NAME

sbd − STONITH Block Device daemon

SYNOPSIS

sbd −d */dev/...* [*options*] *command* [*parameters*...]

DESCRIPTION

SBD provides a node fencing mechanism (Shoot the other node in the head, STONITH) for Pacemaker-based clusters through the exchange of messages via shared block storage such as for example a SAN, iSCSI, FCoE. This isolates the fencing mechanism from changes in firmware version or dependencies on specific firmware controllers, and it can be used as a STONITH mechanism in all configurations that have reliable shared storage.

SBD can also be used without any shared storage. In this mode, the watchdog device will be used to reset the node if it loses quorum, if any monitored daemon is lost and not recovered or if Pacemaker decides that the node requires fencing.

The *sbd* binary implements both the daemon that watches the message slots, as well as the management tool for interacting with the block storage device(s). This mode of operation is specified via the *command* parameter; some of these modes take additional parameters.

To use **SBD** with shared storage, you must first *create* the messaging layout on each block device. Second (assuming the cluster stack is down), configure */etc/sysconfig/sbd* to list those devices (and possibly adjust other options), then starting the cluster stack on each node, so that **SBD** is started. Third, configure the `external/sbd` fencing resource in the Pacemaker CIB.

Each of these steps is documented in more detail when describing the commands and their options.

SBD can only be used as root.

OPTIONS

General Options

−D This option does not have any effect. Maybe it's there for compatibility with older versions.

−d */dev/...*

Specify the block device(s) to be used. If you have more than one, specify this option up to three times. This parameter is mandatory for all modes, since **SBD** always needs a block device to interact with.

This man page uses */dev/sda1*, */dev/sdb1*, and */dev/sdc1* as example device names for brevity. However, in your production environment, you should instead always refer to them by using the long, stable device name (e.g., */dev/disk/by−id/dm−uuid−part1−mpath−3600508b400105b5a0001500000250000*).

−h Display a concise summary of **sbd** options.

−I *N*

Set the Async IO timeout to *N*. This is the time within each single read or write operation to the disk device must have finished. You should not need to adjust this unless your IO setup is really very slow. The default value is 3.

(In daemon mode, the watchdog is refreshed when the majority of devices could be read within this time. That means ''(loop timeout plus io timeout) times the number of required devices'' must not exceed the watchdog timeout.)

−n *node*

Use *node* to identify the local node. This should not need to be set. The default value is the name `uname -n` would report.

−R Do **not** enable realtime priority. This is a debugging option. The default is disabled, using Round-Robin scheduling (`SCHED_RR`) with the highest possible priority, and locking its memory via `mlockall(2)`.

−v Increase logging. This option can be used up to three times to increase verbosity of messages being output. The default is no verbosity.

Commands

Command “create”

sbd −d*device...* **create**

This is the command to initialize each specified device with a metadata header and messaging slots for 255 nodes.

Warning: This command will not prompt for confirmation. Roughly the first megabyte of the specified block device(s) will be overwritten immediately and without backup.

This command uses additional options to adjust the default timings that are written to the metadata header, where they are read on each node running *SBD*. To ensure that identical parameters are used on each node, make sure that each SBD device is initialized with the same timing parameters.

−1 *N*

Set watchdog timeout to N seconds. This depends mostly on your storage latency; the majority of devices must be successfully read within this time, or else the node will self-fence.

If your sbd device(s) reside on a multipath setup or iSCSI, this should be the time required to detect a path failure. You may be able to reduce this if your device outages are independent, or if you are using the Pacemaker integration. The default value is 5 for most platforms, 15 for S390.

−2 *N*

Set slot allocation timeout to N seconds. You should not need to tune this. Actually this is not a timeout value, but the delay between retrying to allocate a slot for a host. The default value is 2.

−3 *N*

Set daemon loop timeout to N seconds. You should not need to tune this. Actually this is not a timeout value, but the delay between each round of trying to read the disks. In addition it is the delay being used as delay between attempts to connect to the CIB. The default value is 1.

−4 *N*

Set *msgwait* timeout to N seconds. This should be twice the *watchdog* timeout. This is the time after which a message written to a node’s slot will be considered delivered. (Or long enough for the node to detect that it needed to self-fence.)

