Commit Write Facility; dml2.sql

Author: Craig Shallahamer (craig@orapub.com), Version 1i, 15-Feb-2012.

Background and Purpose

The purpose of this notepad is to see if using Oracle's commit write facility increases performance measured by a decrease in response time (ms/commits) and increase in arrival rate.

Because it can be difficult to compare the response times when the arrival rate changes, at the bottom of the notebook I complete the relationship between work and time, and then set the work to be same then derived the response time. This allows a pretty good apples-to-apples comparision.

The data used in the notebook is based on the dml2.sql script, which creates a heavy IO load and a light CPU load on the system. But the top wait event during the wait,immediate option is still log file sync.

The terminalolgy used is a mix of queury theory and Oracle database-ease. Without getting into details, CPU time per unit of work is equal to the service time, the non-idle wait time per unit of work is the queue time, and respons time is service time plus queue time. Key to understanding this is knowing the time is based on a single unit of work. For this experiment the chosen unit of work was the Oracle statistic "user commit". Every commit issued by an Oracle client/user process will tick-up the user commit statistics. As a result, the user commit is a very good unit of work when understanding the commit rate of an application or workload.

Experimental Data

Below is all the experimental data. The experiment was run on a Dell single four-core CPU, Oracle 11.2G. According to "cat /proc/version": Linux version 2.6.18-164.el5PAE (mockbuild@ca-build10.us.oracle.com) (gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)) #1 SMP Thu Sep 3 02:28:20 EDT 2009.
There was an intense DML update load by five sessions. **The DML script is dml2.sql**, which in the experimental enviroment cause a severe IO bottleneck with lots of log file sync time. This is stark contrast with the dml1.sql script, which places a very heavy CPU bottleneck on the system with a minimal IO load. You can see this reflected in difference in the service time vs queue time.

After each update and commit, there is no delay, as there is with the dml1.sql statement.

During each of the data collection periods, the follow data was gathered. The percentage figures were visually observed and recorded.

In[1]:=

```
Grid[{
  {"Setting", "Commit/sec", "CPU %",
   "log file sync\nWait Time %", "log file par write\nWait Time %"},
  {"wait immediate", "0.521", "4%", "84%", "14%"},
  {"wait batch", "0.526", "4%", "80%", "13%"},
  {"nowait batch", "12.639", "45%", "0%", "14%"},
  {"nowait immediate", "13.004", "42%", "0%", "14%"}
 }, Frame → All
]
```

Out[1]=

Setting	Commit/sec	CPU %	log file sync Wait Time %	log file par write Wait Time %
wait immediate	0.521	4%	84%	14%
wait batch	0.526	4%	80%	13%
nowait batch	12.639	45%	0%	14%
nowait immediate	13.004	42%	0%	14%

The order of sample data is sample number, elapsed time (seconds), commits (user commits), instance non-idle wait time (sec), and instance CPU consumption (sec).

In[2]:=

```
ssWaitImmediate = {1, 60.008406, 30785, 430.85, 7.703822, 2, 60.008711, 31480, 430.36, 7.690829, 3,
   60.012054, 31487, 431.8, 7.970787, 4, 60.011059, 31295, 429.52, 7.571844, 5, 60.010173, 31544,
   431.65, 7.765815, 6, 60.009867, 31980, 428.93, 7.749821, 7, 60.009749, 30002, 431.8, 7.372878, 8,
   60.010644, 30259, 432.3, 7.524857, 9, 60.01019, 32366, 428.79, 7.861801, 10, 60.008361, 32114,
   429.71, 7.68483, 11, 60.011048, 31795, 430.92, 7.840805, 12, 60.008273, 32438, 430.44, 7.915797,
   13, 60.008556, 32311, 429.05, 7.864801, 14, 60.008422, 32183, 430.76, 7.920795, 15, 60.008963,
   31708, 432.62, 7.683831, 16, 60.009383, 32423, 430.66, 7.66783, 17, 60.009748, 31085, 431.1,
   7.427874, 18, 60.009899, 29004, 430.83, 7.15891, 19, 60.009035, 22894, 427.52, 5.834111, 20,
   60.009977, 31155, 432.65, 7.466864, 21, 60.009028, 32147, 431.36, 7.802813, 22, 60.010415, 32026,
   430.58, 7.919795, 23, 60.009577, 31833, 431.61, 7.757821, 24, 60.009814, 31735, 430.51, 7.838808,
   25, 60.009899, 32171, 429.71, 7.838803, 26, 60.011159, 31226, 432.3, 7.559851, 27, 60.008331,
   31500, 428.97, 7.63884, 28, 60.009205, 30950, 434.3, 7.63784, 29, 60.008219, 30643, 431.23,
   7.707828, 30, 60.001083, 31825, 429.15, 7.517856, 31, 60.009353, 32028, 429.47, 7.806817};
ssWaitBatch = {1, 60.009641, 30881, 431.13, 7.714822, 2, 60.008586, 31817, 432.48, 7.735821, 3,
   60.008945, 31861, 432.41, 7.659832, 4, 60.008505, 31937, 429.32, 7.826805, 5, 60.00836, 31837,
   433.77, 7.784815, 6, 60.009848, 33030, 431.61, 7.973784, 7, 60.008275, 29362, 431.23, 7.270888,
   8, 60.009245, 31286, 431.9, 7.64084, 9, 60.010145, 31554, 429.29, 7.333883, 10, 60.01033, 31536,
   431.85, 7.445865, 11, 60.010792, 31189, 431.87, 7.513857, 12, 60.009273, 32244, 430.35, 7.679823,
   13, 60.009289, 31786, 433.97, 7.517857, 14, 60.009592, 31458, 431.08, 7.632837, 15, 60.008821,
   32065, 428.26, 7.611843, 16, 60.009427, 31000, 431.45, 7.59984, 17, 60.008932, 30885, 432.5,
   7.678829, 18, 60.009505, 31075, 431.5, 7.622839, 19, 60.008194, 31694, 429.15, 7.836806, 20,
   60.008865, 31375, 433.94, 7.581849, 21, 60.008736, 31345, 431.34, 7.654836, 22, 60.011288, 31267,
   433.48, 7.506856, 23, 60.009067, 31480, 429.44, 7.659837, 24, 60.010203, 31242, 431.21, 7.697827,
   25, 60.009333, 31442, 429.32, 7.784812, 26, 60.009741, 32200, 430.56, 7.604843, 27, 60.00888,
   32447, 429.73, 7.731821, 28, 60.008177, 31839, 429.22, 7.61484, 29, 60.009339, 32178, 433.02,
   7.826808, 30, 60.011494, 33052, 430.68, 7.612839, 31, 60.008221, 31047, 431.41, 7.545852};
ssNowaitImmediate = {1, 60.014233, 704566, 351.85, 91.513086, 2, 60.015941, 709865, 351.36,
   90.018315, 3, 60.048768, 720186, 349.26, 92.859879, 4, 60.019765, 844098, 332.23, 108.576494,
   5, 60.038182, 778142, 343.93, 100.75768, 6, 60.012766, 761136, 347.3, 98.499027, 7, 60.02937,
   772002, 345.46, 99.757831, 8, 60.007971, 699253, 353.27, 91.275119, 9, 60.009925, 769054,
   341.54, 100.924656, 10, 60.017388, 699061, 357.02, 91.037156, 11, 60.025007, 724373, 348.65,
   92.750899, 12, 60.01599, 875475, 328.93, 112.809846, 13, 60.010936, 779319, 347.89, 100.818675,
   14, 60.017474, 623452, 374.45, 80.669736, 15, 60.008167, 715967, 350.84, 93.148839, 16, 60.006185,
   680493, 350.46, 87.899632, 17, 60.006519, 792380, 332.78, 101.9285, 18, 60.007605, 710246,
   351.18, 91.405104, 19, 60.006086, 775069, 341.38, 102.460416, 20, 60.008553, 780974, 341.03,
   101.898506, 21, 60.008447, 835669, 334.51, 106.91974, 22, 60.013039, 772112, 339.38, 99.858819,
   23, 60.004641, 834981, 328.27, 106.892747, 24, 60.019324, 761265, 348.07, 97.852123, 25, 60.04922,
   793573, 341.81, 101.973492, 26, 60.045199, 770164, 342.58, 98.541018, 27, 60.034915, 718676,
   351.11, 91.825036, 28, 60.070325, 802175, 344.76, 102.571407, 29, 60.095324, 783637, 343.95,
   100.6247, 30, 60.007424, 771040, 347.04, 100.627699, 31, 60.026255, 758240, 351.73, 97.682152};
ssNowaitBatch = {1, 60.016962, 743487, 346.34, 97.633153, 2, 60.008339, 756281, 346.11, 97.416189,
   3, 60.013531, 731830, 347.67, 94.32566, 4, 60.047524, 816435, 339.76, 105.173013, 5, 60.008653,
   781651, 344.52, 98.510026, 6, 60.017361, 780417, 342.54, 102.255455, 7, 60.034785, 806230,
   346.71, 104.669086, 8, 60.007203, 781086, 345.59, 101.213614, 9, 60.095628, 723871, 352.83,
   91.328114, 10, 60.017674, 870313, 329.86, 113.993667, 11, 60.026442, 800735, 343.13, 103.64624,
   12, 60.008774, 800785, 345.25, 103.489265, 13, 60.008967, 720795, 351, 93.557775, 14, 60.026278,
   764945, 348.17, 100.716688, 15, 60.027589, 760403, 350.62, 98.243062, 16, 60.007748, 812654,
   343.27, 103.072323, 17, 60.069637, 737380, 349.83, 95.734447, 18, 60.015943, 785307, 338.83,
   102.88236, 19, 60.010426, 833832, 332.75, 110.21524, 20, 60.009513, 815090, 334.51, 105.268995,
   21, 60.015481, 754827, 349.83, 98.062091, 22, 60.010747, 819027, 336.94, 107.607638, 23,
   60.014711, 781002, 343.65, 102.166464, 24, 60.008223, 772320, 345.48, 100.635697, 25, 60.00204,
   761687, 347.42, 98.438033, 26, 60.033799, 780991, 341.55, 101.209613, 27, 60.008022, 768536,
   342.86, 102.287451, 28, 60.003977, 792038, 337.22, 102.926348, 29, 60.00675, 778765, 342.38,
   102.137471, 30, 60.009828, 765151, 347.05, 98.810978, 31, 60.037474, 797805, 343.91, 103.294291};
```

Data Loading

All the data sets are contained in the above section.

In[6]:=
```
ssNum = 4;
sampleNum = 31;
ss[1] = ssWaitImmediate; ssName[1] = "Wait Immediate";
ss[2] = ssWaitBatch; ssName[2] = "Wait Batch";
ss[3] = ssNowaitImmediate; ssName[3] = "Nowait Immediate";
ss[4] = ssNowaitBatch; ssName[4] = "Nowait Batch";

ncols = 5;
sampleCol = 1;
elapsedTCol = 2;
workCol = 3;
waitTCol = 4;
cpuTCol = 5;

Do[
  ssWork[ssidx] = {}; ssL[ssidx] = {}; ssSt[ssidx] = {}; ssQt[ssidx] = {}; ssRt[ssidx] = {};
  theSS = ss[ssidx];
  Table[
   elapsedT = theSS[[ncols sampleidx + elapsedTCol]];
   workTot = theSS[[ncols sampleidx + workCol]];
   cpuSecTot = theSS[[ncols sampleidx + cpuTCol]];
   waitSecTot = theSS[[ncols sampleidx + waitTCol]];
   (*Print[ssidx," ",sampleidx," elapsedT=",elapsedT];*)

   λ = workTot / (1000 elapsedT);
   St = (cpuSecTot 1000) / workTot;
   Qt = (waitSecTot 1000) / workTot;
   Rt = St + Qt;

   AppendTo[ssWork[ssidx], workTot];
   AppendTo[ssL[ssidx], λ];
   AppendTo[ssSt[ssidx], St];
   AppendTo[ssQt[ssidx], Qt];
   AppendTo[ssRt[ssidx], Rt];

   , {sampleidx, 0, sampleNum - 1}
  ];
  , {ssidx, ssNum}
 ];
ssName[1]
ss[1]
Length[ssWork[1]]
Take[ssWork[1], 5]
N[Mean[ssWork[1]]]
```

Out[19]=
```
Wait Immediate
```

Out[20]=
```
{1, 60.0084, 30 785, 430.85, 7.70382, 2, 60.0087, 31 480, 430.36, 7.69083, 3, 60.0121, 31 487, 431.8,
 7.97079, 4, 60.0111, 31 295, 429.52, 7.57184, 5, 60.0102, 31 544, 431.65, 7.76582, 6, 60.0099,
 31 980, 428.93, 7.74982, 7, 60.0097, 30 002, 431.8, 7.37288, 8, 60.0106, 30 259, 432.3, 7.52486,
 9, 60.0102, 32 366, 428.79, 7.8618, 10, 60.0084, 32 114, 429.71, 7.68483, 11, 60.011, 31 795,
 430.92, 7.84081, 12, 60.0083, 32 438, 430.44, 7.9158, 13, 60.0086, 32 311, 429.05, 7.8648, 14,
 60.0084, 32 183, 430.76, 7.9208, 15, 60.009, 31 708, 432.62, 7.68383, 16, 60.0094, 32 423, 430.66,
 7.66783, 17, 60.0097, 31 085, 431.1, 7.42787, 18, 60.0099, 29 004, 430.83, 7.15891, 19, 60.009,
 22 894, 427.52, 5.83411, 20, 60.01, 31 155, 432.65, 7.46686, 21, 60.009, 32 147, 431.36, 7.80281,
 22, 60.0104, 32 026, 430.58, 7.9198, 23, 60.0096, 31 833, 431.61, 7.75782, 24, 60.0098, 31 735,
 430.51, 7.83881, 25, 60.0099, 32 171, 429.71, 7.8388, 26, 60.0112, 31 226, 432.3, 7.55985, 27,
 60.0083, 31 500, 428.97, 7.63884, 28, 60.0092, 30 950, 434.3, 7.63784, 29, 60.0082, 30 643,
 431.23, 7.70783, 30, 60.0011, 31 825, 429.15, 7.51786, 31, 60.0094, 32 028, 429.47, 7.80682}
```

Out[21]=
```
31
```

Out[22]=
```
{30 785, 31 480, 31 487, 31 295, 31 544}
```

Out[23]=
```
31 238.5
```

Basic Statistics

In this section I calculate the basic statistics, such as the mean and median. My objective is to ensure the data has been collected and entered correctly and also to compare the two datasets to see if they appear to be different.

In[24]:=
```
myData = Table[
   {
    ssName[ssidx], N[Mean[ssWork[ssidx]]],
    Mean[ssL[ssidx]], Mean[ssSt[ssidx]], Mean[ssQt[ssidx]], Mean[ssRt[ssidx]],
    Length[ssWork[ssidx]], N[StandardDeviation[ssL[ssidx]]], N[StandardDeviation[ssSt[ssidx]]],
    N[StandardDeviation[ssQt[ssidx]]], N[StandardDeviation[ssRt[ssidx]]]
   }, {ssidx, 1, ssNum}
  ];
toGrid = Prepend[myData, {"Settings", "Avg Work\n(cmt)", "Avg L\n(cmt/ms)",
    "Avg CPUt\n(ms/cmt)", "Avg Wt\n(ms/cmt)", "Avg Rt\n(ms/cmt)", "Samples",
    "Stdev L\n(cmt/ms)", "Stdev CPUt\n(ms/cmt)", "Stdev Wt\n(ms/cmt)", "Stdev Rt\n(ms/cmt)"}];
Grid[
 toGrid,
 Frame →
  All]
```

Out[26]=

Settings	Avg Work (cmt)	Avg L (cmt/ms)	Avg CPUt (ms/cmt)	Avg Wt (ms/cmt)	Avg Rt (ms/cmt)	Samples	Stdev L (cmt/ms)	Stdev CPUt (ms/cmt)	Stdev Wt (ms/cmt)	Stdev Rt (ms/cmt)
Wait Immediate	31238.5	0.52056	0.24456	13.8391	14.0836	31	0.0288566	0.00434349	0.969638	0.971996
Wait Batch	31593.9	0.526483	0.241944	13.656	13.898	31	0.0114818	0.00468948	0.313133	0.315873
Nowait Immediate	758601.	12.6386	0.129128	0.45859	0.587718	31	0.886997	0.00113723	0.0447581	0.0449296
Nowait Batch	780506.	13.0041	0.129801	0.441475	0.571276	31	0.552949	0.00156522	0.0246232	0.0242879

In this section I calculate the key parameters for understanding the change. The columns heading are a mix of queuing theory and Oracle-ease centric.

