***Platform Performance Testing, Analysis, and Benchmarking:***

**Updates & Customization (a): Table Operations**

**JJ Shen**

Sept 5, 2016

1. **Introduction**

Customizations to application tables, fields, and records are tracked through update sets or through team development tools. Conditions for tracking updates in a table include having upadate\_synch dictionary attribute, or special handlers to track changes to multiple tables. Dictionary entries are added, and records in Update Version [sys\_update\_version] and Customer Update [sys\_update\_xml] tables are created to track updates in most cases, among other changes.

Table is the basic element in the PaaS platform. In this performance test, customization operations related to tables are included, they are:

1. Add new columns to a table.
2. Modify table column values: column length, data type, and certain flags.
3. Delete table columns.
4. Create new tables.
5. Delete tables.

In the above tests, new tables are created in Global scope and extend ‘Task’ table. New columns are added to the created tables, and column types, max length, etc. are modified. The added columns are then deleted. The tables created are deleted at the last steps.

In each of the above case, dictionary records are added, and versions [sys\_update\_version and sys\_update\_xml] are created. The test measures the responses for the operation’s transactions, such as the time needed to modify a column length, from the action submission [http(s) requests] to getting the response back (versions are created in the background within this time). Other performance data are also collected.

1. **Performance Testing Approach**

JMeter is used for the performance test. JMeter scripts are developed to simulate real environment.

The testing data for performance analysis and benchmarking are collected on a single node instance at the data center. JMeter scripts are launched on a local laptop, from command line (instead of from JMeter console, to reduce overhead).

The following strategy and experiments are used and done to make certain that the tests are valid, and to come up with the range (number of users, wait time, etc.) where the performance data is valid, and to determine the system performance degradation range.

1. Several different wait time numbers are used between operations (such as between querying table form, dictionary form, and modify a table column), ranging from half seconds to 5 seconds.
2. Different number of concurrent users are used, to also test when the response will de-grade. Number of users are from 1 to 50.
3. Different total script running time is used, from 5 minutes to 60 minutes.
4. Different thread ramp up time is used.

Other than those, also run the scripts on lab instances for the main configurations for comparison, as well as on two different data center instances with the same releases.

1. **Performance Benchmarks & Analysis: Helsinki Release**

In this section, data from running on a Helsinki (patch 4) instance are collected and analyzed. (see next section for comparison of basic performance of Fuji, Geneva, Helsinki, and Istanbul). Based on the data collected with different number of users and different wait time, for performance benchmark, the data with 5 users, and wait time 3 seconds, are used. Firefox 45 is the user agent for the simulation. Note, the response time for the actions does not include browser render time.

***3.1: Response Times***

The following table summarize the average response time for the operations measured. The data are collected with a 10 to 30 minutes run with 3 to 5 concurrent users, and 3 seconds wait time in between requests, on a single node instance at data center (sjc6.service-now.com).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Average (msec) | Std. Dev. (msec) |
|  | Insert Table Column | 2217 | 468 |
|  | Delete Table Column | 1679 | 342 |
|  | Modify Column Length | 1576 | 348 |
|  | Multiple Updates for a Column | 1620 | 281 |
|  | Create Table | 1562 | 464 |
|  | Delete Table | 4810 | 998 |

From the data above, we noticed the following:

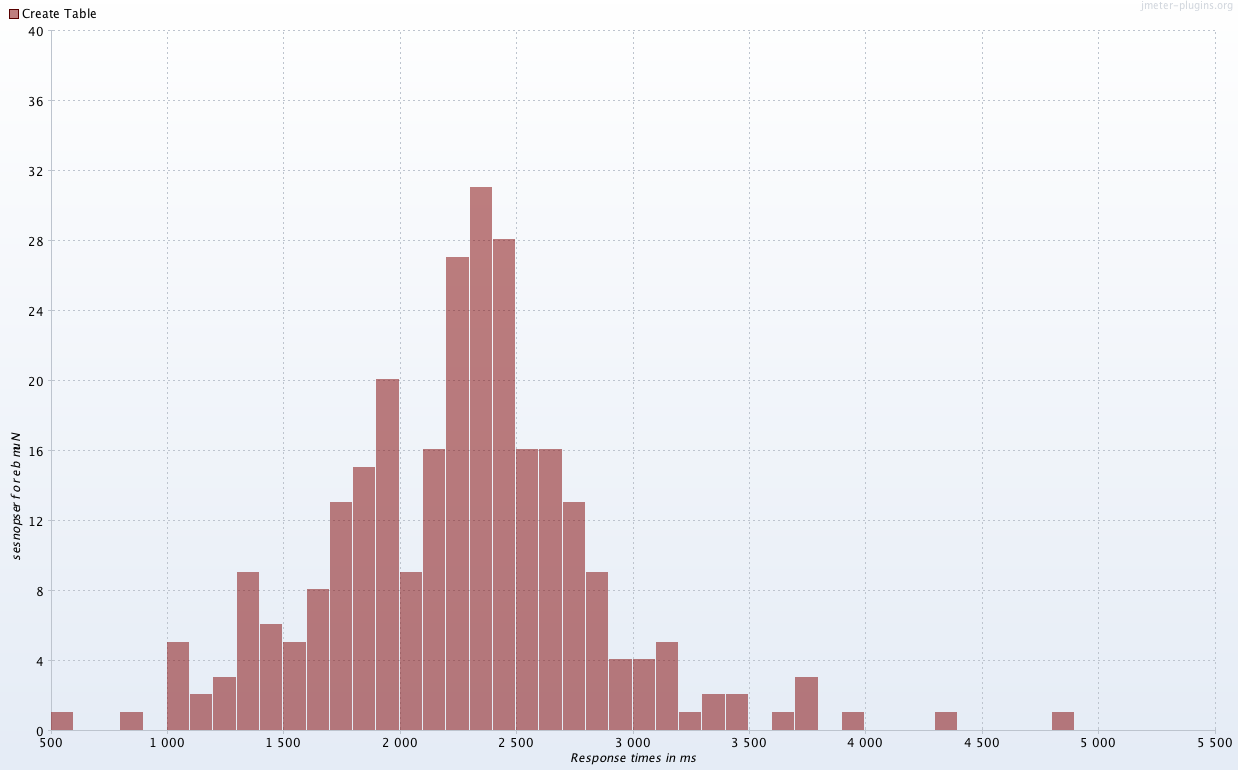
1. Delete a table takes close to 5 seconds, much longer than creating a table.
2. Insert a table column takes about 2 seconds, and delete a table column takes a little bit less time.
3. Update a column takes less than 2 seconds. Also, update one value for a column and update multiple values for a column take about the same amount of time in one request.
4. The standard deviation for all the operations are high. In other words, the response time is highly distributed (see the distribution graphs below for example).

