PowerShell in Parallel

Abstract:

(jump to TOC)

This document shows how to parallelize the appropriate parts of your PowerShell scripts in order to greatly increase performance over normal, serial scripts.

Warning:

(jump to TOC)

Be sure to check the revision history of this document to ensure you have the most recent version, and to see what updates, additions, and corrections have been made.

Document Revision and History:

(jump to TOC)

date	version	author	notes
2015.12.23	1.0	Daniel L. Benway	• initial release
2015.12.24	1.0.1	Daniel L. Benway	• corrected the footer

Special Thanks:

(jump to TOC)

Special thanks to Mr. Boe Prox, of http://www.learn-powershell.net. It's his site that got me up and running with PowerShell parallelization, and it's his module and notes that I draw from heavily in this PDF. I've taken some of the work he's done and boiled it down into a small, simple, bite-sized chunk that is a little easier for PowerShell beginners and intermediates to work with.

Freeware License and Disclaimer:

(jump to TOC)

About the Author:

(jump to TOC)

Daniel L. Benway

Systems & Network Administrator / Engineer
BSc CS, MCSE (NT4, 2000), MCTS (SCCM 2012), CCNA (2.0), Network+, CLP (AD R4)

LinkedIn http://www.LinkedIn.com/in/DanielLBenway

Blog http://www.DanielLBenway.net

twitter @Daniel_L_Benway

Table of Contents:

Getting Started:

(jump to TOC)

The majority of my work on parallelizing PowerShell comes from Mr. Boe Prox, from these two articles:

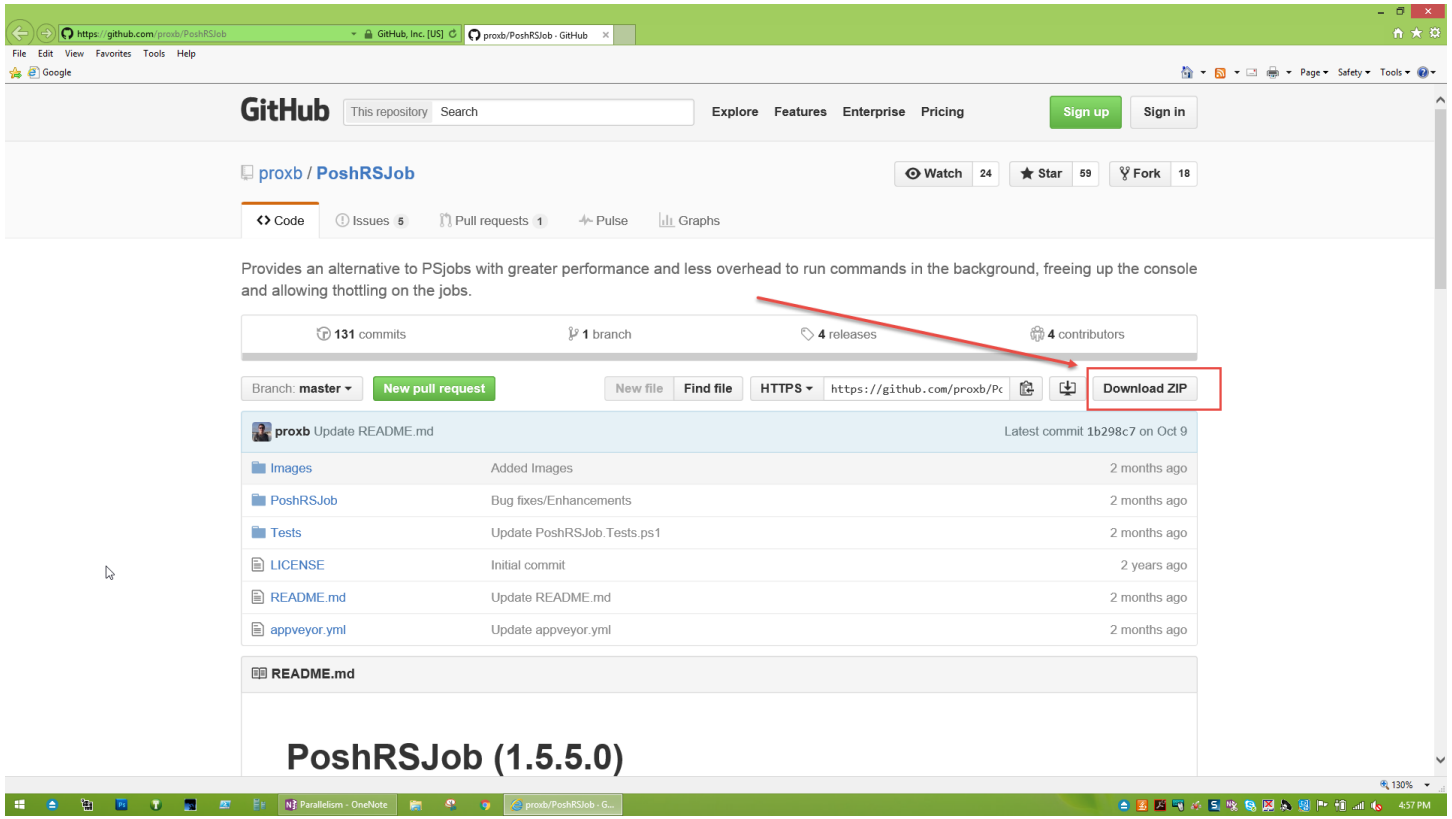
> http://learn-powershell.net/2015/03/31/introducing-poshrsjob-as-an-alternative-to-powershell-jobs/
> Introducing PoshRSJob as an Alternative to PowerShell Jobs
>
> http://learn-powershell.net/2015/04/19/latest-updates-to-poshrsjob/
> Latest Updates to PoshRSJob

