

Using Serviceguard Node Capacities and Package Weights

Table of contents	
Introduction	2
Scope.....	2
Overview	2
Node Capacity parameters.....	3
Weight parameters	3
Case-insensitive parameters	4
Using a default weight	4
Meaning of Capacities and Weights	5
package_limit capacity	6
Use case 1: Limiting number of packages on a node	6
Use case 2: Limiting package load/weight on a node	7
Solution 1	7
Solution 2	9
Solution 3	10
Use case 3: Using multiple Capacities or Weights.....	11
Using Serviceguard Manager.....	13
Use case 4: Using Weights with Package Dependencies	14
For more information.....	15

Introduction

Currently, when selecting a node on which to place a package, Serviceguard does not take into account the resource capacity of the node or the amount of node resources that the package requires. All packages are assumed to use the same amount of the node's resources and all nodes are assumed to have infinite capacity for running packages. For example, say that you have a two node cluster, consisting of a large system and a smaller system. You would like all packages to be able to run on the large system at the same time. However, if the large node fails, you would like only the critical packages to run on the smaller system. Currently, there is no mechanism for configuring Serviceguard to achieve this behavior.

Serviceguard's new node capacity and package weight feature will address this problem. Nodes can be configured to support a certain capacity. Packages can be configured to have a certain weight. Serviceguard will take into account the weight of the package and the capacity of the node when selecting a node to place the package.

Scope

This document describes how you can configure and use the node capacity and package weight feature. It provides an overview of the feature and several use cases. For a complete description of rules and restrictions, please refer to the manpage for the `cmquerycl(1m)` and `cmmakepkg(1m)` commands. For further discussion see the *Managing Serviceguard* manual.

Overview

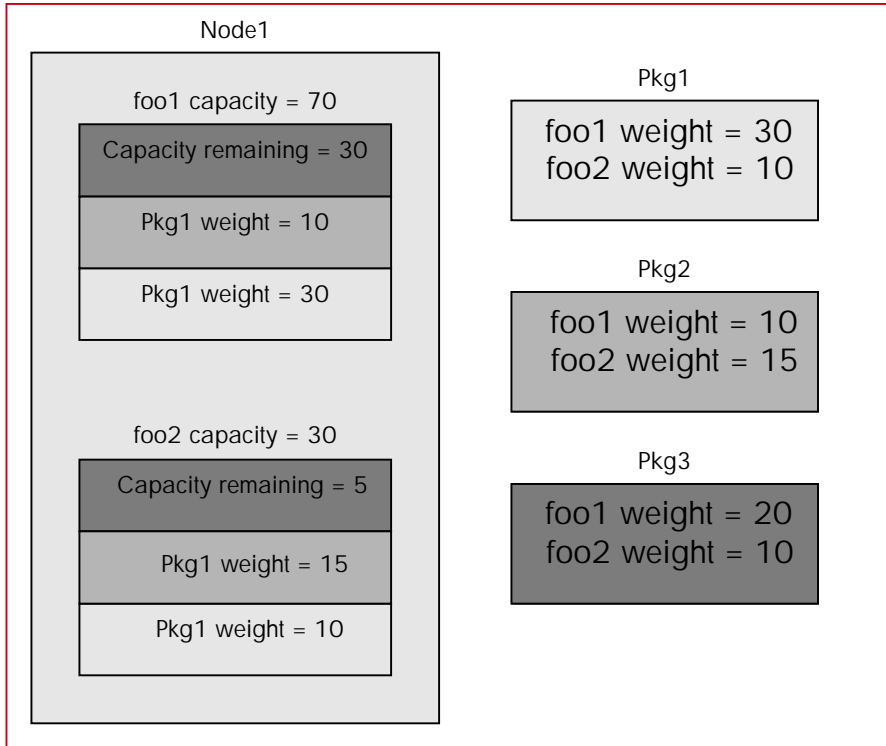
Serviceguard's new node capacity and package weight feature allows you to configure the nodes to support a certain capacity and packages to have a certain weight. The package weight is the amount of node capacity that the package uses. The package weight, the capacity of the nodes configured to run the package, and the priorities and package dependencies determine where the packages should be placed.

