MySQL Performance Blog

Measuring failover time for ScaleArc load balancer

Peter Boros
August 19, 2014

Measuring failover time for ScaleArc load balancer

August 19, 2014 by Peter Boros

ScaleArc hired Percona to benchmark failover times for the ScaleArc database traffic management software in different scenarios. We tested failover times for various clustered setups, where ScaleArc itself was the load balancer for the cluster. These tests complement other performance tests on the ScaleArc software – sysbench testing for latency and testing for WordPress acceleration.
We tested failover times for Percona XtraDB Cluster (PXC) and MHA (any traditional MySQL replication-based solution works pretty much the same way).

In each case, we tested failover with a rate limited sysbench benchmark. In this mode, sysbench generates roughly 10 transactions each second, even if not all 10 were completed in the previous second. In that case, the newly generated transactions are queued.

The sysbench command we used for testing is the following.

```
Shell
# while true ; do sysbench --test=sysbench/tests/db/oltp.lua
                           --mysql-host=172.31.10.231
                           --mysql-user=root
                           --mysql-password=admin
                           --mysql-db=sbtest
                           --oltp-table-size=10000
                           --oltp-tables-count=4
                           --num-threads=4
                           --max-time=0
                           --max-requests=0
                           --report-interval=1
                           --tx-rate=10
                           run ; sleep 1; done
```

The command is run in a loop, because typically at the time of failover, the application receives some kind of error while the virtual IP is moved, or while the current node is declared dead. Well-behaving applications are reconnecting and retrying in this case. Sysbench is not a well-behaving application from this perspective – after failover it has to be restarted.

This is good for testing the duration of errors during a failover procedure – but not the number of errors. In a simple failover scenario (ScaleArc is just used as a regular load balancer), the number of errors won't be any higher than usual with ScaleArc's queueing mechanism.

ScaleArc+MHA

In this test, we used MHA as a replication manager. The test result would be similar regardless of how its asynchronous replication is managed – only the ScaleArc level checks would be different. In the case of MHA, we tested graceful and non-graceful failover. In the graceful case, we stopped the manager and performed a manual master switchover, after which we informed the ScaleArc software via an API call for failover.

We ran two tests:

- A manual switchover with the manager stopped, switching over manually, and informing the load balancer about the change.
- An automatic failover where the master was killed, and we let MHA and the ScaleArc software discover it.

ScaleArc+MHA manual switchover

The manual switchover was performed with the following command.

```
# time (
          masterha_master_switch --conf=/etc/cluster_5.cnf
          --master_state=alive
          --new_master_host=10.11.0.107
          --orig_master_is_new_slave
          --interactive=0
          &&
          curl -k -X PUT https://172.31.10.30/api/cluster/5/manual_failover
          -d '{"apikey": "0c8aa9384ddf2a5756299a9e7650742a87bbb550"}' )

{"success":true,"message":"4214   Failover status updated successfully.","timestamp":1404465709,"data
real    0m8.698s
user    0m0.302s
sys     0m0.220s
```

The curl command calls ScaleArc's API to inform the ScaleArc software about the master switchover.

During this time, sysbench output was the following.

```
[  21s] threads: 4, tps: 13.00, reads/s: 139.00, writes/s: 36.00, response time: 304.07ms (95%)
[  21s] queue length: 0, concurrency: 1
[  22s] threads: 4, tps: 1.00, reads/s: 57.00, writes/s: 20.00, response time: 570.13ms (95%)
[  22s] queue length: 8, concurrency: 4
[  23s] threads: 4, tps: 19.00, reads/s: 237.99, writes/s: 68.00, response time: 976.61ms (95%)
[  23s] queue length: 0, concurrency: 2
[  24s] threads: 4, tps: 9.00, reads/s: 140.00, writes/s: 40.00, response time: 477.55ms (95%)
[  24s] queue length: 0, concurrency: 3
[  25s] threads: 4, tps: 10.00, reads/s: 105.01, writes/s: 28.00, response time: 586.23ms (95%)
[  25s] queue length: 0, concurrency: 1
```

Only a slight hiccup is visible at 22 seconds. In this second, only 1 transaction was done, and 8 others were queued. These results show a sub-second failover time. The reason no errors were received is that ScaleArc itself queued the transactions during the failover process. If the transaction in question were done from an interactive client, the queuing itself would be visible as increased response time – for example a START TRANSACTION or an INSERT command is taking longer than usual, but no errors result. This is as good as it gets for graceful failover. ScaleArc knows about the failover (and in the case of a switchover initiated by a DBA, notifying the ScaleArc software can be part of the failover process).

The queueing mechanism is quite configurable. Administrators can set up the timeout for the queue – we set it to 60 seconds, so if the failover doesn't complete in that timeframe transactions start to fail.

