LOSUG January 2009
MySQL/DTrace and Memcached

Martin MC Brown
Technical Writer, Database Group
Sun Microsystems
Today’s Topics

• DTrace and MySQL
  > How it works
  > What you can do with the DTrace Probes
  > Live Demo!

• Memcached
  > What it isnt
  > What it is
  > How to use It
  > Live Demo?
MySQL and DTrace

- MySQL
  - Database
  - SQL Based
  - Query Optimization is Key

- DTrace
  - Monitors Application Execution through Probes
  - Monitors anything, including time
  - Monitor production applications

- MySQL+DTrace
  - Best way to get execution info from MySQL
Availability

- OpenSolaris/Solaris Compatible Probes in 6.0.8
- Extended set of probes coming in 6.0.10
- Extended probes (based on 6.0.10) in OpenSolaris/MySQL 5.1
How MySQL Executes a Query

1. Query
2. Query Cache?
3. Parse
4. Start Execution
5. Query Optimizer
6. Locking
7. Execute Statement
8. Filesort
9. Results
Query Cache, Parsing and Locks

• Query Cache
  > Returns queries from memory if the SQL statement matches
  > Not perfect for all environments
  > Knowing when QC is used can be vital

• Parsing
  > Determines tables, required fields, and core information used by the optimizer

• Locks
  > Read, Write, External Locks
  > Locks can delay execution on busy servers between threads
Storage Engine

- MySQL Supports multiple Storage Engines
- MySQL/SE Interface is based on individual rows
- Engines provide hints to optimizer on execution
- Different Engines return information in different ways
- Slow downs in one engine aren’t replicated
EXPLAIN

mysql> explain select * from t1 order by s limit 10;
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>t1</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>2097152</td>
<td>Using filesort</td>
</tr>
</tbody>
</table>
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
1 row in set (0.03 sec)

mysql> explain select * from t1 order by s limit 10;
+----+-------------+-------+-------+---------------+------+---------+------+------+-------------+
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>t1</td>
<td>index</td>
<td>NULL</td>
<td>tlb</td>
<td>86</td>
<td>NULL</td>
<td>10</td>
<td>Using index</td>
</tr>
</tbody>
</table>
+----+-------------+-------+-------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
Probe Sets

- Query
- Query Parsing
- Query Cache
- Query Execution
- Locks
- Statements
- Row-Level
- Filesort
- Network
Getting Execution Times

- **query-start**(query, connectionid, database, user, host)
  - query - query text
  - connectionid - MySQL process ID
  - database - DB name
  - user - user name
  - host - client host

- **query-done**(status)

- Combine with the built-in timestamp to get some execution times
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
    printf("%-20s %-20s %-40s %2s %-9s\n", "Who", "Database", "Query", "QC", "Time(ms)" );
}

mysql*:::query-start
{
    self->query = copyinstr(arg0);
    self->connid = arg1;
    self->db     = copyinstr(arg2);
    self->who    = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
    self->querystart = timestamp;
    self->qc = 0;
}

mysql*:::query-cache-hit
{
    self->qc = 1;
}

mysql*:::query-cache-miss
{
    self->qc = 0;
}

mysql*:::query-done
{
    printf("%-20s %-20s %-40s %-2s %-9d\n", self->who,self->db,self->query,(self->qc ? "Y" : "N"),
        (timestamp - self->querystart) / 1000000);
Live Demo

• Here is where the fun begins....
Getting More Detail

- Find out how much time is spent parsing
- Time spent purely *executing* statement
- Time spent in locks
- Time spent transferring data
- Time spent doing a filesort
Live Demo

• Watch the birdie!
DTrace and Enterprise Monitor

• MySQL Enterprise Monitor
  > Monitors an entire enterprise of MySQL servers
  > Provides live query analysis
  > Works using proxy/redirection
  > Adds tiny overhead
DTrace Feeding Enterprise Monitor

- Monitor Supports REST interface
- Take the DTrace query stats
- Pass the query stats up to Enterprise Monitor
- Doesn’t need the proxy
- Lower overhead
- Collates the data for multiple servers of DTrace probes
Where Next

- Go deeper into Storage Engines
- Get statistics on global server operations
- Get statistics on general locks and structures
- Get probes into other parts of the Webstack
Query Optimization Only Gets <> Far

• Query optimization speeds up queries
• You don’t always need to execute a query
• Query Cache isn’t quite what we mean
• Bigger cache
• More general purpose cache
• Flexible
• Cluster-like features
What memcached is

- Big memory cache into which you can store what you want
- Accessible from multiple applications, languages, environments
- Client-driven fault tolerance
- Client-driven data distribution (*not* replication)
- Exceedingly easy to use
- Unix/Linux
- In OpenSolaris soon
What memcached isn’t

• Not a database
  > It’s a cache
• Not persistent
  > It’s a cache
• Not clustered
  > It’s a cache
• No replication
  > It’s a cache
Execution during Load

1. Start
2. Request Key from memcached
3. Does key exist?
   - No: Load data from database
   - Yes: Update memcached
4. Use data
Execution during Save

1. Start
2. Store data into database
3. Update memcached
4. Use data
What do I mean by Client Driven

- Imagine you have multiple servers
- You store data by a unique ID (user-1234)
- Client chooses which server to store the data on using hash on key ID
- Client writes data
- Another client, looking for user-1234, has the same list of servers, runs the same hash algorithm and chooses the same server, and loads the data
- Hashing algorithms
Cache Management

• Keys exist in cache until:
  > Explicitly removed (delete)
  > Removed through lack of use (Least Recently Used (LRU))
  > Entry expires

• Specific Expiry
  > Allows finer control over expiry
  > Useful for sessions
  > Specify an absolute time (epoch)
  > Specify a relative time; object will expire within # seconds of store
Some things to ponder

• It’s a cache
• Cache what you need; not everything
• Don’t worry about ‘filling it up’
• Don’t worry about ‘seeding it’
• Don’t worry about replication
Some more things to ponder

• Don’t panic about server failures
  > But do consider the consequences

• Don’t panic about cache misses
  > You can load it from the DB (it’s a cache!)
  > But do investigate the reasons if they are excessive

• Don’t cache things you don’t need
  > Images
  > Files that can be accessed directly through Apache
Questions

Martin ‘MC’ Brown
Technical Writer, Database Group
Sun Microsystems
mc.brown@sun.com

sun.com/mysql