A package can have multiple weights and a node can have multiple capacities. This allows you to identify and limit different types of resource on a given node, and to quantify the usage of each of these types of resource by a given package. Each weight/capacity pair has a name; you can define up to four names (that is, four types of capacity and corresponding weight) per cluster.

If any capacity is defined for a node, Serviceguard will never permit the combined weight of the packages running on the node to exceed the capacity. If a package fails over, it can move only to a node that has capacity for it. If a package has `same_node` dependencies on other packages, the node must have capacity for both the package and its dependencies. A lower-priority package can be halted to make room for a higher-priority package.

Figure 1 below shows a node and three packages with `pkg1` and `pkg2` running on the node. Node1 has a `foo1` capacity of 70 and a `foo2` capacity of 30. The boxes on the right of the drawing show the weight for each capacity for `pkg1`, `pkg2`, and `pkg3`. The combined weight of `pkg1` and `pkg2` is 40 for the `foo1` capacity and 25 for the `foo2` capacity. Therefore, node1 has 30 of `foo1` capacity and 5 of `foo2` capacity available to run other packages. This means that `pkg3` cannot run on node1; while the node has sufficient `foo1` capacity remaining to run `pkg3` (which has `foo1` weight of 20), it does not have enough `foo2` capacity (the package needs 10 and the node has only 5 to spare).

Figure 1: Capacity and weight example



Node Capacity parameters

The node parameters, `capacity_name` and `capacity_value`, defined in the cluster configuration file, are used to define a node capacity. The `capacity_name` sets the name and the `capacity_value` sets the limit for that capacity for the node.

If you do not specify a capacity for a given node, Serviceguard assumes the capacity is infinite on that node.

See Table 1 below for an example.

Weight parameters

There are two methods for defining weight for packages:

1. You can define weight for a specific package by means of the `weight_name` and `weight_value` parameters in the package configuration file.
 - The `weight_name` sets the name of the weight.
Note: A package weight corresponds to a node capacity of the same name. This `weight_name` must exactly match the corresponding `capacity_name` for the node.
 - The `weight_value` sets capacity usage for that capacity for the package.
 - See the following Table 1 for an example.

2. You can define a cluster-wide default weight for packages by means of the `weight_name` and `weight_default` parameters in the cluster configuration file.
 - The `weight_name` sets the name of the weight.

Note: A package weight corresponds to a node capacity of the same name. This `weight_name` must exactly match the corresponding `capacity_name` for the node.
 - The `weight_default` sets the default weight for all packages that do not have the weight defined in their package configuration file.

Note: A package weight defined in the package configuration file overrides the default weight.
 - See the following Table 2 for an example.
 - See "Using a Default Weight" below for more information.

You cannot define a package weight if you have not defined a capacity of the same name for at least one of the package's configured nodes.

If you do not specify a `weight_name` and `weight_default` corresponding to a capacity in the cluster configuration file, and you also do not specify a `weight_name` and `weight_value` for this capacity in a package's configuration file, Serviceguard assumes that the package is "weightless" for this capacity (that is it has weight value of 0).

Case-insensitive parameters

Note that, as with all cluster and package configuration parameters, the parameters `capacity_name`, `capacity_value`, `weight_name`, `weight_default` and the `weight_value` are case insensitive. This means that "CAPACITY_NAME" is the same as "capacity_name", and so on. But the name of the capacity/weight, the value you assign to the `capacity_name` and `weight_name` parameters, is case sensitive. This means that `weight_name Foo` does not match `capacity_name foo`.

For the node and packages defined in the preceding Figure 1, the following table shows the entries for the capacity definition for node1 in the cluster configuration file, and weight definition for pkg1, pkg2 and pkg3 in the corresponding package configuration files.

Table 1: Using Capacity and Weight parameters

Cluster configuration file:		Package configuration file: weight definition		
Capacity definition for Node1		Pkg1	Pkg2	Pkg3
<code>node_name</code>	<code>node1</code>	<code>package_name pkg1</code>	<code>package_name pkg2</code>	<code>package_name pkg3</code>
...	
<code>capacity_name</code>	<code>foo1</code>	<code>weight_name foo1</code>	<code>weight_name foo1</code>	<code>weight_name foo1</code>
<code>capacity_value</code>	<code>70</code>	<code>weight_value 30</code>	<code>weight_value 10</code>	<code>weight_value 20</code>
<code>capacity_name</code>	<code>foo2</code>	<code>weight_name foo2</code>	<code>weight_name foo2</code>	<code>weight_name foo2</code>
<code>capacity_value</code>	<code>30</code>	<code>weight_value 10</code>	<code>weight_value 15</code>	<code>weight_value 10</code>

Using a default weight

As Table 1 shows, the `weight_value` parameter, defined in the package configuration file, specifies the weight of each package in relation to the corresponding node capacity, defined in the cluster configuration file. But you do not always need to configure the `weight_value`.

