Global Endpoint Deduplication 2

Bacula Systems Documentation

Contents

1 Executive Summary **3**

2 Deduplication **3**
- 2.1 Advantages of Deduplication . . . 3
- 2.2 Cautions About Using Deduplication . . . 4
- 2.3 Global Endpoint Deduplication . . . 4
- 2.4 How Bacula Global Endpoint Deduplication Works . . . 4
- 2.5 Global Endpoint Deduplication During Backup Jobs . . . 5
- 2.6 New Storage Daemon Directives . . . 6
- 2.7 New Dedupengine Resource . . . 6
- 2.8 New Device Directives . . . 8
- 2.9 Director Daemon Fileset Directive . . . 9
- 2.10 About Fileset Compression . . . 9
- 2.11 Things to Know About Bacula . . . 10
- 2.12 Deduplication Engine Vacuum . . . 10
- 2.13 Deduplication Engine Status . . . 11
- 2.14 Disaster Recovery . . . 13

3 Dedupengine **13**
- 3.1 Sizing the Index . . . 14
- 3.2 Punching holes in containers . . . 14
- 3.3 Detect, Report and Repair Dedupengine Inconsistencies . . . 14

4 Hardware Requirements **16**
- 4.1 CPU . . . 16
- 4.2 Memory . . . 16
- 4.3 Disks . . . 16

5 Installation **17**
- 5.1 Linux . . . 17

6 Restrictions and Limitations **17**

7 Best Practices **18**
- 7.1 RAID . . . 18
- 7.2 ZFS . . . 18
- 7.3 Maximum Container Size . . . 19
- 7.4 Vacuum . . . 19
- 7.5 Holepunching . . . 20

Contents

- *Executive Summary*
- *Deduplication*
- *Dedupengine*

- *Hardware Requirements*
- *Installation*
- *Restrictions and Limitations*
- *Best Practices*

1 Executive Summary

IT organizations are constantly being challenged to deliver high quality solutions with reduced total cost of ownership. One of those challenges is the growing amount of data to be backed up, together with limited time to run backup jobs (backup window). Bacula Enterprise offers several ways to tackle these challenges, one of them being *Global Endpoint Deduplication 2*, which minimizes the network transfer and Bacula Volume size using deduplication technology.

This document is intended to provide insight into the considerations and processes required to successfully implement this innovative backup technique.

2 Deduplication

Deduplication is a complex subject. Generally speaking, it detects that data being backed up (usually chunks) has already been stored and rather than making an additional backup copy of the same data, the deduplication software keeps a pointer referencing the previously stored data (chunk). Detecting that a chunk has already been stored is done by computing a hash code (also known as signature or fingerprint) of the chunk, and comparing the hash code with those of chunks already stored.

The picture becomes much more complicated when one considers where the deduplication is done. It can either be done on the server and/or on the client machines. In addition, most deduplication is done on a block by block basis, with some deduplication systems permitting variable length blocks and/or blocks that start at arbitrary boundaries (sliding blocks), rather than on specific alignments.

2.1 Advantages of Deduplication

- Deduplication can significantly reduce the disk space needed to store your data. In good cases, it may reduce disk space needed by half, and in the best cases, it may reduce disk space needed by a factor of 10 or 20.
- Deduplication is combined with compression to further reduce the storage space needed. Compression depends on data type and deduplication depends on the data usage (on the need or the will of the user to keep multiple copies or versions of the same or similar data). Bacula takes advantage that both techniques work perfectly together and combines them in it's original Dedup Engine.
- Deduplication can significantly reduce the network bandwidth required because both ends can exchange references instead of the actual data itself. It works when the destination already has a copy of the original chunks.
- Handling references instead of the data can speed up most of the processing inside the Storage Daemon. For example, Bacula features like copy/migrate and Virtual Full can be up to 1,000 times faster. See the following article for more information on this subject.

2.2 Cautions About Using Deduplication

Here are a few of the things that you should be aware of before using deduplication techniques.

- To do efficient and fast deduplication, the Storage Daemon will need additional CPU power (to compute hash codes and do compression), as well as additional RAM (for fast hash code lookups). Bacula Systems can help you to calculate memory needs.
- For effective performance, the deduplication Index should be stored on SSDs as the index will have many random accesses and many updates.
- It is highly recommended that you use the dedup index with storage that handles random read/writes quickly, such as NVMe or SSD. Ceph, NFS, and spinning disks are not recommended.
- Due the extra complexity of deduplication, performance tuning is more complicated.
- We recommend Index and Containers are stored in xfs or ext4 file systems. But we are also familiar with the zfs file system.
- Deduplication collisions can cause data corruption. This is more likely to happen if the deduplicating system uses a weak hash code such as MD5 or Fletcher. The problem of hash code collisions is mitigated in Bacula by using a strong hash code (SHA512/256).
- Deduplication is not implemented on tape devices. It works only with disk-based backups.
- Deduplication 2 is compatible with cloud based infrastructure taking into consideration the virtual hardware specification required.
- The immutable flag is not compatible or does not apply to the dedup index or dedup containers.

