ECSE 429

Software Validation

Team 9

Automation Project - Part C Deliverables

Christopher Cui

Student ID: 260867703

Bo Wen Li

Student ID: 260787692

Dominic Wener

Student ID: 260871052

Glen Xu

Student ID: 260767363

TABLE OF CONTENTS

Summary of Deliverables

Description of Implementation of the Performance Test Suite

- Include charts(excel or other tool) showing transaction time, memory use and cpu use versus number of objects for each experiment
- Summarize any recommendations for code enhancements related to results of performance testing
- Highlight any observed performance risk

Description of Implementation of Static Analysis with Sonar Qube

- Summarize how static analysis was performed on the Rest API to do list manager using Sonar Qube static analysis tool
- Summarize code complexity, code statement counts, and any technical risks, technical debt or code smells highlighted by the static analysis

Recommendations For Improving Code

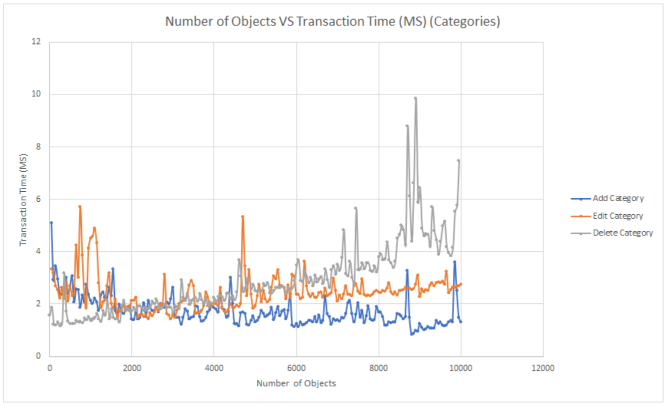
- Provide recommendations regarding changes to the source code to minimize risk during future modifications, enhancements, or bug fixes

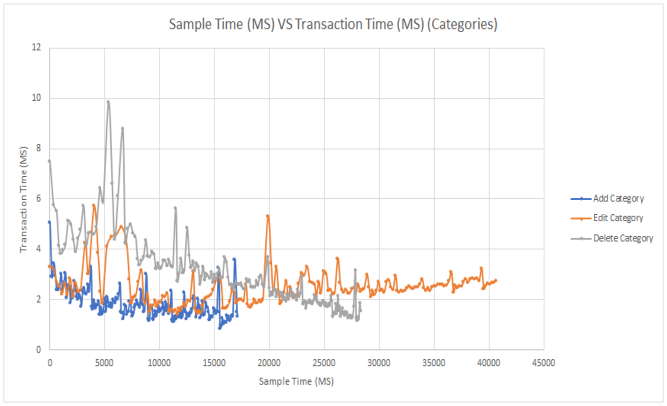
Description of Implementation of the Performance Test Suite

The performance tests for adding an object are done by an iteration up to 10,000, adding one object each time and measuring the transaction time, current time, and current number of objects on the server. Similarly, changing an object is also done in an iteration up to 10,000 adding one object each time, but we also modify an object after to measure the data required for that. For deleting an object, we first populate the server with 10,000 objects, then for run an iteration up to 10,000 and delete an object one by one, recording the required information. Thus, the transaction time graphs are all output csv files from our program. The available memory and CPU use graphs are from Windows Perfmon in which are run before the unit tests begin, and terminated after the unit tests end. In the case of deleting, we started Windows Perfmon when the add iteration is almost complete, and the delete iteration and starting.

Categories

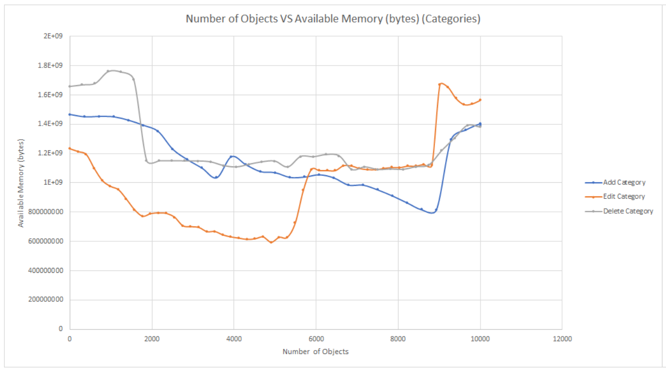
Objects, Sample Time VS Transaction Time

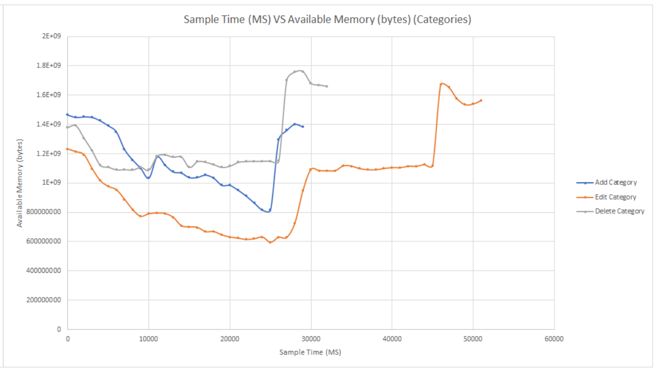




It can be observed above that as sample time or number of objects increases, the transaction time for adding and deleting a category decreases, and the transaction time for changing a category increases. The trend for deleting a category is appropriate because as we are deleting categories, the total number of categories on the server decreases. Thus the trends in the graphs for deleting are opposite from each other for sample time and number of objects. The trend of adding a category is interesting and unexpected since the transaction time decreases as the number of categories on the server increases.

