HammerDB TPC-C Distribution of Response Times

This guide contributed by Wes Vaske gives a detailed guide on generating a distribution of response times for the TPC-C workload and the original post can be found here.

Context

HammerDB provides only a couple metrics at the end of a run (NOPM and TPM). These are good but don't tell the whole story of database performance. Many of us would like to get Response Time data as well but there's an issue to doing this--measuring the response time will have an impact on the response time.

One method for capturing response times is to use the **etprof** package for tcl. This works reasonably well (very low overhead) but only provides the Average response time for each transaction. I would also like to see the 99.9th percentile response time, median, and--at times--the histogram for the distribution of response times.

Below I describe how I've implemented this in my workload scripts.

Solution

NOTE: This solution requires a secondary script to parse the log files after a run and do the analysis outside of HammerDB / tcl

This solution will have a percentage of the users to capturing the response times while running their transactions. It is up to you, the benchmarker, to test and determine to what degree this data capture influences the performance.

At the top of the workload file I define two parameters that define the percentage of users used to measured response times and the number of times each user will keep before flushing to the log file.

```
set measpct 5;  # Percent of users to measure ART
set listlimit 1000; # Size of lists for ART. Larger uses more memory.
```

At the beginning of the Switch -> Default section we define whether this specific user is a timing user. The intent is that with a percentage greater than 0 we always have at least one user collecting timing data.

```
                # Timing User
                # The minuses and plusses are to ensure that values
perform correctly
```

```
                # with the int() calls
                if { [expr int(($myposition - 3)*($measpct/100.0) +
1)] != [expr int(($myposition - 2)*($measpct/100.0) + 1)] } {
                        set TIMING 1
                } else {
                        set TIMING 0
                }
```

Note: I wrote this section almost a year ago. If someone can test to be sure this behaves how you'd expect I'd appreciate the feedback

Now that we know which users are collecting data, we need to actually collect data. At the start of the data run, just after "RUN TPC-C" I initialize the variables we'll use for the response time lists:

```
                if { $TIMING } {
                        set newordrts {}
                        set paymentrts {}
                        set ostatrts {}
                        set deliveryrts {}
                        set slevrts {}
                        set RAMPING 1
                }
```

I don't want to capture data during the ramp up period so I do an estimation to only get data during the rampup. Add this section before the main execution loop:

```
                if { $TIMING } {
                        puts "     User will provide time data..."
                        set rampstart [clock seconds]
                        set rampstop [ expr $rampstart+(60*$rampup)]
                }
```

And immediately after the check for application abort inside the execution loop:

```
if { $TIMING } { if { $rampstop <= [clock seconds] } { set RAMPING 0 } }
```

Now for each transaction we modify the transaction block a bit to add a check for TIMING and RAMPING as well as the start and stop timers when the check passes. After executing the transaction we add the time to the list we initialized above and when the list hits our limit we flush to the log files:

```
set choice [ RandomNumber 1 23 ]
if {$choice <= 10} {
if { $TIMING && !$RAMPING } { set newordstart [clock microseconds] }
if { $KEYANDTHINK } { keytime 18 }
neword $lda $w_id $w_id_input $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 12 }
if { $TIMING && !$RAMPING } {
set newordstop [clock microseconds]
set newordrt [expr { $newordstop - $newordstart}]
if { [llength $newordrts] >= $listlimit } {
puts "newordrts: $newordrts"
set newordrts {}
                }
lappend newordrts $newordrt
        }
```

```
} elseif {$choice <= 20} {
if { $TIMING && !$RAMPING } { set paymentstart [clock microseconds] }
if { $KEYANDTHINK } { keytime 3 }
payment $lda $w_id $w_id_input $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 12 }
if { $TIMING && !$RAMPING } {
set paymentstop [clock microseconds]
set paymentrt [expr { $paymentstop - $paymentstart}]
if { [llength $paymentrts] >= $listlimit } {
puts "paymentrts: $paymentrts"
set paymentrts {}
                }
lappend paymentrts $paymentrt
   }

} elseif {$choice <= 21} {
if { $TIMING && !$RAMPING } { set deliverystart [clock microseconds] }
if { $KEYANDTHINK } { keytime 2 }
delivery $lda $w_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 10 }
if { $TIMING && !$RAMPING } {
set deliverystop [clock microseconds]
set deliveryrt [expr { $deliverystop - $deliverystart}]
if { [llength $deliveryrts] >= $listlimit } {
puts "deliveryrts: $deliveryrts"
set deliveryrts {}
        }
lappend deliveryrts $deliveryrt
   }
} elseif {$choice <= 22} {
if { $TIMING && !$RAMPING } { set slevstart [clock microseconds] }
if { $KEYANDTHINK } { keytime 2 }
slev $lda $w_id $stock_level_d_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 5 }
if { $TIMING && !$RAMPING } {
set slevstop [clock microseconds]
set slevrt [expr { $slevstop - $slevstart}]
if { [llength $slevrts] >= $listlimit } {
puts "slevrts: $slevrts"
set slevrts {}
    }
lappend slevrts $slevrt
}
} elseif {$choice <= 23} {
if { $TIMING && !$RAMPING } { set ostatstart [clock microseconds] }
if { $KEYANDTHINK } { keytime 2 }
ostat $lda $w_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 5 }
if { $TIMING && !$RAMPING } {
set ostatstop [clock microseconds]
set ostatrt [expr {$ostatstop - $ostatstart}]
if { [llength $ostatrts] >= $listlimit } {
puts "ostatrts: $ostatrts"
set ostatrts {}
    }
lappend ostatrts $ostatrt
}
```

```
                        }
```