2.3 Global Endpoint Deduplication

Bacula Systems' latest step in deduplication technology is to offer the *Global Endpoint Deduplication* feature. With Global Endpoint Deduplication, Bacula will analyze data at the block level, then Bacula will store only new chunks in the deduplication engine, and use references in standard Bacula volumes to chunks stored in the deduplication engine. The deduplication can take place at the File Daemon side (saving network and storage resources), and/or at the Storage Daemon side (saving storage resources).

The remainder of this white paper will discuss only Global Endpoint Deduplication.

2.4 How Bacula Global Endpoint Deduplication Works

- First, be aware that you need the Dedup bacula-sd-dedup-driver-x.y.z.so Storage Daemon plugin for Global Endpoint Deduplication to work. This plugin holds both *Legacy* and *Dedup2* drivers.
- Dedup2 includes new directives that allows to have both Legacy and Dedup2 engine types in the same Storage Daemon. The goal of these new directives is to support multiple Dedup Engine in the same Storage Daemon. This will allow to handle smooth migration between two Dedup of different generations. To accomplish that, you must modify your current dedup engine to use the new directives and setting "Driver=Legacy".
- Dedup devices are enabled by specifying the `Dedup` keyword as a DeviceType directive in each disk Device resource in the bacula-sd.conf where you want to use deduplicated Volumes. If you have multiple Dedud Engine you must also specify the name that you have defined in the Dedup Engine.

```
Device {
  Name = DiskDedup_Dev0
  ...
  Device Type = Dedup
```

(continues on next page)

(continued from previous page)

```
  Dedupengine = Dedupengine_name_with_dedup2
}
```

- You must pay particular attention to define a unique Media Type for devices that are Dedup as well as for each Virtual Autochanger that uses a different Archive Device directory. If you use the same Media Type for a Dedup device type as for a normal disk Volume, you run the risk that you will have data corruption on disk Volumes that are used on Dedup and non-Dedup devices.
- When Global Endpoint Deduplication is enabled, the Device will fill in disk volumes with chunk references instead of the chunks. Bacula encrypted data, and very small files will be stored in the Volumes as usual. The deduplicated chunks are stored in the "Containers" of the Dedupengine, and are shared by all other dedup-aware devices (related to the same Dedup LegacyDedupEngine) in the same Storage Daemon.
- We advise to set a limit on the number of Jobs or the usage duration when working with dedup Volumes. In case you prefer to use `Maximum Volume Bytes`, consider that two Catalog fields are considered when computing the volume size. `VolBytes` represents the volume size on disk and `VolaBytes` considers the amount of non-dedup data stored in the volumes, i.e., the rehydrated data. If the directive `Maximum Volume Bytes` is used for a dedup Volume, Bacula will consider both VolBytes and VolaBytes values to check the limits.