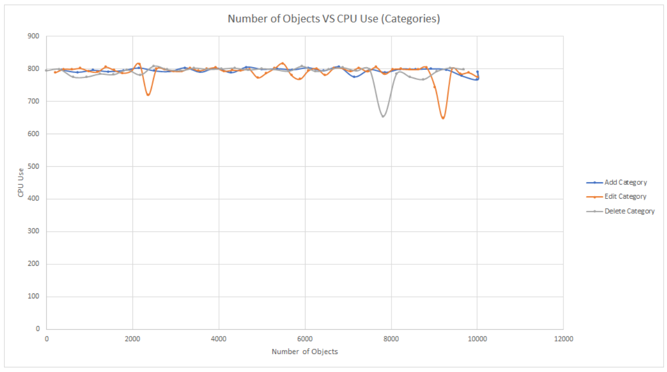
Objects, Sample Time VS Available Memory

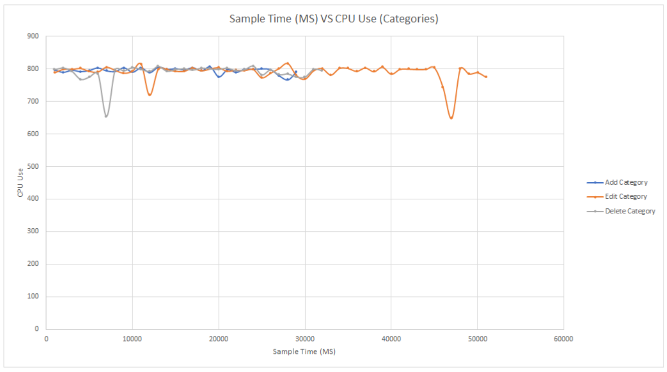




It can be observed that as sample time or number of objects increase, the available memory decreases to a certain point, then increases significantly. This demonstrates that as the unit tests run, the available memory decreases. The increase near the end of the test is because perfmon is run before and after the unit tests, thus the unit tests end before perfmon. This demonstrates that a lot of memory is freed up after the unit test execution. The above trend is consistent for adding, changing, and deleting a category (note that deleting graph is opposite in both graphs because as time goes on, objects are being reduced on the server).

Objects, Sample Time VS CPU Use

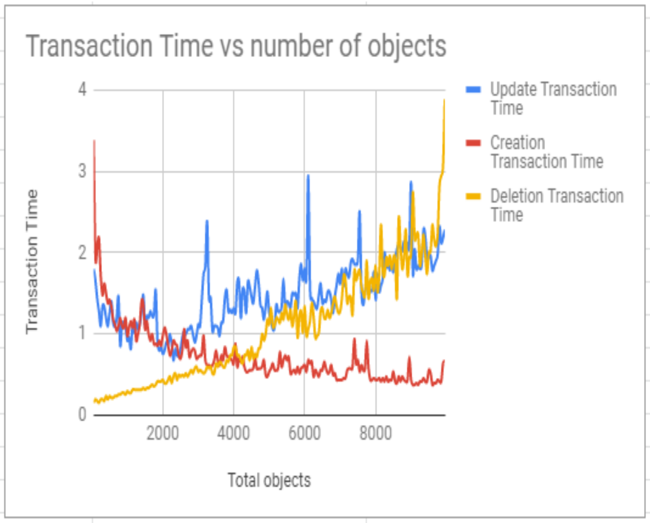


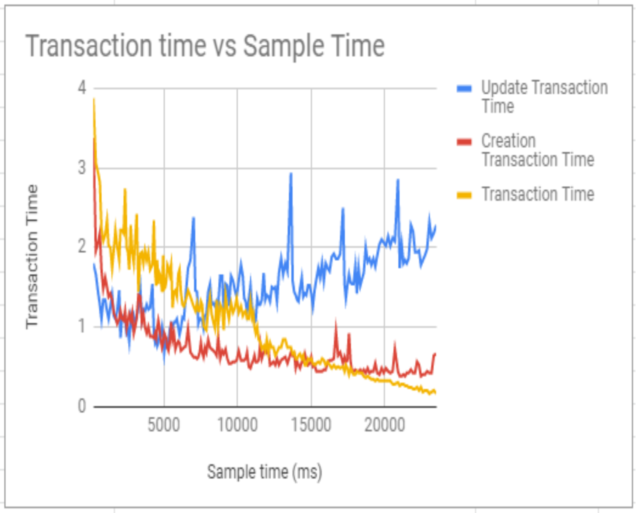


There are not any significant changes to CPU Use as sample time or number of objects increase. It is noticeable that CPU Use drops near the end for adding, changing, and deleting a category and this is because perfmon observes a few seconds after the unit test execution (note that deleting graph is opposite in both graphs because as time goes on, objects are being reduced on the server).