This also affects the *stonith-timeout* in Pacemaker’s CIB; see below. The default value is 10 for most platforms, and 30 for S390.

Example:

```
sbd −d /dev/sda1 −d /dev/sdb1 create
```

Command “list”

sbd −d*device...* **list**

List all allocated slots with their corresponding message (and possibly sender) on each device. You should see a slot for every cluster node that ever has been started with the corresponding device. Nodes that are currently running should have a `clear` state; nodes that have been fenced, but not yet restarted, will show the appropriate fencing message (e.g. `reset`). See also ‘‘Command ’’message"" for details.

Example:

```
# sbd −d /dev/sda1 list
0       hex−0   clear
1       hex−7   clear
2       hex−9   clear
```

Command “dump”

sbd −d*device*... **dump**

Dump meta-data header of each specified device.

Example:

```
# sbd −d /dev/sda1 dump
==Dumping header on disk /dev/sda1
Header version     : 2.1
UUID               : c345a982−627b−4cb0−b340−86ddd046950d
Number of slots    : 255
Sector size        : 512
Timeout (watchdog) : 15
Timeout (allocate) : 2
Timeout (loop)     : 1
Timeout (msgwait)  : 30
==Header on disk /dev/sda1 is dumped
```

Command "watch"

sbd −d*device*... **watch**

This command will make **sbd** start in *daemon mode*. It will constantly monitor the message slot assigned to the local node, checking incoming messages, reachability, and optionally Pacemaker's state.

A node slot is automatically allocated on the specified devices the first time the daemon starts watching the particular device. Hence, manual pre-allocation of slots is not required.

Monitoring connectivity to the specified devices, **SBD** guarantees that it does not disconnect from fencing messages. In case of disconnection **SBD** self-fences.

If a watchdog is used together with the **sbd** as is strongly recommended, the watchdog is activated at initial start of the **sbd** daemon. The watchdog is refreshed every time the majority of SBD devices has been successfully read. Using a watchdog provides additional protection against **sbd** hanging or crashing.

SBD must be started before the cluster stack! See below for enabling this according to your boot environment.

The options for this mode are rarely specified directly on the command line directly, but most frequently set via */etc/sysconfig/sbd*.

If the *Pacemaker integration* is activated, **sbd** will **not** self-fence if device majority is lost, and one of the following is true:

1. The partition the node is in is still quorate according to the CIB;
2. it is still quorate according to Corosync's node count;
3. the node itself is considered online and healthy by Pacemaker.

This allows **sbd** to survive temporary outages of the majority of devices. However, while the cluster is in such a degraded state, it can neither successfully fence nor be shutdown cleanly (as taking the cluster below the quorum threshold will immediately cause all remaining nodes to self-fence). In short, it will not tolerate any further faults. Please repair the system before continuing.

There is one **sbd** process that acts as a master to which all watchers report; one per device to monitor the node's slot; and, optionally, one that handles the Pacemaker integration. Such watchers are named *servants*.

−5 *N*

Warn if the time interval for tickling the watchdog exceeds this many seconds. That interval will be at least the loop timeout. Since the node is unable to log the watchdog expiry (it reboots immediately without a chance to write its logs to disk), this is very useful for getting an indication that the watchdog timeout is too short for the IO load of the system.

Default is 3 seconds, set to zero to disable. If the watchdog timeout is set to a value exceeding 5, that value times 3/5 is being used.

−C *N*

Watchdog timeout to set before crash-dumping. If **SBD** is set to crash-dump instead of reboot − either via the trace mode settings or the *external/sbd* fencing agent's parameter −, **SBD** will adjust the watchdog timeout to this setting before triggering the dump. Otherwise, the watchdog might trigger and prevent a successful crash-dump from ever being written.

The value set seems to be unused.

Defaults to 240 seconds. Set to zero to disable.

−c Force a cluster check. If enabled, additional cluster checks are done periodically.

Usually cluster checks are enabled automatically.

−F *N*

Number of times a failing servant process will be restarted within the servant restart interval. If set to zero, servants will be restarted immediately and indefinitely. If set to one, a failed servant will be restarted once every servant restart interval. See also option `−t`. Defaults to *1*.