In[27]:=

```
myData = Table[
   {
    ssName[ssidx],
    N[Mean[ssWork[ssidx]]], N[100 * (Mean[ssWork[ssidx]] - Mean[ssWork[1]]) / Mean[ssWork[1]]],
    Mean[ssL[ssidx]], 100 * (Mean[ssL[ssidx]] - Mean[ssL[1]]) / Mean[ssL[1]],
    Mean[ssSt[ssidx]], 100 * (Mean[ssSt[ssidx]] - Mean[ssSt[1]]) / Mean[ssSt[1]],
    Mean[ssQt[ssidx]], 100 * (Mean[ssQt[ssidx]] - Mean[ssQt[1]]) / Mean[ssQt[1]],
    Mean[ssRt[ssidx]], 100 * (Mean[ssRt[ssidx]] - Mean[ssRt[1]]) / Mean[ssRt[1]],
    Length[ssWork[ssidx]]
   }, {ssidx, 1, ssNum}
  ];
toGrid = Prepend[myData,
   {"Settings", "Avg Work\n(cmt)", "%\nChange", "Avg L\n(cmt/ms)", "%\nChange", "Avg CPUt\n(ms/cmt)",
    "%\nChange", "Avg Wt\n(ms/cmt)", "%\nChange", "Avg Rt\n(ms/cmt)", "%\nChange", "Samples"}];
Grid[
 toGrid,
 Frame →
  All]
```

Out[29]=

Settings	Avg Work (cmt)	% Change	Avg L (cmt/ms)	% Change	Avg CPUt (ms/cmt)	% Change	Avg Wt (ms/cmt)	% Change	Avg Rt (ms/cmt)	% Change	Samples
Wait Immediate	31238.5	0.	0.52056	0.	0.24456	0.	13.8391	0.	14.0836	0.	31
Wait Batch	31593.9	1.13787	0.526483	1.13783	0.241944	-1.06973	13.656	-1.32262	13.898	-1.31823	31
Nowait Immediate	75860 1.	2328.42	12.6386	2327.88	0.129128	-47.1998	0.45859	-96.6863	0.587718	-95.8269	31
Nowait Batch	78050 6.	2398.54	13.0041	2398.09	0.129801	-46.9245	0.441475	-96.8099	0.571276	-95.9437	31

In this section I'm showing change and if that change is statictially significant, all compared to our baseline sample set, wait,immediate. This is pretty cool: As mentioned in the Normality Tests section below, distributions in a ttest must be normal. If not, then a location test can be performed. If you look closely at the code segment, below I tested and adjusted for this.

In[30]:=
```
myData = Table[
   {
    ssName[ssidx],
    100 * (Mean[ssL[ssidx]] - Mean[ssL[1]]) / Mean[ssL[1]],
    If[DistributionFitTest[ssL[ssidx]] ≥ 0.050,
     TTest[{ssL[1], ssL[ssidx]}],
     LocationEquivalenceTest[{ssL[1], ssL[ssidx]}]
    ],
    100 * (Mean[ssRt[ssidx]] - Mean[ssRt[1]]) / Mean[ssRt[1]],
    If[DistributionFitTest[ssRt[ssidx]] ≥ 0.050,
     TTest[{ssRt[1], ssRt[ssidx]}],
     LocationEquivalenceTest[{ssRt[1], ssRt[ssidx]}]
    ]
   }, {ssidx, 1, ssNum}
  ];
toGrid = Prepend[myData, {"Settings", "Avg L\n% Change",
    "Avg L\nTtest pvalue", "Avg Rt\n% Change", "Avg Rt\nTtest pvalue"}];
Grid[
 toGrid,
 Frame → 
  All]
```

TTest::nortst : At least one of the p-values in {0, 0.325329}, resulting from a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. ≫

TTest::nortst : At least one of the p-values in {0, 0.506652}, resulting from a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. ≫

TTest::nortst : At least one of the p-values in {0, 0.0792448}, resulting from a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. ≫

General::stop : Further output of TTest::nortst will be suppressed during this calculation. ≫

Out[32]=

Settings	Avg L % Change	Avg L Ttest pvalue	Avg Rt % Change	Avg Rt Ttest pvalue
Wait Immediate	0.	1.	0.	1.
Wait Batch	1.13783	0.292553	-1.31823	0.315888
Nowait Immediate	2327.88	6.24857×10^{-36}	-95.8269	3.40131×10^{-36}
Nowait Batch	2398.09	1.42674×10^{-42}	-95.9437	3.90295×10^{-36}

Sample Set Normality Tests

Before we can perform a standard t-test hypothesis tests on our data, we need to ensure it is normally distributed...because that is one of the underlying assumptions and requirements for properly performing a t-test.

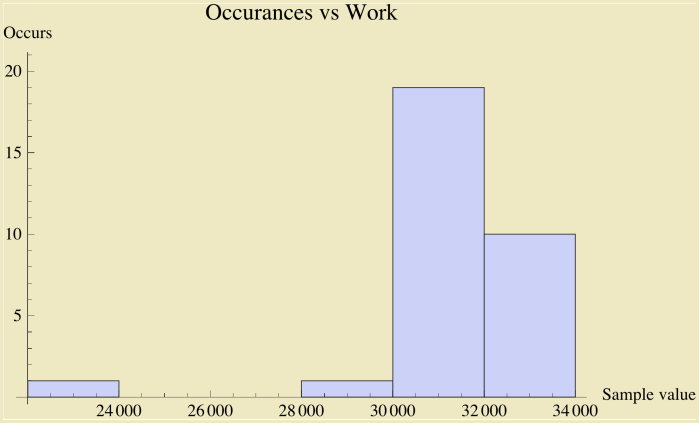
Statistical and vIsual normality test

Our alpha will be 0.05, so if the distribution fit test results in a value greater than 0.05 then we can assume the data set is indeed normally distributed.

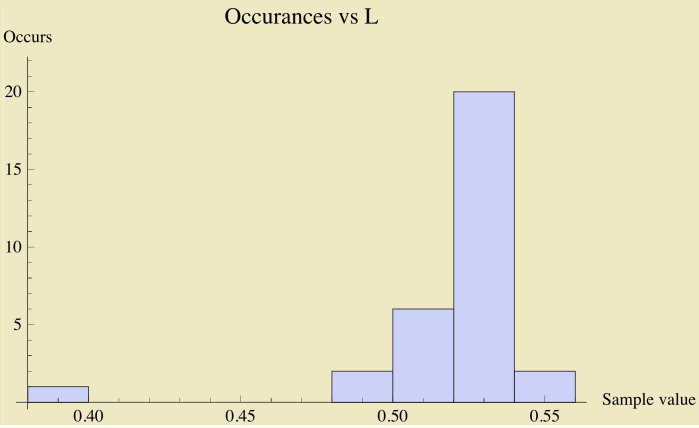
The first test is just to double check to make sure my thinking is correct. Since I creating a normal distribution based on a mean and standard deviation (just happens to be based on the my sample set data), I would expect a p-value (the result) to greatly exceed 0.05. Notice that the more samples I have created (the final number), the closer the p-value approaches 1.0.

In[33]:=
```
check = DistributionFitTest[
   RandomVariate[NormalDistribution[Mean[ssSt[1]], StandardDeviation[ssSt[1]]], 10000]];
Print["This number should be much greater than 0.05: ", check,
  " If not try again by re-evaluating."];
Do[
  Print["================================================================================"];
  Print[ssName[i], ", ", Length[ssSt[i]], " sample values"];
  pValueWork = DistributionFitTest[ssWork[i]];
  pValueL = DistributionFitTest[ssL[i]];
  pValueSt = DistributionFitTest[ssSt[i]];
  pValueQt = DistributionFitTest[ssQt[i]];
  pValueRt = DistributionFitTest[ssRt[i]];
  Print["Work pvalue=", pValueWork];
  Print[
   Histogram[ssWork[i], PlotLabel → "Occurances vs Work", AxesLabel → {"Sample value", "Occurs"}]];
  Print["---------------------------"];
  Print["L pvalue=", pValueL];
  Print[Histogram[ssL[i], PlotLabel → "Occurances vs L", AxesLabel → {"Sample value", "Occurs"}]];
  Print["---------------------------"];
  Print["St pvalue=", pValueSt];
  Print[Histogram[ssSt[i], PlotLabel → "Occurances vs St", AxesLabel → {"Sample value", "Occurs"}]];
  Print["---------------------------"];
  Print["Qt pvalue=", pValueQt];
  Print[Histogram[ssQt[i], PlotLabel → "Occurances vs Qt", AxesLabel → {"Sample value", "Occurs"}]];
  Print["---------------------------"];
  Print["Rt pvalue=", pValueRt];
  Print[Histogram[ssRt[i], PlotLabel → "Occurances vs Rt", AxesLabel → {"Sample value", "Occurs"}]];
  , {i, 1, ssNum}
 ];
```

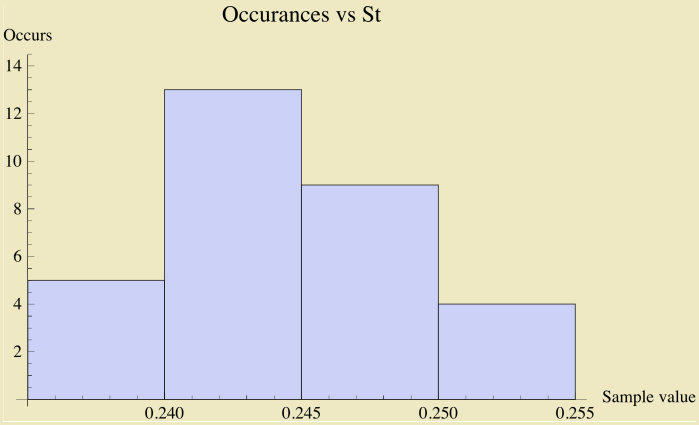
```
This number should be much greater than 0.05: 0.545721 If not try again by re-evaluating.

================================================================================

Wait Immediate, 31 sample values

Work pvalue=0
```



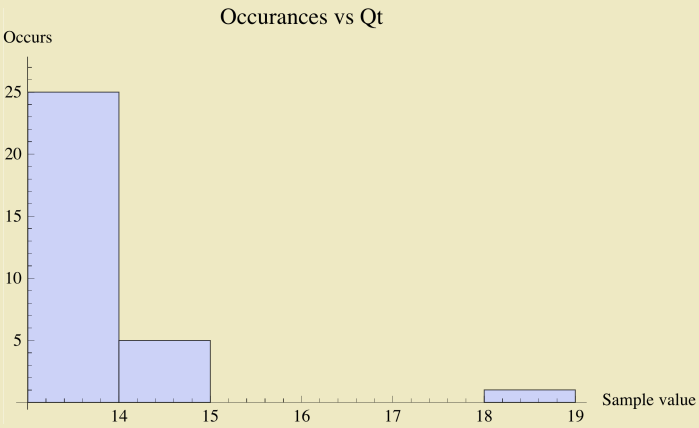

```
---------------------------

L pvalue=0
```



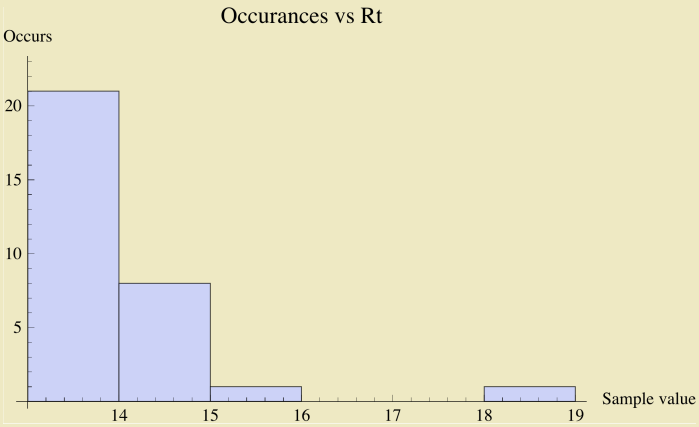
St pvalue=0.334333



Qt pvalue=0



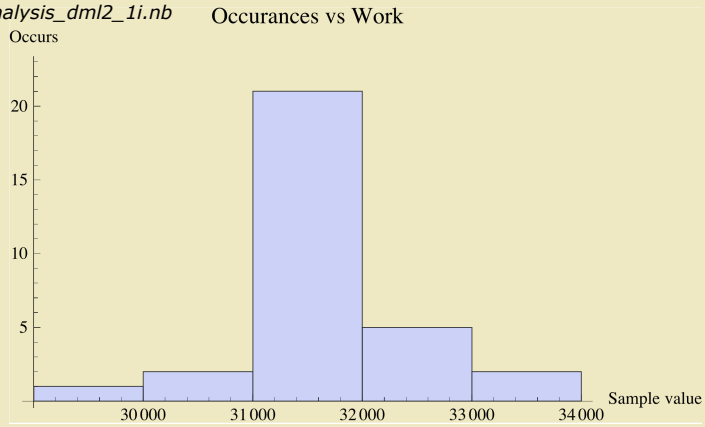
Rt pvalue=0



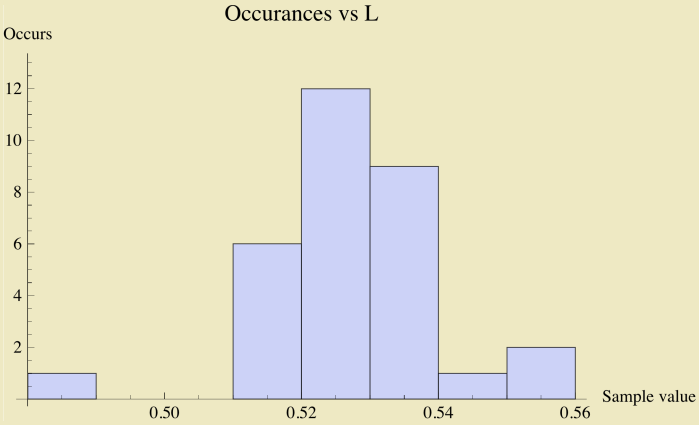
==========

Wait Batch, 31 sample values

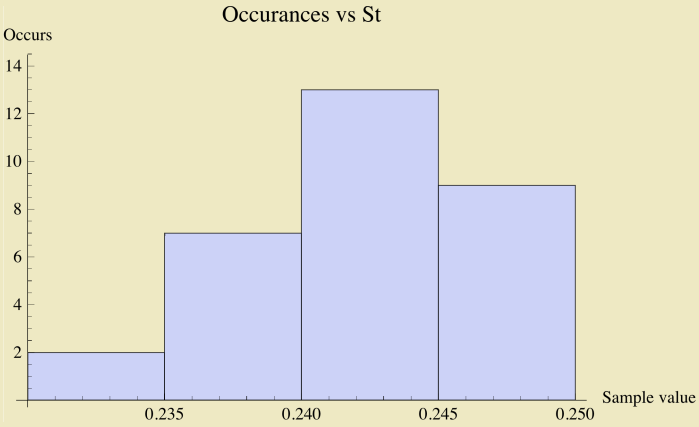
Work pvalue=0.323358



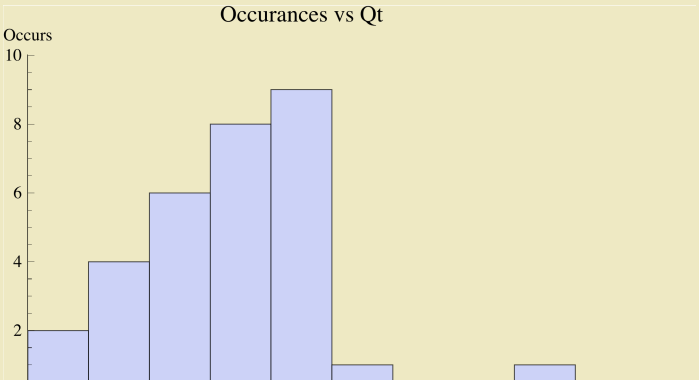
L pvalue=0.325329



St pvalue=0.557386

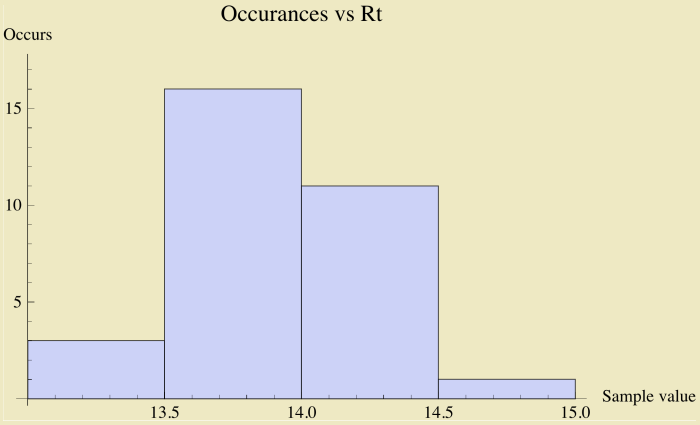


Qt pvalue=0.511312



```
---------------------------
```

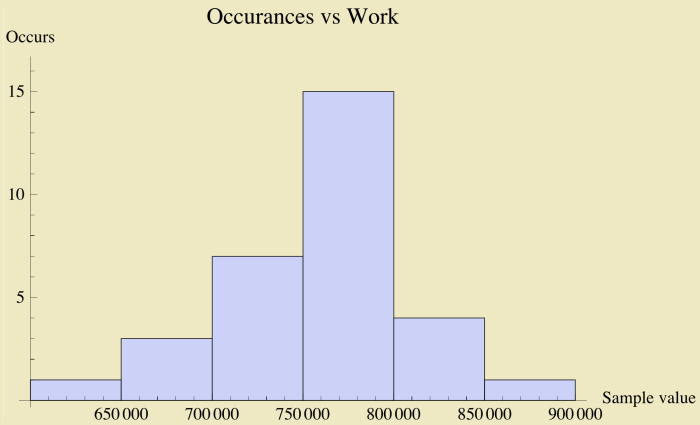
```
Rt pvalue=0.506652
```




```
================================================================================
```