Projects

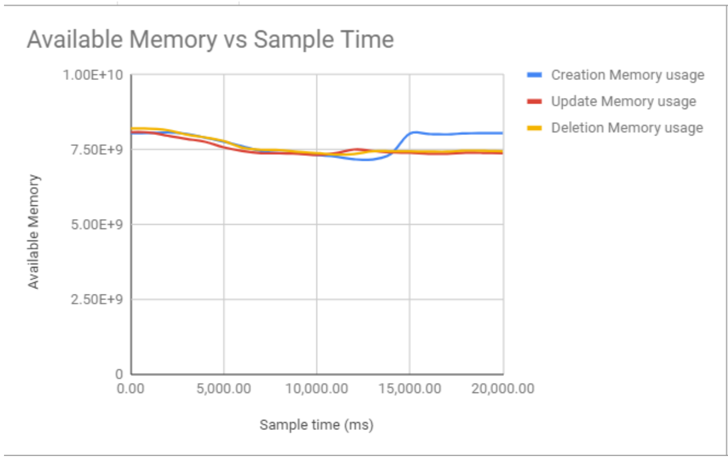
The graphs on the left show sample time on the x axis and the graphs on the right show the number of objects on the x axis.

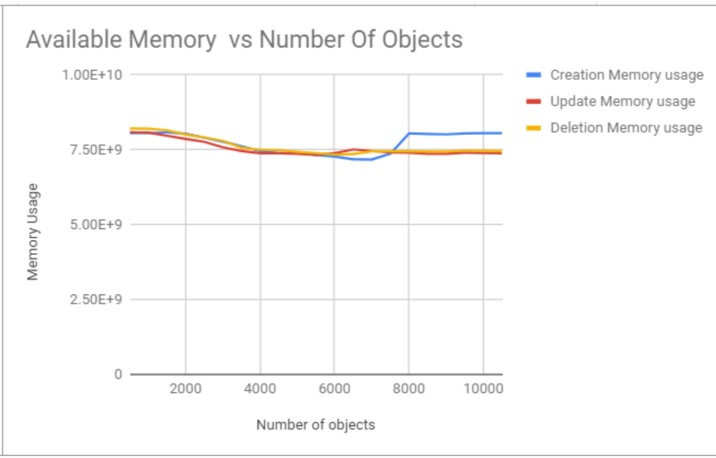




__Transaction time__

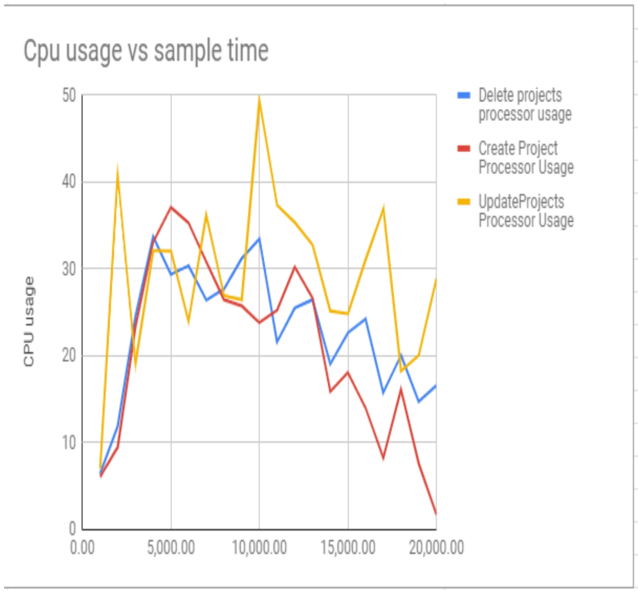
We can see a sharp trend in increase of transaction time as the number of objects increases for the deletion and update operation. However, this behavior is not exhibited for the creation time. Creation time seems to not have this same effect as the objects increase. We have a large transaction time at the start of the sample time. It is assumed this is because the API server has to start up and may be slower at the start of our object creations.

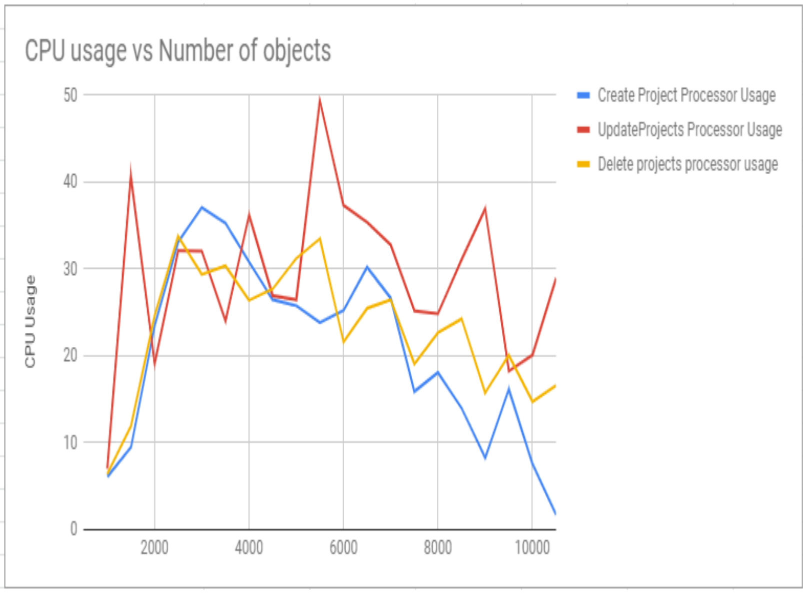




Available Memory (Memory Usage)

Charted on this graph is available memory. Higher is better, and a lower value means the api is using more of the memory. We can see the available memory dip lower as the number of objects increases. However, it seems to rise at the 15000.00 sample time mark. This could be due to PC interference on my windows machine. Many background apps are running at any time on Windows.