−P Enable Pacemaker integration which checks Pacemaker quorum and node health. Specify this an odd number of times to enable, an even number of times to disable.

The default is enabled.

−p *pid_file*

Set the file to use as PID file to *pid_file*. There is no default value, meaning a PID file will not be written.

−S *N*

Set the start mode.

If this is set to *zero*, **sbd** will always start up unconditionally, regardless of whether the node was previously fenced or not.

If set to *one*, **sbd** will only start if the node was previously shutdown cleanly (as indicated by an exit request message in the slot), or if the slot is empty. A reset, crash-dump, or power-off request in any slot will halt the start up.

This is useful to prevent nodes from rejoining if they were faulty. The node must be manually ''unfenced'' (cleared) by sending an empty message to it:

```
sbd −d /dev/sda1 message node1 clear
```

See also ''Command ''message"" for details. The default value is *0*.

−s *N*

Set the start-up wait time for devices to *N*. When starting, **sbd** will wait up to *N* seconds to read the header of the first disk device. If set to 0, start-up will be aborted immediately if no devices are available. Dynamic block devices such as iSCSI might take some time to become connected and thus operational. The default value is 120. =item **−T**

By default, the daemon will set the watchdog timeout as specified in the device metadata. However, this does not work for every watchdog device. In this case, you must manually ensure that the watchdog timeout used by the system correctly matches the SBD settings, and then specify this option to allow **sbd** to continue with start-up.

−t *N*

Set the servant restart interval to *N*. That interval is the time in which faulty servants are restarted. See also option `−F 1`, Default is 5 seconds.

If set to zero, processes will be restarted indefinitely and immediately.

−W

Enable or disable use of the system watchdog. Use an odd number of times to enable, or an even number of times to disable. The default is enabled.

−w *watchdog_device*
Specify the watchdog device to use. If set to */dev/null*, then no watchdog is being used. The default value is */dev/watchdog*.

−Z Enable *debug mode*. Using debug mode is unsafe for production, use at your own risk! Using *once* will turn all reboots or power-offs, be they caused by self-fence decisions or messages, into a crash-dump. Specifying this *twice* will just log them but not continue running. Specifying this *three times* will call `sync()` and add a ten second delay before the actual fencing operation takes place. The default value is off.

Example:

```
sbd −d /dev/sda1 −d /dev/sdb1 −P watch
```

Command "allocate"

sbd −d*device...* **allocate** *node_name*

Explicitly allocates a slot for the specified node name. This should rarely be necessary, as every node will automatically allocate itself a slot the first time it starts up in watch mode.

Example:

```
sbd −d /dev/sda1 allocate node1
```

Command "message"

sbd −d*device...* **message** *target_node msg*

Writes message *msg* to the slot allocated for *target_node*. This is rarely done directly, but rather abstracted via the `external/sbd` fencing agent configured as a cluster resource.

Supported messages are:

`test`
This is like a built-in PING for **SBD** that also generates a log message on *target_node* and can be used to check whether **SBD** can communicate using the specified *device*.

As each message slot can only hold one message, this could overwrite an unprocessed fencing request in the same slot that had been sent by the cluster. So better avoid sending `test` messages to live cluster nodes.

`reset`
Reset the target by writing `b` to */proc/sysrq−trigger*. Before that an emergency syslog message is sent, and *sync* (2) is called.

`off`
Power-off the target by writing `o` to */proc/sysrq−trigger*. Before that an emergency syslog message is sent, and *sync* (2) is called.

`crashdump`
Cause the target node to crash-dump by writing `c` to */proc/sysrq−trigger*. Before that an emergency syslog message is sent, and *sync* (2) is called.

`exit`
This will initiate a clean exit of the **sbd** daemon on the target. The disk servant processes (and also the master process) will terminate after having read the `exit` message.

As **SBD** fencing for the target node is lost when the daemon exited, you should **not** send this message to a live cluster node; also it is not necessary, because a shutdown of the cluster stack will do that.

`clear`
This message indicates that no real message has been sent to the node, meaning it cancels any unprocessed message found in the message slot. **SBD** will write a `clear` message to its slot automatically during start-up.

Example:

```
sbd -d /dev/sda1 message node1 test
```

Command "query-watchdog"

sbd query-watchdog

Check for available watchdog devices and print some info.