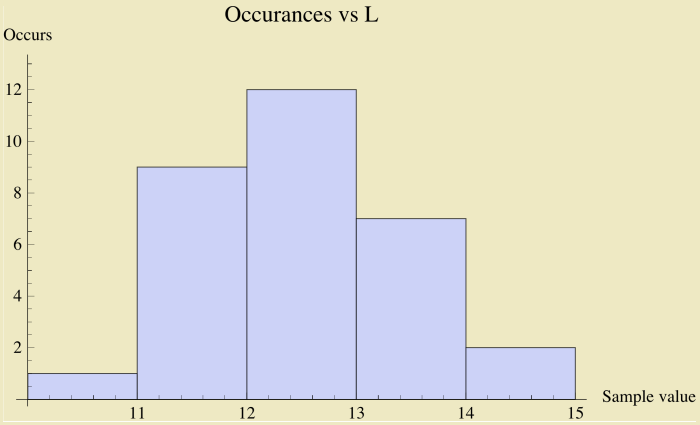
```
Nowait Immediate, 31 sample values
```

```
Work pvalue=0.0844514
```



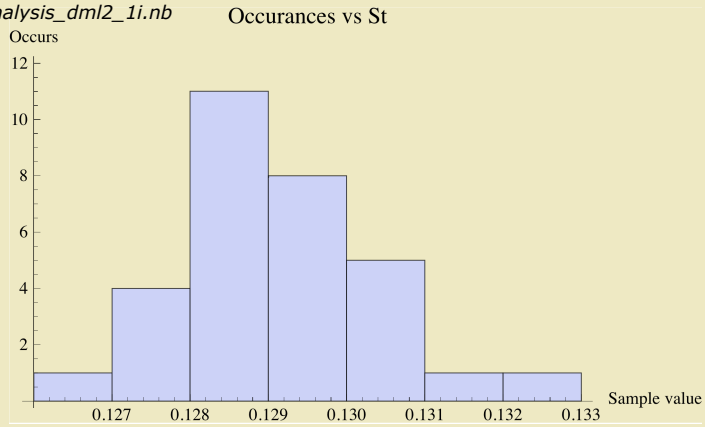

```
---------------------------
```

```
L pvalue=0.0792448
```

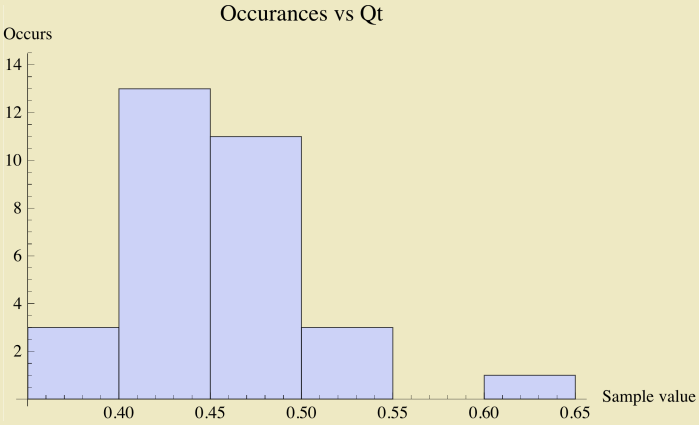



```
---------------------------
```

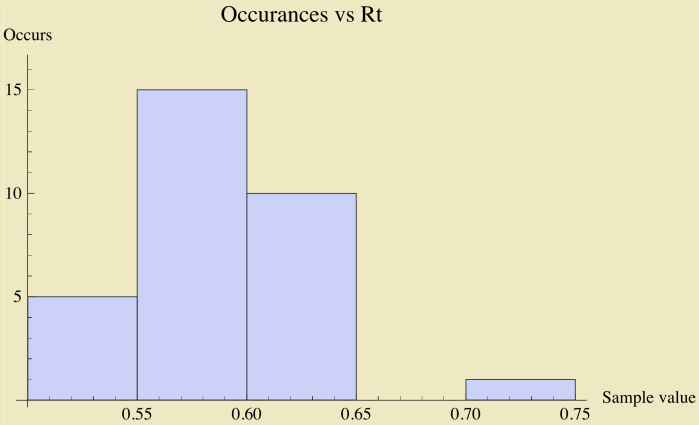
```
St pvalue=0.409065
```



Qt pvalue=0.146078



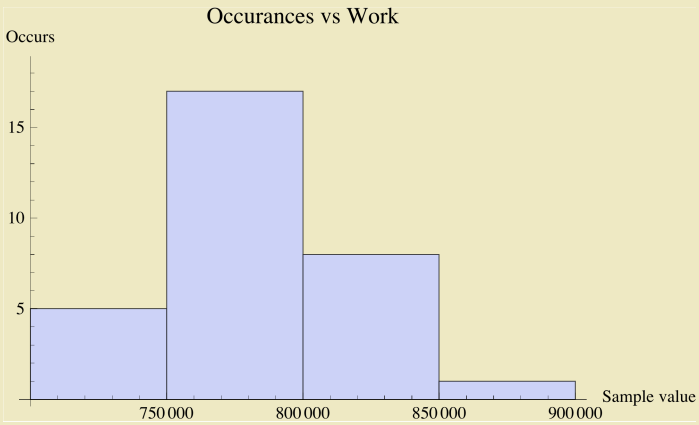
Rt pvalue=0.169537



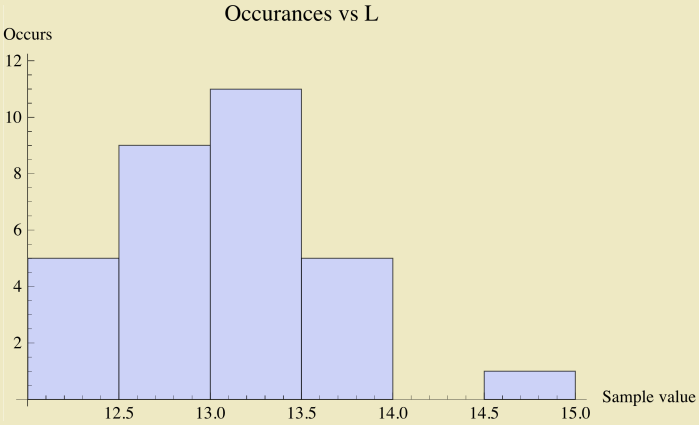
==========

Nowait Batch, 31 sample values

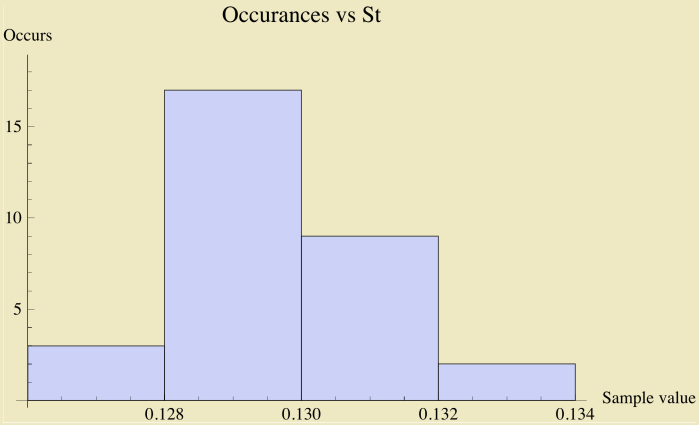
Work pvalue=0.858574



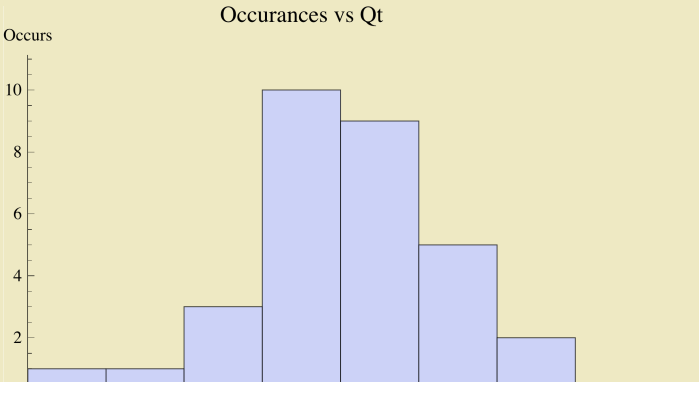
L pvalue=0.863761



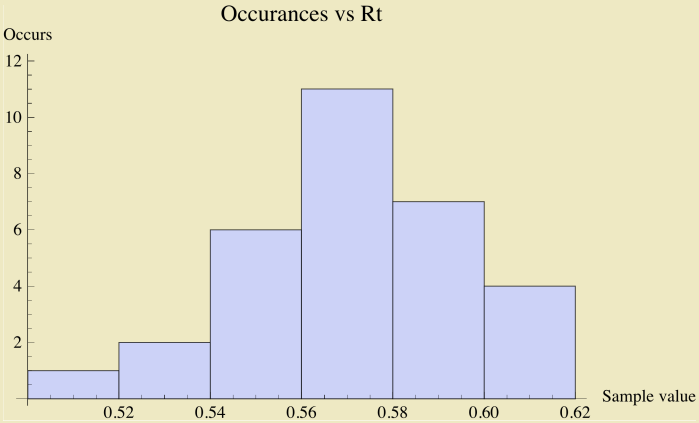
St pvalue=0.0724144



Qt pvalue=0.931557



```
Rt pvalue=0.986393
```



Sample Comparison Tests (when normality exists)

Assuming our samples **are normally distributed**, now it's time to see if they are significantly different. If so, then we know changing the commit write optoins indeed makes a significant performance difference...at least statistically.

The null hypothesis is; there is no real difference between our samples sets. We need to statistically prove that any difference is the result of randomness; like we just happened to pick poor set of samples and it makes their difference look much worse than it really is.

A t-test will produce a statistic p. The p value is a probability, with a value ranging from zero to one. It is the answer to this question: If the populations really have the same mean overall, what is the probability that random sampling would lead to a difference between sample means larger than observed?

For example, if the p value is 0.03 we can say a random sampling from identical populations would lead to a difference smaller than you observed in 97% of the experiments and larger than you observed in 3% of the experiments.

Said another way, suppose I have a single sample set and I copy it, resultling in two identical sample sets. Now suppose we perform a significance test on these two identical sample sets. The resuting p-value will be 1.0 because they are exactly the same. We are essentially doing the same thing here except we have to different sample sets... but we still want to see if they "like" each other..and in our case we hope they are NOT like each other, which means the p-value will low... below our cut off value of 0.05.

For our analysis we choose alpha of 0.05. To accept that our two samples are statistically similar the p value would need to be less than 0.05 (our alpha).

Good reference about the P-Value and significance testing: http://www.graphpad.com/articles/pvalue.htm

Here we go (assuming our samples are normally distributed):

1. Our P value threshold is 0.05, which is our alpha.
2. The null hypothesis is the two populations have the same mean. (Remember we have to sample sets, which not the population.)
3. Do the statistical test to compute the P value.
4. Compare the result P value to our threshold alpha value. If the P value is less then our threshold, we will reject the null hypothesis and say the difference between our samples is significant. However, if the P value is greater than the threshold, we cannot reject the null hypothesis and any difference between our samples are not statistically significant.

In[36]:=
```
Print["P-values assumming normality exists."];
Table[
  Table[
    If[i ≠ j,
      Print["============================================================"];
      Print[ssName[i], "(", Length[ssL[i]], ") and ", ssName[j], " (", Length[ssL[j]], ")"];
      pValueWork = TTest[{ssWork[i], ssWork[j]}];
      Print["Work: ", pValueWork];
      pValueL = TTest[{ssL[i], ssL[j]}];
      Print["L: ", pValueL];
      pValueSt = TTest[{ssSt[i], ssSt[j]}];
      Print["St: ", pValueSt];
      pValueQt = TTest[{ssQt[i], ssQt[j]}];
      Print["Qt: ", pValueQt];
      pValueRt = TTest[{ssRt[i], ssRt[j]}];
      Print["Rt: ", pValueRt];
     ];
    ,
    {j, 1, ssNum}
   ];
  , {i, 1, ssNum}
 ];
```

```
P-values assumming normality exists.

============================================================

Wait Immediate(31) and Wait Batch (31)

TTest::nortst : At least one of the p-values in {0, 0.323358}, resulting from
      a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. »

Work: 0.292536

TTest::nortst : At least one of the p-values in {0, 0.325329}, resulting from
      a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. »

L: 0.292553

St: 0.0262514

TTest::nortst : At least one of the p-values in {0, 0.511312}, resulting from
      a test for normality, is below 0.025`. The tests in {T} require that the data is normally distributed. »

General::stop : Further output of TTest::nortst will be suppressed during this calculation. »

Qt: 0.321246

Rt: 0.315888

============================================================

Wait Immediate(31) and Nowait Immediate (31)

Work: 6.33512 × 10^-36

L: 6.24857 × 10^-36

St: 5.61565 × 10^-49

Qt: 4.09416 × 10^-36

Rt: 3.40131 × 10^-36

============================================================

Wait Immediate(31) and Nowait Batch (31)

Work: 1.34728 × 10^-42

L: 1.42674 × 10^-42

St: 1.35589 × 10^-52

Qt: 4.67269 × 10^-36

Rt: 3.90295 × 10^-36

------------------------------------------------------------
```

Wait Batch(31) and Wait Immediate (31)

Work: 0.292536

L: 0.292553

St: 0.0262514

Qt: 0.321246

Rt: 0.315888

==========

Wait Batch(31) and Nowait Immediate (31)

Work: 7.11514×10^{-36}

L: 7.01919×10^{-36}

St: 6.48177×10^{-47}

Qt: 3.98069×10^{-52}

Rt: 4.15085×10^{-52}

==========

Wait Batch(31) and Nowait Batch (31)

Work: 1.90334×10^{-42}

L: 2.01294×10^{-42}

St: 6.27681×10^{-50}

Qt: 4.97031×10^{-51}

Rt: 5.26187×10^{-51}

==========

Nowait Immediate(31) and Wait Immediate (31)

Work: 6.33512×10^{-36}

L: 6.24857×10^{-36}

St: 5.61565×10^{-49}

Qt: 4.09416×10^{-36}

Rt: 3.40131×10^{-36}

==========

Nowait Immediate(31) and Wait Batch (31)

Work: 7.11514×10^{-36}

L: 7.01919×10^{-36}

St: 6.48177×10^{-47}

Qt: 3.98069×10^{-52}

Rt: 4.15085×10^{-52}

==========

Nowait Immediate(31) and Nowait Batch (31)

Work: 0.056551

L: 0.0562214

St: 0.0574033

Qt: 0.0684293

Rt: 0.079636

Nowait Batch(31) and Wait Immediate (31)

Work: 1.34728×10^{-42}

L: 1.42674×10^{-42}

St: 1.35589×10^{-52}

Qt: 4.67269×10^{-36}

Rt: 3.90295×10^{-36}

==========

Nowait Batch(31) and Wait Batch (31)

Work: 1.90334×10^{-42}

L: 2.01294×10^{-42}

St: 6.27681×10^{-50}

Qt: 4.97031×10^{-51}

Rt: 5.26187×10^{-51}

==========

Nowait Batch(31) and Nowait Immediate (31)

Work: 0.056551

L: 0.0562214

St: 0.0574033

Qt: 0.0684293

Rt: 0.079636

If the above T-Test results (p value) are less then our threshold we can say there is a significant difference between the two sample sets.

Sample Comparison Tests (when normality may NOT exist)

If our sample sets are **not normally distributed,** we can not perform a simple t-test. We can perform what are called location tests. I did some research on significance testing when non-normal distributions exists. I found a very nice reference:

http://www.statsoft.com/textbook/nonparametric-statistics

The paragraph below (which is from the reference above) is a key reference to what we're doing here:

...the need is evident for statistical procedures that enable us to process data of "low quality," from small samples, on variables about which nothing is known (concerning their distribution). Specifically, nonparametric methods were developed to be used in cases when the researcher knows nothing about the parameters of the variable of interest in the population (hence the name nonparametric). In more technical terms, nonparametric methods do not rely on the estimation of parameters (such as the mean or the standard deviation) describing the distribution of the variable of interest in the population. Therefore, these methods are also sometimes (and more appropriately) called parameter-free methods or distribution-free methods.

Being that I'm not a statistician but still need to determine if these sample sets are significant different, I let *Mathematica* determine the appropriate test. Notice that one of the above mentioned tests will probably be the test *Mathematica* chooses.

Note: If we run our normally distributed data through this analysis (speically, the "LocationEquivalenceTest"), *Mathematica* should detect this and use a more appropriate significant test, like a t-test.
Here we go with the hypothesis testing (assuming our sample sets are not normally distributed):

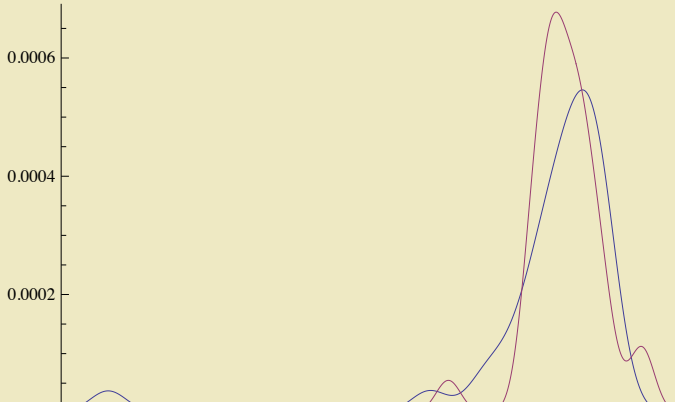
1. Our P value threshold is 0.05, which is our alpha.
2. The null hypotheses is the two populations have the same mean. (Remember we have to sample sets, which is not the

3. Do the statistical test to compute the P value.

4. Compare the result P value to our threshold alpha value. If the P value is less then our threshold, we will reject the null hypothesis and say the difference between our samples is significant. (Which is what I'm hoping to see.) However, if the P value is greater than the threshold, we cannot reject the null hypothesis and any difference between our samples are not statistically significant; randomness, picked the "wrong" samples, etc.

