**Dealing with performance issues of TPL**

**Overview:** **TPL**(Task Parallel Library) In our project, we have python scripts running in parallel across multiple stations. But here when we try to run more than one script in parallel, we faced up many milestones. This blog would suggest the best practices for efficient utilization of resources like CPU and memory that improves application’s performance.

**Prominent Problems**

* High CPU utilization.
* Application becomes non responsive.
* Incorrect output from script.
* Script execution terminates abruptly.
* Memory leakage.

**Prerequisites to run scripts parallely…**

We have implemented TPL(Task Parallel Library) concept to run multiple scripts at a time.

Task.run() method in TPL runs the script and we delegated an event listener called outputlinereceived() to process the output redirected from the script.

**Solutions to improve performance**

1. **Dispatcher.Priority()** method - for efficient CPU utilization,

We can set the dispatcher priority. If we increase the priority of the dispatcher the CPU usage increases and when we decrease the priority of the dispatcher the CPU usage gets decreased.

**DispatcherPriority.Background**

We have used this priority type because it makes the scripts run faster and at the same time to make the application more **responsive**.

1. **ManualRestEvent.wait ()** method- to process output accurately.

We need to process the output redirected from the script. As we are using TPL, we may not have guarantee that whatever the output is retrieved is processed correctly. So, we need to wait until the retrieved output got processed completely before trying to read the next pipelined output.

**EventWaitHandle.WaitOne ()** method does the above work for us. It is used to check the status of the dispatcher every time. If the dispatcher status is under process then the manualResetEvent is set to wait else it would resume the next task.

1. Use **StringBuilder** objectrather than **String** to write the output to the output screen every time which reduces the **memory leakage** issue.
2. Use **StringBuilder.Append()** method to append the script output instead of string concatenations using ‘+’ operator.
3. **CustomTaskScheduler –** Limits Concurrency

We can specify the maximum number of sub tasks to be created when you schedule a main Task which helps us to configure the task creation count depending on the system configuration thereby we can reduce CPU overhead.

1. **Lock variable**

When working with TPL, there may be some dependent tasks running in parallel.

If we don’t handle them perfectly, there could be loss of data or data processed might be incorrect.

Lock variable ensures resources common to multiple tasks be accessed in synchronous fashion.