The following shows a few samples of details of the data collected for the above results.

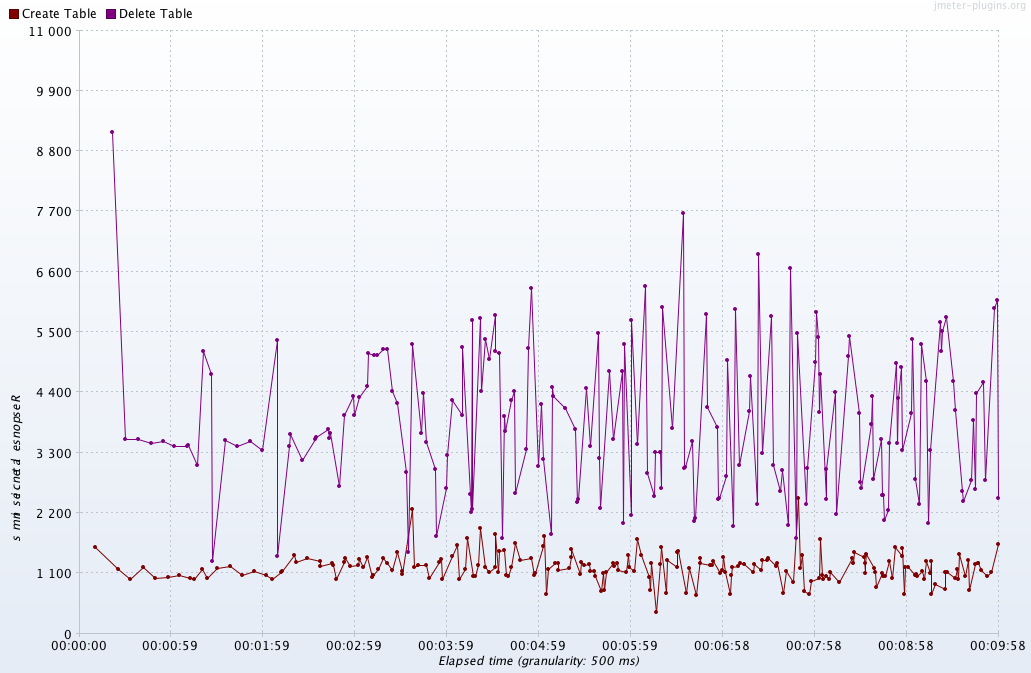
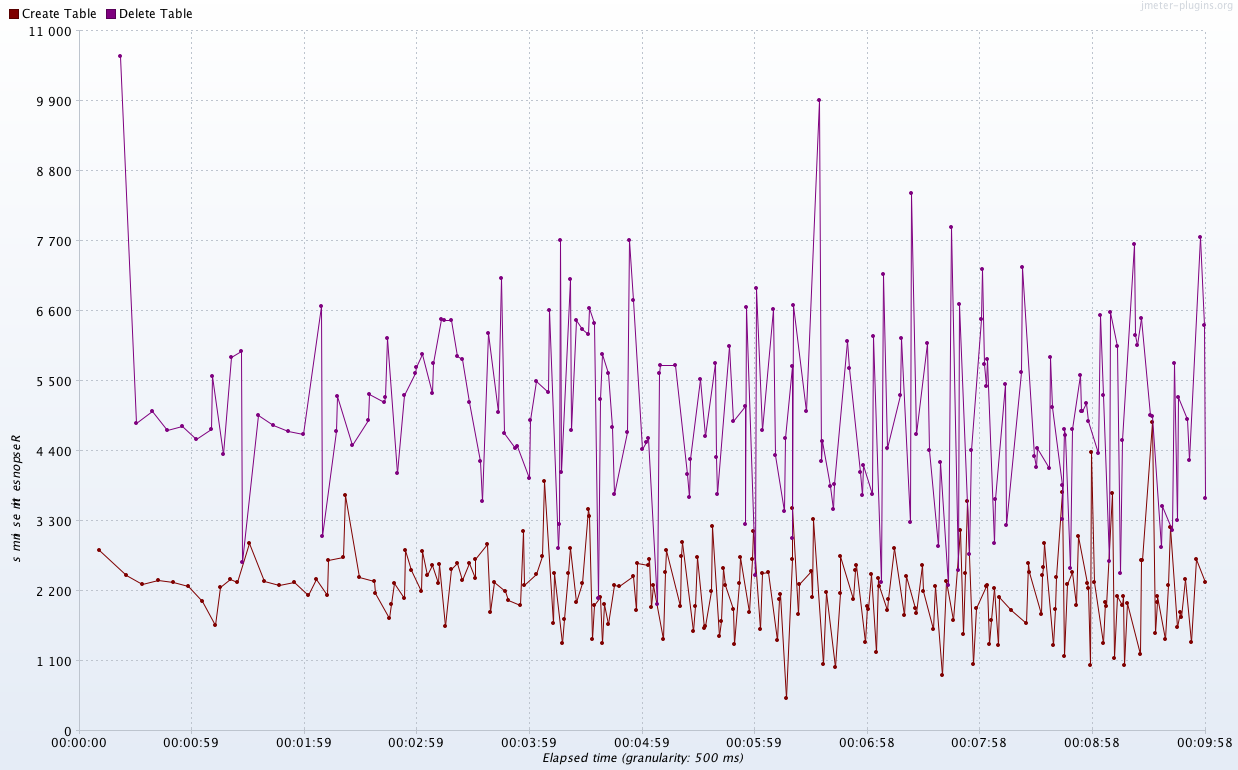
For comparison, we also tried to get the response time for creating incidents (no versions [sys\_update\_version] are created for creating incidents), which takes about the order of 300 milliseconds in average.