Warning: This command will arm the watchdog during query, and if your watchdog refuses disarming (for example, if its kernel module has the `nowayout` parameter set) this will reset your system.

Example:

```
sbd query-watchdog
```

Command "test-watchdog"

sbd test-watchdog

Test configured watchdog device.

Warning: This command will arm the watchdog and have your system reset if your watchdog is working properly!

If issued from an interactive session, it will prompt for confirmation.

Example:

```
sbd [-w /dev/watchdog3] test-watchdog
```

Base System Configuration

Configure a Watchdog

It is highly recommended that you configure your Linux system to load a watchdog driver with hardware assistance (as is available on most modern systems), such as *hpwdt*, *iTCO_wdt*, or others. As a fall-back, you can use the *softdog* module.

No other software must access the watchdog timer; it can only be accessed by one process at any given time. Some hardware vendors ship systems management software that use the watchdog for system resets (f.e. HP ASR daemon). Such software has to be disabled if the watchdog is to be used by **SBD**.

Choosing and initializing the Block Device(s)

First, you have to decide if you want to use one, two, or three devices.

If you are using multiple ones, they should reside on independent storage devices. For example, putting more than one on the same logical unit would not provide any additional redundancy.

The SBD device can be connected via Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), or even iSCSI.

The SBD partitions themselves **must not** be mirrored (via MD, DRBD, or the storage layer itself), since this could result in a split-mirror scenario. Nor can they reside on cLVM2 volume groups, since they must be accessed by the cluster stack before it has started the cLVM2 daemons; hence, these should be either raw partitions or logical units on (multipath) storage.

The block device(s) must be accessible from all nodes. (While it is not necessary that they share the same path name on all nodes, this is considered a very good idea.) When there are multiple paths to the device, the use of multipathd is highly recommended. Then you can define a convenient alias name as well (e.g. */dev/disk/by-id/dm-name-SBD_1*).

SBD will only use about one megabyte per device, so you can easily create a small partition, or very small logical units. (The space required on the SBD device depends on the block size of the underlying device. Thus, 1MB is fine on plain SCSI devices and SAN storage with 512 byte blocks. On the IBM s390x architecture in particular, disks default to 4k blocks, and thus require roughly 4MB.)

The number of devices will affect the operation of **SBD** as follows:

One device
: In its most simple implementation, you use one device only. This is appropriate for clusters where all your data is on the same shared storage (with internal redundancy) anyway; the SBD device does not introduce an additional single point of failure then.

  If the SBD device is not accessible, the daemon will fail to start and inhibit openais startup.

Two devices
: This configuration is a trade-off, primarily aimed at environments where host-based mirroring is used, but no third storage device is available.

  SBD will not commit suicide if it loses access to one mirror leg; this allows the cluster to continue to function even in the face of one outage.

  However, **SBD** will not fence the other side while only one mirror leg is available, since it does not have enough knowledge to detect an asymmetric split of the storage. So it will not be able to automatically tolerate a second failure while one of the storage arrays is down. (Though you can use the appropriate crm command to acknowledge the fence manually.)

  It will not start unless both devices are accessible on boot.

Three devices
: In this most reliable and recommended configuration, **SBD** will only self-fence if more than one device is lost; hence, this configuration is resilient against temporary single device outages (be it due to failures or maintenance). Fencing messages can still be successfully relayed if at least two devices remain accessible.

  This configuration is appropriate for more complex scenarios where storage is not confined to a single array. For example, host-based mirroring solutions could have one SBD device per mirror leg (not mirrored itself), and an additional tie-breaker on iSCSI.

  It will only start if at least two devices are accessible on boot.

After having prepared the devices, use the `create` command described above to initialize the SBD metadata on them. Optionally you may allocate slots for each node that will use the SBD devices. Dumping the headers and listing the slots could be a final verification step.

Sharing the Block Device(s) between multiple Clusters

It is possible to share the block devices between multiple clusters, provided the total number of nodes accessing them does not exceed *255* nodes, and they all must share the same SBD timeouts (since these are part of the metadata).

If you are using multiple devices this can reduce the setup overhead required. However, you should **not** share devices between clusters in different security domains, because in principle each node can fence each other node using the same device.