2.5 Global Endpoint Deduplication During Backup Jobs

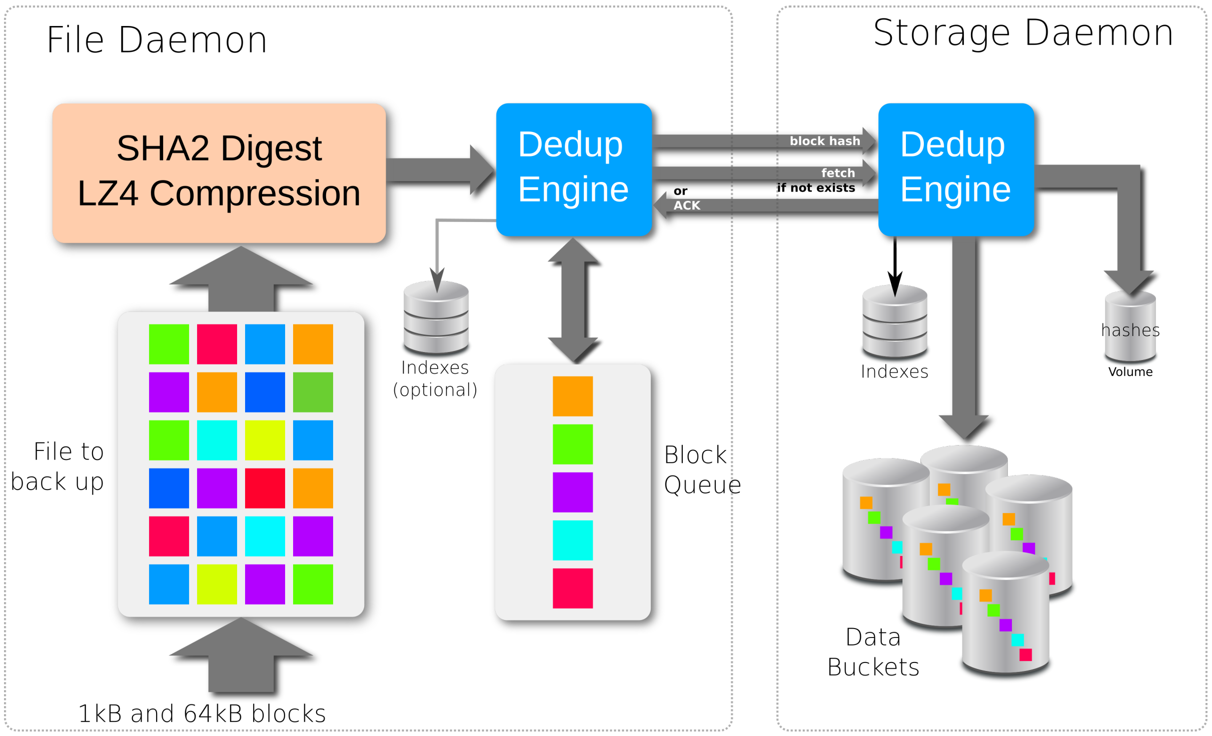

Fig. 1: Backup Scenario with bothsides deduplication

- When starting a Backup Job, the Storage Daemon will inform the File Daemon that the Device used for the Job can accept dedup data.
- If the Fileset uses the `dedup = bothsides` option, the File Daemon will compute a strong hash code for each chunk and send *references* including these hashes to the Storage Daemon which will request the original chunks from the File Daemon for any of them that the Dedupengine is unable to resolve.

- If the Fileset uses the `dedup = storage` option, the File Daemon will send data as usual to the Storage Daemon, and the Storage Daemon will compute hash codes and store chunks in the Dedupengine and the references in the disk volume.
- If the Fileset uses the `dedup = none` option, the File Daemon will send data as usual to the Storage Daemon, and the Storage Daemon will store the chunks in the Volume without performing any deduplication functions.
- If the File Daemon doesn't support Global Endpoint Deduplication, the deduplication will be done on the Storage side if the Device is configured with `DeviceType = dedup`.

2.6 New Storage Daemon Directives

Remember to specify the directive Plugin Directory.

- Plugin Directory = `<directory-path>`
  This directive tells the Storage Daemon where to find plugins. The file bacula-sd-dedup-driver-x.y.z.so must be present in this directory before starting the Storage Daemon.

2.7 New Dedupengine Resource

As said earlier, it is possible to have both the legacy and the new *Dedup2* engines in the same Storage Daemon. A new format has been introduced to allow it. The new resource *Dedupengine* will allow to have your current legacy dedup engine and a new *Dedup2* engine in the same Storage Daemon. All Dedup Devices that refer to the same *DedupEngine* shares the same storage space.

```
Dedupengine {
  Name = Dedupengine_name_with_dedup2
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index
  Driver = Dedup2
  MaximumContainerSize = 2TB
}
```

- Name = `<name>`
  The name of the DedupEngine that will be used in the Device resource. The Dedup name is required.
- Dedup Directory = `<directory-path>`
  Deduped chunks will be stored in the Dedup Directory. This directory is shared by all Dedup devices configured on a Storage Daemon and should have a large amount of free space. We advise you to use LVM on Linux Systems to ensure that you can extend the space in this directory. The Dedup Dedup Directory directive is mandatory. We recommend that you do not change this directory afterward, because if you make a mistake, it would invalidate all of your backups. If you do change the `Dedup Directory` directive, the following files must be moved to the new directory:
  - `beed2-XXXXXXXX.ctn`
  - `beed2.esm0`
  - `beed2.esm1`

  The .esm0 and .esm1 files hold the state of every extent in each container.
- Dedup Index Directory = `<directory-path>`
  Indexes will be stored in the Dedup Index Directory. Indexes will have a lot of random update accesses, and will benefit from fast drives such as SSD drives. By default, the Dedup Index Directory is set to the Dedup Directory.

As with the Dedup Directory, we recommend against changing the `Dedup Index Directory` directive. If you do, the following files and directories must be moved to the new directory:

  - `beeindex.tch`
  - `beeindex.tch.new`

The file `beeindex.tch.new` is a temporary file created by the `optimize` part of the vacuum that remain when the process is interrupted. This file holds a new version of the index. At startup the DedupEngine rename this file into *beeindex.tch* and use this new file. If this *beeindex.tch.new* file exists you can just copy it as is and ignore *beeindex.tch*

- Maximum Container Size = `<size>`

  No container will be allowed to grow to more than <size> bytes. When this size is reached, a new container file will be created. The default value is 100GiB. This value is used at initialization time and cannot be modified later, any change would be ignored. This limit is useful when you store your containers on a filesystem that limits the size of the file to a pretty low value. For now, the number of containers is limited to 8192 but we recommend to keep the number below 1024. This directive should be used to limit the size of container files to any limit from the file system where containers are stored. Dedup2 allows up to 100 million container files.

- Maximum Container Open = `<count>`

  This is the maximum number of containers that can be simultaneously open. When *Maximum Container Size* has been setup to a small size, this directive force Bacula to close unused containers before to open a new one and avoid to have 1000th of simultaneously open files. The default value is 100 GB. The recommended value is 20 + the number of simultaneous backup + the number of simultaneous restore.

- Driver = `<driver>`

  You have the choice between the old Dedup **Legacy** or the new one **Dedup2**.