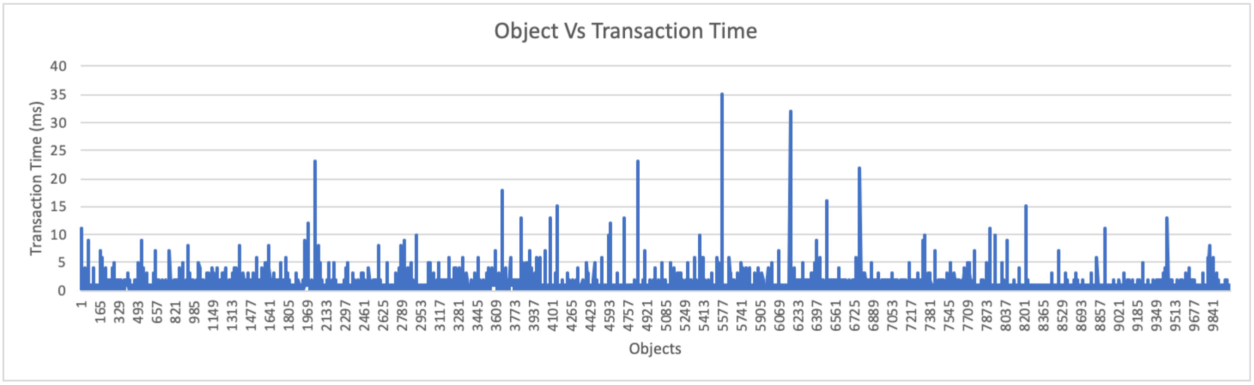
<u>CPU usage</u>

The CPU usage is a bit sporadic. It peaks towards the middle of the sample, then it has a steady trendline downwards. The same behavior is witnessed for all 3 transaction types.

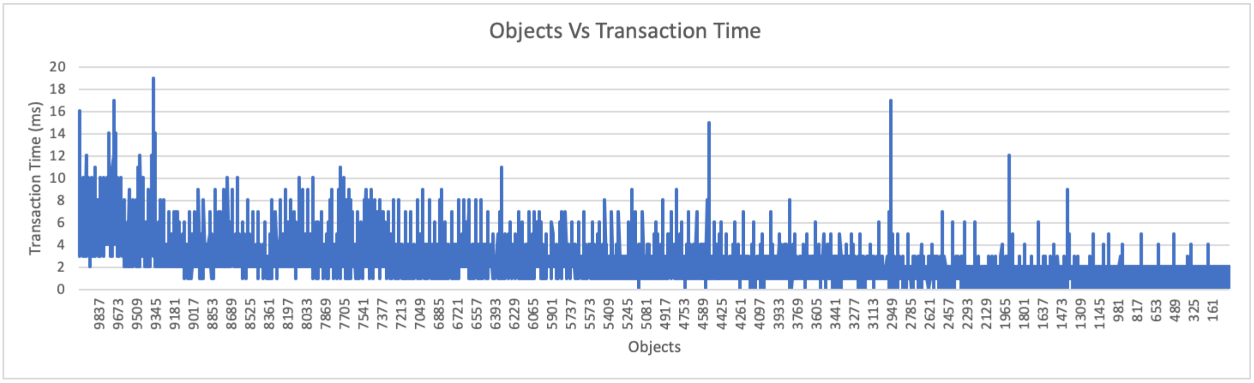
Interoperability

According to the documentation, there are no APIs allowing users to change, modify or update the information of a capability from another instance of capability. For example, if we want to modify the todo id=1 attached to project id=2, we cannot do POST /projects/2/tasks/1, because such a command is unsupported. We have to find the id of this todo, and update it directly in todo by POST /todos/1. However, if we do it this way, it is already tested in the capability of todo, instead of being part of interoperability. As a result, only "create" and "delete" are tested in the interoperability session. Since the connections between capabilities are symmetrical, we have used the connection between todos and projects as a representative.

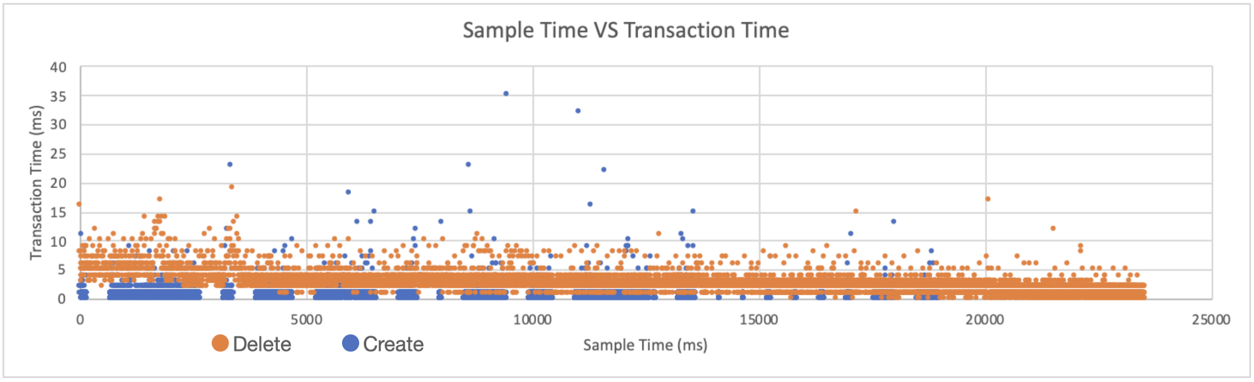
<u>Objects VS Transaction Time, Create</u>