In[38]:=
```
Print["P-values assumming normality MAY not exist."];
Table[
  Table[
    If[i ≠ j,
      Print["================================================================"];
      Print[ssName[i], "(", Length[ssL[i]], ") and ", ssName[j], " (", Length[ssL[j]], ")"];

      test1 = MannWhitneyTest[{ssWork[i], ssWork[j]}];
      test2 = LocationEquivalenceTest[{ssWork[i], ssWork[j]}, {"TestDataTable", "AutomaticTest"}];
      Print["Work: Test1=", test1, " Test2=", test2];
      Print[SmoothHistogram[{ssWork[i], ssWork[j]}]];

      Print["------------"];
      test1 = MannWhitneyTest[{ssL[i], ssL[j]}];
      test2 = LocationEquivalenceTest[{ssL[i], ssL[j]}, {"TestDataTable", "AutomaticTest"}];
      Print["L: Test1=", test1, " Test2=", test2];
      Print[SmoothHistogram[{ssL[i], ssL[j]}]];

      Print["------------"];
      test1 = MannWhitneyTest[{ssSt[i], ssSt[j]}];
      test2 = LocationEquivalenceTest[{ssSt[i], ssSt[j]}, {"TestDataTable", "AutomaticTest"}];
      Print["St: Test1=", test1, " Test2=", test2];
      Print[SmoothHistogram[{ssSt[i], ssSt[j]}]];

      Print["------------"];
      test1 = MannWhitneyTest[{ssQt[i], ssQt[j]}];
      test2 = LocationEquivalenceTest[{ssQt[i], ssQt[j]}, {"TestDataTable", "AutomaticTest"}];
      Print["Qt: Test1=", test1, " Test2=", test2];
      Print[SmoothHistogram[{ssQt[i], ssQt[j]}]];

      Print["------------"];
      test1 = MannWhitneyTest[{ssRt[i], ssRt[j]}];
      test2 = LocationEquivalenceTest[{ssRt[i], ssRt[j]}, {"TestDataTable", "AutomaticTest"}];
      Print["St: Test1=", test1, " Test2=", test2];
      Print[SmoothHistogram[{ssRt[i], ssRt[j]}]];

     ];
    ,
    {j, 1, ssNum}
   ];
  , {i, 1, ssNum}
 ];
```

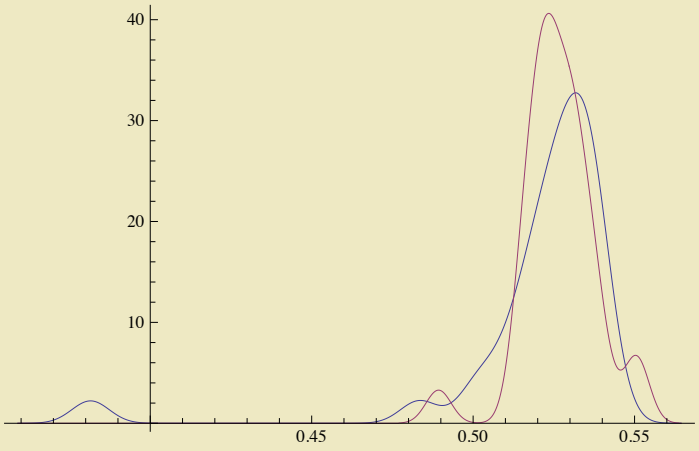
```
P-values assumming normality MAY not exist.

================================================================

Wait Immediate(31) and Wait Batch (31)

Work: Test1=0.949485 Test2={ | Statistic | P-Value |
                            | Kruskal-Wallis | 0.0031713 | 0.955645 | , KruskalWallis}
```



L: Test1=0.955091 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.00242803	0.961185

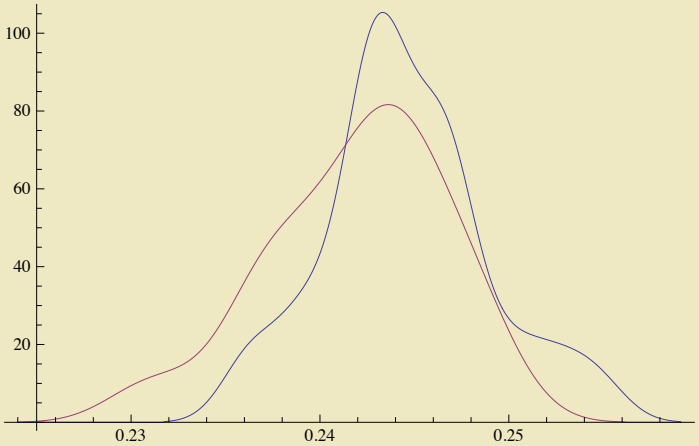
, KruskalWallis}



St: Test1=0.0693495 Test2={

	Statistic	P-Value
K-Sample T	5.19297	0.0262514

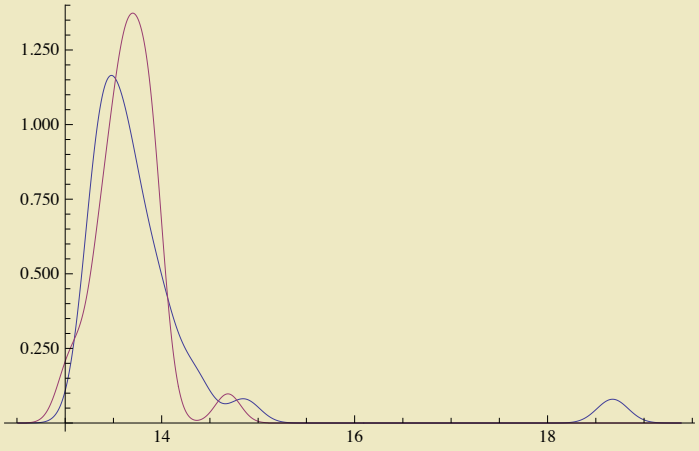
, KSampleT}



Qt: Test1=0.966311 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.00123879	0.97227

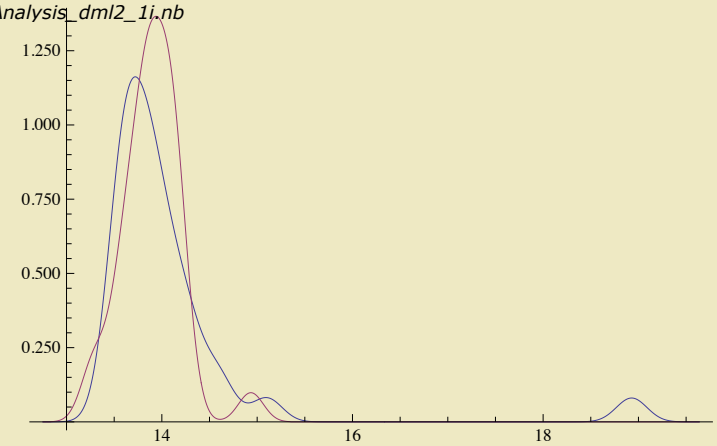
, KruskalWallis}



St: Test1=0.977537 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.000445964	0.98336

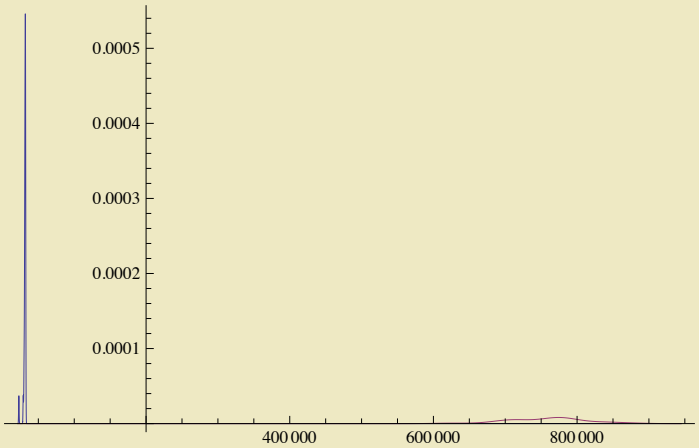
, KruskalWallis}



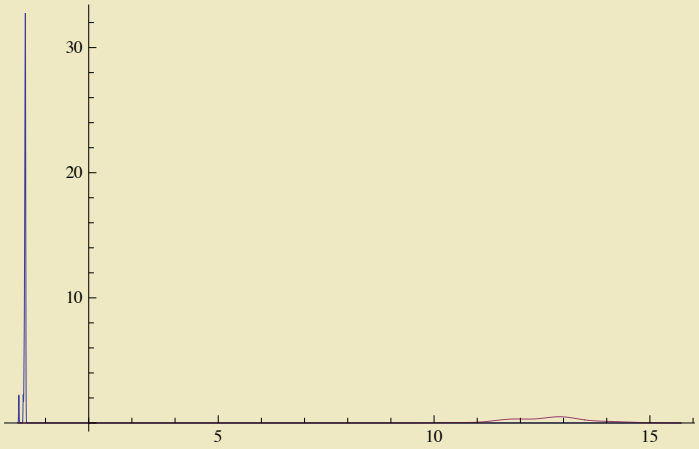
==========

Wait Immediate(31) and Nowait Immediate (31)

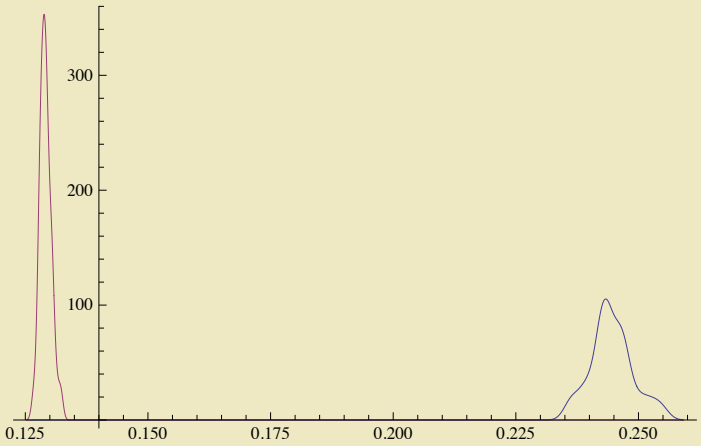
Work: Test1=1.27195×10^{-11} Test2=$\left\{\begin{array}{c|cc} & \text{Statistic} & \text{P-Value} \\ \hline \text{Kruskal-Wallis} & 45.7619 & 9.98137\times10^{-20} \end{array}, \text{KruskalWallis}\right\}$



L: Test1=1.27195×10^{-11} Test2=$\left\{\begin{array}{c|cc} & \text{Statistic} & \text{P-Value} \\ \hline \text{Kruskal-Wallis} & 45.7619 & 9.98137\times10^{-20} \end{array}, \text{KruskalWallis}\right\}$



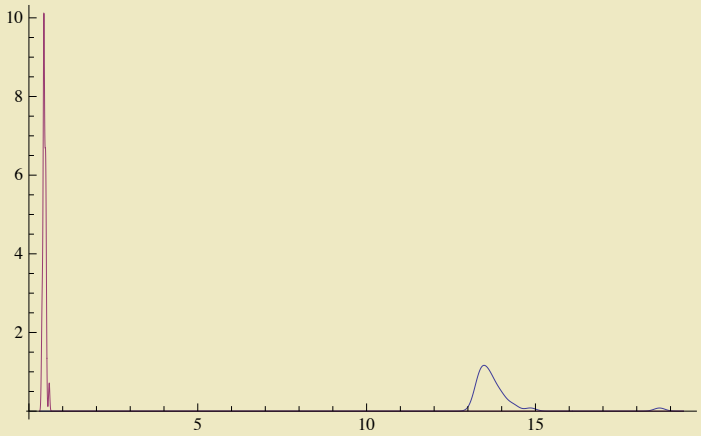
St: Test1=1.40185×10^{-11} Test2=$\left\{\begin{array}{c|cc} & \text{Statistic} & \text{P-Value} \\ \hline \text{K-Sample T} & 20489.8 & 9.37463\times10^{-78} \end{array}, \text{KSampleT}\right\}$



Qt: Test1=1.40185 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

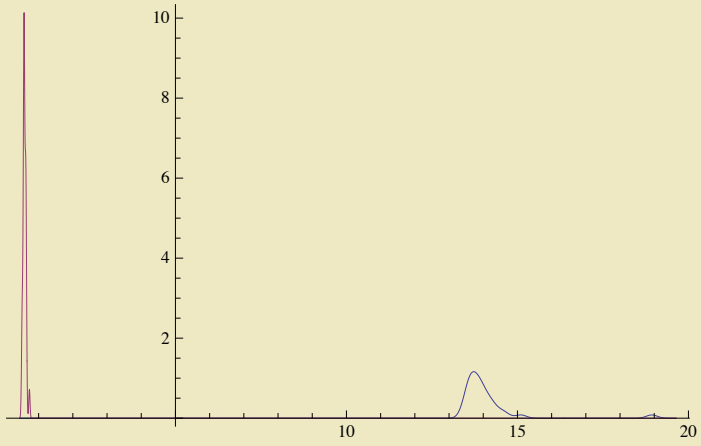
, KruskalWallis}



St: Test1=1.40185 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

, KruskalWallis}



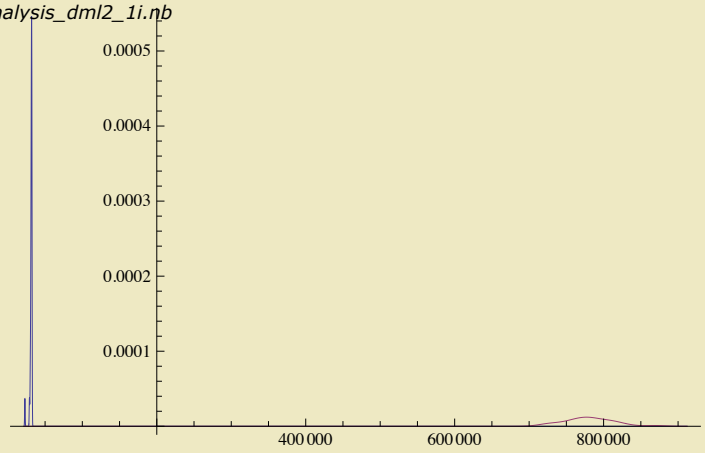
==========

Wait Immediate(31) and Nowait Batch (31)

Work: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

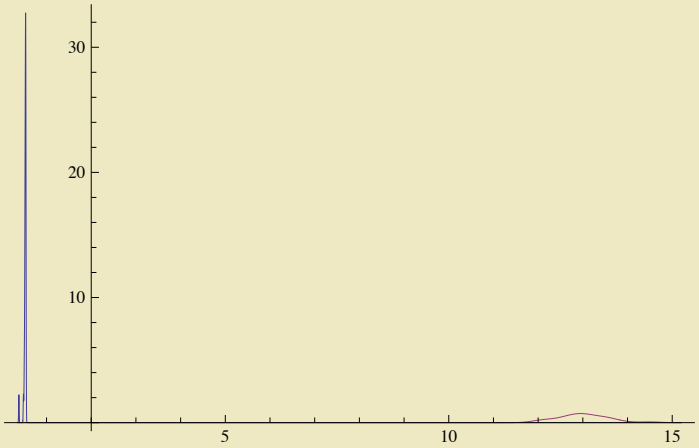
, KruskalWallis}



L: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137×10^{-20}

, KruskalWallis}



St: Test1=1.40185×10^{-11} Test2={

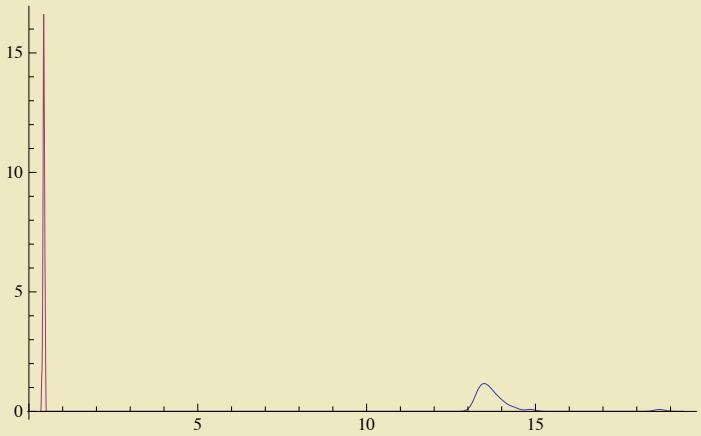
	Statistic	P-Value
K-Sample T	19 152.6	7.05724×10^{-77}