Configure SBD to start on boot

On systems using `sysvinit`, the `openais` or `corosync` system start-up scripts must handle starting (and stopping) of the **sbd** daemon as required before starting the rest of the cluster stack.

For `systemd`, **sbd** simply has to be enabled using

```
systemctl enable sbd.service
```

The daemon is brought online before corosync and Pacemaker are started, and terminated only after all other cluster components have been shut down − ensuring that cluster resources are never activated without **SBD** supervision.

Configuration via sysconfig

The system instance of **sbd** is configured via */etc/sysconfig/sbd*. In this file, you must specify the device(s) used, as well as any options to pass to the daemon:

```
SBD_DEVICE="/dev/sda1;/dev/sdb1;/dev/sdc1"
SBD_PACEMAKER="true"
```

SBD will fail to start if no `SBD_DEVICE` is specified. See the installed template for more options that can be configured here.

Testing the SBD installation

As root send a `test` message to any node being part of the SBD configuration (i.e. using the same devices):

```
sbd -d /dev/sda1 message node1 test
```

When a SBD daemon on the receiving node is properly configured, the node will acknowledge the receipt of the message in the system logs:

```
Aug 29 14:10:00 node1 sbd: [13412]: info: Received command test from node
```

This confirms that **SBD** is indeed up and running on the node, and that it is ready to receive messages.

Make **sure** that */etc/sysconfig/sbd* is identical on all cluster nodes, and that all cluster nodes are running the daemon.

Pacemaker CIB Integration

Fencing Resource

Pacemaker can only interact with **SBD** to issue a node fence if there is a fencing resource configured. That should be a primitive, not a clone, as follows:

```
primitive fencing-sbd stonith:external/sbd \
        params pcmk_delay_max=30
```

This will automatically use the same devices as configured in */etc/sysconfig/sbd*.

As it is possible in a split-brain scenario that each node sends a fencing message to the other node at the same time, causing both nodes to be fences an instant later, the *pcmk_delay_max* setting defines a random fencing delay which reduces the likelihood that both nodes are fenced (assuming node1 is fenced by **SBD** while still delaying its fencing request for node2).

SBD also supports turning the reset request into a crash request, which may be helpful for debugging if you have kernel crash-dumping configured; then, every fence request will cause the node to dump core. You can enable this via the `crashdump="true"` parameter on the fencing resource. This is **not** recommended for production use, but only for debugging phases.

General Cluster Properties

You must also enable STONITH in general, and set the STONITH timeout to be at least twice the *msgwait* timeout you have configured, to allow enough time for the fencing message to be delivered and processed. If your *msgwait* timeout is 60 seconds, this is a possible configuration:

```
property stonith-enabled="true"
property stonith-timeout="120s"
```

Caution: if *stonith-timeout* is too low for *msgwait* and the system overhead, **sbd** will never be able to successfully complete a fence request. This will create a fencing loop.

Note that the sbd fencing agent will try to detect this and automatically extend the *stonith-timeout* setting to a reasonable value, on the assumption that **sbd** modifying your configuration is preferable to not fencing.

Management Tasks

Recovering from temporary SBD Device Outage

If you have multiple devices, failure of a single device is not immediately fatal. **sbd** will retry to restart the monitor for the device every 5 seconds by default. See option **−t** and ‘‘Command ’’watch"".

SIGNALS

SIGUSR1
Force an immediate restart of all currently disabled monitor processes by sending *SIGUSR1* to the **sbd** *inquisitor* process.

To be completed...

EXIT STATUS

0 Invocation was successful, or there were usage errors.

1 Some error has been detected.

To be refined...

ENVIRONMENT

SBD_DELAY_START
To be documented...

SBD_DEVICE
To be documented...

SBD_PACEMAKER
To be documented...

SBD_PIDFILE
To be documented...

SBD_STARTMODE
To be documented...

SBD_WATCHDOG
To be documented...

SBD_WATCHDOG_DEV
To be documented...

SBD_WATCHDOG_TIMEOUT

FILES

/proc/sys/kernel/sysrq
To be documented...

/proc/sysrq−trigger
To be documented...

/sys/class/watchdog
To be documented...

LICENSE