

Writing a Resource Based (Adaptive) Server Agent

Technical Note

UPDATED: 28 July 2023

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

#1 Load Balancer in Price/Performance, 360 Central, 360 Vision, Chef, Chef (and design), Chef Habitat, Chef Infra, Code Can (and design), Compliance at Velocity, Corticon, Corticon.js, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Driving Network Visibility, Flowmon, Inspec, Ipswitch, iMacros, K (stylized), Kemp, Kemp (and design), Kendo UI, Kinvey, LoadMaster, MessageWay, MOVEit, NativeChat, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), Sitefinity Insight, SpeedScript, Stylized Design (Arrow/3D Box logo), Stylized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Workstation, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Classic, Fiddler Everywhere, Fiddler Jam, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, InstaRelinker, JustAssembly, JustDecompile, JustMock, KendoReact, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the NOTICE.txt or Release Notes – Third-Party Acknowledgements file applicable to a particular Progress product/hosted service offering release for any related required third-party acknowledgements.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

Table of Contents

1 Document Purpose 4

2 Custom Server Agents and Adaptive Scheduling 5

3 Sample Linux Agent Script 9

4 Sample Windows Agent Script 12

Last Updated Date 14

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

1 Document Purpose

This document outlines how to create your own load balancing server agent to use in conjunction with the resource based (adaptive) Virtual Service (VS) scheduling method, which is one of the VS **Scheduling Method** choices in the VS **Standard Options**. This document focuses primarily on the requirements for creating and configuring such agents on arbitrary server operating systems (for example, the protocol used by the agent to communicate load values to the LoadMaster), and less on the mechanics of generating a load value using specific Operating System (OS) utilities. Nevertheless, this document presents simple examples of implementing and deploying a server agent on both a Windows and Linux system.

In the remainder of this document, the resource based (adaptive) scheduling method is referred to simply as adaptive scheduling.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

2 Custom Server Agents and Adaptive Scheduling

Adaptive scheduling requires the following on every Real Server (RS) in the VS:

- There is an HTTP server running on the RS and the endpoint for this HTTPS server is accessible to the LoadMaster.
- There is a server agent installed and running on the RS that does the following:
    - Periodically queries the RS operating system for whatever system load statistics are available and desired.
    - Uses the load values obtained from the operating system to arrive at an integer value.
    - Places the integer value in a file underneath the root directory of the HTTP server running on the RS. This is the file that the LoadMaster requests using an HTTP GET command. It must contain a single integer value and nothing else.

The server agent and an appropriately configured HTTP server must be running on all RSs in a VS before selecting adaptive scheduling on the VS.

On the LoadMaster side:

- Once adaptive scheduling is selected on a VS, the LoadMaster periodically attempts to retrieve the file populated by the server agent from each RS, with the integer load value.
    - Until the LoadMaster successfully retrieves an integer load value from a Real Server, it is considered unavailable.
- Once an integer load value is retrieved, the LoadMaster modifies the status of the Real Server by interpreting the integer retrieved according to the table below.

Agent Response	Action Taken	Description
No HTTP Response	**Dynamic Weight = 0**	If the LoadMaster gets no response when it attempts to get the file containing the load value, then the Real Server is assumed to be unavailable and it is marked down. This is done by setting the server's dynamic weightto 0 so that no new connections are sent

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

		to it. Currently open connections are not affected. This typically happens when either the Real Server is unavailable, or the required HTTP server is not responding to the =query from the LoadMaster.
File does not exist	**Dynamic Weight = 100**	If the HTTP server is running on the Real Server and responds to the LoadMaster's query, it may send a response that indicates the file requested does not exist. In this case, the server is assumed to be available, it is marked up, and the server dynamic weight is set to 100.
File is empty	**Dynamic Weight = 100**	If the HTTP server is running on the Real Server and responds to the LoadMaster's query with an empty file (the file contains no data at all), the server is assumed to be available. It is then marked up and the server dynamic weight is set to 100.
File contains a non-integer value	**Dynamic Weight = 100**	If the file returned by the HTTP server running on the Real Server contains a non-integer value, then the server is assumed to be available. It is then marked up and the server dynamic weight is set to 100.
< 0	**Dynamic Weight = 100**	If the file returned by the HTTP server running on the Real Server contains a negative integer value (for example, -1), then the server is assumed to be available, it is marked up, and the server dynamic weight is set to 100.
0	**Switch to the round robin scheduling method**	If the file returned by the HTTP server running on the Real Server contains an integer value of 0 (zero), this forces the LoadMaster to temporarily fall back to using the round robin scheduling method until the server agent returns a value other than 0 on a subsequent query. This return value is intended to be used in a situation where the server agent code cannot for some reason arrive at a valid integer value to return to the LoadMaster. This might happen if, for example, the server is not responding to the queries from the agent for load values.
1-100	**Dynamic Weight set appropriately**	If the file returned by the HTTP server running on the Real Server contains an integer value between 1 and 100, this is interpreted as a percentage of fully loaded (1=lowest load, 100 = fully loaded). The LoadMaster sets the dynamic weight appropriately: the higher the number returned by the server agent, the lower that the dynamic weight of the server is set to.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