, KSampleT}

Qt: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137×10^{-20}

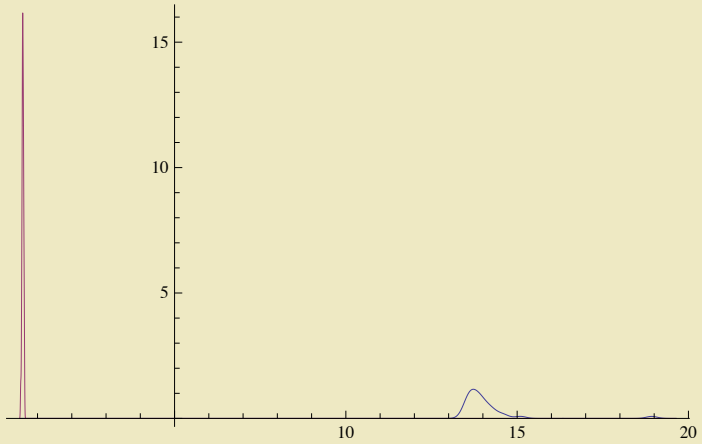
, KruskalWallis}



St: Test1=1.40185 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

, KruskalWallis}



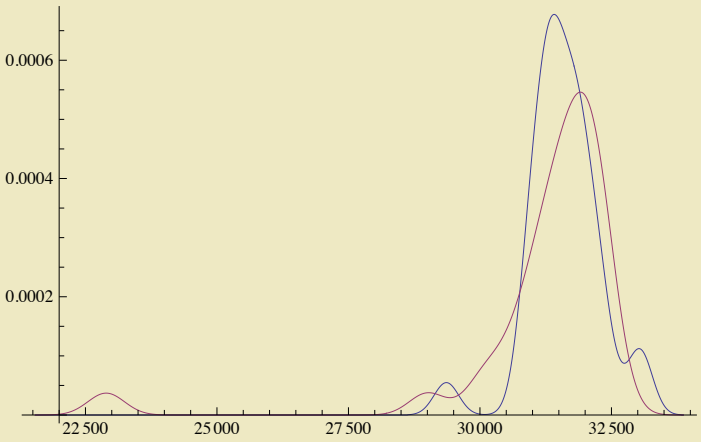
==========

Wait Batch(31) and Wait Immediate (31)

Work: Test1=0.9607 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.0031713	0.955645

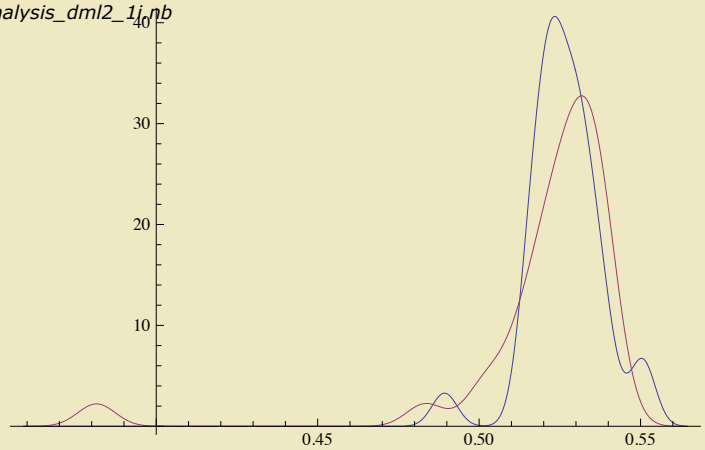
, KruskalWallis}



L: Test1=0.966311 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.00242803	0.961185

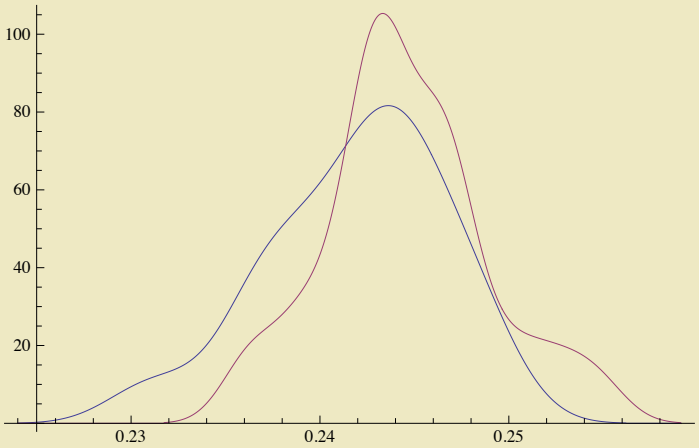
, KruskalWallis}



St: Test1=0.0672179 Test2={

	Statistic	P-Value
K-Sample T	5.19297	0.0262514

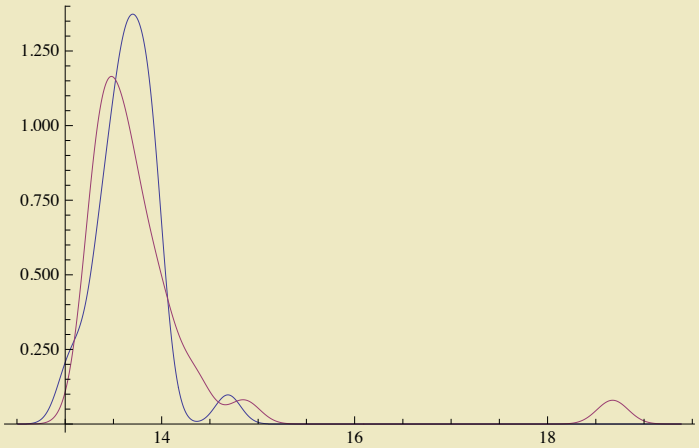
, KSampleT}



Qt: Test1=0.977537 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.00123879	0.97227

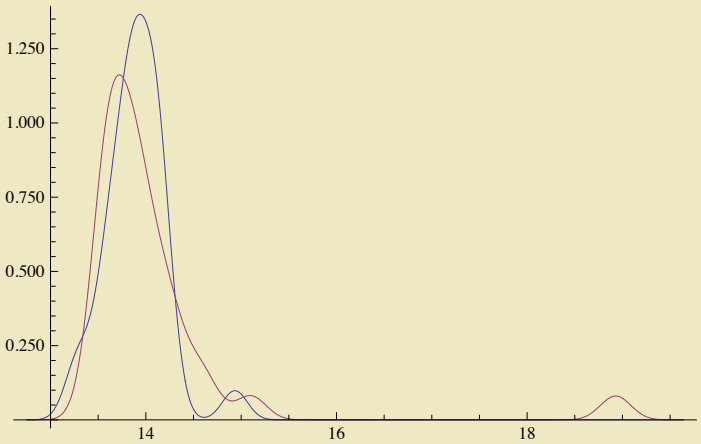
, KruskalWallis}



St: Test1=0.988767 Test2={

	Statistic	P-Value
Kruskal-Wallis	0.000445964	0.98336

, KruskalWallis}



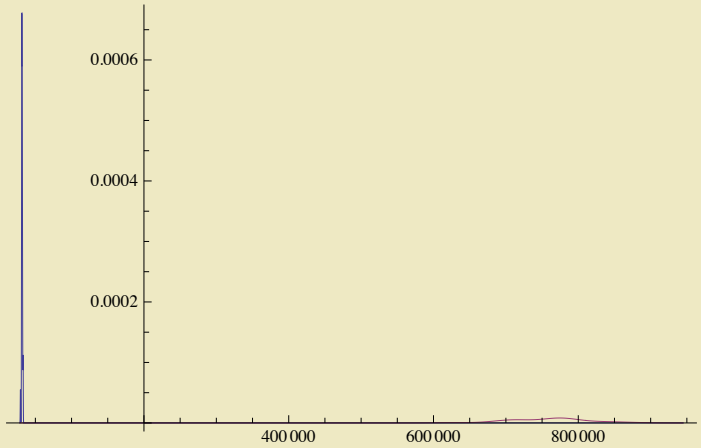
==========

Wait Batch(31) and Nowait Immediate (31)

Work: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	5774.14	2.39034×10^{-61}

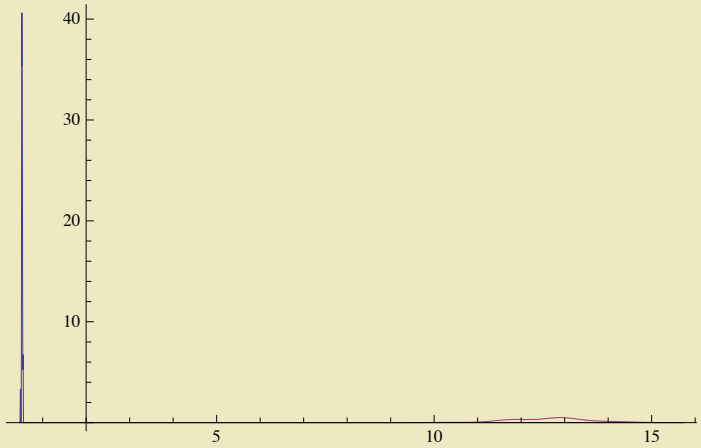
, KSampleT}



L: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	5779.38	2.32676×10^{-61}

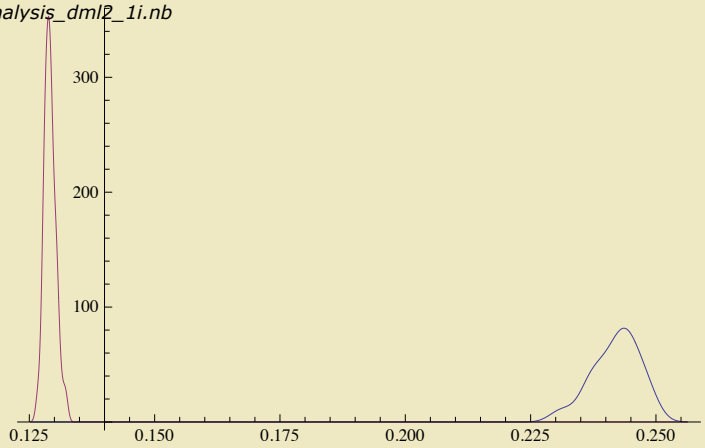
, KSampleT}



St: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	16944.7	2.74971×10^{-75}

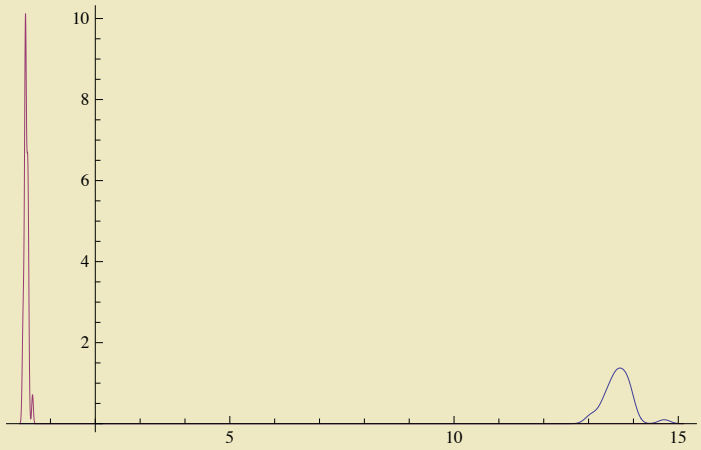
, KSampleT}



Qt: Test1=1.40185×10^{-11} Test2=

	Statistic	P-Value
K-Sample T	53963.3	2.38991×10^{-90}

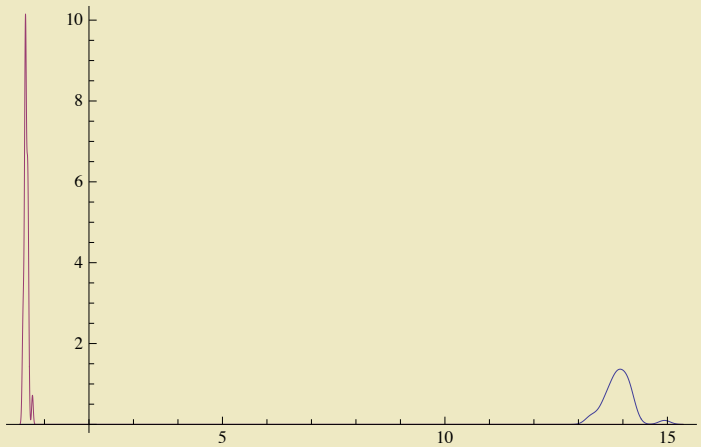
, KSampleT



St: Test1=1.40185×10^{-11} Test2=

	Statistic	P-Value
K-Sample T	53952.2	2.40472×10^{-90}

, KSampleT



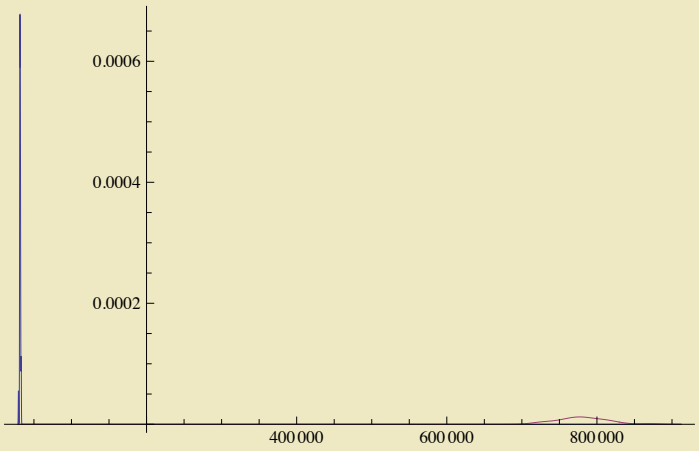
==========

Wait Batch(31) and Nowait Batch (31)

Work: Test1=1.27195×10^{-11} Test2=

	Statistic	P-Value
K-Sample T	15837.3	2.07347×10^{-74}

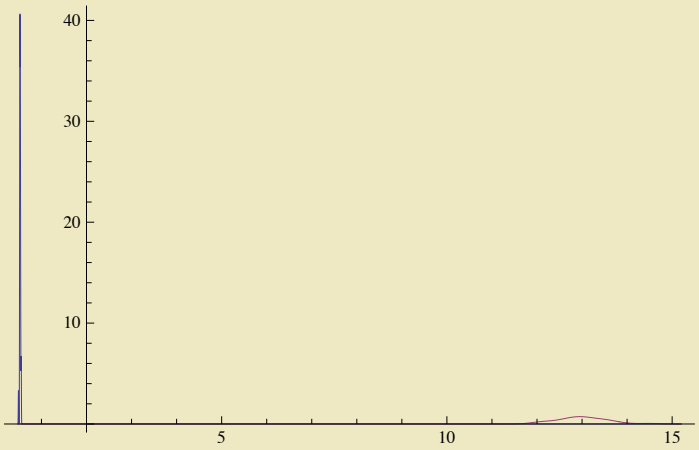
, KSampleT



L: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	15 778.5	2.31728×10^{-74}

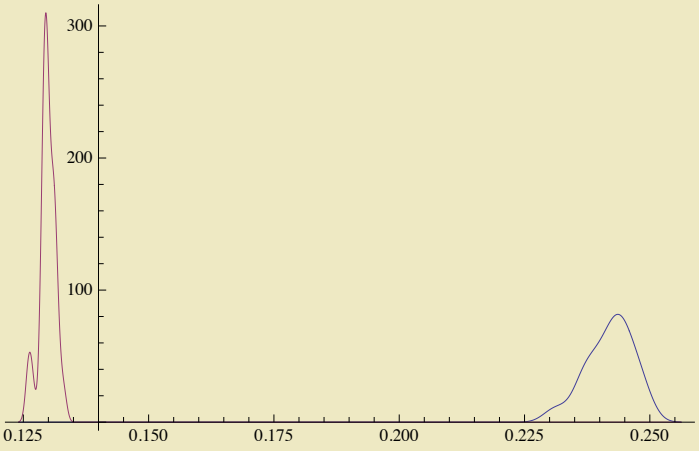
, KSampleT}



St: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	15 950.7	1.67521×10^{-74}

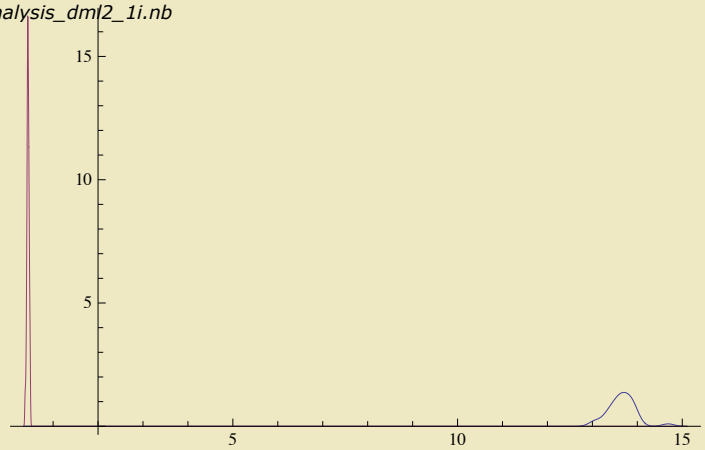
, KSampleT}



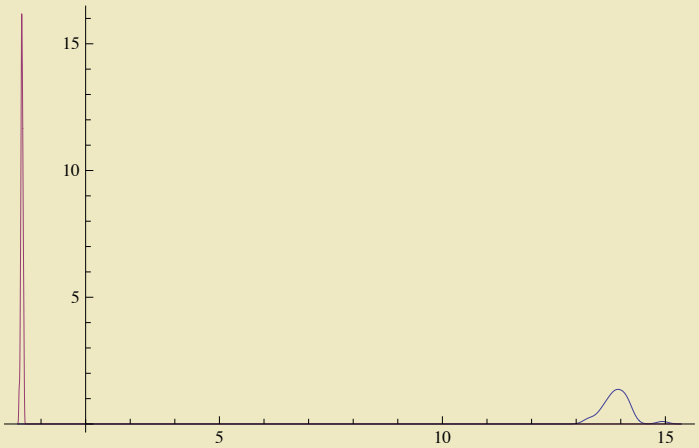
Qt: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	54 869.5	1.45095×10^{-90}