The graph to the right shows the response times distribution for table creation. We can see that table creation, as well as other operations, take different amount of time for different users at different time, with average 4810 milliseconds, and normal distribution

standard deviation of 998 (which means, for example, 67% of the time, table creation takes between 3812 milliseconds and 5808 milliseconds [average plus/minus delta]).

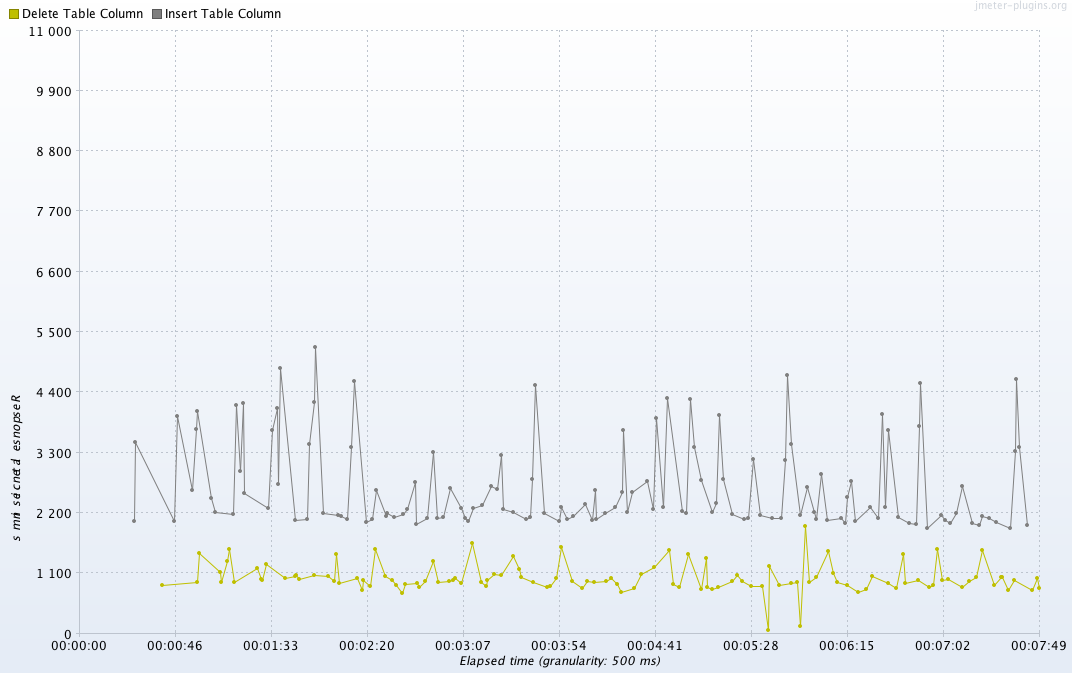
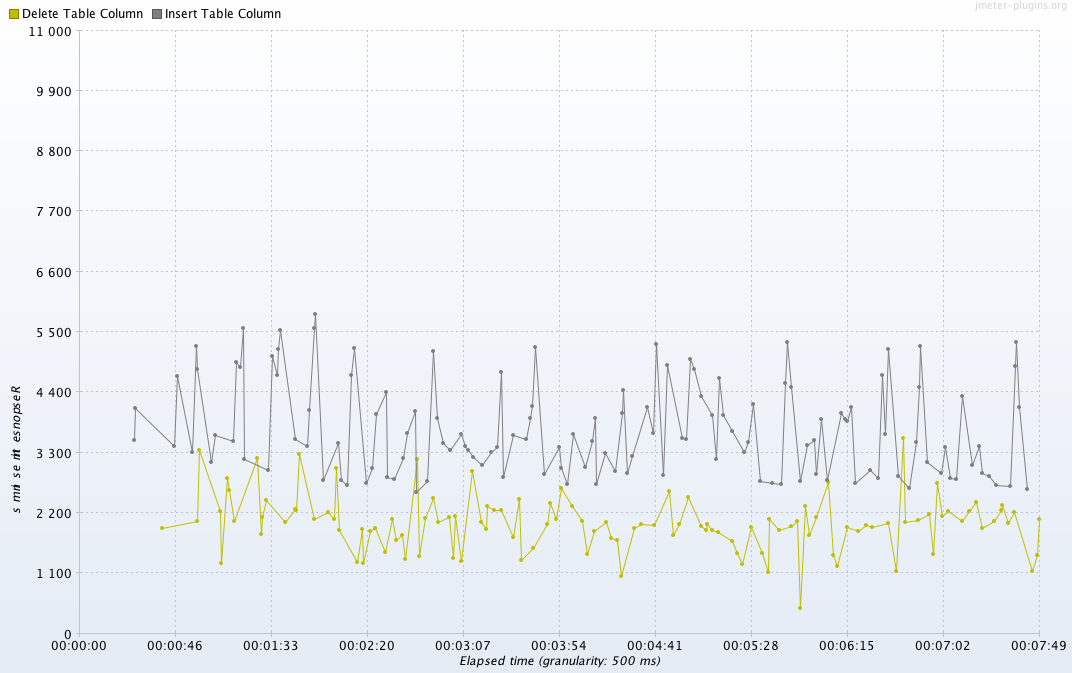


The following graph on the left shows the response times over time, for a 10 min run with 3 through 5 concurrent users, for table creation and deletion. On the right, the response latencies over time is plotted for the same test run.