There may be cases in which most packages use the same amount of a given resource (this could be quantified, for example, as 10) but some packages use a different amount of that resource (for example 15). You can use the `weight_default` parameter in the cluster configuration file to specify a default weight for all packages that do not have a weight defined in their package

configuration file (in this example, the `weight_default` would be 10). You could then override this default weight for the packages that use a different amount of this resource by specifying their `weight_value` in their package configuration file. In this example you would specify a `weight_value` of 15 for the packages that use a different amount of the resource. For these packages Serviceguard would use the value in the package configuration file (15) to determine the package's usage of this resource. For the other packages, you would not specify a value in the package configuration file, and the value specified by the `weight_default` parameter in the cluster configuration file (10) would be used.

The following table illustrates this use of the `weight_default` parameter. For the node and packages defined in the preceding Figure 1, we define a weight default value of 10 for capacity `foo2` in the cluster configuration file. Serviceguard will use this as the weight for packages that do not define a `foo2 weight_value`. `pkg1` and `pkg3` do not need to define a weight value for the `foo2` capacity because they use the default (10); `pkg2` uses 15 of `foo2` capacity, so this value does need to be specified in `pkg2` package configuration file.

Table 2: Using default weight

Cluster configuration file:		Package configuration file: weight definition					
Weight default definition for foo2 and Capacity definition for Node1		Pkg1	Pkg2	Pkg3			
<code>weight_name</code>	<code>foo2</code>	<code>package_name</code>	<code>pkg1</code>	<code>package_name</code>	<code>pkg2</code>	<code>package_name</code>	<code>pkg3</code>
<code>weight_default</code>	<code>10</code>	<code>...</code>		<code>...</code>		<code>...</code>	
<code>...</code>		<code>weight_name</code>	<code>foo1</code>	<code>weight_name</code>	<code>foo1</code>	<code>weight_name</code>	<code>foo1</code>
		<code>weight_value</code>	<code>30</code>	<code>weight_value</code>	<code>10</code>	<code>weight_value</code>	<code>20</code>
<code>node_name</code>	<code>node1</code>			<code>weight_name</code>	<code>foo2</code>		
<code>...</code>				<code>weight_value</code>	<code>15</code>		
<code>capacity_name</code>	<code>foo1</code>						
<code>capacity_value</code>	<code>70</code>						
<code>capacity_name</code>	<code>foo2</code>						
<code>capacity_value</code>	<code>30</code>						

Meaning of Capacities and Weights

The real-world meanings of the names you assign to node capacities and package weights are outside the scope of Serviceguard. Serviceguard simply ensures that for each capacity configured for a node, the combined weight of packages currently running on that node do not exceed that capacity.

For example, if you define a `capacity_name` and `weight_name` `processor`, and a `capacity_name` and `weight_name` `memory`, and a node has a `processor` capacity of 10 and a `memory` capacity of 1000, Serviceguard ensures that combined `processor` weight of packages currently running on the node does not exceed 10 and the combined `memory` weight of packages does not exceed 1000. But Serviceguard has no knowledge of the real-world meanings of the names `processor` and `memory`; there is no mapping to actual `processor` and `memory` usage. You would get exactly the same results if you used the names `apples` and `oranges`, for example.

package_limit capacity

Serviceguard supports a capacity with the reserved name `package_limit`. This can be used to limit the number of packages that can run on a node. If you use `package_limit`, you cannot define any other capacities for this cluster, and the default weight for all packages is always 1. Note that the capacity `package_limit` is case sensitive and must be entered in lower case. For more information, see the Managing Serviceguard manual.

The following use cases illustrate how the capacity/weight parameters can be used.