Objects VS Transaction Time, Delete

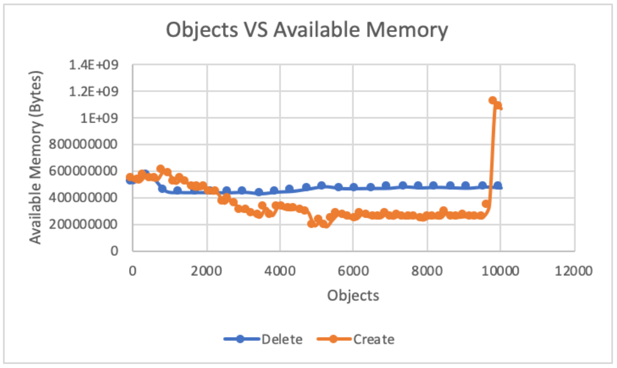


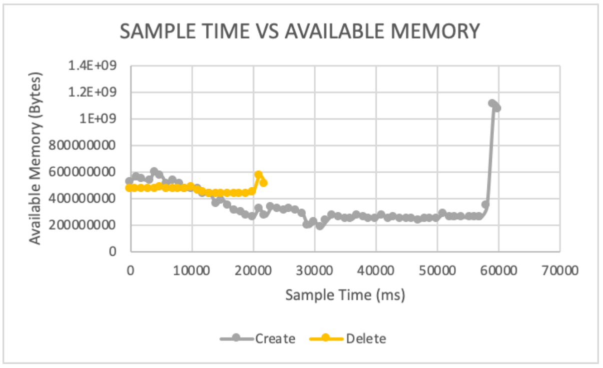
Sample Time VS Transaction Time



From the graphs, we can observe that when there are more objects in the server, the time required for deleting an object increases. This makes sense, because when there are more objects, it takes more time to find the target object to be deleted. There are no obvious relationships shown for creating objects, since the time taken is always pretty short no matter how many objects there are. However, it is obvious that when there are more objects, there are more outliers that spike from the rest of the data. This may indicate that when there are more objects, irregularities are more likely to occur.

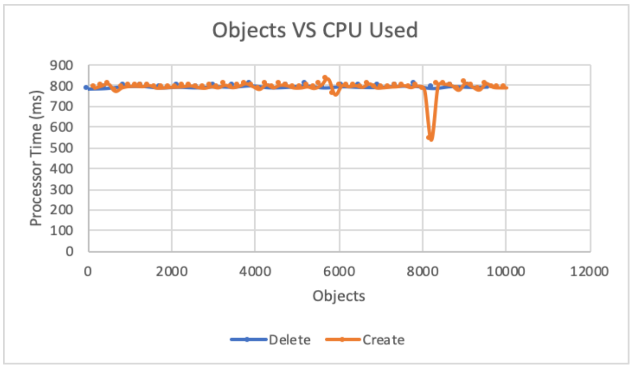
Objects & Sample Time VS Available Memory

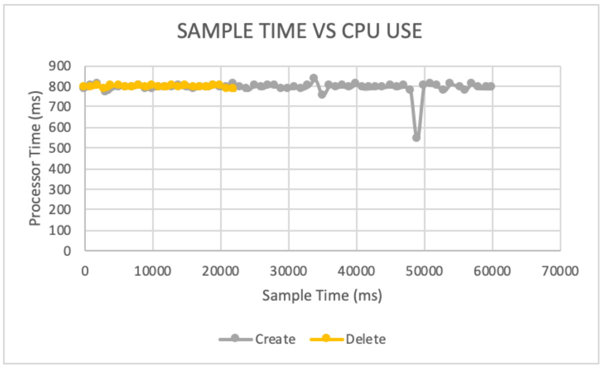




For both creating and deleting operations, the available memory first gradually decreases, then dramatically increases. The gradual decrease indicates that the tests slowly take up more available space in the memory. The dramatic increase indicates that the execution of the test takes up a lot of memory, and the taken memory is free-ed at once after the execution of the tests.

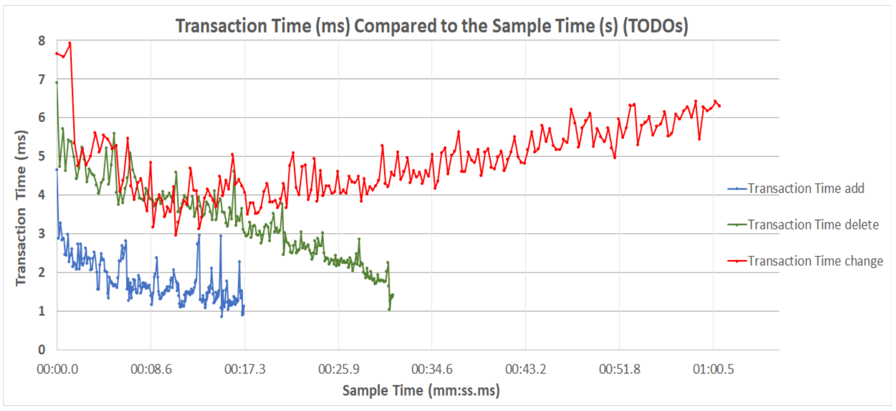
Objects & Sample Time VS CPU Use





There are no specific patterns or tendencies observed from CPU use in either of the operations.