From the above plots, the average latencies for create and delete tables are about 1.1 seconds and 3.2 seconds, respectively. Compare that with total response time, it’s clear that server processing time is about 1 seconds and 1.6 seconds for creating and deleting a table, respectively.

The following shows examples of response time over time and the latencies over time for insert and delete columns.



From the plots, we can see that the average server process time and the response latencies account for 1 and 2.1 seconds respectively for column insertion, and 0.6 and 1 seconds respectively for column deletion, in the response time.

For column updates, the average server process time and the response latencies account for about 0.6 seconds and 1.1 seconds, in the response time (graphs are not shown).

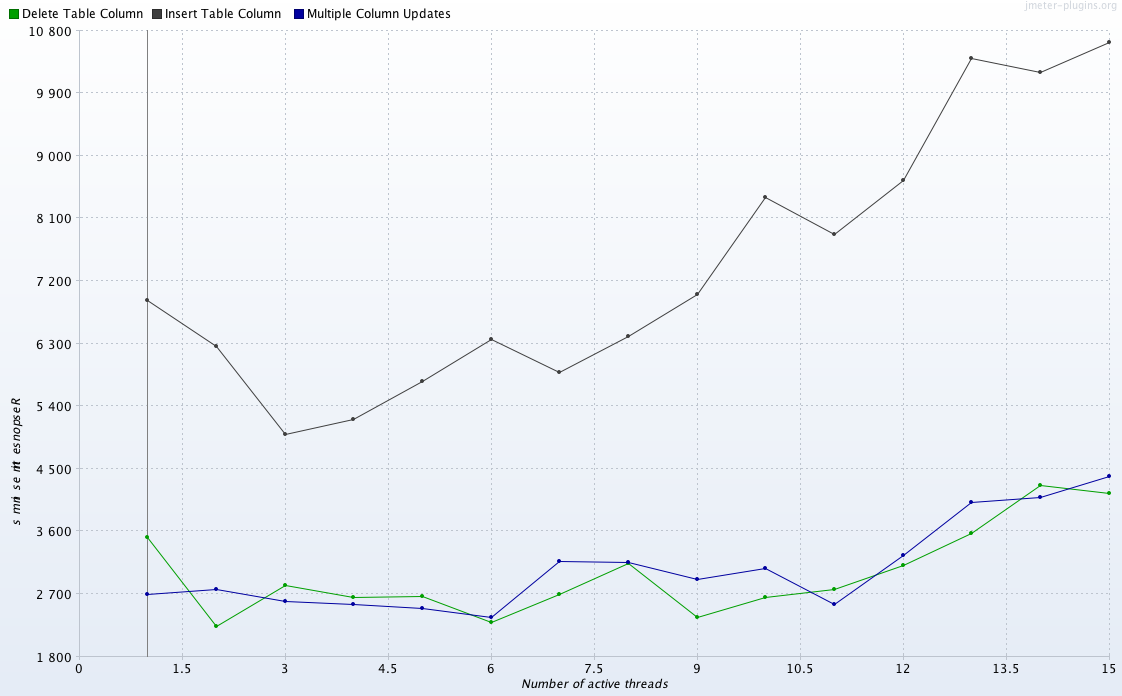
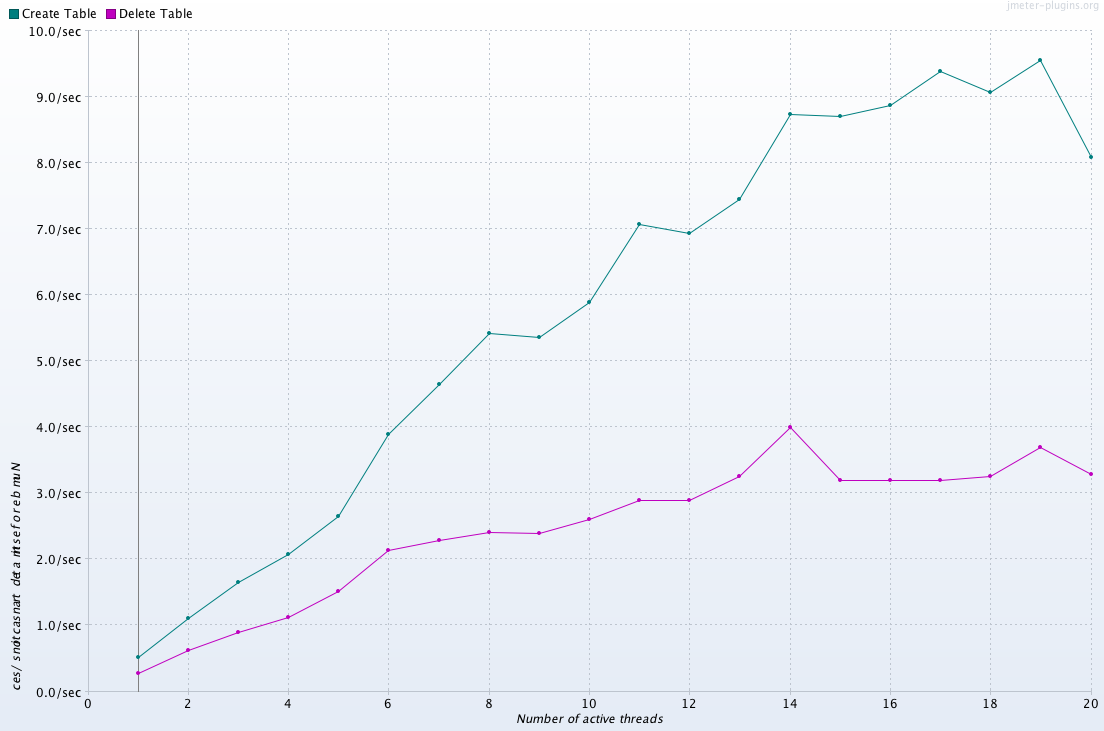
In Chapter 5, the sever processing time is analyzed in detail for Istanbul release.