- MaximumIndexMemorySize = `<size>`

  This is the maximum memory the DedupEngine will allocate for the `Index`. If unspecified, Bacula uses the available RAM as a starting point, saving enough RAM for the system. When the Index is created the first time, the half of this value or the half of the RAM is used to size the hash table. The initial hash table should not use more than 8GiB, this is about 1 billion entries.

To modify your current legacy Dedup engine to the new configuration format:

- Stop the Storage Daemon
- Modify the current dedup engine settings by creating a DedupEngine resource with Dedup=Legacy and the same index and containers directories of your current dedup engine:

```
Dedupengine {
  Name = Dedupengine_legacy
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index
  Driver = Legacy
  MaximumContainerSize = 2TB
}
```

- Remove the index and containers directories from the Storage resource configuration
- Start the Storage Daemon

2.8 New Device Directives

```
Device {
  Name = FileDedup1
  ...
  Device Type = Dedup
  Dedupengine = Dedupengine_Name_1
}
```

- Device Type = Dedup

  This directive is required to make the Device write Dedup volumes. Once turned on, Bacula will use references in Volumes and will store data chunks into the specified DedupEngine.

  Once a Device has been defined with a certain Type (such as Dedup, Aligned, File or Tape), it cannot be changed to another Type. If you do so, the Bacula Storage Daemon will not be able to properly mount volumes that were created before the change.

- Dedupengine = <dedupengine-name>

  This directive defines the DedupEngine where this device will store its chunks.

```
# From bacula-sd.conf
Storage {
 Name = my-sd
 Working Directory = /opt/bacula/working
 Pid Directory = /opt/bacula/working
 Subsys Directory = /opt/bacula/working
 Plugin Directory = /opt/bacula/plugins
}

Dedupengine {
  Name = Dedupengine_Name_1
  Dedup Directory = /mnt/bacula/dedup/containers
  Dedup Index Directory = /mnt/SSD/dedup/index  # Recommended to be on fast local SSD␣
↪storage
  Driver = Dedup2
  Maximum Container Size = 2TB # Try to get a maximum of 1000 containers at most
}

Device {
  Name = "DedupDisk"
  Archive Device = /mnt/bacula/dedup/volumes
  Media Type = DedupVolume
  Device Type = Dedup                   # Required
  dedupengine = Dedupengine_Name_1
  LabelMedia = yes
  Random Access = Yes
  AutomaticMount = yes
  RemovableMedia = no
  AlwaysOpen = no
}
```

2.9 Director Daemon Fileset Directive

Within the Director, the Global Endpoint Deduplication system is enabled with a Fileset Option directive called Dedup. Each Include section can have a different behavior depending on your needs.

```
# Use the default dedup option of 'storage' side deduplication
 Fileset {
   Name = FS_BASE
   Include {
     Options {
       Dedup = storage
     }
     File = /opt/bacula/etc
   }

   # Do not dedup my encrypted data
   Include {
     Options {
       Dedup = none
     }
     File = /encrypted
   }

   # Minimize the network transfer by using 'bothsides' dedup option
   Include {
     Options {
       Dedup = bothsides
     }
     File = /bigdirectory
   }

 }
```

The Dedup Fileset option can have the following values:

- **Dedup = storage** - All the deduplication work is done on the Storage Daemon side if the device type is Dedup. The File Daemon will send all data to the SD just as it normally would. (Default value)
- **Dedup = none** - Disable deduplication on both the File Daemon and Storage Daemon.
- **Dedup = bothsides** - The deduplication work is done on the File Daemon and the Storage Daemon. This reduce the network traffic between both Daemon as soon as the Storage Daemon already have the same data.

2.10 About Fileset Compression

The data stored by the Global Endpoint Deduplication Engine is automatically compressed using the LZ4 algorithm. Using the Fileset `Compression = LZO|GZIP` option might reduce the deduplication efficiency, and compressing the data twice consumes extra CPU cycles on the client side. Thus we advise that you do not use client-side GZIP or LZO compression when using a Dedup Device. To prevent such an inefficient configuration, we recommend setting the `Allow Compression` directive in a Director Storage resource "No":

```
# cat bacula-dir.conf
...
Storage {
```

(continues on next page)

(continued from previous page)

```
   Name = Dedup
   Allow Compression = No    # Disable Fileset Compression
                             # option automatically
   Address = baculasd.lan
   Password = xxx
   Media Type = DedupMedia
   ...

}
```

2.11 Things to Know About Bacula

- You must pay particular attention to define a unique Media Type for devices that are Dedup as well as for each Virtual Autochanger that uses a different Archive Device directory. If you use the same Media Type for a Dedup device type as for a normal disk Volume, you run the risk that you will have data corruption on disk Volumes that are used on Dedup and non-Dedup devices.
- Dedup devices are compatible with Bacula's Virtual Disk Changers
- We strongly recommend that you not use the Bacula `disk-changer` script, because it was written only for testing purposes. Instead of using `disk-changer`, we recommend using the Virtual Disk Changer feature of Bacula, for which there is a specific white paper.
- We strongly recommend that you update all File Daemons that are used to write data into Dedup Volumes. It is not required, but old File Daemons do not support the newer FD to SD protocol, and consequently the Global Endpoint Deduplication cannot not be done on the FD side.
- The immutable flag is compatible with dedup volumes, see more details in Volume Protection Enhancements and Volume Protection.