Use case 1: Limiting number of packages on a node

Say that you have a cluster with the following specification:

A two node cluster which consists of a large system (node1) and a smaller system (node2).

- You have 5 packages. `pkg1`, `pkg2`, `pkg3`, `pkg4` and `pkg5` with priorities 10, 20, 30, 40, and 50 respectively.
- All packages are configured to run on both nodes.
- All packages consume roughly the same amount of resources and therefore "weigh" the same.
- You would like all packages to be able to run on the large system at the same time. However, if the large node fails, you would like only two packages to run on the smaller system.

You can use the reserved capacity `package_limit` to set a limit on the number of packages that can run on node2 at the same time. In this example, you set the capacity `package_limit` to 2 for node2 and you do not set any limit on node1.

The following table shows the relevant portion of the cluster configuration file for the above specification.

Table 3: Configuration for Use Case 1 example

Cluster configuration file:	CLUSTER_NAME	cluster_11
	...	
	#	
	#	No capacity limit for node1. This node can
	#	run all packages configured to run on the node.
	#	
Node capacity definition	NODE_NAME	node1
	...	
	#	
	#	node2 can run only two packages
	#	
	NODE_NAME	node2
	...	
	CAPACITY_NAME	package_limit
	CAPACITY_VALUE	2

When you use the reserved capacity `package_limit`, Serviceguard sets the default weight to 1 for all packages. So, you do not need to set a weight for any package in the package configuration file.

If node1 fails, the package priority and dependencies determine which packages will run on node2. Serviceguard will give preference to running the higher priority packages. In this example, `pkg1` and `pkg2` are the highest priority packages. Since node2 has capacity for two packages, if `pkg1` and `pkg2` have no dependencies, they will be placed on node2 regardless of which packages were running on this node previously. If `pkg1` has a dependency on `pkg5`, `pkg1` needs `pkg5` in order to

run. So, pkg1 and pkg5 will run on node2. If pkg1 depends on pkg4 and pkg5, pkg1 cannot run on node2 because this node can not run three packages. If this is the case, node2 will run pkg2 and pkg3 which are the next highest priority packages.

Use case 2: Limiting package load/weight on a node

In this example, different packages require different amounts of the system's resources. You need to estimate the overall resource requirement for packages relative to one another and also resource capacity for cluster nodes relative to one another.

Say that you have a cluster with the following specification:

- Two node cluster which consists of a large system (node1) and a smaller system (node2). You estimate that node1 has twice the amount of resources for running packages than node2.
- You have 5 packages: pkg1, pkg2, pkg3, pkg4 and pkg5 with priorities 10, 20, 30, 40, 50 respectively.
- All packages are configured to run on both nodes.
- You estimate that:
 - pkg2, pkg3, pkg4 require the same amount of resources in order to run and therefore “weigh” the same.
 - pkg1 requires twice the amount of resources pkg2 requires.
 - pkg5 requires half the amount of resources pkg2 requires.
- If all packages are running, you would like node1 to carry twice as much package load as node2. If node1 fails, you would like pkg1 to run on node2.

There are many ways to assign capacities and weights to achieve the desired behavior. Three solutions are presented below:

Solution 1

We can assign a weight of 10 to pkg5, which consumes the least resources, a weight of 20 to pkg2, pkg3, and pkg4, and a weight of 40 to pkg1. This means node2 needs a capacity of 40 in order to run pkg1 and node1 will have a capacity of 80. We can use the reserved capacity `package_limit` for this configuration. Since for capacity `package_limit`, the default weight for packages is 1, we need to override the weight for all packages.

The following table shows the relevant portions of the configuration files for the cluster and the packages:

Table 4: Configuration for solution 1

Cluster	CLUSTER_NAME	cluster_21
configuration file:	...	
	#	
	# node1 has capacity value of 80 for package_limit	
	#	
Node capacity	NODE_NAME	node1
definition	...	
	CAPACITY_NAME	package_limit
	CAPACITY_VALUE	80
	#	
	# "node2" has a capacity of 40 for package_limit	
	#	
	NODE_NAME	node2
	...	
	CAPACITY_NAME	package_limit
	CAPACITY_VALUE	40
Package	package_name	pkg1
configuration file	...	
for pkg1:	weight_name	package_limit
	weight_value	40
Weight definition		
Package	package_name	pkg2
configuration file	...	
for pkg2:	weight_name	package_limit
	weight_value	20
Weight definition		
Package	package_name	pkg3
configuration file	...	
for pkg3:	weight_name	package_limit
	weight_value	20
Weight definition		
Package	package_name	pkg4
configuration file	...	
for pkg4:	weight_name	package_limit
	weight_value	20
Weight definition		
Package	package_name	pkg5
configuration file	...	
for pkg5:	weight_name	package_limit
	weight_value	10
Weight definition		