Downloading and Importing the Module:

(jump to TOC)

First, go to GitHub and download Mr. Prox's PoshRSJob Module:

https://github.com/proxb/PoshRSJob



After you download and extract the zip to a directory of your choice, unblock the files and import the module, similarly to this:

```
Get-ChildItem -recurse C:\Install_Files\GitHub\PoshRSJob | Unblock-File
Import-Module -name C:\Install_Files\GitHub\PoshRSJob\PoshRSJob -verbose
```

Simple Demonstration Script:

(jump to TOC)

Next, experiment with this simple demonstration script until you get a feel for what parallelization is all about:

```
cls
#################################################################################################################
Get-ChildItem -recurse C:\Install_Files\GitHub\PoshRSJob  | Unblock-File
Import-Module -name C:\Install_Files\GitHub\PoshRSJob\PoshRSJob -verbose
#################################################################################################################
1..20 | Start-RSJob -throttle 5 -name {$_} -scriptBlock {Start-Sleep -Seconds (Get-Random (1..7))} | Wait-RSJob
```

Here's the output:

```
VERBOSE: Loading module from path 'C:\Install_Files\GitHub\PoshRSJob\PoshRSJob\PoshRSJob.psd1'.
VERBOSE: Loading module from path 'C:\Install_Files\GitHub\PoshRSJob\PoshRSJob\PoshRSJob.psm1'.
VERBOSE: Importing function 'Get-RSJob'.
VERBOSE: Importing function 'Receive-RSJob'.
VERBOSE: Importing function 'Remove-RSJob'.
VERBOSE: Importing function 'Start-RSJob'.
VERBOSE: Importing function 'Stop-RSJob'.
VERBOSE: Importing function 'Wait-RSJob'.
VERBOSE: Importing alias 'gsj'.
VERBOSE: Importing alias 'rmsj'.
VERBOSE: Importing alias 'rsj'.
VERBOSE: Importing alias 'spsj'.
VERBOSE: Importing alias 'ssj'.
VERBOSE: Importing alias 'wsj'.

Id      Name            State         HasMoreData   HasErrors    Command
--      ----            -----         -----------   ---------    -------
1       1               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
5       5               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
2       2               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
3       3               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
7       7               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
10      10              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
4       4               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
9       9               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
6       6               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
8       8               Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
14      14              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
12      12              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
11      11              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
16      16              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
13      13              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
15      15              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
18      18              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
17      17              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
20      20              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))
19      19              Completed     False         False        Start-Sleep -Seconds (Get-Random (1..7))


PS C:\Users\D.L.Benway>
```

The key thing to see here is that the contents of the scriptBlock are what gets run in parallel.

The output shows that only 5 processes were allowed to run concurrently, the processes were started in order of 1, 2, 3 … 20, as soon as one process completed another one was started so that 5 were always running (until the queue was emptied), and the completion order happened to be 1, 5, 2, 3, 7, 10, 4, 9, 6, 8, 14, 12, 11, 16, 13, 15, 18, 17, 20, 19 based on how long each process was doing its work (which was just a simple sleep statement).

