for Server Additions

- **Horizon set**
  - Servers are added only from horizon set

- **Warm-up period**
  - Allow packet arrival from affected connections
  - Paced server addition $\rightarrow$ small horizon
    - E.g., if slower than TCP idle timeout then horizon can be a *single server*

- **Removed servers are handled instantly**
  - Transient failures are put in horizon set
    - Expected to be added back
Which Connections to Track?

- Answer: Ask the Consistent-Hash

- We implemented this for several consistent hash algorithms
  - Ring Hash
  - Highest Random Weight (HRW)
  - Table-based HRW
  - AnchorHash

- Very little overhead
  - Only 1 extra bit per entry in CH data structure

- See paper of details
JET Flow

Algorithm 1 JET

```plaintext
1: function GET_DESTINATION(k)
2: s ← CT[k]  
3: \textbf{if} not s \textbf{then}
4: s ← CH(W, k)  
5: \textbf{if} s ≠ CH(W ∪ H, k) \textbf{then}
6: CT[k] ← s
7: return s  
```

- Receive Packet
- Already Tracked?
- Compute Destination
- Forward to Destination
- Extract 5-Tuple
- Connection Tracking
- Consistent Hash
- Should Track

Flow:

- **Receive Packet**
- **Extract 5-Tuple**
- **Connection Tracking**
- **Consistent Hash**
- **Should Track**

Decision Path:

- **Already Tracked?**
  - yes: Forward to Destination
  - no: **Compute Destination**

Consistency Check:

- **Consistent Hash**
  - yes: Track
  - no: Already Tracked?

Tracking Logic:

- **Receive Packet**
- **Already Tracked?**
  - yes: Forward to Destination
  - no: **Compute Destination**

Tracking Algorithm:

```plaintext
1: function GET_DESTINATION(k)
2: s ← CT[k]  
3: \textbf{if} not s \textbf{then}
4: s ← CH(W, k)  
5: \textbf{if} s ≠ CH(W ∪ H, k) \textbf{then}
6: CT[k] ← s
7: return s  
```
A Word on AnchorHash

- A new scalable consistent hash we developed
  - Ultra fast, small memory footprint, excellent balance
  - See our paper in ToN ‘21
  - Code available at https://github.com/anchorhash

- Works especially well with JET – no warmup period needed
Evaluation

– Event-based simulations
  - Inspired by evaluation of Cheetah, NSDI ‘20
    » 468 servers
    » Up to 40 backend changes per minute
    » Varying connection rates

– Traces
  - Real traces
  - Synthetic traces

– Reproducibility
  - Code available at https://github.com/anchorhash/jetlb
PCC Violations

- 468 servers
- 100K active connections on average at any time
- 1K seconds (~16 minutes)
- JET (overlayed in black) with 10% horizon (47 servers)
Balance, Tracking and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

<table>
<thead>
<tr>
<th>34.1M Packets</th>
<th>1.6M flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum oversubscription</td>
<td></td>
</tr>
<tr>
<td>Tracked connections</td>
<td></td>
</tr>
<tr>
<td>Rate pkt/sec [millions]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n=500 Table-based HRW</th>
<th>AnchorHash</th>
<th>MaglevHash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full CT</strong></td>
<td><strong>JET</strong></td>
<td><strong>Full CT</strong></td>
</tr>
<tr>
<td>1.139 ±0.017</td>
<td>1.139 ±0.017</td>
<td>1.052 ±0.004</td>
</tr>
<tr>
<td>1,602,007 ±0</td>
<td>145,378 ±895,286</td>
<td>1,602,007 ±0</td>
</tr>
<tr>
<td>22.883 ±2.573</td>
<td>45.567 ±4.113</td>
<td>22.702 ±0.134</td>
</tr>
</tbody>
</table>
Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

<table>
<thead>
<tr>
<th>34.1M Packets</th>
<th>1.6M flows</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Full CT</th>
<th>JET</th>
<th>AnchorHash</th>
<th>Full CT</th>
<th>JET</th>
<th>MaglevHash</th>
<th>Full CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table-based HRW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum oversubscription</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.602,007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>145,578</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±0.017</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.052</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±0.004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.054</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±0.005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tracked connections</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.883</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±2.573</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.702</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±0.134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.446</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>±2.839</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rate pkt/sec [millions]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