Solution 2

This solution takes advantage of the fact that the default package weight for the reserved capacity `package_limit` is 1. We can assign a weight of 0.5 to `pkg5`, which consumes the least resources, a weight of 1 to `pkg2`, `pkg3`, and `pkg4`, and a weight of 2 to `pkg1`. This means `node2` needs a capacity of 2 in order to run `pkg1`, and `node1` will have a capacity of 4. Since we are using the default package weight for `pkg2`-`pkg4`, we do not need to assign weights to `pkg2`-`pkg4` in the package configuration file. We only need to override the weight for `pkg1` and `pkg5`.

The following table shows portions of the configuration files for the cluster, `pkg1`, and `pkg5`.

Table 5: Configuration for solution 2

Cluster configuration file:	<code>CLUSTER_NAME</code>	<code>cluster_22</code>
	<code>...</code>	
Node capacity definition	<code>#</code>	<code># node1 has capacity value of 4 for package_limit</code>
	<code>#</code>	<code>#</code>
	<code>NODE_NAME</code>	<code>node1</code>
	<code>...</code>	
	<code>CAPACITY_NAME</code>	<code>package_limit</code>
	<code>CAPACITY_VALUE</code>	<code>4</code>
	<code>#</code>	<code># "node2" has capacity value of 2 for package_limit</code>
	<code>#</code>	<code>#</code>
	<code>NODE_NAME</code>	<code>node2</code>
	<code>...</code>	
	<code>CAPACITY_NAME</code>	<code>package_limit</code>
	<code>CAPACITY_VALUE</code>	<code>2</code>
Package configuration file for <code>pkg1</code> :	<code>package_name</code>	<code>pkg1</code>
	<code>...</code>	
Weight definition	<code>weight_name</code>	<code>package_limit</code>
	<code>weight_value</code>	<code>2</code>
Package configuration file for <code>pkg5</code> :	<code>package_name</code>	<code>pkg5</code>
	<code>...</code>	
Weight definition	<code>weight_name</code>	<code>package_limit</code>
	<code>weight_value</code>	<code>0.5</code>

Solution 3

This solution combines solutions 1 and 2. It allows you to assign the same capacity/weight values as solution1 and use a default package weight so you do not need to change configuration files for pkg2-pkg4. You can define your own capacity name, say "node_resources" and define a weight_default of 20 (equal to the required weight of pkg2-pkg4) for that capacity.

The following table shows the relevant portions of the configuration files for the cluster and the packages:

Table 6: Configuration for solution 3

Cluster configuration file:	CLUSTER_NAME	cluster_23
	...	
	#	
Node capacity and weight default definition	# node1 has capacity value of 80 for capacity	
	# node_resources	
	#	
	NODE_NAME	node1
	...	
	CAPACITY_NAME	node_resources
	CAPACITY_VALUE	80
	#	
	# node2 has a capacity of 40 for capacity	
	# node_resources	
	#	
	NODE_NAME	node2
	...	
	CAPACITY_NAME	node_resources
	CAPACITY_VALUE	40
	...	
	#	
	# package default weight for capacity/weight	
	# node_resources.	
	# 20 is the weight value for packages that	
	# do not specify a package weight.	
	#	
	WEIGHT_NAME	node_resources
	WEIGHT_DEFAULT	20
Package configuration file for pkg1:	package_name	pkg1
	...	
	weight_name	node_resources
Weight definition	weight_value	40
Package configuration file for pkg5:	package_name	pkg5
	...	
	weight_name	node_resources
Weight definition	weight_value	10

Use case 3: Using multiple Capacities or Weights

If you do not wish to assign an overall capacity to nodes, you can assign multiple capacities. The following shows an example using memory and processor capacities.