***3.2: Performance Under Load***

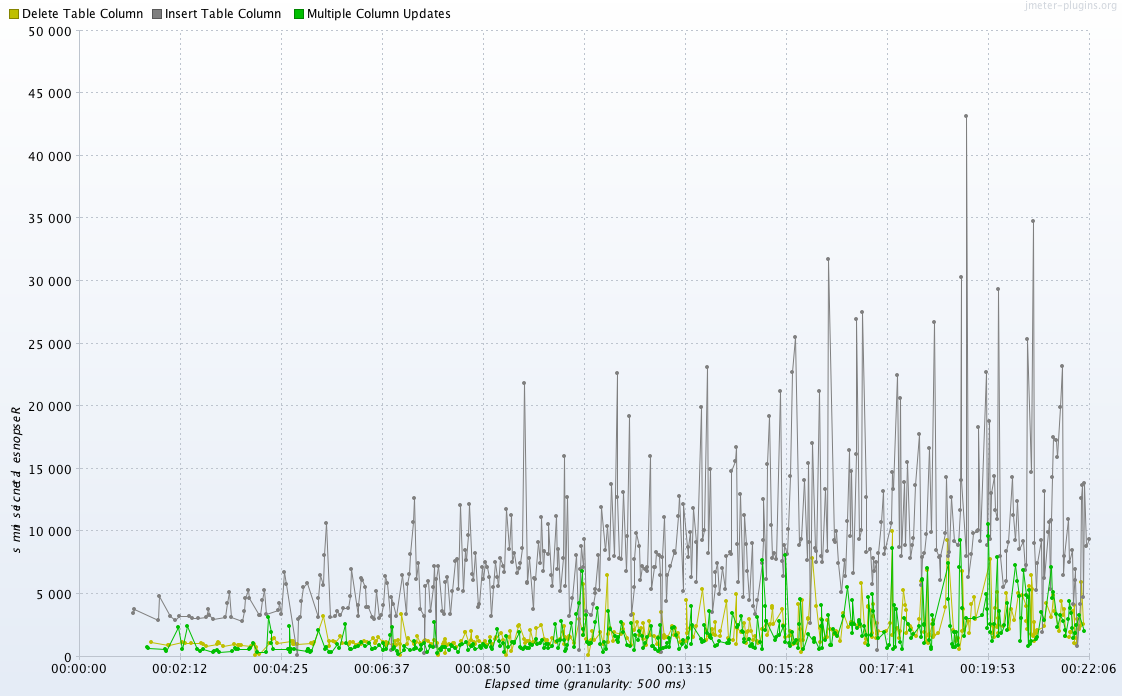
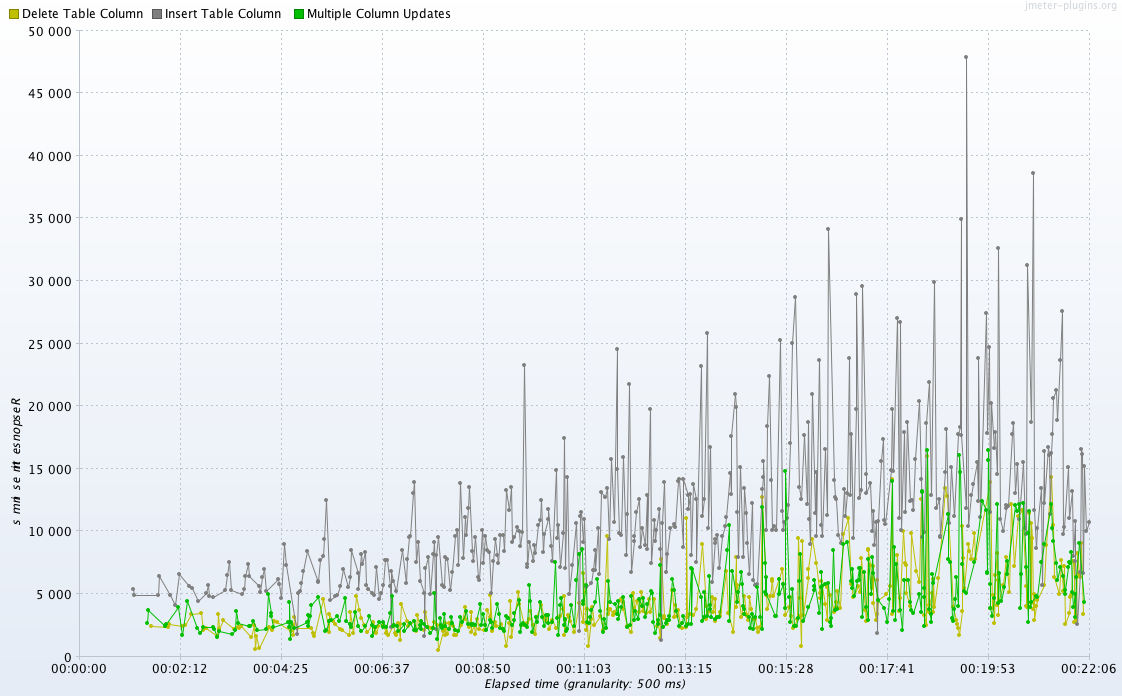
Number of concurrent users are increased gradually in the tests, and to determine performance degradation under load.