102	**Dynamic Weight = 0 plus optional connection drop**	If the file returned by the HTTP server running on the Real Server contains an integer value of 102, this means essentially the same as returning 101. The only difference is that the LoadMaster will also drop all currently open connections if the **Drop connections on RS failure** option is set on the associated VS.
> 102	**Dynamic Weight = 0**	A return of 103 or greater is treated the same as a return of 101.

On any Real Server operating system, a straightforward manner in which to implement a server agent is to write a shell script or executable program that runs periodically using a standard system scheduler, like **cron** on Linux systems or the **Task Scheduler** on Windows systems. This script would use standard utilities to generate the load value and place it in a file under the HTTP server root.

Such a script could be as simple as this Linux bash script.:

```
#! /bin/bash
top –ll | awk ‘/CPU Usage/ {printf “%d\n”, $7} > /<path>/load
```

The above uses the top command and a short **awk** script to copy the current CPU load value in a file – the **<path>** above specifies the path to the root of an HTTP server on the Real Server. The file name ‘load’ is the default file name retrieved by the LoadMaster. [This and other adaptive load balancing options are located in the User Interface (UI) under **Rules & Checking > Check Parameters > Adaptive Parameters**.]

Alternatively, you could instead write a Common Gateway Interface (CGI) program or any other program that the HTTP server can be configured to run when the LoadMaster sends a GET for the required file to the HTTP server. This program would generate the integer load value when the LoadMaster requests the file from the server and the HTTP server would return it to the LoadMaster.

There are advantages and disadvantages to either approach. For example:

- Using a static file populated using a cron job is a simple and easy-to-maintain option, but also means that the data in the file could be stale when it is accessed by the LoadMaster. The interval at which the load value is generated should be based on the known performance profile of the server.
- Using a program that runs each time the LoadMaster checks performance means that you will get the most accurate data, but the server agent itself could contribute to system load if CPU, Disk I/O, and network performance were all being continually checked by the server agent.

The approach you choose most likely depends on the needs of your configuration and the tools you typically use.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

In terms of internal processing, a server agent can be as complicated as you like, depending on your application. If your application is completely CPU-intensive, then a simple script like the two-line bash script above may be enough. But, if your application also generates significant disk I/O or network-traffic, you will probably want to consider the load on those resources in addition to CPU usage to get a more detailed view of server performance.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

3 Sample Linux Agent Script

The following script is an example of gathering load information on CPU and memory, and then combining them into a single integer value that the LoadMaster can use for adaptive scheduling. It is written as a bash shell script and is intended to be run on a Linux server using a privileged crontab file. See the Linux manual pages for bash, crontab, and the commands used in the script to gather the required statistics.

This script is intended solely as an example, although it could be used in a demonstration or proof of concept of how to configure adaptive scheduling.

```
#! /bin/bash
#
# Very simple adaptive server agent program, intended to be run via a root
# or similarly privileged crontab(1) entry to periodically populate
# the file LoadMaster expects to be available via HTTP on the Real Server.

#### DEFAULTS
# This is the default filename expected by LoadMaster. This setting must
# match what's on "Rules & Checking > Adaptive Parameters" in the LM WUI.
LOAD_FILE_PATH=/load
# This is the default doc root of apache2 on an ubuntu server, and where
# we'll place the 'load' file defined above for LoadMaster to GET.
DOCROOT=/var/www/html
# Define the weights to be used for CPU and Memory.
# Change these numbers to indicate the percentage weight that each
# statistic should carry when figuring the overall load value to
# return to LoadMaster. By default, these are set to 0.25 for CPU and 0.75
# for Memory, so the effect of the weighting is apparent. Weights must
# add up to 1.
CPUweight=0.25
MEMweight=0.75
#### GET MEMORY STATS
# Get memory usage from the free(1) command using a simple awk command and
# calculate the percentage of memory used. Since the shell does integer
# arithmetic only, we'll use bc(1) to do the math.
MEMtotal=`free | awk '/Mem:/ {print $2}'`
MEMstatus=$?
MEMused=`free | awk '/Mem:/ {print $3}'`
MEMload=`echo "scale=3;( $MEMused / $MEMtotal ) * 100" | bc`
# Round the answer to the nearest integer
MEMloadrnd=`echo "$MEMload + 0.5" | bc`
MEMloadrnd=`echo "scale=0; $MEMloadrnd/1" | bc`
#### GET CPU LOAD
# Note: Some version of top(1) report avg. CPU usage since time of last
# reboot in the first iteration, so we'll take the second iteration value.
CPUidletop=`top -b -n2 -d.2 | awk '/%Cpu/ {print $8}'`
CPUidle=`echo $CPUidletop | cut -d" " -f2`
CPUload=`echo "scale=3;100 - $CPUidle" | bc`
CPUstatus=$?
# Round the answer to the nearest integer
CPUloadrnd=`echo "$CPUload + 0.5" | bc`
```


© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

```
CPUloadrnd=`echo "scale=0; $CPUloadrnd/1" | bc`
#### CHECK IF STATS ARE A VALID % (between 1 and 100)
# VALID_STATS=1 means stats are valid.
# if stats are not valid, we want to return 0 to LM to disable adaptive
# LB and use round robin until a different value is received.
VALID_STATS=1
if [ $MEMloadrnd -lt 1 -o $MEMloadrnd -gt 100 -o $CPUloadrnd -lt 1 \
       -o $CPUloadrnd -gt 100 ]
then
       VALID_STATS=0
       RETURNED_LOAD=0
fi
#### CALCULATE THE WEGHTED AVERAGE OF $CPUload AND $MEMload.
# Only do this if stats were valid - i.e., $VALID_STATS=1.
if [ $VALID_STATS -eq 1 ]
then
       RSload=`echo "( $CPUweight * $CPUloadrnd ) + \
              ( $MEMweight * $MEMloadrnd )" | bc`
       RSloadrnd=`echo "$RSload + 0.5" | bc`
       RSloadrnd=`echo "scale=0; $RSloadrnd/1" | bc`
       RETURNED_LOAD=$RSloadrnd
fi
#### DETERMINE IF WE SHOULD RETURN 101 OR 102.
# Returning 101 or 102 in the case of a highly loaded system is somewhat
# a matter of policy. In this example, we'll return 101 (take the server
# out of rotation) when stats are valid AND the aggregated load value
# is above 90%, to allow more resources to become available. If the
# aggregate load goes above 96%, we'll return 102 (which takes the server
# out of rotation AND drops all current connections). Presumably, once
# more resources are available, load values will decrease, and the
# server agent will return a lower value on a subsequent run.
if [ $VALID_STATS -eq 1 -a $RSloadrnd -gt 90 ]
then
       RETURNED_LOAD=101
elif [ $VALID_STATS -eq 1 -a $RSloadrnd -gt 96 ]
then
       RETURNED_LOAD=102
fi
#### PUT THE LOAD VALUE IN THE FILE.
# If stats were not valid above, then return 0 -- this disables adaptive
# scheduling and uses round robin instead; else, return the computed load.
if [ $VALID_STATS -eq 0 ]
then
       RETURNED_LOAD=0
fi
echo $RETURNED_LOAD > ${DOCROOT}${LOAD_FILE_PATH}
#### PRINT EVERYTHING TO STDOUT FOR DEBUGGING
# This allows for running the script from the command line to test in
# your environment -- or, if the script is run via cron, cron will
# capture the output and email it to you so you can debug any issues.
# Once you're happy the script is working properly, comment these lines
# out to avoid being emailed output by cron every time the script runs.
echo RETURNED_LOAD = $RETURNED_LOAD
echo RSload = $RSload
echo RSloadrnd = $RSloadrnd
echo MEMweight = $MEMweight
echo MEMtotal = $MEMtotal
echo MEMused = $MEMused
echo MEMload = $MEMload
echo MEMloadrnd = $MEMloadrnd
echo MEMstatus = $MEMstatus
```


© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

```
echo CPUweight = $CPUweight
echo CPUidle = $CPUidle
echo CPUload = $CPUload
echo CPUloadrnd = $CPUloadrnd
echo CPUstatus = $CPUstatus
#### EXIT with appropriate status
if [ $VALID_STATS -eq 1 -a $MEMstatus -eq 0 -a $CPUstatus -eq 0 ]
then
# success
       exit 0
else
# error
       exit 1
fi
#### EOF
```


© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

4 Sample Windows Agent Script

The following script is similar in function to the Linux script presented in the previous section, but is written in Windows PowerShell rather than the Linux bash shell and uses Windows Management Instrumentation (WMI) commands to do the job of gathering performance data. The script can be set to run periodically as a Windows Task and assumes that there is an already running IIS server on the Windows Real Server that will respond to the GET request from the LoadMaster for the load value file created by the server agent.

