Scale out the test cases

X-BU Partner Integration Engineering

Tao Zhou

Tao.Zhou@emc.com

Abstract:

This paper will introduce a simple way to scale out test cases on many machines. The way to scale out test cases will not require code changes to existing test codes, which using reflection technology to decouple the synchronization code and test cases code.

1. Introduction

For automation testing, there will have more than thousands of test cases. In regression phase, there is always pain to handle tremendous cases in short time: if all test cases run sequentially in separated environments, that will takes more time to complete testing than expected ,but if cases are parallel running, that will require extra effort to synchronize the critical resource requests between the test cases. Is there an easy way to scale out these test cases to parallel execution without resource confliction? The paper will describe how to scale out your test cases in an easy way.

1. The synchronization architecture

How to synchronize the resource request in automation test cases? For the test cases are running on different machines, so it is intuitional to have a server (Figure 1) to synchronize the requests.

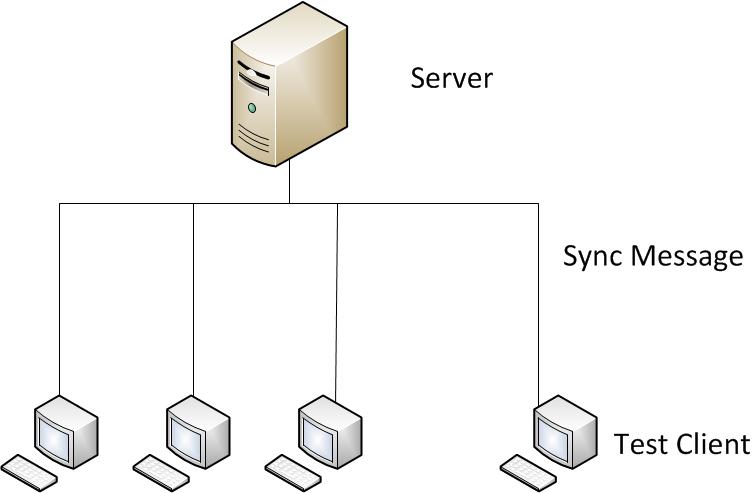


Figure 1

The server side implementation could simple and straightforward, but the difficult part is the client code, for the following reasons:

* The existed test cases are designed without consider the parallel execution cases.
* The testing environment may change frequently.
* Adding the synchronize code in test cases needs extra effect for QA.
* There will heterogeneous test cases, for example, test case A will require resource 1, 2, 3 in different phases, but test case B will only require resource 2 in specified. So when A is using resource 1 and 3 the case B could use resource 2.

Is there a panacea to serve all test cases? Yes, the paper will introduce the method which will easily transfer the single-process test cases to multi-processes test cases.

1. Client side software design

As previous section mentioned, there are many issues for design the synchronize code for client side, in the section, there is a simple scenario to demonstrate how to resolve these issues.

The picture (Figure 2) shows the simplified code which contains a test case of creating a Lun. When several instance of the case run simultaneously, the storage will get more than one request for creating Lun at a moment, then the instances will be failed.

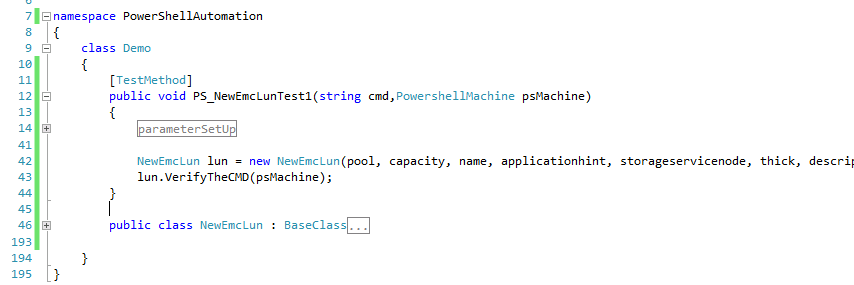


Figure 2

And the following picture (Figure 3) shows the synchronized code added to the test case, now when several instances are running simultaneously, but on storage side, the request for creating Lun is one by one, the order of requesting is synchronized by server, so these instances will be success.

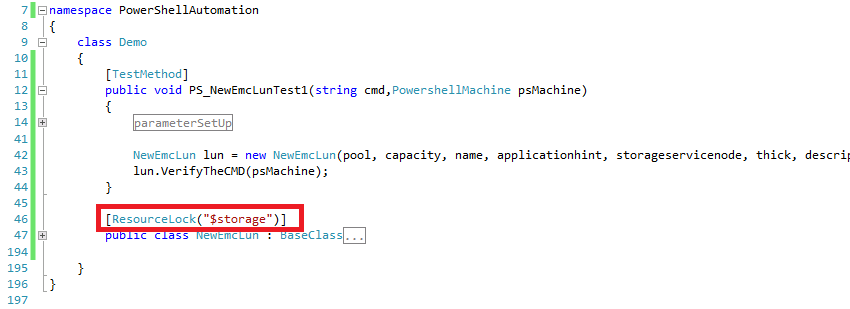


Figure 3

As showing the picture above, there is no extra work for test case code, only work is just add an attribute to the New-EmcLun class. The attribute is a decoupled with test cases, and easily be implemented.

So the attribute class perfectly resolves all the issues mentioned in previous section, but why the attribute class has the magic power?