, KSampleT}



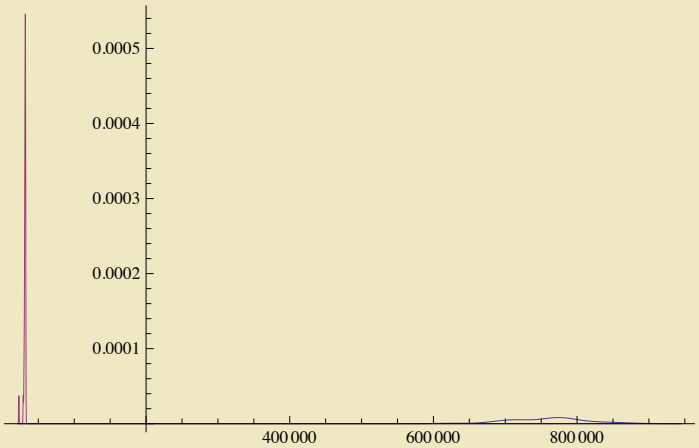
St: Test1=1.40185 × 10^-11 Test2={ | Statistic | P-Value | / K-Sample T | 54855.5 | 1.46206 × 10^-90 , KSampleT}



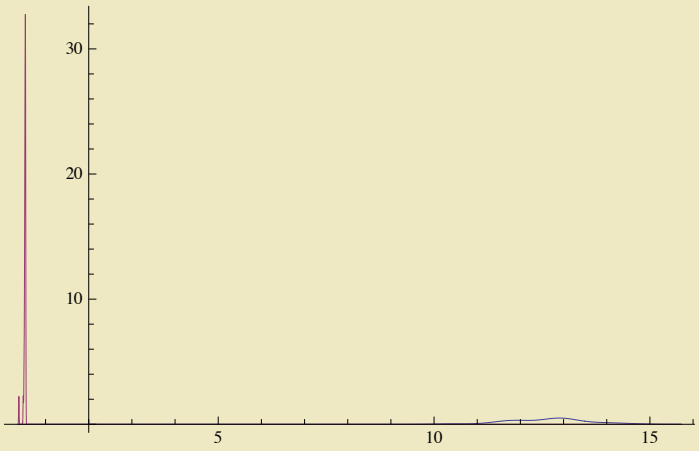
==========

Nowait Immediate(31) and Wait Immediate (31)

Work: Test1=1.40185 × 10^-11 Test2={ | Statistic | P-Value | / Kruskal-Wallis | 45.7619 | 9.98137 × 10^-20 , KruskalWallis}



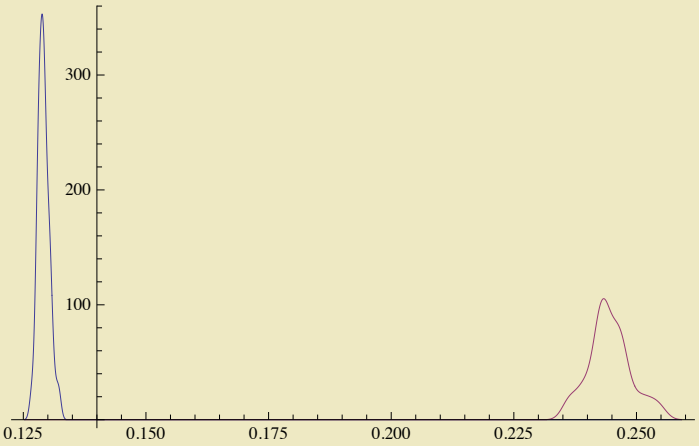
L: Test1=1.40185 × 10^-11 Test2={ | Statistic | P-Value | / Kruskal-Wallis | 45.7619 | 9.98137 × 10^-20 , KruskalWallis}



St: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
K-Sample T	20489.8	9.37463 × 10^-78

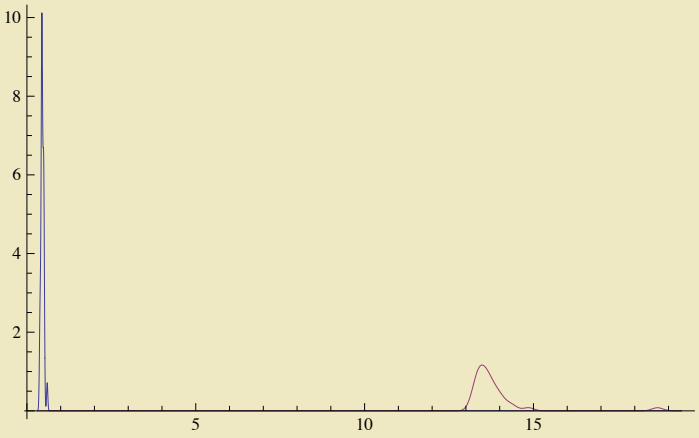
, KSampleT}



Qt: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

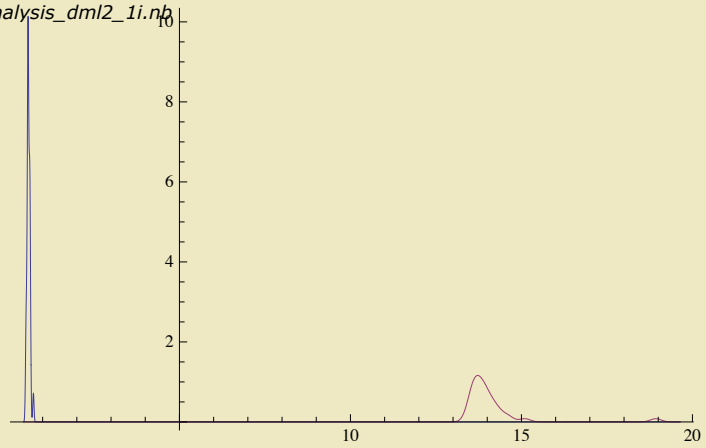
, KruskalWallis}



St: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

, KruskalWallis}



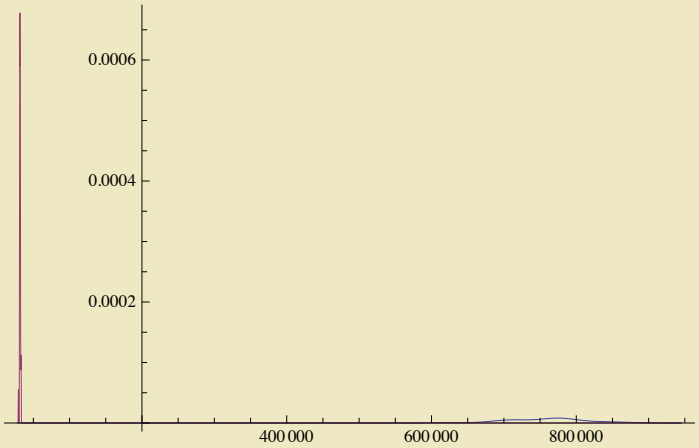
==========

Nowait Immediate(31) and Wait Batch (31)

Work: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	5774.14	2.39034×10^{-61}

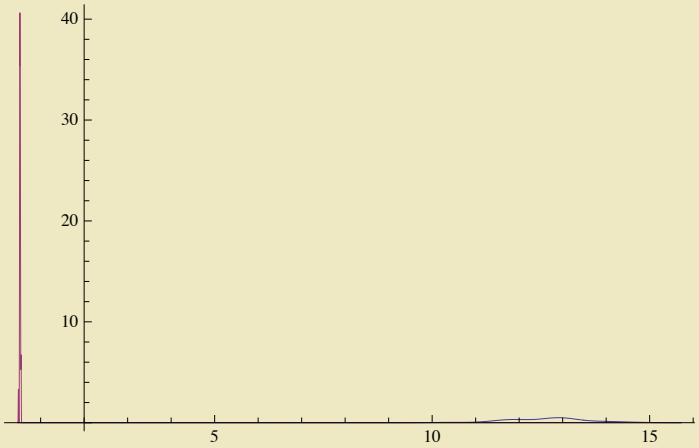
, KSampleT}



L: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	5779.38	2.32676×10^{-61}

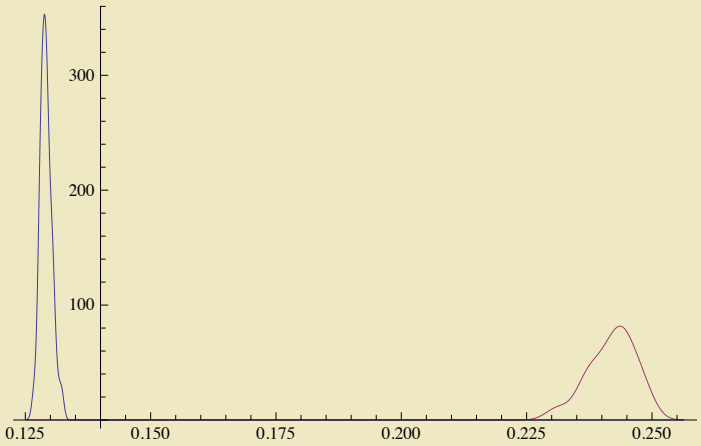
, KSampleT}



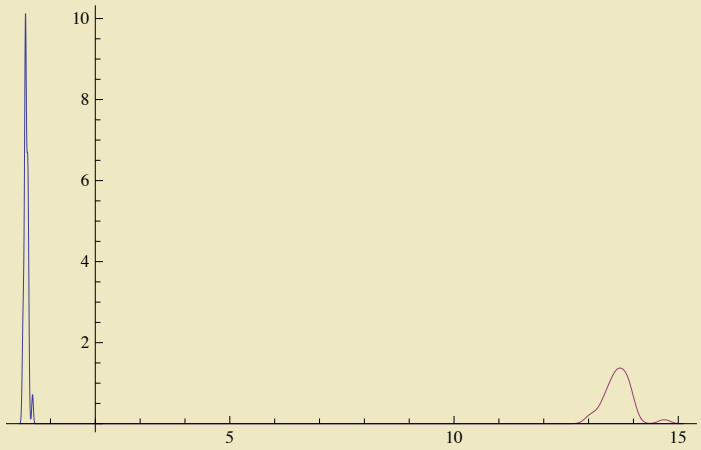
St: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	16 944.7	2.74971×10^{-75}

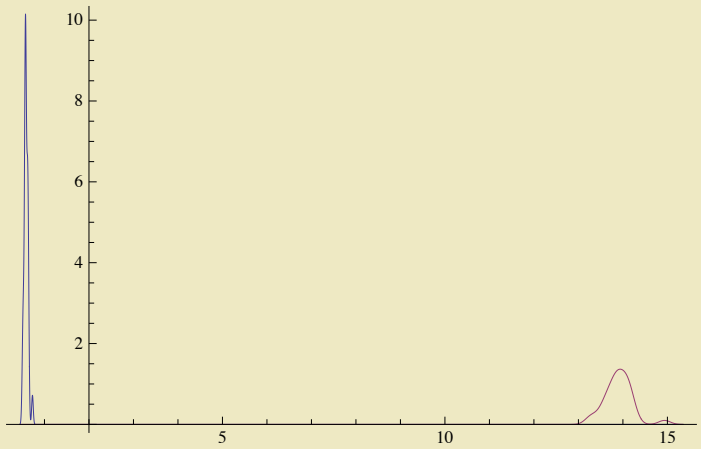
, KSampleT}



Qt: Test1=1.27195 × 10^-11 Test2={ | Statistic | P-Value |
| K-Sample T | 53963.3 | 2.38991 × 10^-90 | , KSampleT}



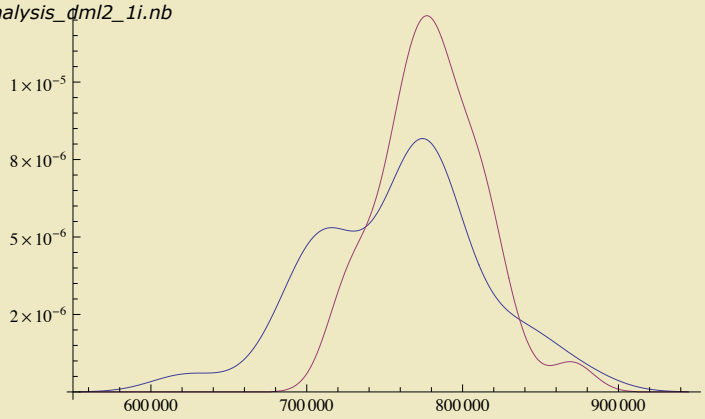
St: Test1=1.27195 × 10^-11 Test2={ | Statistic | P-Value |
| K-Sample T | 53952.2 | 2.40472 × 10^-90 | , KSampleT}



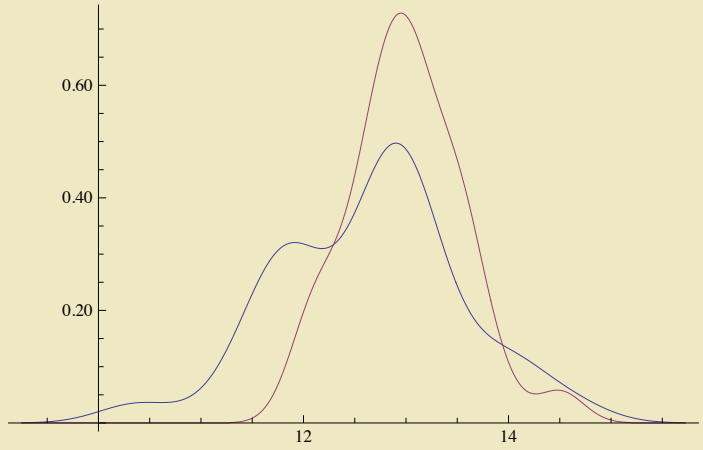
==========

Nowait Immediate(31) and Nowait Batch (31)

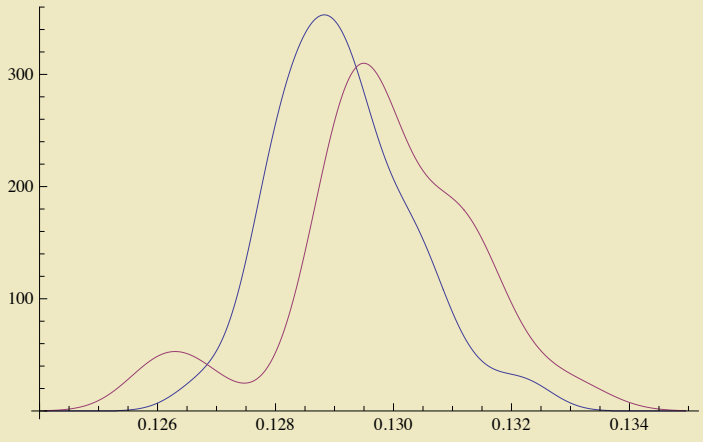
Work: Test1=0.0573535 Test2={ | Statistic | P-Value |
| K-Sample T | 3.78033 | 0.056551 | , KSampleT}



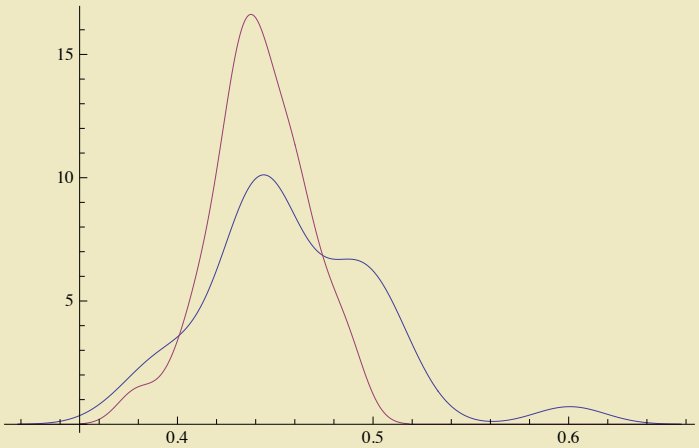
L: Test1=0.0611443 Test2={ | Statistic | P-Value | K-Sample T | 3.79077 | 0.0562214 | , KSampleT}



St: Test1=0.0132184 Test2={ | Statistic | P-Value | K-Sample T | 3.75366 | 0.0574033 | , KSampleT}



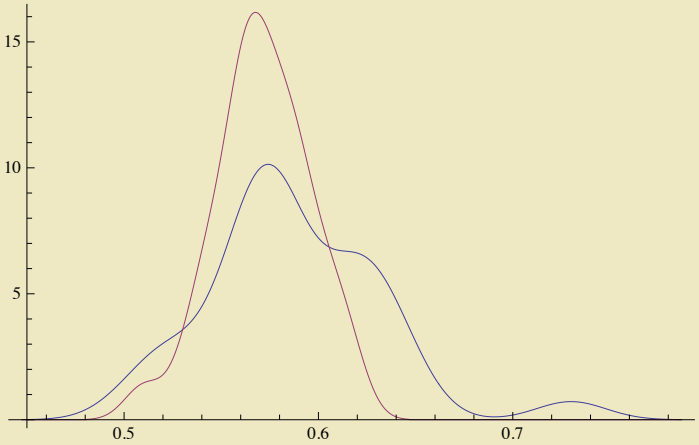
Qt: Test1=0.0884732 Test2={ | Statistic | P-Value | K-Sample T | 3.47956 | 0.0670227 | , KSampleT}