<table>
<thead>
<tr>
<th>34.1M Packets</th>
<th>1.6M flows</th>
<th>n=500</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Table-based HRW</td>
<td>AnchorHash</td>
</tr>
<tr>
<td></td>
<td>Full CT</td>
<td>JET</td>
</tr>
<tr>
<td>Maximum oversubscription</td>
<td>1.139</td>
<td>1.139</td>
</tr>
<tr>
<td>Tracked connections</td>
<td>1,602,007</td>
<td>145,378</td>
</tr>
<tr>
<td>Rate pkt/sec [millions]</td>
<td>22.883</td>
<td>45.367</td>
</tr>
</tbody>
</table>
Balance, Tracking volume and Rate

- JET and full CT achieve the same balance
  - Use the same CH
- JET tracks less than 10% compared to full CT
- JET achieves higher rate due to smaller CT tables
  - Better caching

<table>
<thead>
<tr>
<th>34.1M Packets</th>
<th>1.6M flows</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Maximum oversubscription</th>
<th>Table-based HRW</th>
<th>AnchorHash</th>
<th>MaglevHash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full CT</td>
<td>JET</td>
<td>Full CT</td>
</tr>
<tr>
<td></td>
<td>1.139 ± 0.017</td>
<td>1.139 ± 0.017</td>
<td>1.052 ± 0.004</td>
</tr>
<tr>
<td>Tracked connections</td>
<td>1,602,007</td>
<td>145,378</td>
<td>1,602,007</td>
</tr>
<tr>
<td></td>
<td>+0</td>
<td>+895,286</td>
<td>+0</td>
</tr>
<tr>
<td>Rate pkt/sec [millions]</td>
<td>22.883 ± 2.573</td>
<td>45.567</td>
<td>22.702</td>
</tr>
</tbody>
</table>

n=500
More In The Paper

- JET formulation
- Pseudo-code for several consistent hash algorithms
- Theoretical guarantees
- Extensive evaluation
- Contact: galmen@stanford.edu
Thank you!
EXAMPLE
Adapting Ring hash to JET
Ring Hash 101

Ring: sorted list of tuples

\[( \text{hash}(S_3), S_3 ) \]
\[( \text{hash}(S_1), S_1 ) \]
\[( \text{hash}(S_2), S_2 ) \]

Ring.get(key):

Search the sorted list for the successor of hash(key)

Example:

Ring.get(key_1) = S_3
Ring.get(key_2) = S_2
What if we add server $H_1$? (it is in the horizon set)

Ring: sorted list of tuples

- $(hash(H_1), H_1)$
- $(hash(S_3), S_3)$
- $(hash(S_1), S_1)$
- $(hash(S_2), S_2)$

If we add server $H_1$ then:

- $Ring.get(key_1) = H_1 \Leftarrow$ changed
  - $key_1$ should be tracked
- $Ring.get(key_2) = S_2 \Leftarrow$ unchanged
  - $key_2$ should not be tracked
Add a “tracking” bit to each entry

Ring: sorted list of tuples

- $(\text{hash}(H_1), H_1, \text{Track}=\text{TRUE})$
- $(\text{hash}(S_3), S_3, \text{Track}=\text{FALSE})$
- $(\text{hash}(S_1), S_1, \text{Track}=\text{FALSE})$
- $(\text{hash}(S_2), S_2, \text{Track}=\text{FALSE})$

Ring.get(key):
   Also return whether tracking is needed

Example:

- Ring.get(key$_1$) = $H_1$, Track=TRUE
- Ring.get(key$_2$) = $S_2$, Track=FALSE
Should still not return $H_1$

Ring: sorted list of tuples

- $(\text{hash}(H_1), S_3, \text{Track}=\text{TRUE})$
- $(\text{hash}(S_3), S_3, \text{Track}=\text{FALSE})$
- $(\text{hash}(S_1), S_1, \text{Track}=\text{FALSE})$
- $(\text{hash}(S_2), S_2, \text{Track}=\text{FALSE})$

Ring.get(key):

Return whether tracking is needed

Example:

- $\text{Ring.get}(key_1) = S_3, \text{Track}=\text{TRUE}$
- $\text{Ring.get}(key_2) = S_2, \text{Track}=\text{FALSE}$