And outside the execution loop we add the last values to the log file:

```
                if { $TIMING } {
                        puts "Timing Data Complete"

                        puts "newordrts: $newordrts"
                        puts "paymentrts: $paymentrts"
                        puts "ostatrts: $ostatrts"
                        puts "deliveryrts: $deliveryrts"
                        puts "slevrts: $slevrts"
                }
```

Note: The values collected are in microseconds

Analysis

Now that we have a sample of our transaction response times in the log files, we need to do something with it. I personally use python for much of my automation tasks and a parsing script would look something like this:

```
 1 #!/usr/bin/python
 2
 3 import numpy as np
 4
 5 logfile = "hammerdb_56DFB10252D903E233039393.log"
 6 values = {}
 7 totalrts = []
 8
 9 # Each list will hold the cummulative response time data for
10 # each transaction type. The response time lists in the log
11 # file are not part of the default HammerDB script.
12 rtfields = {'deliveryrts': [], 'slevrts': [], 'newordrts': [],
13     'ostatrts': [], 'paymentrts': []}
14
15 with open(logfile, 'r') as openLog:
16     for line in openLog:
17         # If the line contains one of these keys then it contains
18         # response time data. We append this to the list.
19         # Not part of the standard HammerDB script.
20         for f in rtfields.keys():
21             if f in line:
22                 rtfields[f] += line.split(':')[2].split()
23
24         # Get NOPM and TPM from the test results output section.
25         # This is standard for any run of HammerDB
26         if "TEST RESULT" in line:
27             ln = line.split()
28             values["TPM"] = int(ln[6])
29             values["NOPM"] = int(ln[10])
30             print "%s:\t%s NOPM"%(logfile,values["NOPM"])
```

```
31
32 # Calculate totals and statistics for manually calculated
33 # lists of response times.
34 for rt in rtfields.values():
35     totalrts.extend(rt)
36
37 rtfields['totalrts'] = totalrts
38
39 for f in rtfields.keys():
40     # There are two options:
41     # 1. Create Numpy arrays and do statistics on those
42     # 2. Do statistics on lists of ints
43     #
44     # Creating Numpy arrays takes slightly longer than converting
45     # the lists of strings to lists of ints but the statistics on
46     # a Numpy array is SIGNIFICANTLY faster than statistics on abs
47     # list of ints.
48     rtfields[f] = np.array(rtfields[f])
49     rtfields[f] = rtfields[f].astype(np.int)
50
51     # Calculate different cutoffs. eg: 95th percentile, 99th percentile
52     # Get length of list and find member that is xx% to the end of the
53 list.
54     percents = [90, 95, 98, 99, 99.9]
55     pdata = map(lambda x: int(x)/1000.0, np.percentile(rtfields[f],
56 percents))
57
58     for i, p in enumerate(percents):
59         values["%s %.1f%%"%(f,p)] = pdata[i]
60
61     for func in [np.min, np.mean, np.median, np.std, np.max]:
62         values["%s %s"%(f,func.func_name)] =
63 int(func(rtfields[f]))/1000.0

   for i in values.items():
       print i
```

This will give an output like:

```
('deliveryrts amin', 8.049)
('totalrts median', 10.655)
('paymentrts 9.0%', 3.088)
('totalrts 99.9%', 92.889)
('totalrts std', 9.873)
('newordrts 98.0%', 30.679)
('ostatrts amax', 82.099)
('ostatrts median', 3.87)
('deliveryrts amax', 159.678)
('deliveryrts 9.0%', 15.152)
('deliveryrts 95.0%', 47.85)
('deliveryrts 99.9%', 98.814)
('newordrts 9.0%', 8.584)
('newordrts amax', 322.164)
('deliveryrts mean', 25.793)
('ostatrts amin', 0.436)
('paymentrts 99.0%', 30.1)
('paymentrts amax', 205.827)
```

```
('slevrts 98.0%', 84.684)
('paymentrts median', 6.059)
('ostatrts std', 5.461)
('newordrts amin', 3.227)
('totalrts 99.0%', 49.975)
('paymentrts mean', 7.925)
('newordrts std', 7.017)
('deliveryrts median', 22.847)
('paymentrts amin', 1.019)
('totalrts 9.0%', 3.403)
('newordrts mean', 15.27)
('totalrts amin', 0.436)
('newordrts 99.9%', 80.88)
('newordrts median', 14.181)
('totalrts 98.0%', 39.036)
('ostatrts 99.0%', 26.596)
('slevrts mean', 23.033)
('paymentrts 98.0%', 25.257)
('newordrts 99.0%', 35.634)
('totalrts amax', 322.164)
('TPM', 265303)
('slevrts 95.0%', 69.99)
('slevrts 99.9%', 155.144)
('ostatrts 95.0%', 15.98)
('slevrts amax', 190.533)
('slevrts 9.0%', 2.525)
('totalrts 95.0%', 27.684)
('slevrts amin', 1.63)
('paymentrts std', 6.168)
('ostatrts mean', 5.536)
('ostatrts 99.9%', 44.429)
('paymentrts 99.9%', 56.708)
('ostatrts 9.0%', 1.164)
('deliveryrts std', 11.36)
('deliveryrts 99.0%', 64.438)
('newordrts 95.0%', 25.765)
('slevrts median', 11.562)
('NOPM', 115216)
('deliveryrts 98.0%', 57.18)
('slevrts 99.0%', 95.868)
('slevrts std', 24.537)
('paymentrts 95.0%', 19.158)
('totalrts mean', 12.438)
('ostatrts 98.0%', 22.327)
```

Summary

I'll leave it as an exercise for the reader to use matplotlib to draw the full histogram for the latency distribution.