St: Test1=0.0995184 Test2=

	Statistic	P-Value
K-Sample T	3.21245	0.0781217

, KSampleT



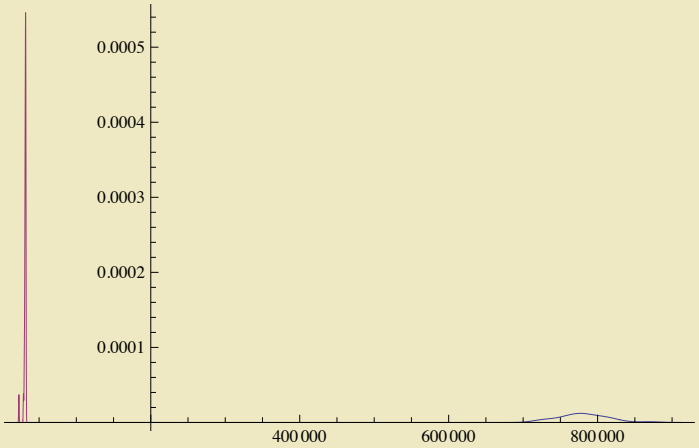
==========

Nowait Batch(31) and Wait Immediate (31)

Work: Test1=1.40185×10^{-11} Test2=

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137×10^{-20}

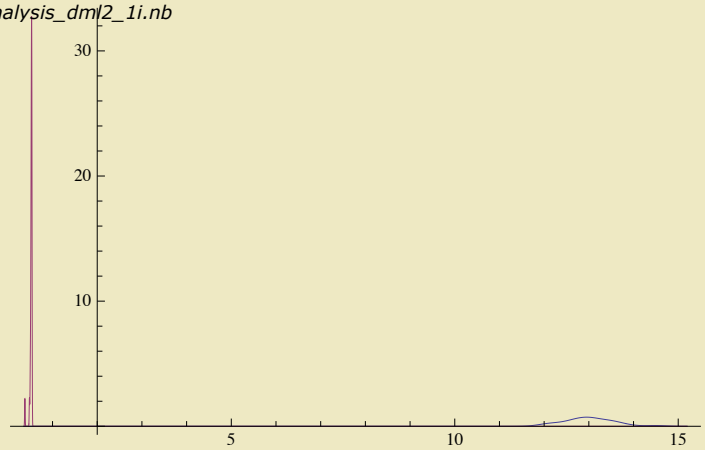
, KruskalWallis



L: Test1=1.40185×10^{-11} Test2=

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137×10^{-20}

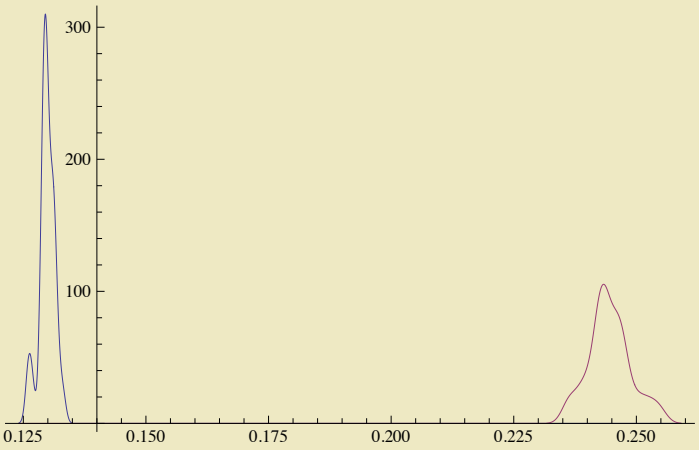
, KruskalWallis



St: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
K-Sample T	19 152.6	7.05724 × 10^-77

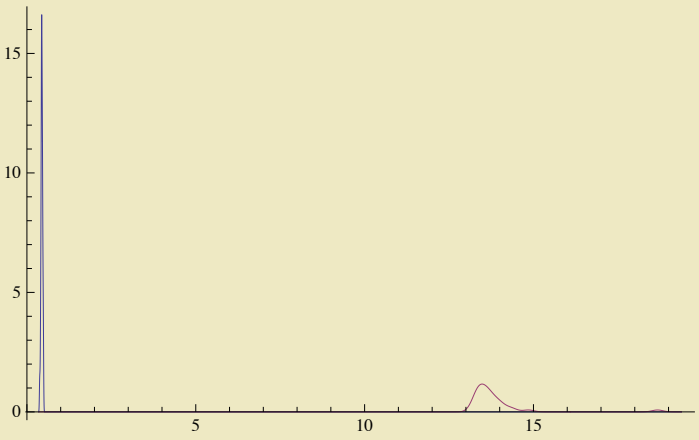
, KSampleT}



Qt: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

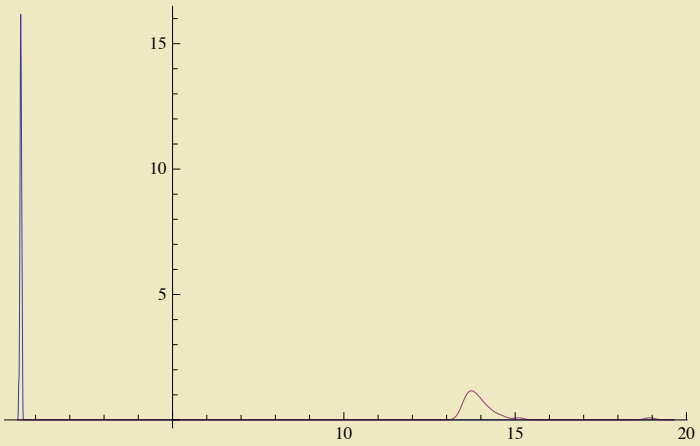
, KruskalWallis}



St: Test1=1.27195 × 10^-11 Test2={

	Statistic	P-Value
Kruskal-Wallis	45.7619	9.98137 × 10^-20

, KruskalWallis}



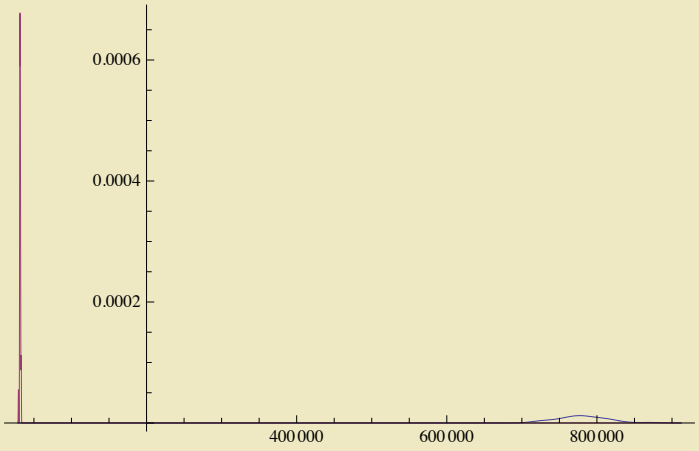
==========

Nowait Batch(31) and Wait Batch (31)

Work: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	15837.3	2.07347×10^{-74}

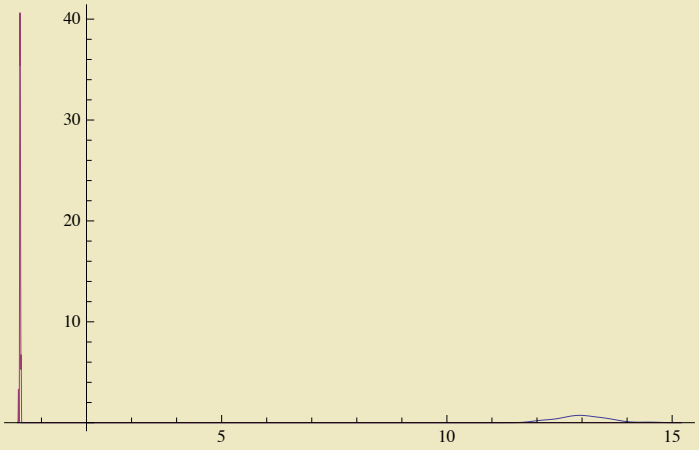
, KSampleT}



L: Test1=1.40185×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	15778.5	2.31728×10^{-74}

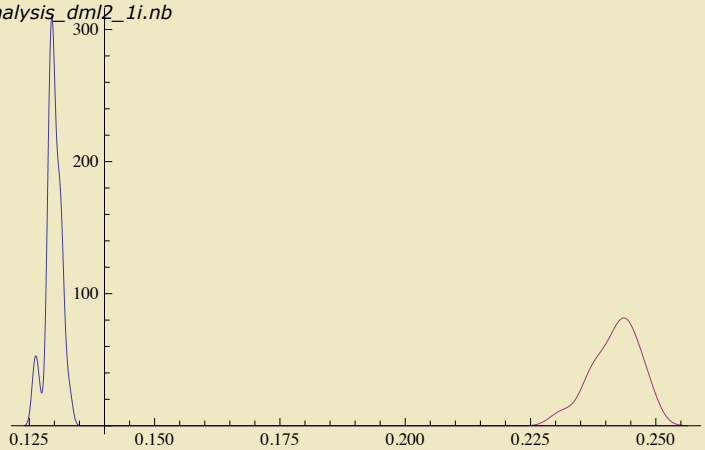
, KSampleT}



St: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	15950.7	1.67521×10^{-74}

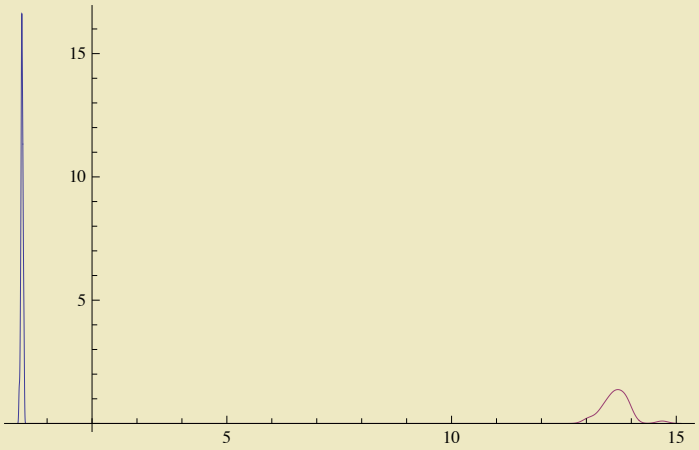
, KSampleT}



Qt: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	54869.5	1.45095×10^{-90}

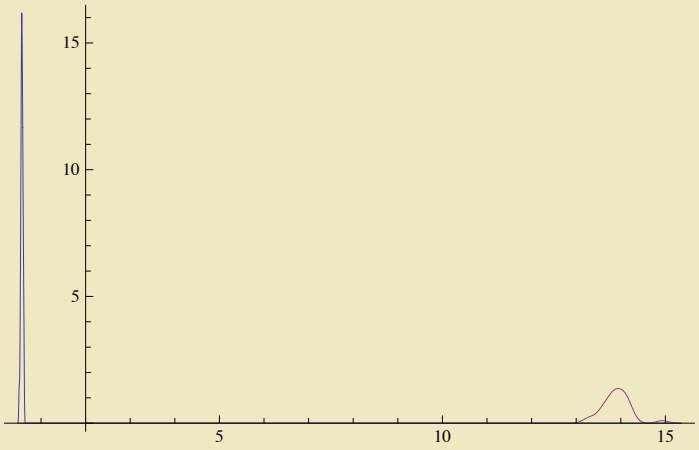
, KSampleT}



St: Test1=1.27195×10^{-11} Test2={

	Statistic	P-Value
K-Sample T	54855.5	1.46206×10^{-90}

, KSampleT}



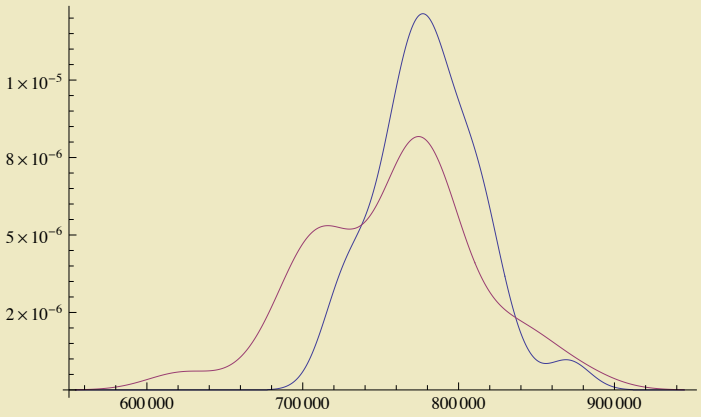
==========

Nowait Batch(31) and Nowait Immediate (31)

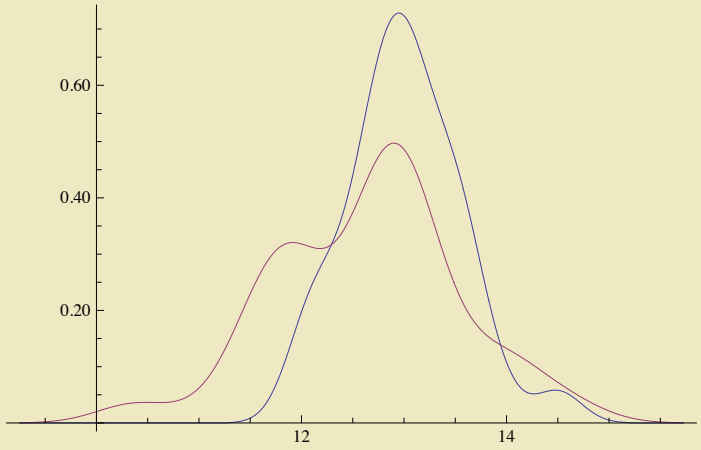
Work: Test1=0.0592237 Test2={

	Statistic	P-Value
K-Sample T	3.78033	0.056551

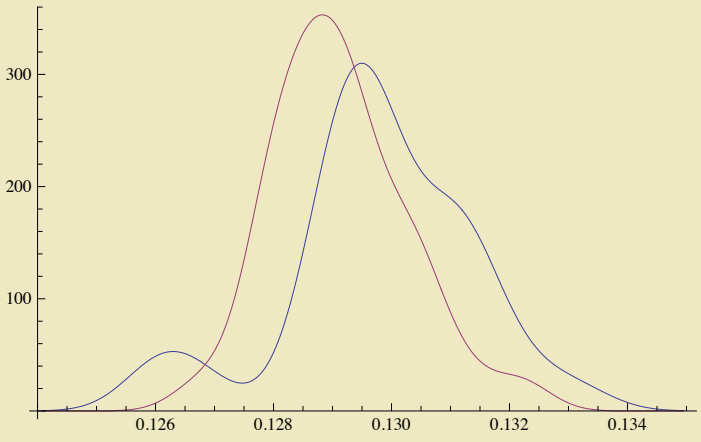
, KSampleT}



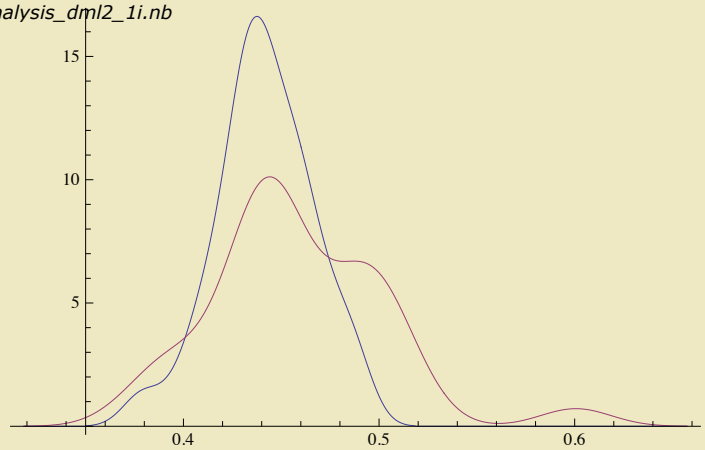
L: Test1=0.0631162 Test2={ | Statistic | P-Value |
| K-Sample T | 3.79077 | 0.0562214 | , KSampleT}



St: Test1=0.0137492 Test2={ | Statistic | P-Value |
| K-Sample T | 3.75366 | 0.0574033 | , KSampleT}



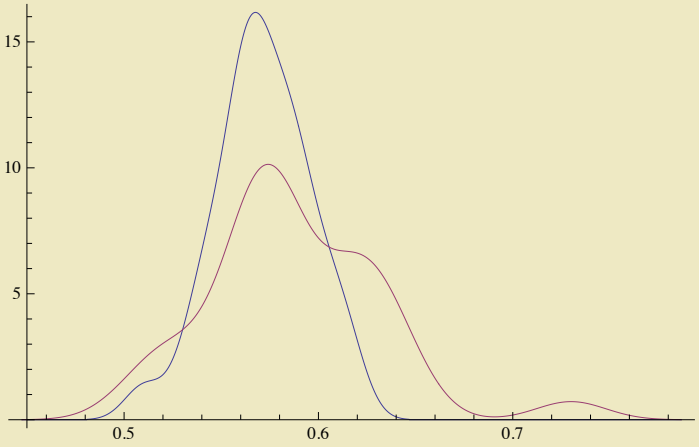
Qt: Test1=0.0858722 Test2={ | Statistic | P-Value |
| K-Sample T | 3.47956 | 0.0670227 | , KSampleT}



St: Test1=0.0966589 Test2={

	Statistic	P-Value
K-Sample T	3.21245	0.0781217

, KSampleT}



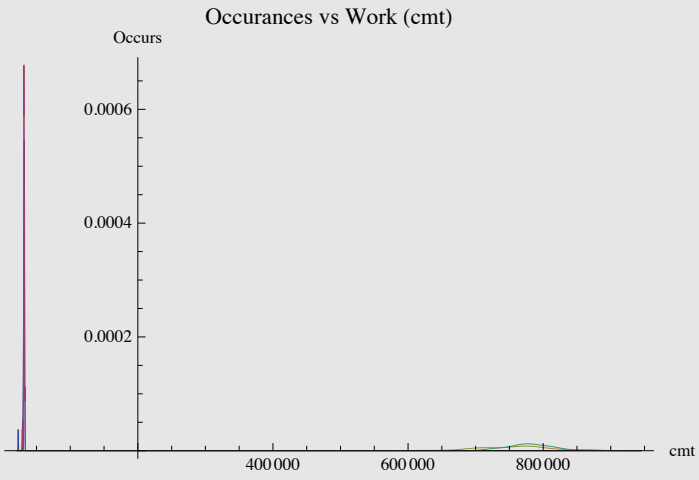
Visually Comparing All Samples

I also wanted to get a nice visual picture of my sample sets...together. Sometimes I include all the sample sets and sometimes I don't. It's just based on what I want to convey. Sometimes you get a more appropriate view if all the data is not included.