The picture below (Figure 4) is a typically test cases structure, the test cases call the command classes, and command classes finally call all kinds of CLI via adapter class to execute the command in test cases.

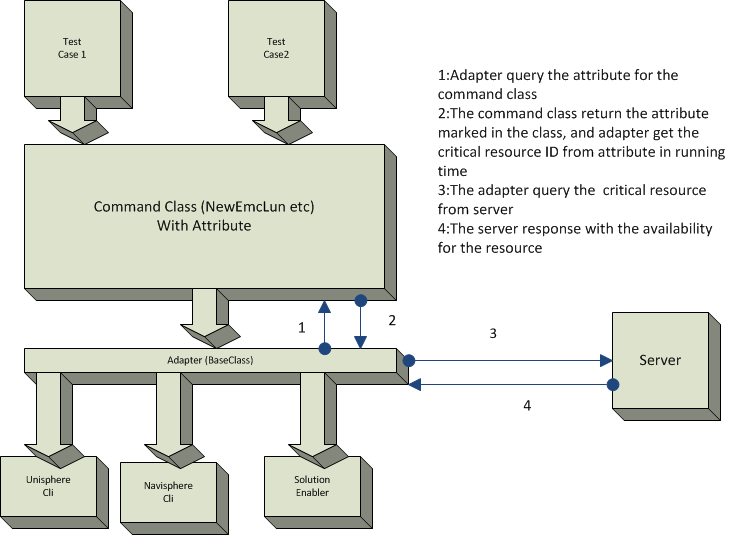


Figure 4

What’s the attribute class do is mark the critical resource for command classes. When the command class calls the adapter class to execute CLI command, then the adapter class queries the attributes of command class then get the critical resource ID in running time environment from attribute values, then query to server to make sure the critical resource is not using by other test cases.

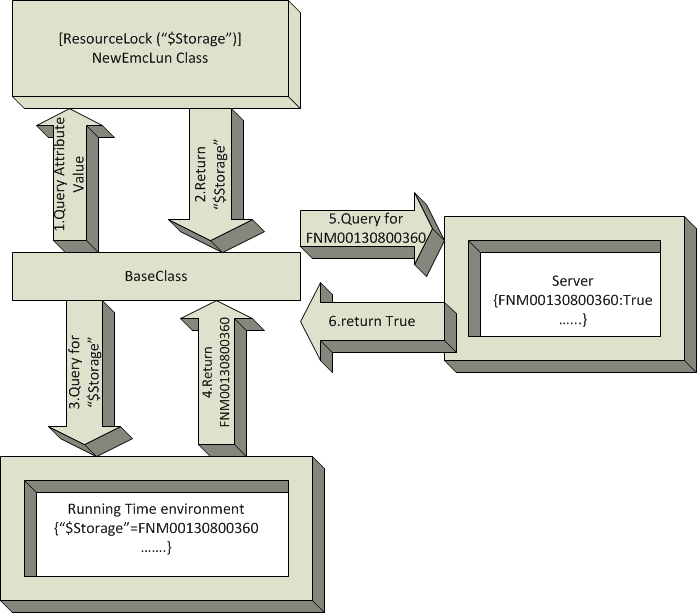


Figure 5

The magic power of the design is that the critical resource ID is get from running time environment, only thing the test cases designer need to know is just setting the attribute of command class. For the same command class, when the environment is different, there will generate different critical environment ID. And the test cases code is decoupled with synchronization code, there will get synchronized test cases without change any test cases code.

1. Server side software design

Server side software design is straightforward and simple, only one concern is resource deadlock. For example if client A request resource 1 and 2, meanwhile client B request 2 and 3, client C request 3 and 1, if resource is assigned separately, the deadlock scenario will happen. The simple way to resolve the issue is to set atomic lock for each resource request, only all requested resource are available, the resource will be acquired.

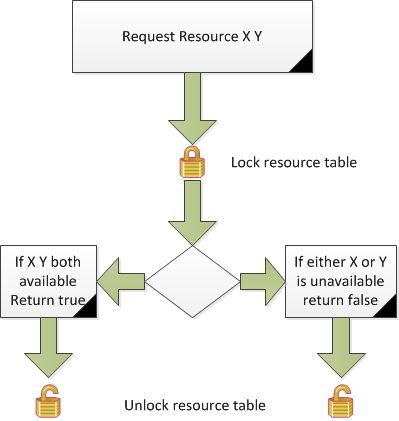


Figure 6

1. Application

To get more strictly testing result of the scale-out ability of the design. There are choosing the most resource consuming cases and all cases requests same resources. There defines two parameters to measure the scale-out ability:

The over head rate: ; is when test cases running on N machines, the sum of time for N machines cost for test cases, is the time with only one machine, the time of running all test cases. The rate is near the value 1 means the system is a scalable system.

The response time rate: ; is the average response time of server (the resource [Like storage, Host] which test cases are using) when test cases are running on N machine, is the average response time when test cases are running on 1 machine.

Form the picture below, the value is near 1 when N =3 and 2, which means the overhead is very low if test cases are running on 3 machines; the turn high when N=4, which means when 4 clients are requesting the service on server, the server performance is degraded sharply, that may caused by reaching the maximum of server capability. So the increase on N=4 is causes by increasing. So the scalability of the test is depends on the server capability.

Figure 7

1. Summary

This paper introduced an easy implementation of synchronization. With the implementation, the test cases could easily scale out on many machines.