Say that you have a cluster with the following specification:

- Two node cluster which consists of node1 with 650M of memory and 2 processors and node2 with 600M of memory and 1 processor. We assume that node1 and node2 use identical processors. Therefore, node1 has twice as much capacity for processor as node2.
- Three packages: pkg1, pkg2, and pkg3 with priorities 10, 20, 30 respectively.
 - Pkg1 requires 400M of memory and 0.6 of processor capacity.
 - Pkg2 requires 200M of memory and 0.6 of processor capacity.
 - Pkg3 requires 100M of memory and 0.2 of processor capacity.
 - Node1 is the primary node for all packages
- All packages are configured to run on both nodes.

You can assign capacities and weights as follows:

- You need to choose a unit of memory capacity to represent a certain amount of actual memory. For example, for each Mbyte of memory, you can assign one unit of memory capacity. This means node1's memory capacity will be 650 and the memory weight for pkg1, pkg2, and pkg3 will be 400, 200, and 100 respectively.
- You can enter the values for the processor capacity and weight as described above or choose, for example, 10 to represent 1 processor. We will set processor capacity of node1 to 20 and node2 to 10. This means that the processor weight of pkg1, pkg2, and pkg3 will be 6, 6, and 2 respectively.

Based on the above configuration, pkg1, pkg2, and pkg3 combined require 700 of memory capacity and 14 of processor capacity. To run all three packages at the same time, a node must meet the requirement for both memory and processor capacities. This means that node1 cannot run these packages at the same time; while the node has sufficient processor capacity for the three packages, it does not have enough memory capacity. Since node1 is the primary node for all three packages, the highest priority packages, pkg1 and pkg2 will run on node1 and pkg3 will run on node2. If node1 fails, pkg1 and pkg3 will run on node2. Even though pkg2 is higher priority than pkg3, there is not sufficient processor capacity on node2 to run both pkg1 and pkg2 at the same time (although there is enough memory capacity for both).

The following table shows portions of the configuration files for the cluster and packages pkg1, pkg2 and pkg3.

Table 7: Configuration for Use Case 3 example

Cluster configuration file:	CLUSTER_NAME	cluster_3
	...	
	#	
Node capacity definition	# node1 has capacity value of 650 for "memory"	
	# and capacity value of 20 for "processor"	
	#	
	NODE_NAME	node1
	...	
	CAPACITY_NAME	memory
	CAPACITY_VALUE	650
	CAPACITY_NAME	processor
	CAPACITY_VALUE	20
	#	
	# node2 has capacity value of 600 for "memory"	
	# and capacity value of 10 for "processor"	
	#	
	NODE_NAME	node2
	...	
	CAPACITY_NAME	memory
	CAPACITY_VALUE	600
	CAPACITY_NAME	processor
	CAPACITY_VALUE	10
Package configuration file for pkg1:	package_name	pkg1
	...	
	weight_name	memory
	weight_value	400
Weight definition	weight_name	processor
	weight_value	6
Package configuration file for pkg2:	package_name	pkg2
	...	
	weight_name	memory
Weight definition	weight_value	200
	weight_name	processor
	weight_value	6
Package configuration file for pkg3:	package_name	pkg3
	...	
	weight_name	memory
Weight definition	weight_value	100
	weight_name	processor
	weight_value	2

Using Serviceguard Manager

You can also use Serviceguard Manager, a graphical user interface for Serviceguard, to configure node capacities and package weights. The following figure shows how Serviceguard Manager displays information about the above configuration.

Figure 2: Serviceguard Manager Node Capacity tab

The screenshot shows the HP Serviceguard Manager web interface in Internet Explorer. The browser address bar shows <https://sysman5:2381/>. The page title is "HP Serviceguard Manager: Cluster cluster_3". The cluster status is "up", and there are no alerts. The "Node Capacity" tab is selected in the navigation menu.

Configured Node Capacities

Node Name	Capacity Name	Capacity Value	Total Capacity Used By Packages	Percent Of Capacity Used By Packages
node1	memory	650	600.000	92
node1	processor	20	12.000	60
node2	memory	600	100.000	16
node2	processor	10	2.000	20

Packages utilizing this capacity on selected node.