Here is the colors in order of sample set; blue, red, yellow, green.

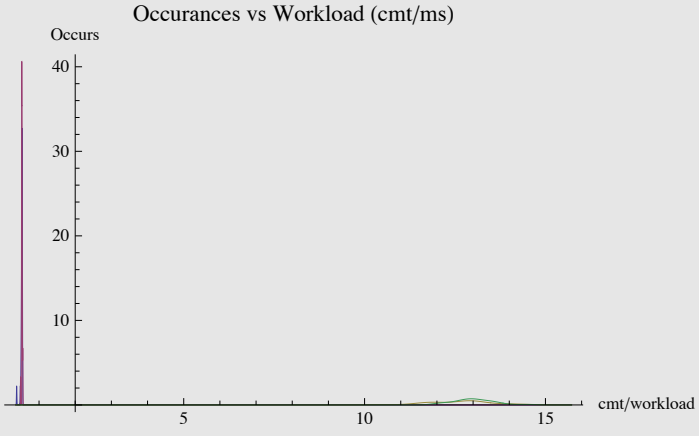
In[40]:=
```
gset = {};
Table[
  AppendTo[gset, ssWork[i]];
  , {i, 1, ssNum}
 ];
SmoothHistogram[gset,
 PlotLabel → "Occurances vs Work (cmt)", AxesLabel → {"cmt", "Occurs"}]
```

Out[42]=



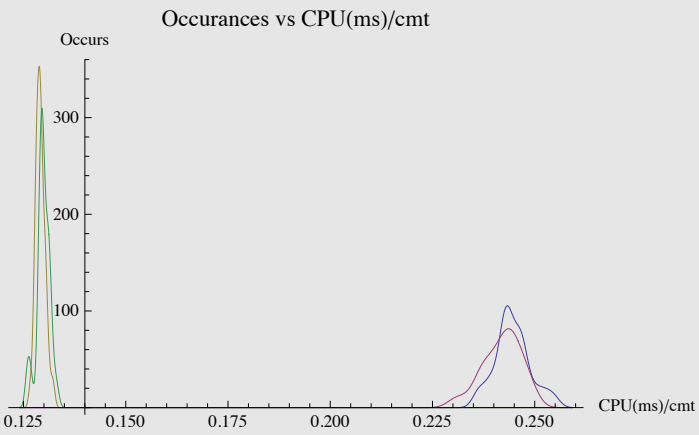
In[43]:=
```
gset = {};
Table[
  AppendTo[gset, ssL[i]];
  , {i, 1, ssNum}
 ];
SmoothHistogram[gset,
 PlotLabel → "Occurances vs Workload (cmt/ms)", AxesLabel → {"cmt/workload", "Occurs"}]
```

Out[45]=



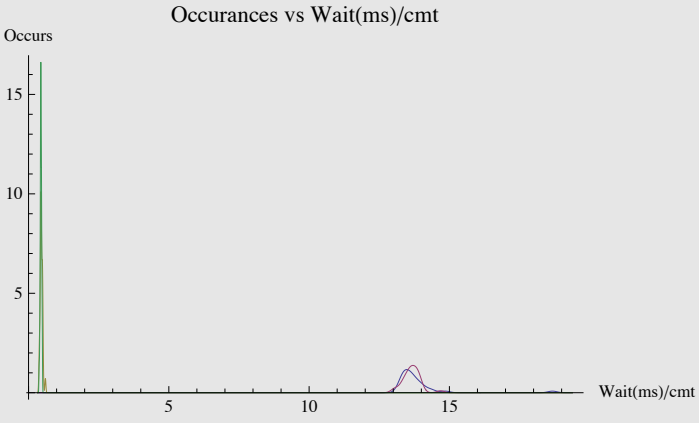
In[46]:=
```
gset = {};
Table[
  AppendTo[gset, ssSt[i]];
  , {i, 1, ssNum}
 ];
SmoothHistogram[gset,
 PlotLabel → "Occurances vs CPU(ms)/cmt", AxesLabel → {"CPU(ms)/cmt", "Occurs"}]
```

Out[48]=



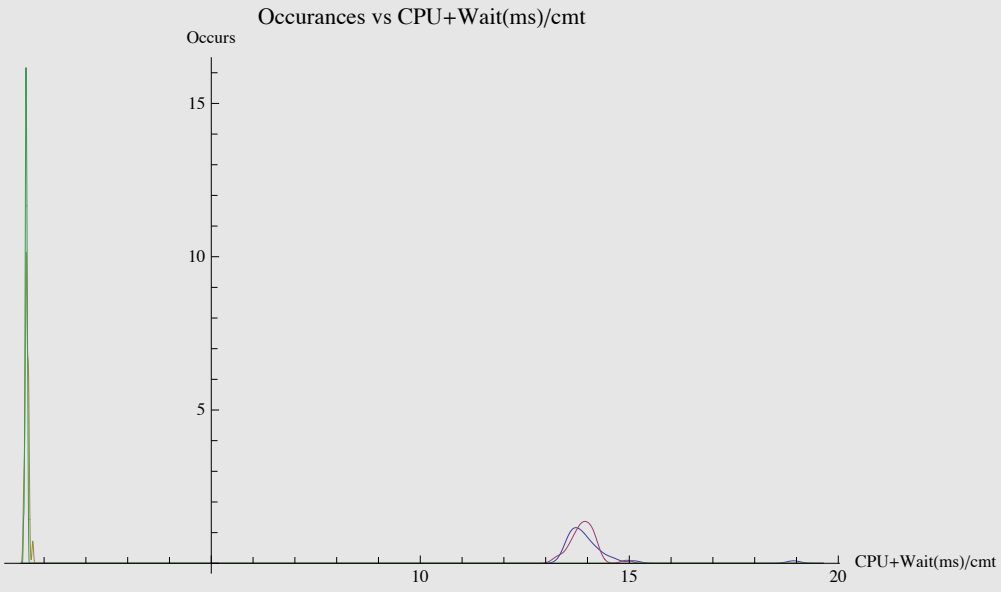
In[49]:=
```
gset = {};
Table[
  AppendTo[gset, ssQt[i]];
  , {i, 1, ssNum}
 ];
SmoothHistogram[gset,
 PlotLabel → "Occurances vs Wait(ms)/cmt", AxesLabel → {"Wait(ms)/cmt", "Occurs"}]
```

Out[51]=



In[52]:=
```
gset = {};
Table[
  AppendTo[gset, ssRt[i]];
  , {i, 1, ssNum}
 ];
SmoothHistogram[gset,
 PlotLabel → "Occurances vs CPU+Wait(ms)/cmt", AxesLabel → {"CPU+Wait(ms)/cmt", "Occurs"}]
```

Out[54]=



Fair: Comparing situations fairly

As workload increases, we can expect response time to also increase. If the workload is decreases response time *may* decrease. When comparing two sample sets, if their arrival rates are different, comparing their queue times and response times using statitcal significant tests is problematic and downright unfiar.

To get around this problem, we need to do the testing at the same arrival rate. However, our sample data may not have been collected at the same arrival rate! That's a problem.

To get this problem, we need to develop an equation for each of our four sample sets relating the arrival rate to the response time. Fortunately, there already exists a formal mathematical equation related the arrival rate, service time, queue time, the number of transaction processors (think: CPU cores), and the response time.

For a CPU constained system, the equation is r = s / (1-(sL/m)^m)

For an IO constrained system, the equation is r = s / (1-(sL/m))

There are many ways to go about this process. Here is who I'm choosing to do it. Within in each of the four sample sets (ssNum), for each sample value (sampleIdx), I derive the missing variable M and store these in mList. Then I take the average M and save that in bestMList. I could have weighted the average or taken the median, but I just kept it simple. You see the details in the code segment below.

Note that this can take awhile to run. The "Solve" is cpu intensive.

In[55]:=
```
Clear[s, q, r, l, sol, bestM, mList, bestMList];
bestMList = {};
Table[
  mList = {};
  Table[
   s = ssSt[ssidx][[sampleIdx]];
   q = ssQt[ssidx][[sampleIdx]];
   r = s + q;
   l = ssL[ssidx][[sampleIdx]];
   (* Using the IO model, not CPU *)
   sol = Solve[s / (1 - (s * l / m)) == r && m > 0, m];
   {mm} = m /. sol;
   (*Print[ssidx," ",sampleIdx," mm=",mm];*)
   AppendTo[mList, mm];
   , {sampleIdx, 1, sampleNum}
  ];
  bestM = Mean[mList];
  (*Print[ssidx," bestM=",bestM];*)
  AppendTo[bestMList, bestM];
  , {ssidx, 1, ssNum}
 ];
```

In[58]:=
```
Print["The best M values are:"];
Table[
  Print[ssName[ssidx], " M=", bestMList[[ssidx]]];
  , {ssidx, 1, ssNum}
 ];
```

```
The best M values are:

Wait Immediate M=0.129503

Wait Batch M=0.129606

Nowait Immediate M=2.09822

Nowait Batch M=2.1874
```

The baseline arrival rate will be the average from our first sample set, (nowait, immediate). I expect the other three options to provide better performance and the nowwait,immediate is the Oracle default, hence this is my chosen baseline.

In[60]:=
```
baselineL = Mean[ssL[1]]
```

Out[60]=
```
0.52056
```

Now that we have a good M for each of our four sample sets (bestMList) along with the standard arrival rate (baselineL), and the observed service time for each individual sample, we will derive the queue time and the response time for each indivisual sample (within each of our four sample sets). All the inputs and derived queue time and response time are stored in the standardized lists, stndL, stndM, stndSt, stndQt, and stndRt. At this point, we essentially have an entirely new (i.e., standardized) set of experimental data. This allows us to run this data through the same statistical analysis as I did above!

In[61]:=
```
Clear[stndRt, stndSt, stndL, stndQt, stndM];
Clear[s, l, m, r, q];
Table[
  stndRt[ssidx] = {};
  stndQt[ssidx] = {};
  stndSt[ssidx] = {};
  stndL[ssidx] = {};
  stndM[ssidx] = {};
  Table[
   s = ssSt[ssidx][[sampleIdx]];
   l = baselineL;
   m = bestMList[[ssidx]];
   r = s / (1 - (s l / m)^m);
   q = r - s;
   (*Print[ssidx," ",sampleIdx," s=",s," l=",l," m=",m," r=",r];*)
   AppendTo[stndRt[ssidx], r];
   AppendTo[stndSt[ssidx], s];
   AppendTo[stndL[ssidx], l];
   AppendTo[stndQt[ssidx], q];
   AppendTo[stndM[ssidx], m];
   , {sampleIdx, 1, sampleNum}
  ];
  , {ssidx, 1, ssNum}
 ];
stndM[1]
stndL[1]
stndSt[1]
stndQt[1]
stndRt[1]
```

Out[64]=
```
{0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503,
 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503,
 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503,
 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503, 0.129503}
```

Out[65]=
```
{0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056,
 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056,
 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056, 0.52056}
```

Out[66]=
```
{0.250246, 0.244308, 0.253145, 0.241951, 0.24619, 0.242333, 0.245746,
 0.248682, 0.242903, 0.239298, 0.246605, 0.244029, 0.243409, 0.246117, 0.242331,
 0.236494, 0.238954, 0.246825, 0.254831, 0.239668, 0.242723, 0.247293, 0.243704,
 0.247008, 0.243661, 0.242101, 0.242503, 0.24678, 0.251536, 0.236225, 0.24375}
```

Out[67]=
```
{-328.035, 103.991, -112.388, 67.0415, 181.829, 71.2004, 154.76, 5081.39, 78.3949, 47.456, 217.168,
 97.6839, 86.0713, 176.784, 71.173, 35.9505, 45.6871, 241.992, -81.9441, 49.5021, 75.973,
 319.265, 91.2385, 267.41, 90.4429, 68.6205, 73.2042, 236.473, -176.123, 35.1185, 92.1011}
```

Out[68]=
```
{-327.785, 104.235, -112.135, 67.2834, 182.075, 71.4427, 155.006, 5081.64, 78.6378, 47.6953,
 217.414, 97.928, 86.3148, 177.03, 71.4154, 36.1869, 45.926, 242.239, -81.6892, 49.7418, 76.2158,
 319.512, 91.4823, 267.657, 90.6865, 68.8626, 73.4467, 236.72, -175.871, 35.3547, 92.3448}
```

Fair: Basic Statistics

Using our standardized data, in this section I calculate the basic statistics, such as the mean and median. My objective is to ensure the data has been collected and entered correctly and also to compare the two datasets to see if they appear to be different.

In[69]:=
```
myData = Table[
   {
    ssName[ssidx], Mean[stndL[ssidx]], Mean[stndSt[ssidx]],
    Mean[stndQt[ssidx]], Mean[stndRt[ssidx]], Length[stndL[ssidx]],
    N[StandardDeviation[stndL[ssidx]]], N[StandardDeviation[stndSt[ssidx]]],
    N[StandardDeviation[stndQt[ssidx]]], N[StandardDeviation[stndRt[ssidx]]]
   }, {ssidx, 1, ssNum}
  ];
toGrid = Prepend[myData, {"Settings", "Avg L\n(cmt/ms)", "Avg CPUt\n(ms/cmt)",
    "Avg Wt\n(ms/cmt)", "Avg Rt\n(ms/cmt)", "Samples", "Stdev L\n(cmt/ms)",
    "Stdev CPUt\n(ms/cmt)", "Stdev Wt\n(ms/cmt)", "Stdev Rt\n(ms/cmt)"}];
Grid[
 toGrid,
 Frame → 
  All]
```

Out[71]=

Settings	Avg L (cmt/ms)	Avg CPUt (ms/cmt)	Avg Wt (ms/cmt)	Avg Rt (ms/cmt)	Samples	Stdev L (cmt/ms)	Stdev CPUt (ms/cmt)	Stdev Wt (ms/cmt)	Stdev Rt (ms/cmt)
Wait Immediate	0.52056	0.24456	240.627	240.871	31	2.25715×10^{-16}	0.00434349	907.48	907.48
Wait Batch	0.52056	0.241944	116.912	117.154	31	2.25715×10^{-16}	0.00468948	274.83	274.831
Nowait Immediate	0.52056	0.129128	0.000094614	0.129223	31	2.25715×10^{-16}	0.00113723	2.59613×10^{-6}	0.00113982
Nowait Batch	0.52056	0.129801	0.0000646176	0.129866	31	2.25715×10^{-16}	0.00156522	2.46726×10^{-6}	0.00156769

Using our standardized data, in this section I calculate the key parameters for understanding the impact or change of the settings.

In[72]:=
```
myData = Table[
   {
    ssName[ssidx],
    Mean[stndL[ssidx]], 100 * (Mean[stndL[ssidx]] - Mean[stndL[1]]) / Mean[stndL[1]],
    Mean[stndSt[ssidx]], 100 * (Mean[stndSt[ssidx]] - Mean[stndSt[1]]) / Mean[stndSt[1]],
    Mean[stndQt[ssidx]], 100 * (Mean[stndQt[ssidx]] - Mean[stndQt[1]]) / Mean[stndQt[1]],
    Mean[stndRt[ssidx]], 100 * (Mean[stndRt[ssidx]] - Mean[stndRt[1]]) / Mean[stndRt[1]],
    Length[stndL[ssidx]]
   }, {ssidx, 1, ssNum}
  ];
toGrid = Prepend[myData, {"Settings", "Avg L\n(cmt/ms)", "%\nChange", "Avg CPUt\n(ms/cmt)",
    "%\nChange", "Avg Wt\n(ms/cmt)", "%\nChange", "Avg Rt\n(ms/cmt)", "%\nChange", "Samples"}];
Grid[
 toGrid,
 Frame → 
  All]
```

Out[74]=

Settings	Avg L (cmt/ms)	% Change	Avg CPUt (ms/cmt)	% Change	Avg Wt (ms/cmt)	% Change	Avg Rt (ms/cmt)	% Change	Samples
Wait Immediate	0.52056	0.	0.24456	0.	240.627	0.	240.871	0.	31
Wait Batch	0.52056	0.	0.241944	-1.06973	116.912	-51.4135	117.154	-51.3624	31
Nowait Immediate	0.52056	0.	0.129128	-47.1998	0.000094614	-100.	0.129223	-99.9464	31
Nowait Batch	0.52056	0.	0.129801	-46.9245	0.0000646176	-100.	0.129866	-99.9461	31