2.12 Deduplication Engine Vacuum

Over time, you will normally delete files from your system, and in doing so, it may happen that there will be chunks that are stored in dedup containers that are no longer referenced.

In order to reclaim these unused chunks in containers, the administrator needs to schedule a `vacuum` option of the `dedup` command. The `vacuum` option will analyze dedup volumes and mark as free any chunks that are not referenced, thus allowing the disk space to be reused. The vacuum command can run while other jobs are running. During the index optimization backups are temporary suspended and resume just after the optimization.

```
* dedup
Dedup Engine choice:
     1: Vacuum data files
     2: Cancel running vacuum
     3: Display data files usage
Select action to perform on Dedup Engine (1-3): 1
The defined Storage resources are:
     1: File1
     2: Dedup
Select Storage resource (1-2): 2
Connecting to Storage daemon File at localhost:9103 ...
dedupvacuum start inventory
```

(continues on next page)

(continued from previous page)

```
Found 1 volumes (4.547 MB) to scan for MediaType=File
Ready to read from volume "TestVolume001" on Dedup device "FileStorage" (/home/bac/
↪workspace2/bee/regress/tmp).
End of Volume "TestVolume001" at addr=4547374 on device "FileStorage" (/home/bac/
↪workspace2/bee/regress/tmp).
Found 0 volumes (0 B) to scan for MediaType=File1
dedupvacuum start index cleanup and container defrag
Vacuum OK.
```

2.13 Deduplication Engine Status

Is it possible to query the Deduplication Engine to get some information and statistics. Here is an example output of the dedup `usage` command, followed by an explanation of each section in the output:

```
Dedupengine status: name="dedup2-Dedupengine" now="11-Nov-2022 10:07:26"
Index: usage=0% hash_count=928859 index_capacity=1006632947
Storage: usage=5% used=62.80GB free=1.002TB unused=54.30GB
References: count=1881851 size=123.3GB count_ratio=2.03 size_ratio=1.96
Last Vacuum: start="04-Nov-2022 20:16:20" end="04-Nov-2022 20:18:48"
 total=0:02:28 inventory=0:00:06 index_cleanup=0:00:12 defrag=0:00:48
 volume=5 ref=479198 ref_size=31.40G
 suspect=0 orphan=0 2miss=0 miss=0 idx_err=0
```

Dedupengine: General information about the *Dedupengine*

- `name` This is the name of the Dedupengine
- `now` This is when the status has been generated in local time

Index: Information about the *Index*

- `usage` This is the usage of the index in %. A value above 100% means that the hash table is overloaded and that a vacuum should be run to optimize the index and increase the size of the hash table.
- `hash_count` This is the number of entries in the index. This is the number of chunk that are indexed.
- `index_capacity` This is the size of the hash table of the index

Storage: Information about the storage and the containers

- `usage` This is usage in % of space used by the *containers* of the Dedupengine. This is the ratio of the used space vs the total amount of space that ould be used.
- `used` This is the amount of space used inside the containers.
- `free` This is the amount of space that is available for new data. This include the space unused inside the container and the free space in the filesystem.
- `unused` This is amount of space that is unused inside the *containers*. This space will be used first by the Dedupengine before to grow the last *container* or create new ones.

References: Information about the references inside the *Volume* from the last *Vacuum*

- `count` This is the number of references inside all the volumes.
- `size` This is the total amount of data that the volumes refer to.
- `count_ratio` This is ratio between the number of reference and the number of chunks (*hash_count*)

- size_ratio This is ratio between the size without deduplication and the size with deduplication. The difference with the count_ratio give and idea of the efficiency of the LZ4 compression.

Last Vacuum: These information have been generated during the last *Vacuum*

- start This is the time of the start of the last *vacuum*.
- end This is when the last *vacuum* ended.
- total This is the total amount of time used by the vacuum in H:m:s
- inventory This is the time used read the volumes and make the *inventory* of the chunk that must be kept.
- index_cleanup This is the time to cleanup and optimize the index.
- defrag This is the time to reorganize the data inside the containers and release unused extends.
- volume This is the number of volume that have been read to create the inventory.
- ref This is the number of references found in the volumes.
- ref_size This is the total of all rehydrated references in all the volumes. This is the size that would be needed if deduplication was not in use.
- suspect This is the number of references that are missformed. This can come from an unexpected shutdown or a volume corruption. When the *checkmiss* option is used this also count the *orphan* references.
- orphan When the *checkmiss* option is used, this is the number of orphan references. Orphan references are reference that don't have a matching entry in the *index* and a matching chunk in the *containers*.
- 2miss This is the number of new *missing* entries created in the index by the the last vacuum when the *checkmiss* option is used.
- miss This is the number of *missing* entries in the index. This is entries that are known to be needed but that don't have a matching chunk in the container. One missing entry can have multiple related *orphan* references.
- idx_err This is number of errors when creating *missing* entries in the index. If the this counter is greater than zero this can confirm that the old index was suffering of some problem. The value should goes to zero at the next use of the *checkmiss* option

Dedup command with the qcontainer option can be used to query used and unused space in the containter files.