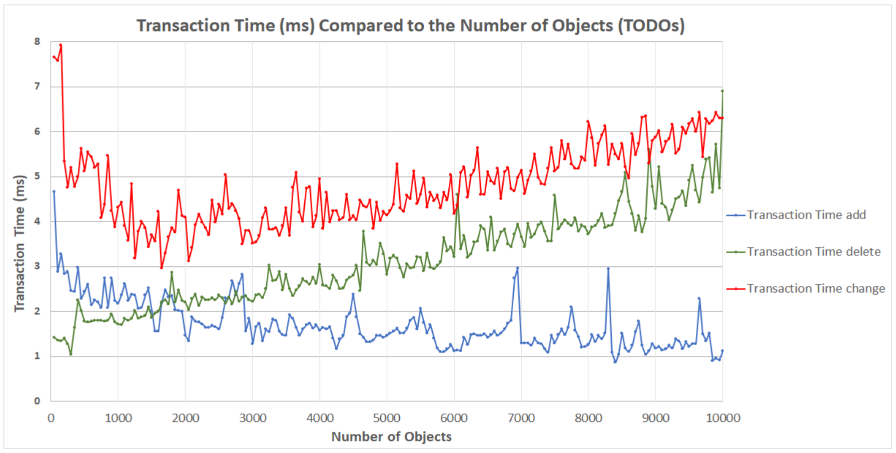
TODOs



Transaction Time Analysis

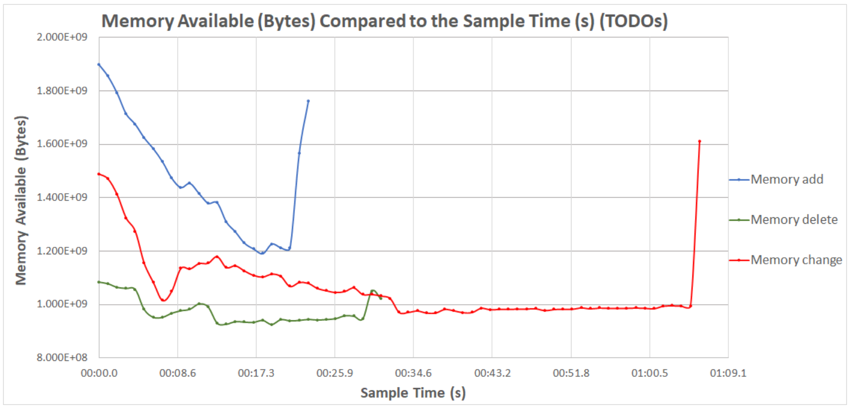
The graph on the left highlights the correlation between the Transaction time (y-axis) and the overall sampling time (x-axis) required to do 10,000 of each transaction on a TODO instance. The red line shows that the average transaction time for changing a TODO instance gradually increases. The blue line shows that the transaction time for creating new TODO instances decreases slightly as we create more entries, and stabilises between 1 and 2 milliseconds per transaction. The green line indicates that the transaction time decreases as we delete more and more TODO instances. However, since the Performance Test is designed to start deleting when there are 10,000 TODO entries, until there is 1 left, we could also extrapolate that the Transaction time decreases when the amount of TODOs in the API decreases.

Furthermore, the first samples of each Performance Test are noticeably larger than the average samples following them. These could be outliers caused by the start of our Tests or the start of the Performance Monitoring Program, or they could be indicators of code weaknesses and possible improvements.

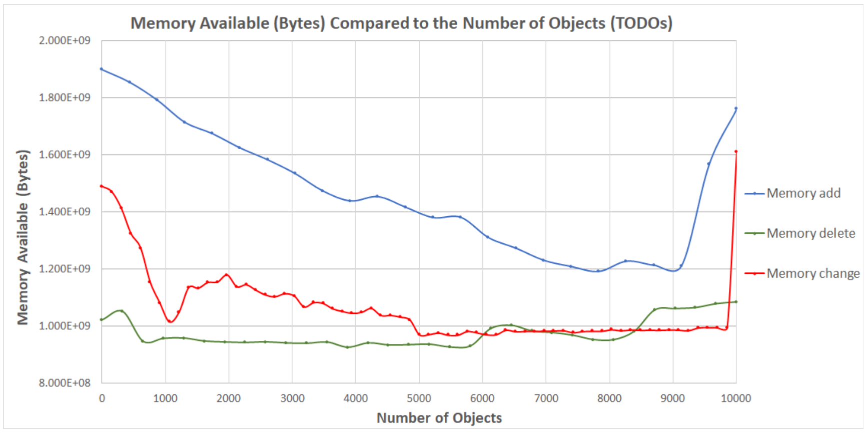


The above graph highlights the correlation between the Transaction time (y-axis) and the Number of Objects (x-axis) that exist in the API database, i.e. 0 to 10,000 TODO instances. Once more, the red line indicates that the transaction time for changing a TODO instance gradually increases and the blue line shows that the transaction time for creating new TODO instances decreases slightly until it stabilises between 1 and 2 milliseconds per transaction. The green line shows the transaction time increasing for deleting TODOs, which corresponds to the previous conclusions: the transaction time increases when there are more objects.