On a customer development instance using platform features, the number of users is not big, only a handful most of the time, so there is no heavy load of table operations, as well other operations in platform features. But it’s good to know the limit of the platform performance under load or stress conditions.

For the table operations under the testing environment, where all users are performing table operations, the system responses start to slow down when the number of concurrent user sessions exceeds 10. See the following graphs of transaction throughput vs number of threads, and response time vs number of threads.



When the number of concurrent users are larger, the latency becomes larger, and therefore the response time increases dramatically. In this case, the server process time is a small portion of the response time. The following graphs show a case where the number of users is gradually increased from 1 to 25 in 20 minutes or so.



Note, network connections/conditions, amount of data sent through the HTTP requests, JMeter server capacities, etc. all play some roles in the performance. More research and testing are needed to collect more details on latencies and server process time (currently there is some limitations to easily collect server process time statistics with JMeter). Also distributed performance testing (launch scripts from multiple places at the same time and sending requests to the same instance) may be needed to further analyze performance under load. So the above declaration on performance degradation is not conclusive.

1. **Performance Comparison: Fuji, Geneva, Helsinki, and Istanbul**

In this section, performances of the table operations are compared for last four releases, including current to be released Istanbul version. This comparison is important to determine release to release performance changes, and also to validate and to monitor performance changes for future releases.

For reliable performance comparison, testing environment, settings, and conditions should be the same. For example, launch testing scripts from the same machine, testing instances should be at the same or similar locations and at similar state (after zboot, for example), same network conditions, running with the same number of threads, use the same amount of wait time, ramp up time, and duration.

The following data center instances are used to collect the performance data.

|  |  |  |
| --- | --- | --- |
| [teamdevb.service-now.com/](https://hi.service-now.com/cmdb_ci_service.do?sys_id=013f1e650ffd42402f42938172050e06&sysparm_view=data_center&sysparm_record_target=cmdb_ci_service&sysparm_record_row=4&sysparm_record_rows=6&sysparm_record_list=operational_statusNOT+IN2%2C7%2C6%5Eused_for%21%3DManagement%5Eassigned_to%3D97a6bfd38c537000772e1b3b0e285498%5EORDERBYname) | (app129142.sjc6) | FP13 |
| teamdevc.service-now.com/ | (app129163.sjc6) | GP7 |
| teamdeva.service-now.com/ | (app129163.sjc6) | HP4 |
| [teamdevd.service-now.com/](https://hi.service-now.com/cmdb_ci_service.do?sys_id=21e11c256f5bca402c9f8e4c2c3ee457&sysparm_view=data_center&sysparm_record_target=cmdb_ci_service&sysparm_record_row=6&sysparm_record_rows=6&sysparm_record_list=operational_statusNOT+IN2%2C7%2C6%5Eused_for%21%3DManagement%5Eassigned_to%3D97a6bfd38c537000772e1b3b0e285498%5EORDERBYname) | (app128177.sjc6) | IP0 |

The average response times are shown in the table below, and also shown in the following graphs (unit in milliseconds).