To see more details on the containers usage, you can use the dedup command:

```
*dedup qcontainer storage=dedup2-autochanger
Connecting to Storage daemon bacula-sd at bacula-sd:9103...
...
dedup device=dedup2-Chgr drive=-1 cmd=qcontainer
container[00]  [0-476836]  00% free=476827 used=8 unavailable=1 other=0 file=/mnt/dedup2/
↪containers/beed2-00000000.ctn
container[00] 0.............................................
```

2.14 Disaster Recovery

Catalog

The Catalog doesn't contain any reference to the deduplication engine. However, the dedup volumes' records in the Catalog are used during the vacuum process. For this reason, you must make sure to have the Catalog properly restored before starting a dedup vacuum process.

Volumes

If a dedup Volume is in the Catalog but not on disk, a dedup vacuum process will stop and report an error.

Index

The Index is essential for the deduplication engine. In the case of a disaster, contact Bacula Systems Support Team.

Extent Space Map (ESM)

The DedupEngine use two ESM file. One is active the other is ready to receive a new version of the ESM. The active version can be guessed with a field that works like the Rock, Paper Scissor logic. There is no reconstruction procedure for ESM yet.

Containers

Containers are cut into extents that hold the chunks of data. When a container (or part of a container) file is lost, the data is lost and it is not recoverable by Bacula. There is no specific tool able to identify and update the index with the data that are missing.

3 Dedupengine

The deduplication engine is the heart of Bacula's Global Endpoint Deduplication. It has to store, to index and to retrieve the chunks. All chunks are stored in the `Container Area` and registered in the `Index`. The `Index` is the bottleneck of the deduplication process because all operations need to access it randomly, and very quickly. Memory caching and storing the Index on SSD drives will help to maintain good performance.

The Deduplication Index maintains all the hashes of all chunks stored in the Dedup Containers. To get effective performance very fast low latency storage is critical. For large back up data it is best to have the Containers and Deduplication Index on the same hardware server with the Deduplication Index on solid-state drives (SSDs). The faster the disk performance, the faster and more efficient the deduplication process and the data protection will be. In production environments it is best to avoid configurations which introduce latency and delays in the storage infrastructure for the Deduplication Index. It is therefore best to avoid spinning disks, remote access configurations like NFS or CIFS and virtualized SDs. These can be acceptable for a small amount of data (1-2TB) or to perform tests but will normally not provide acceptable performance in larger production environments.

3.1 Sizing the Index

The size of the index depends on the number of chunks that you want to store in the deduplication engine. An upper limit would be 1 chunk per file plus 1 chunk per 64K block of data.

$$\text{number\_of\_chunks} = \text{number\_of\_files} + \frac{\text{data\_amount}}{64\text{K}}$$

If all you have is the storage capacity of your Storage Daemon and want to maximize it, you must know the average compressed size of the chunks you expect to store in Containers. If you don't know the size, you may use 16K.

$$\text{number\_of\_chunks} = \frac{\text{storage\_capacity}}{16\text{K}}$$

When you know the number of chunks, you can calculate the size of your index.

$$\text{index\_size} = 1.3 * \text{number\_of\_chunk} * (8 + 70)$$

The index can be split into two parts: the hash table and the records.

$$\text{index\_size} = \text{table\_size} + \text{record\_size}$$

$$\text{table\_size} = 1.3 * \text{number\_of\_chunk} * 8$$

$$\text{record\_size} = 1.3 * \text{number\_of\_chunk} * 70$$

The hash table part is small and is accessed for all operations. The record part is bigger.

For good performance, Bacula try to lock the hash table into memory. The OS will keep in memory a part or the whole part of the records in memory depending the the amount of RAM and the size of the Index.

But these are not the only requirements. Bacula needs some extra space on disk and in memory to optimize and resize the Index. We recommend the following:

- Be sure to have 3 times the index_size on an SSD drive for the Index.
- Try to have index_size+table_size of RAM for the Index.
- At least be sure to have 3 times the "table_size" of RAM for the Index.

3.2 Punching holes in containers

New Dedup2 does not need and does not provide any kind of `hole punching`. The `vacuum` moves the `chunks` inside the containers to optimize the space. This also improve the backup speed and the restore speed. The vacuum `holepunching` option is then ignored.

3.3 Detect, Report and Repair Dedupengine Inconsistencies

The `dedup vacuum` command should be used to detect, report and possibly repair inconsistencies in the DDE.

When used with the `checkmiss` option, the vacuum process will additionally detect and report references used by backup jobs that are not present in the dedup index.

The `checkindex` option is not used in dedup2.

The vacuum process will log information to the trace file.