This script is intended solely as an example, although it could be used in a demonstration or proof of concept of how to configure adaptive scheduling.

```
#
# Very simple adaptive server agent program, intended to be run via a
# Task Scheduler Task with system level privileges, to periodically
# populate the file LoadMaster expects to be available via HTTP on the
# Real Server.
#### DEFAULTS
# This is the default filename expected by LoadMaster. This setting must
# match what's on "Rules & Checking > Adaptive Parameters" in the LM WUI.
$LOAD_FILE_PATH="\load"
# This is the default root of a Windows IIS server. Change this to match
# the location where your HTTP server will require the load file (above)
# to be placed so that LoadMaster can access it using an HTTP GET request.
$DOCROOT="\inetpub\wwwroot"
# Define the weights to be used for CPU and Memory.
# Change these numbers to indicate the percentage weight that each
# statistic should carry when figuring the overall load value to
# return to LoadMaster. By default, these are set to 0.25 for CPU and 0.75
# for Memory, so the effect of the weighting is apparent. Weights must
# add up to 1.
$CPUweight=0.25
$MEMweight=0.75
#### GET MEMORY STATS
[int]$MEMtotal=(Get-WmiObject -Class win32_
operatingsystem).TotalVisibleMemorySize
[int]$MEMfree=(Get-WmiObject -Class win32_operatingsystem).FreePhysicalMemory
[int]$MEMused=$MEMtotal - $MEMfree
$MEMloadpct=($MEMused / $MEMtotal).tostring("P")
[int]$MEMload=($MEMused / $MEMtotal * 100)
#### GET CPU LOAD
$CPUload=Get-WmiObject win32_processor | select -Expand LoadPercentage
$VALID_STATS=1
$RETURNED_LOAD=1
if ( $MEMload -lt 1 -or $MEMload -gt 100 -or $CPUload -lt 1 -or $CPUload -gt
100 )
{
```


© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

```
        $VALID_STATS=0
        $RETURNED_LOAD=0
}
#### CALCULATE THE WEGHTED AVERAGE OF $CPUload AND $MEMload.
# Only do this if stats were valid - i.e., $VALID_STATS=1.
if ( $VALID_STATS -eq 1 )
{
        [int]$RSload=( $CPUweight * $CPUload ) + ( $MEMweight * $MEMload )
        $RETURNED_LOAD=$RSload
}
#### DETERMINE IF WE SHOULD RETURN 101 OR 102.
# Returning 101 or 102 in the case of a highly loaded system is somewhat
# a matter of policy. In this example, we'll return 101 (take the server
# out of rotation) when stats are valid AND the aggregated load value
# is above 90%, to allow more resources to become available. If the
# aggregate load goes above 96%, we'll return 102 (which takes the server
# out of rotation AND drops all current connections). Presumably, once
# more resources are available, load values will decrease, and the
# server agent will return a lower value on a subsequent run.
if ( $VALID_STATS -eq 1 -and $RSload -gt 90 )
{
        $RETURNED_LOAD=101
}
elseif ( $VALID_STATS -eq 1 -and $RSload -gt 96 )
{
        $RETURNED_LOAD=102
}
#### PUT THE LOAD VALUE IN THE FILE.
# If stats were not valid above, then return 0 -- this disables adaptive
# scheduling and uses round robin instead; else, return the rounded load.
if ( $VALID_STATS -eq 0 )
{
        RETURNED_LOAD=0
}
echo $RETURNED_LOAD > $DOCROOT$LOAD_FILE_PATH
#### PRINT EVERYTHING TO STDOUT FOR DEBUGGING
# This allows for running the script from the command line to test in
# your environment. Once you're happy the script is working properly,
# comment these lines out.
echo "VALID_STATS = $VALID_STATS"
echo "RETURNED_LOAD = $RETURNED_LOAD"
echo "MEMload = $MEMload"
echo "MEMloadpct = $MEMloadpct"
echo "CPUload = $CPUload"
#### EXIT with appropriate status
if ( $VALID_STATS -eq 1 )
{
# success
        exit 0
}
else
{
# invalid stats -- error
        exit 1
}
#### EOF
```


© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

Last Updated Date

This document was last updated on 28 July 2023.

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.