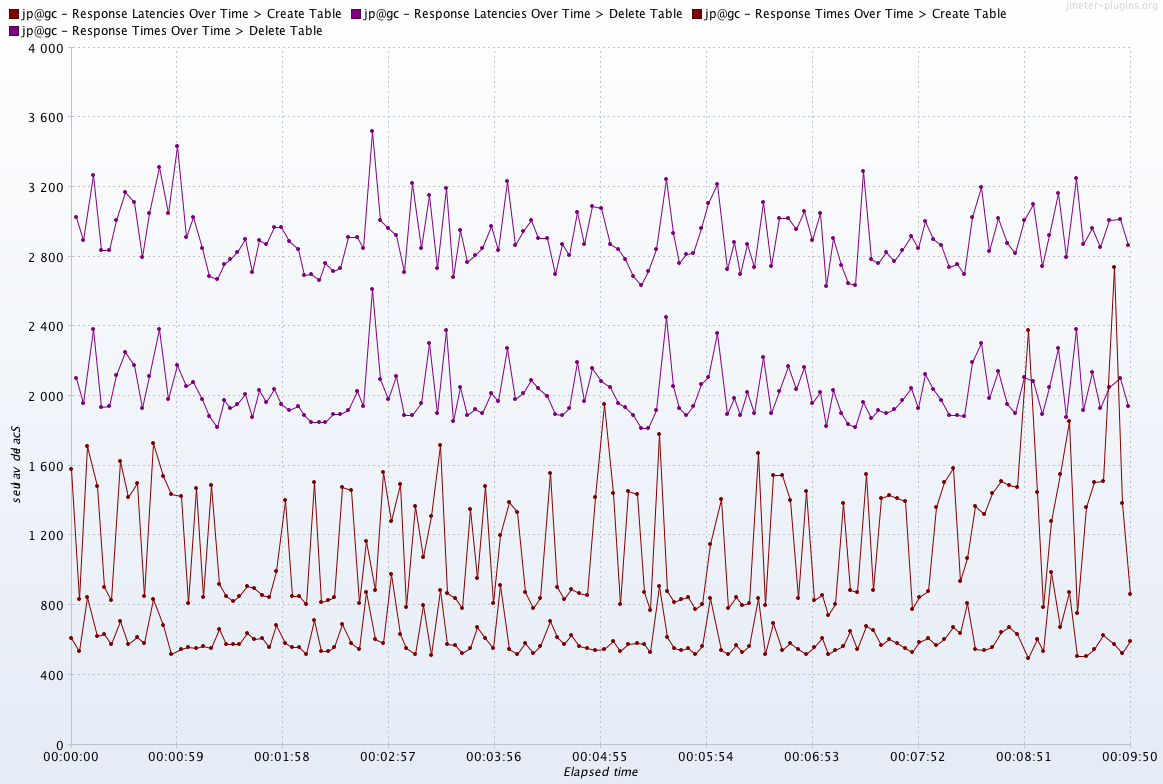
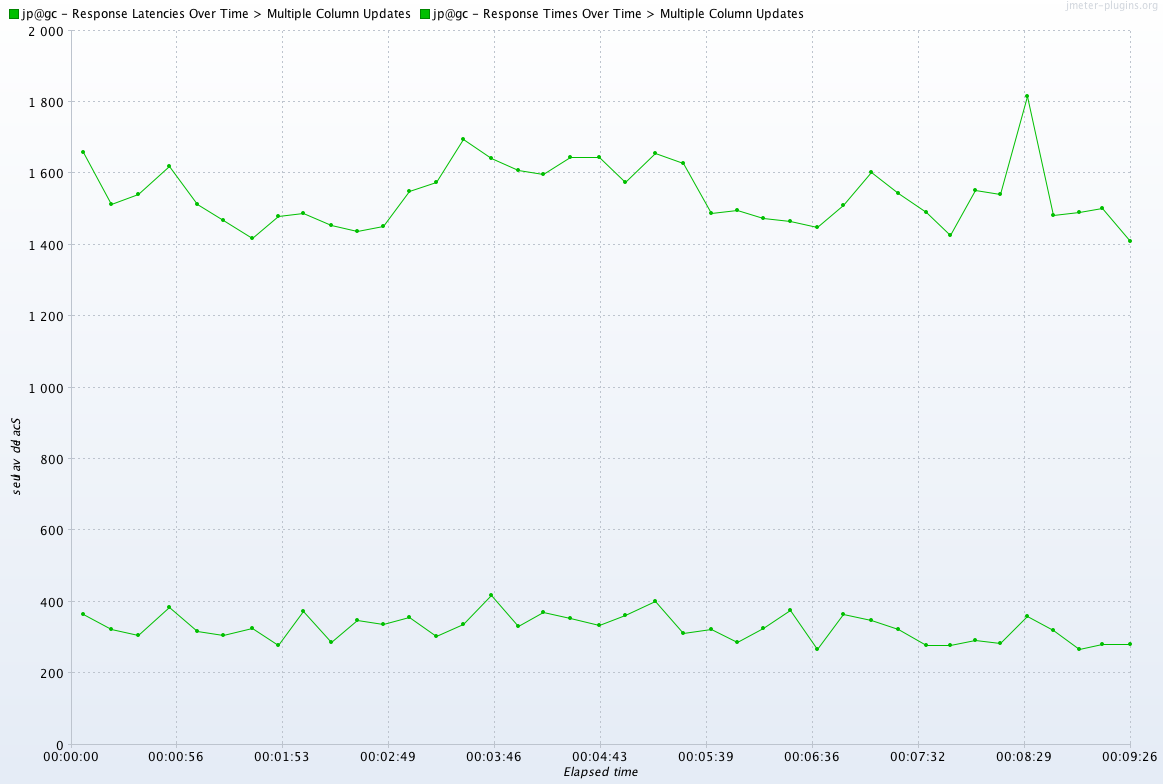
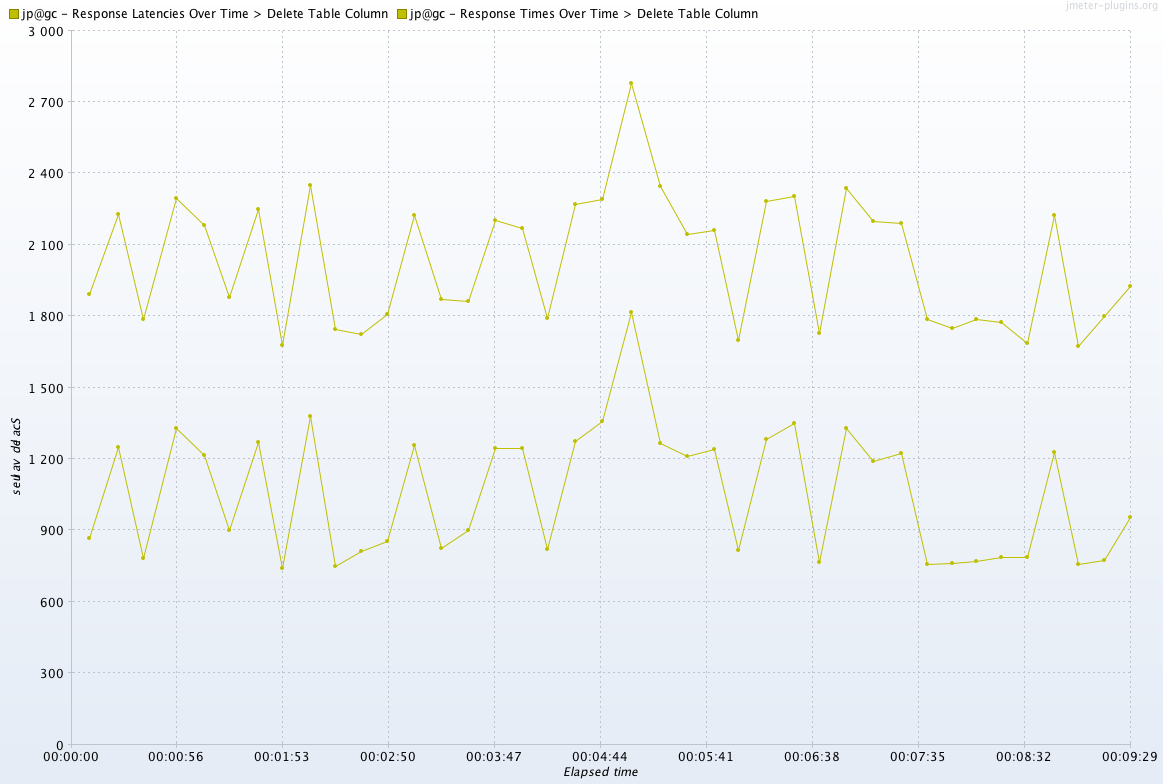
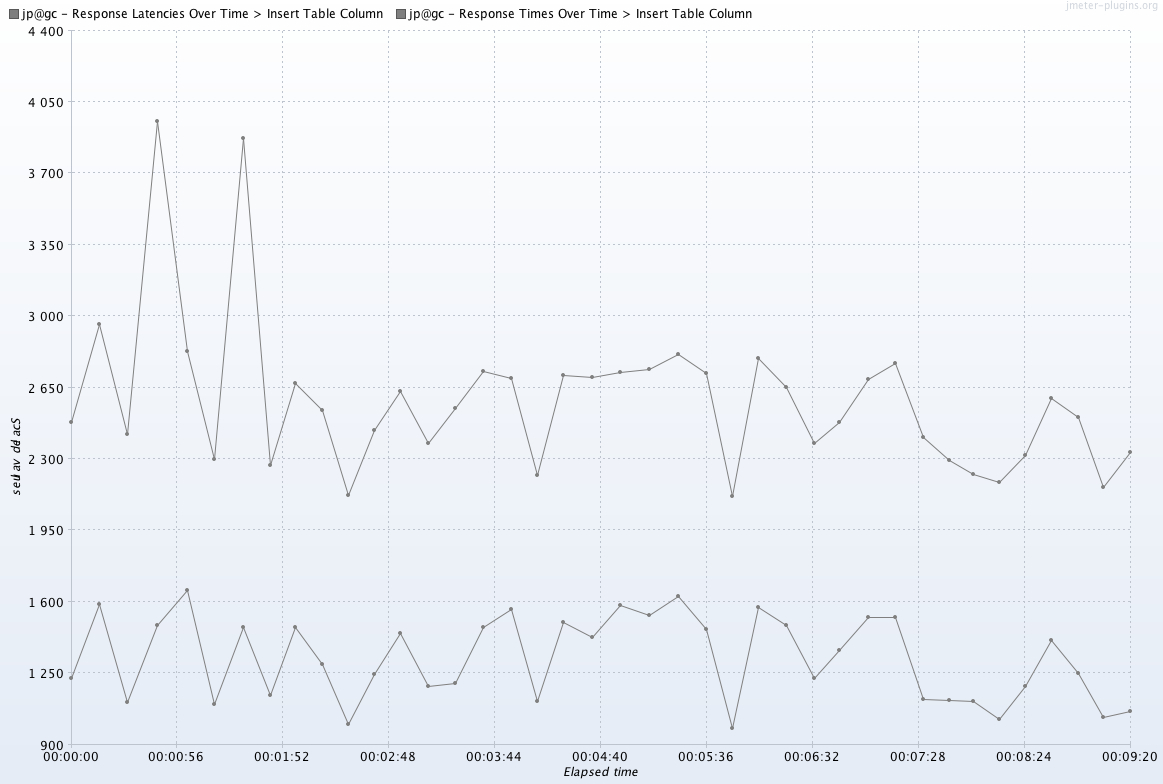
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Fuji | Geneva | Helsinki | Istanbul |
| Insert Table Column | 2258 | 2884 | 2217 | 2021 |
| Delete Table Column | 1768 | 1831 | 1679 | 1941 |
| Table Column Updates | 2442 | 2245 | 1620 | 1598 |
| Create Table | 1752 | 2387 | 1562 | 1977 |
| Delete Table | 2871 | 3546 | 4810 | 3411 |