Package Name	Package Status On Node	Weight Name	Used By This Package
pk01	running	memory	400.0
pk02	running	memory	200.0

Default Package Weights

Weight Name	Weight Value
-------------	--------------

You can launch Serviceguard Manager in a browser by entering the URL, <http://hostname:2301/>, where hostname is a node in a configured Serviceguard cluster. Then click on the Serviceguard cluster name. The Serviceguard Manager online help describes how to configure node capacities and package weights. See <http://www.hp.com/go/serviceguardmanager> for more information about Serviceguard Manager.

Use case 4: Using Weights with Package Dependencies

If you configure weight for a package which requires another package with weight to be up on the same node (that is the package specifies a same node dependency on another package), the node must have sufficient capacity for both packages. For a description of package dependencies, please refer to the manpage for `cmmakepkg (1m)` command and the Managing Serviceguard manual.

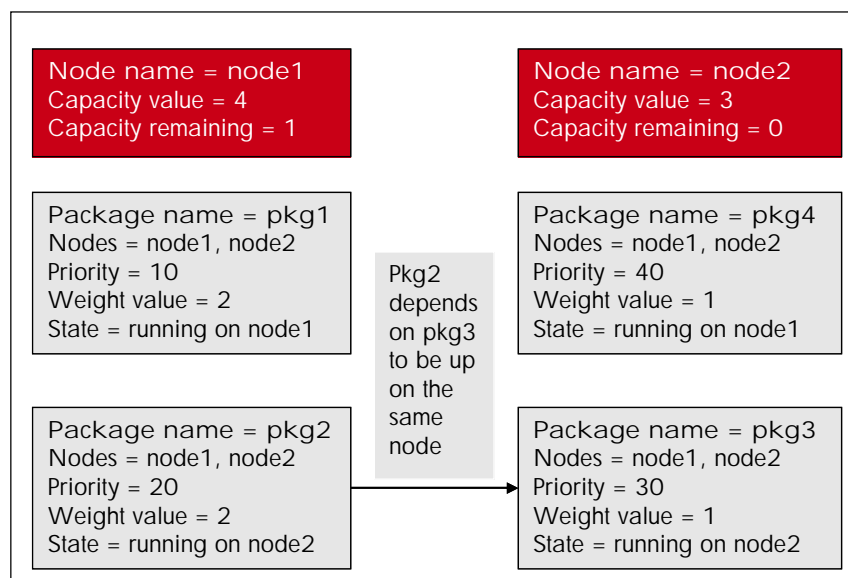
Say you have a cluster with the following specification (see figure below):

- Cluster has two nodes. You estimate that node1 has a capacity of 4 and node2 has a capacity of 3 for running packages.
- You have four packages: pkg1, pkg2, pkg3, and pkg4 with priorities 10, 20, 30, and 40 respectively.
- All packages are configured to run on both nodes and they all prefer to run on node1 (that is node1 is the first node in the package's configured node list).
- You estimate that:
 - pkg1 and pkg2 have a weight of 2
 - pkg3, and pkg4 have a weight of 1

Pkg1 is the highest priority package and it prefers node1. So, it will run on node1. This means that node1 has a capacity of 2 available for running other packages. Pkg2 is the 2nd highest priority package and it also prefers node1. Pkg2 requires pkg3 to run on the same node and together they have a weight of 3. So, pkg2 cannot run on node1 and will run on node2 instead (along with pkg3). This means that node2 has no capacity remaining for running other packages. Pkg4 has a weight of 1 and prefers node1 and node1 has sufficient capacity available. So, it will run on node1.

Now if node1 fails, since node2 has no capacity remaining for running other packages, some packages must be halted to make room for pkg1 which has the highest priority. Pkg1 has a weight of 2. Since node2 does not have sufficient capacity for both pkg1 and pkg2, Serviceguard will halt pkg2 to make room for pkg1. Pkg1 and pkg3 will run on node2. Pkg4 will go down because it is the lowest priority package.

Figure 3: Weight and dependency example



For more information

www.hp.com/go/serviceguardsolutions

www.hp.com/go/sqerac

Technology for better business outcomes

© Copyright 2009 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Linux is a U.S. registered trademark of Linus Torvalds. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. UNIX is a registered trademark of The Open Group.

4AA2-4455ENW, January 2009