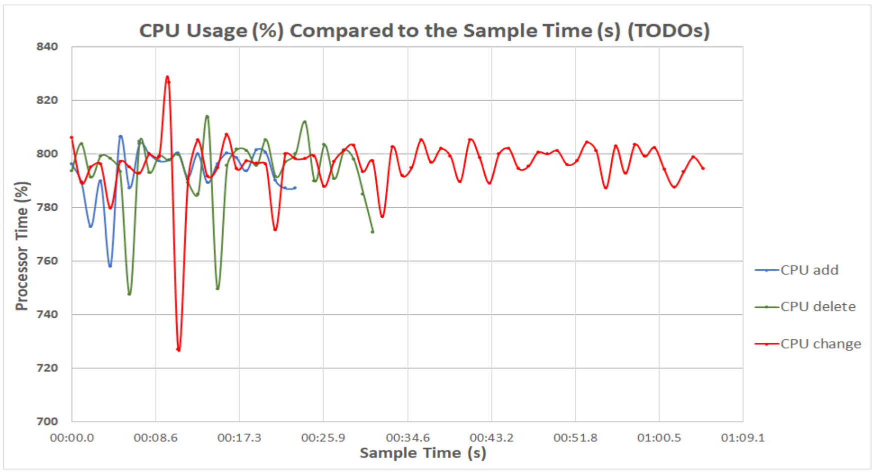
<u>Memory Available Analysis</u>



The graph on the left highlights the correlation between the Available Memory (y-axis) and the overall sampling time (x-axis) required to do 10,000 of each transaction on a TODO instance. The red line shows the available memory decreasing abruptly from 0 to 8 seconds, which corresponds to the start of the performance test and the start of Perf. Monitor. It continues a decreasing trend until it stabilises a little after the halfway point. We could extrapolate that continuing to create more TODO instances and changing them would not decrease the available memory. However, the blue line shows that the available memory will continue decreasing as we create more TODO instances. When the 'Add' Performance Test ends, we see the available memory increase abruptly. The green line shows a very small decrease of Available Memory when the Test starts deleting the 10,000 TODO entries. However, the line stabilizes afterwards as the test deletes the last half of the 10,000 TODOs.

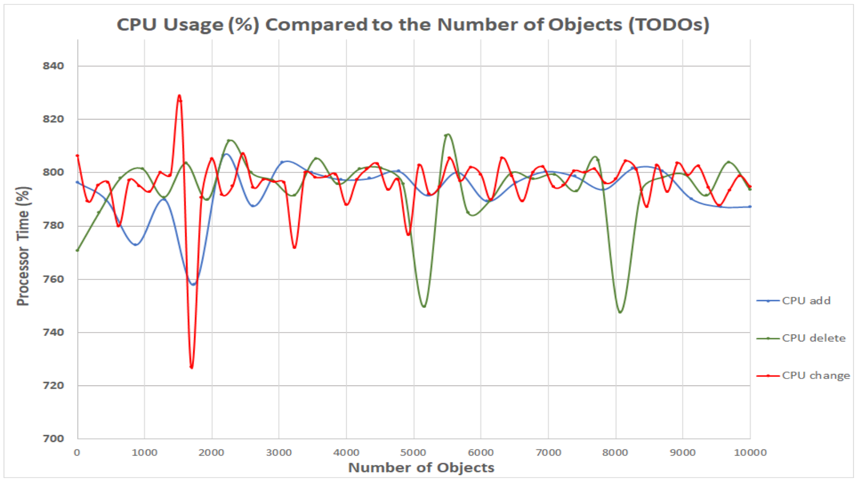


The above graph highlights the correlation between the Available Memory (y-axis) and the Number of Objects (x-axis) that exist in the API database, i.e. 0 to 10,000 TODO instances. The blue line shows once more a decrease in Available Memory as we create more TODOs. The red line shows the same results as the previous Graph. The green line shows constant Available Memory when the test deletes most of the TODOs until the end of the Test where there is a small increase.



<u>CPU Usage Analysis</u>

The beside graph highlights the correlation between the CPU Usage (y-axis) and the overall sampling time (x-axis) required to do 10,000 of each transaction on a TODO instance. The red line, aside from the large variation at the beginning of the test, shows a constant CPU Usage while creating and then changing the 10,000 TODOs. The blue line shows the same conclusion for only creating the TODOs. The green line shows high variations when the test starts creating the 10,000 entries and when it starts deleting the 10,000 entries. Otherwise, it shows small CPU Usage variations until a sudden drop when the Test terminates.



This graph highlights the correlation between the CPU Usage (y-axis) and the Number of Objects (x-axis) that exist in the API database, i.e. 0 to 10,000 TODO instances. The same results as the previous graph can be seen here.

Performance Test Recommendations

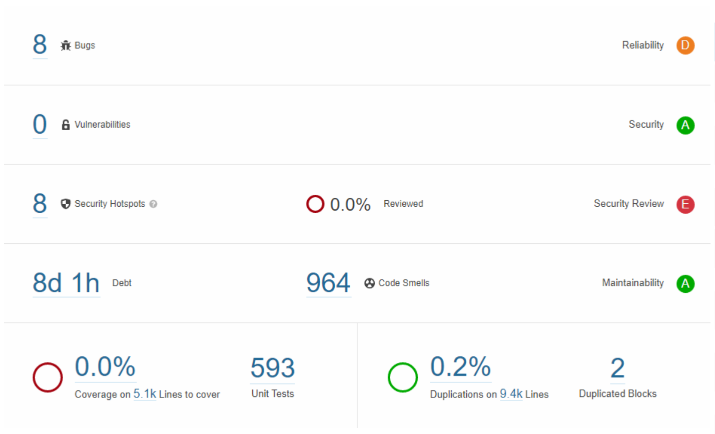
CPU Usage was relatively consistent throughout the dynamic tests and we did not observe any significant trends or results. Improvements can include isolating all background processes that can skew data results. For changing object tests, we implemented them to change only one instance repeatedly after a new object is created. It can be interesting to see how this result would differ from a changing object test where we modify different instances during each iteration.