The rebuild of the dedup2 index is only necessary when requested by Bacula Systems support. Rebuilding the dedup2 index will establish a baseline verify directory, allowing for the identification of any inconsistencies between the data and the index that may not be detected through other methods. Once the baseline is set, it will facilitate recovery from a crash or unexpected shutdown of the dedup2 SD in the future.

checkmiss and checkvolume options of the vacuum command

In Dedup2 there is no more difference between options `checkvolume` and `checkmiss`.

These options search the Index for every reference found in the volumes. This can significantly increase the time of the vacuum if the Index doesn't fit into memory.

The option `checkmiss` simply creates dummy entries when a reference in not found in the Index. This entry indicates that the chunk is missing and could be resolved by future backups.

Every mismatch is logged in the trace file with the coordinate of the file that holds the reference. Only one line is logged per file and per type of mismatch, others are counted in the statistics.

The lines in the trace file look like this:

```
bacula-sd: dedupengine2.cc:872 VacuumBadRef OrphanRef FI=1729 SessId=1
  SessTime=1654854105 : ref(#1f41101f addr=0x0000000000000003 size=10769)
  idx_addr=0x0000000000000002
```

Every related line holds the keyword "VacuumBadRef" followed by one second keyword, see below for the details:

- `RefTooSmall`: The record in the volume that holds the reference is too small to hold a reference and is then unusable and not processed further.
- `RefWrongSize`: The record in the volume that holds the reference has an unexpected size and is then unusable and not processed further.
- `OrphanRef`: The hash related to this reference was not found in the index or is know by the Index to be missing. This reference is an orphan one. The file that holds this reference cannot be fully recovered.

The other fields on the line depend on the type:

- `FI, SessId and SessTime` are the coordinates of the file as written in the Catalog.
- `fields inside ref()` are related to the reference.
- `idx_addr` this is the address in the index. This should always be 0x2

At the end, Bacula displays some statistics in the trace file. these values can be see via the *dedup usage* command and are documented there.

```
Vacuum: volumes=1 ref_count=1975 ref_size=97382548 suspect_ref=1
Vacuum: 2miss=0 orphan=1 idxupd_err=0
```

Self-Healing

It is possible to enable the `self_healing` feature on the dedup Storage to store all chunks of data to the Deduplication Engine even if the chunks are already stored.

```
dedup storage=Dedup self_healing=1
```

This feature is helpful when a restore/migration/copy is failing due to a corrupted chunk. Then, enabling the `self_healing` feature, and re-running the same backup job using Full level can potentially recover the required corrupted chunk.

4 Hardware Requirements

4.1 CPU

Bacula's Global Endpoint Deduplication consumes CPU resources on both File Daemon and Storage Daemon. The table *1* shows operations done by both daemons depending on the deduplication mode.

Note that the Storage Daemon has to re-calculate hashes of the chunks sent by the File Daemon to ensure the validity of the data added to the Dedupengine.

Table 1: Operations done by each daemon

	Dedup=none	**Dedup=storage**	**Dedup=bothside**
Client	N/A	N/A	hash + compress
Storage	N/A	hash + compress + DDE	decompress + hash + DDE

On recent Intel processors, compression and hash calculations each require about 1GHz of CPU power per 100MB/sec (1Gbps). Decompression requires only 0.25GHz for the same throughput. The Dedupengine depends more on IOPs rather than on CPU power (about 0.1GHz per 100MB/sec). Both daemons must also handle network and disks (around 1GHz per 100MB/sec).

The rules of thumb might be to dedicate 3GHz per 100MB/s for the File Daemon or the Storage Daemon when doing deduplication.

Table 2: CPU requirements (Intel & AMD Xen based)

	100MB/sec (Gbps)	**400MB/sec (4Gbps)**	**1000MB/s (10Gbps)**
Client or storage	3GHz	4 cores at 3GHz	10 cores at 3GHz

4.2 Memory

The File Daemon requires additional RAM to do `bothsides` deduplication because it has to keep the chunks in memory until the Storage Daemon sends a command to send or to drop a chunk of data. The extra memory required is about 4MB per running job.

The Storage Daemon also requires about 4MB of memory for every running job. The Dedupengine also needs more of memory for efficient lookups in the index file, see *this section*.

4.3 Disks

On the Storage Daemon, chunks are stored together into *Extent* that are *randomly* stored into Containers free space. The disk systems might have to do a mix of sequential and random I/O during backup and restore. Note that migration/copy and virtual full Jobs do not need to rehydrate data if the destination device is connected to the same dedupengine.

Container files are stored in the `Dedup Directory`. All Volumes use references to those container files.

For effective performance, it is recommended to store the deduplication engine Index on dedicated SSDs, see section *#section:dedupengine*. It is not recommended to store deduplication engine containers with the Catalog.