From the testing data, we can see that there are no significant performance changes for last four releases, from Fuji to Istanbul. Deleting tables take a little bit longer time in Geneva release than others. There is a slight improvement of table column updates performance release by release from Fuji to Istanbul.

For comparison (as a bonus), the following graph plots the average response time of creating an incident under normal condition (no load).

1. **Sever Processing Time Analysis for Istanbul Release**

To show the server processing time clearly, performance testing is run with one concurrent user only, on a datacenter instance that was upgraded to latest Istanbul build. Server processing time is roughly the response time minus latency (JMeter can’t measure server time directly), in milliseconds.



the following table summarizes the result for this particular testing (note, response time may be different from that in earlier testing due to environment differences, instance version differences, etc.). Insert incident is also included for comparison.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Response Time | Latency | Process Time |
| Insert Table Column | 2505 | 1320 | **1185** |
| Delete Table Column | 2007 | 1020 | **977** |
| Table Column Updates | 1542 | 480 | **1062** |
| Create Table | 1180 | 580 | **600** |
| Delete Table | 2909 | 2000 | **1909** |
| Insert incident | 311 | 80 | **231** |

1. **Next Steps and Plans**

Here are next steps for platform updates & customization feature’s performance testing, and plans for platform performance testing overall.

***5.1: Performance Testing of More Customization Operations***

1. Certain other update\_synch table updates.
2. Multiple table updates: Choice lists.
3. Multiple table updates: Field labels.
4. Multiple table updates: Form sections.
5. Multiple table updates: Lists.
6. Multiple table updates: Related lists.

***5.2: JMeter Infrastructure Setting***

1. Perform JMeter script execution from the lab or dedicated environment, remove any potential issues with local machine resource limitation.
2. Distributed JMeter environment for stress testing (stretch goal).
3. For stress/under load testing, monitor system resource usage (CPU, memory, etc) on the instance server, to measure correlation with session load.

***5.3: Performance Testing on Customer Clones***

Perform the testing on customer clones with real time data and load, to compare performance with out-of-box instance.