Performance Risks

It was observed for each Performance Test, that the transaction time values sampled at the beginning of a test were always larger than the average values sampled as the test moved forward. This could be an indication of weaknesses in the code during the early executions when an user sends requests to the API. As users do not enjoy waiting for execution times, this can prove to be a significant performance risk.

Description of Implementation of Static Analysis with SonarQube

SonarQube was used to implement the static analysis of the REST API to do manager. The application is downloaded from https://www.sonarqube.org/downloads/. The static analysis was performed on a Windows 10 operating system. SonaQube is run by the StartSonar.bat file in the windows bin folder. After the initialization completes, we can go on a web browser and go to http://localhost:9000/ and enter admin for username and password. Next, we create a new project and generate a new token on the website. Once the above steps are completed, we can run the provided maven command on the terminal in the thingifier root folder. Once it is complete, we can view the overview of the status of the code and a bug report page to specify individual problems as shown below.



8 Bugs — Reliability D

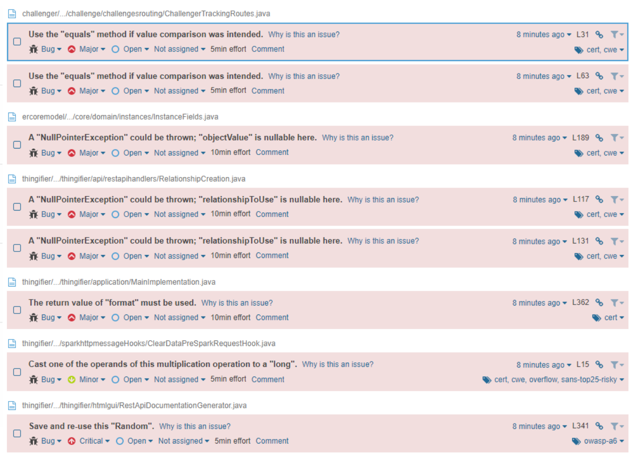
0 Vulnerabilities — Security A

8 Security Hotspots — 0.0% Reviewed — Security Review E

8d 1h Debt — 964 Code Smells — Maintainability A

0.0% Coverage on 5.1k Lines to cover — 593 Unit Tests

0.2% Duplications on 9.4k Lines — 2 Duplicated Blocks



challenger/.../challenge/challengesrouting/ChallengerTrackingRoutes.java

Use the "equals" method if value comparison was intended. Why is this an issue? — 8 minutes ago — L31
Bug — Major — Open — Not assigned — 5min effort — Comment — cert, cwe

Use the "equals" method if value comparison was intended. Why is this an issue? — 8 minutes ago — L63
Bug — Major — Open — Not assigned — 5min effort — Comment — cert, cwe

ercoremodel/.../core/domain/instances/InstanceFields.java

A "NullPointerException" could be thrown; "objectValue" is nullable here. Why is this an issue? — 8 minutes ago — L189
Bug — Major — Open — Not assigned — 10min effort — Comment — cert, cwe

thingifier/.../thingifier/api/restapihandlers/RelationshipCreation.java

A "NullPointerException" could be thrown; "relationshipToUse" is nullable here. Why is this an issue? — 8 minutes ago — L117
Bug — Major — Open — Not assigned — 10min effort — Comment — cert, cwe

A "NullPointerException" could be thrown; "relationshipToUse" is nullable here. Why is this an issue? — 8 minutes ago — L131
Bug — Major — Open — Not assigned — 10min effort — Comment — cert, cwe

thingifier/.../thingifier/application/MainImplementation.java

The return value of "format" must be used. Why is this an issue? — 8 minutes ago — L362
Bug — Major — Open — Not assigned — 10min effort — Comment — cert

thingifier/.../sparkhttpmessageHooks/ClearDataPreSparkRequestHook.java

Cast one of the operands of this multiplication operation to a "long". Why is this an issue? — 8 minutes ago — L15
Bug — Minor — Open — Not assigned — 5min effort — Comment — cert, cwe, overflow, sans-top25-risky

thingifier/.../thingifier/htmlgui/RestApiDocumentationGenerator.java

Save and re-use this "Random". Why is this an issue? — 8 minutes ago — L341
Bug — Critical — Open — Not assigned — 5min effort — Comment — owasp-a6

Recommendations For Improving Code

Code Smells -Minor

Many Test classes use old writing conventions for Method visibility of the tests. They all have a 'public' modifier in their declaration, but it is recommended by SonarQube to not add a modifier. This would give the tests the default package visibility, and therefore improve readability (removing redundancies). There are many unused variables and redundant variable declarations should be deleted for better readability. Also, multiple field declarations can be changed to 'Local' and many unused Imports can be removed. Finally, there are multiple simplifiable assertions in the Test classes.

Code Smells - Major

Many blocks of commented code was found and should be deleted for better readability and to avoid cluttered classes. Some 'if' statements are redundant and could be combined/optimized. Some 'if' statements have common parts that could be combined. Some 'catch' blocks could ignore exceptions in some cases.

Bug Fixes

2 instances of String or Boxed Type comparisons (value comparisons) should use the "equals()" method instead of "==". One instance of an operand of a multiplication that should be cast to a 'long'. There are 3 instances of 'NullPointer' exceptions that could be thrown. There is an instance of a function whose return value is ignored. It should be deleted (if useless) or the behavior should be changed. Finally, the Random value should be re-used instead of creating a brand new pseudo-random number every time.