Simple Production Script:

(jump to TOC)

Once you've played around with the simple demonstration script, move onto something almost as simple but a bit more real:

```
cls
#############################################################################################################
Get-ChildItem -recurse C:\Install_Files\GitHub\PoshRSJob | Unblock-File
Import-Module -name C:\Install Files\GitHub\PoshRSJob\PoshRSJob -verbose
#############################################################################################################
import-module activeDirectory
Get-Date -format "yyyy.MM.dd.HHmmss" > "C:\Temp\ParallelizationTestOutput.txt"
$WindowsServers = $null
$WindowsServers = Get-ADComputer -Filter {OperatingSystem -Like "*Windows*Server*"} -Property Name
If ($WindowsServers -ne $null)
    {
    $WindowsServers | Start-RSJob -throttle 2 -name {$_.Name} -scriptBlock {
        If (Test-Connection -ComputerName $_.Name -Count 2 -Delay 1 -Quiet)
            {
            $NICs = $null
            $NICs = Get-WmiObject -Class Win32_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -ComputerName $_.Name
            If ($NICs -eq $null)
                {
                $outputLine = $ .Name + " had a WMI connection error (or it had no IP NICs, which is unlikely)."
                $outputLine >> "C:\Temp\ParallelizationTestOutput.txt"
                }
            Else
                {
                $NICNumber = 0
                ForEach($NIC in $NICs)
                    {
                    $NICNumber = $NICNumber + 1
                    $outputLine = $_.Name + " " + $NICNumber + " " + $NIC.MACAddress + " " + $NIC.Description + " " + $NIC.IPAddress
                    $outputLine >> "C:\Temp\ParallelizationTestOutput.txt"
                    }
                }
            }
        Else
            {
            $outputLine = $_.Name + " did not respond to PING."
            $outputLine >> "C:\Temp\ParallelizationTestOutput.txt"
            }
        } | Wait-RSJob
    }
Else
    {
    "Get-ADComputer failed." >> "C:\Temp\ParallelizationTestOutput.txt"
    }
```

Here's the screen output:

```
VERBOSE: Loading module from path 'C:\Install_Files\GitHub\PoshRSJob\PoshRSJob\PoshRSJob.psd1'.
VERBOSE: Loading module from path 'C:\Install_Files\GitHub\PoshRSJob\PoshRSJob\PoshRSJob.psm1'.
VERBOSE: Importing function 'Get-RSJob'.
VERBOSE: Importing function 'Receive-RSJob'.
VERBOSE: Importing function 'Remove-RSJob'.
VERBOSE: Importing function 'Start-RSJob'.
VERBOSE: Importing function 'Stop-RSJob'.
VERBOSE: Importing function 'Wait-RSJob'.
VERBOSE: Importing alias 'gsj'.
VERBOSE: Importing alias 'rmsj'.
VERBOSE: Importing alias 'rsj'.
VERBOSE: Importing alias 'spsj'.
VERBOSE: Importing alias 'ssj'.
VERBOSE: Importing alias 'wsj'.

Id      Name            State         HasMoreData   HasErrors    Command
--      ----            -----         -----------   ---------    -------
1       DLBT-ADDS1      Completed     False         False        ...
2       DLBT-ADDS2      Completed     False         False        ...
3       DLBT-PKI2       Completed     False         False        ...


PS C:\Users\D.L.Benway>
```

The script pulled all server names from AD all at once, then it contacted the WMI on 2 servers at a time until each server's WMI had been queried.

In a large production environment the throttle would be set higher than 2, but for a small test lab with only 3 servers 2 was a good value for demonstration.

Here's the file output:

```
2015.12.23.115158
DLBT-ADDS1 1 00:15:5D:3D:E6:1E Microsoft Hyper-V Network Adapter 10.0.1.100
DLBT-ADDS2 1 00:15:5D:3D:E6:1F Microsoft Hyper-V Network Adapter 172.16.1.100
DLBT-PKI2 1 00:15:5D:3D:E6:20 Microsoft Hyper-V Network Adapter 172.16.1.101
```

This is the actual output file generated by the script.

Wrap-Up:

(jump to TOC)

Hopefully, these two simple demonstration scripts will give you an idea of how to work with PowerShell parallelization. It's really not hard! Mr. Boe Prox has done a great job with his PoshRSJob module, which has made this all pretty simple and straightforward. Please visit his site, http://www.learn-powershell.net, to learn more.