ScaleArc+MHA non-graceful failover

In the case of the non-graceful failover MHA and the ScaleArc software have to figure out that the node died.

```
[  14s] threads: 4, tps: 11.00, reads/s: 154.00, writes/s: 44.00, response time: 1210.04ms (95%)
[  14s] queue length: 4, concurrency: 4
( sysbench restarted )
[   1s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   1s] queue length: 13, concurrency: 4
[   2s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   2s] queue length: 23, concurrency: 4
[   3s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   3s] queue length: 38, concurrency: 4
[   4s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   4s] queue length: 46, concurrency: 4
[   5s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   5s] queue length: 59, concurrency: 4
[   6s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   6s] queue length: 69, concurrency: 4
[   7s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   7s] queue length: 82, concurrency: 4
[   8s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   8s] queue length: 92, concurrency: 4
[   9s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[   9s] queue length: 99, concurrency: 4
[  10s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  10s] queue length: 108, concurrency: 4
[  11s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  11s] queue length: 116, concurrency: 4
[  12s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  12s] queue length: 126, concurrency: 4
[  13s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  13s] queue length: 134, concurrency: 4
[  14s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  14s] queue length: 144, concurrency: 4
[  15s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  15s] queue length: 153, concurrency: 4
[  16s] threads: 4, tps: 0.00, reads/s: 0.00, writes/s: 0.00, response time: 0.00ms (95%)
[  16s] queue length: 167, concurrency: 4
[  17s] threads: 4, tps: 5.00, reads/s: 123.00, writes/s: 32.00, response time: 16807.85ms (95%)
[  17s] queue length: 170, concurrency: 4
[  18s] threads: 4, tps: 18.00, reads/s: 255.00, writes/s: 76.00, response time: 16888.55ms (95%)
[  18s] queue length: 161, concurrency: 4
```

The failover time in this case was 18 seconds. We looked at the results and found that a check the ScaleArc software does (which involves opening an SSH connection to all nodes in MHA) take 5 seconds, and ScaleArc declares the node dead after 3 consecutive checks (this parameter is configurable). Hence the high failover time. A much lower one can be achieved with more frequent checks and a different check method – for example checking the read-only flag, or making MHA store its state in the databases.

ScaleArc+Percona XtraDB Cluster tests

Percona XtraDB Cluster is special when it comes to high availability testing because of the many ways one can use it. Many people write only to one node to avoid rollback on conflicts, but we have also seen lots of folks using all the nodes for writes.

Also, graceful failover can be interpreted differently.

- Failover can be graceful from both Percona XtraDB Cluster's and from the ScaleArc software's perspective. First the traffic is switched to another node, or removed from the node in question, and then MySQL is stopped.
- Failover can be graceful from Percona XtraDB Cluster's perspective, but not from the ScaleArc software's perspective. In this case the database is simply stopped with service mysql stop, and the load balancer figures out what happened. I will refer to this approach as semi-graceful from now on.
- Failover can be completely non-graceful if a node is killed, where neither Percona XtraDB Cluster, nor ScaleArc knows about its departure.

We did all the tests using one node at a time, and using all three nodes. What makes these test sets even more complex is that when only one node at a time used, some tests (the semi-graceful and the non-graceful one) don't have the same result if the node removed is the used one or an unused one. This process involves a lot of tests, so for the sake of brevity, I omit the actual sysbench output here – they look either like the graceful MHA and non-graceful MHA case – and only present the results in a tabular format. In the case of active/active setups, to remove the nodes gracefully we first have to lower the maximum number of connections on that node to 0.

Failover type	1 node (active)	1 node (passive)	all nodes
Graceful	sub-second (no errors)	no effect at all	sub-second (no errors)
Semi-graceful	4 seconds (errors)	no effect at all	3 seconds
Non-graceful	4 seconds (errors)	6 seconds (no errors)	7 seconds (errors)

In the previous table, active means that the failed node did receive sysbench transactions and passive means that it didn't.

All the graceful cases are similar to MHA's graceful case.

If only one node is used and a passive node is removed from the cluster, by stopping the database itself gracefully with the

```
Shell
1 # service mysql stop
```

command, it doesn't have an effect on the cluster. For the subsequent cases of the graceful failover, switching on ScaleArc will enable queuing similar to MHA's case. In case of the semi-graceful, if the passive node (which has no traffic) departs, it has no effect. Otherwise, the application will get errors (because of the unexpected mysql stop), and the failover time is around 3-4 seconds for the cases when only 1 node is active and when all 3 are active. This makes sense, because ScaleArc was configured to do checks (using clustercheck) every second, and declare a node dead after three consecutive failed checks. After the ScaleArc software determined that it should fail over, and it did so, the case is practically the same as the passive node's removal from that point (because ScaleArc removes traffic in practice making that node passive).

The non-graceful case is tied to suspect timeout. In this case, XtraDB Cluster doesn't know that the node departed the cluster, and the originating nodes are trying to send write sets to it. Those write sets will never be certified because the node is gone, so the writes will be stalled. The exception here is the case when the active node failed. After ScaleArc figures out that the node dies (three consecutive checks at one second intervals) a new node is chosen, but because only the failed node was doing transactions, no write sets are in the remaining two nodes' queues, which are waiting for certification, so there is no need to wait for suspect timeout here.

Conclusion

ScaleArc does have reasonably good failover time, especially in the case when it doesn't have to interrupt transactions. Its promise of zero-downtime maintenance is fulfilled both in the case of MHA and XtraDB Cluster.

Filed Under: Benchmarks, Insight for DBAs, MySQL, Percona MySQL Consulting, Percona Software, Percona XtraDB ClusterTagged With: failover time, load balancer, MHA, ScaleArc

About Peter Boros

Peter joined the European consulting team in May 2012. Before joining Percona, among many other things, he worked at Sun Microsystems, specialized there in performance tuning and was a DBA at Hungary's largest social networking site. He also taught many Oracle University MySQL courses. He has been using and working with open source software from early 2000s. Peter's first and foremost professional interest is performance tuning.

He currently lives in Budapest, Hungary with his wife and son.