The index file used to associate SHA512/256 digests with data chunk addresses will be constantly accessed and updated for each chunk backed up. Using SSD disks to store the index file will give better performance. The number of I/O operations per second that SSD devices can achieve will drive the global performance of the deduplication engine. For example, if your disk system can do 10,000 operations per second, it means that your Storage Daemon will be able to write between 5,000 and 10,000 blocks per second to your data disks. (310 MB/sec to 620 MB/sec with 64 KB block size, 5 MB/sec to 10 MB/sec with 1 KB block size). The index is shared between all concurrent Jobs.

To ensure that a file system required for container, disk, volumes is mounted before the Storage Daemon starts, you can edit the `bacula-sd.service` unit file

```
# systemctl edit bacula-sd.service
```

This will create the file `/etc/systemd/system/bacula-sd.service.d/override.conf` to add `bacula-sd.service` customized settings. Add the following line line to the file and save it:

```
RequiresMountsFor=/bacula/dedup/index /bacula/dedup/containers /bacula/dedup/volumes
```

5 Installation

The recommended way to install deduplication plugin is using BIM, where the deduplication plugin installation can happen alongside the installation of Storage Daemon, at the point of choosing the plugin.

5.1 Linux

To install deduplication plugin for Linux, visit Linux: Install Storage Daemon and, in step 5, choose the dedup plugin. If you have already installed SD, run the installation again and choose the dedup plugin.

Important: While going through the installation steps again, your configuration file will not be overwritten.

6 Restrictions and Limitations

- You must take care to define unique Media Types for Dedup Volumes that are different from Media Types for non-Dedup Volumes.
- Some files are not good candidates for deduplication. For example, a mail server using maildir format will have a lot of small files, and even if one email was sent to multiple users, SMTP headers will probably interfere with the deduplication process. Small files will tend to enlarge your chunk index file resulting in a poor dedup ratio. A good dedup ratio of 20 for a file of 1024 bytes will save only 19 KB of storage, so much less gain than with a file of 64 KB with a poor dedup ratio of 2.
- Dedup Volumes cannot just be copied for offsite storage. Dedup Volumes should stay where the deduplication engine is running. In order to do offsite copies, it is recommended to run a Copy Job using a second Dedup Storage Daemon for example, or to create rehydrated volumes with a Copy/Migration/Virtual Full job using a regular disk Device. The VirtualFull support was added in version 8.0.7. The Storage Daemon to Storage Daemon Copy/Migration process with deduplication protocol was added in version 8.2.0.
- A Virtual Full between two Storage Daemons is currently not supported.
- Data spooling cannot be used with deduplication. Since versions 8.2.12 and 8.4.10, data spooling is automatically disabled whenever a device resource is configured with Type = Dedup.

- All Bacula Enterprise File Daemons (including Linux, FreeBSD, Solaris, Windows, ...) support the Global Endpoint Deduplication. The Community Bacula File Daemon supports only the Global Endpoint Deduplication in `dedup=storage` mode. The list of the platforms supported by Bacula Enterprise is available on www.baculasystems.com.
- We strongly recommend that you update all File Daemons that are used to write data into Dedup Volumes. It is not required, but old File Daemons do not support the newer FD to SD protocol, and consequently the Global Endpoint Deduplication will be done only on the Storage daemon side.
- The `restart` command has limitations with plugins, as it initiates the Job from scratch rather than continuing it. Bacula determines whether a Job is restarted or continued, but using the `restart` command will result in a new Job.

7 Best Practices

7.1 RAID

If you plan to use RAID for performance and redundancy, note that read and write performances are essential for the deduplication engine. The Index is highly accessed for reading and for writing during backup jobs run and during the maintenance tasks required by the deduplication plugin. Also, the optimize process that rebuilds the Index strongly depend on the read and write performance of the disk infrastructure.

Some RAID solutions don't fit the deduplication engine read and write performance requirements. RAID 1 and RAID 10 with a small strip size are recommended for the dedup index. RAID 5, 6, 10, 50, 60 can be used for the containers. The strip size should be compatible with the size of the *extent* that is 4MiB. The goal is for a *stripe* to never be shared by two *extent*. If the extents are well aligned and are a multiple in size with the size of the stripe, then writing a single 4MiB extent should not require to read and rewrite the stripes that are nearby. The *extent* are aligned on a 4MiB boundary inside the *containers*, up to you to configure your raid array, your partitions, your LVM layers and the filesystem parameters (like the sunit and swidth of XFS) to get the container aligned too.

7.2 ZFS

If you plan to use ZFS file system to store the dedup index, it is important to guarantee that you have enough memory and CPU resources in the Storage Daemon server to have both the deduplication plugin and ZFS in good condition.

The Global Endpoint Deduplication plugin does both deduplication and chunk compression. This means there is no need to have enabled deduplication or compression in the zfs pool that will be used to store containers. In fact, it is not recommended to have them enabled as it may cause slow performance and there will be no gain in space savings.

Aligned disk acess is a key factor when using ZFS. ZFS is able to detect the sector size of the drive, but disks can report the emulated value instead. As we recommend SSD disks for the dedup Index, performance can be improved if the `ashift` property is explicitly set to match the 4096 byte sector size of SSD disks.

The disk block size is defined by the `ashift` value, but as ZFS stores data in records, there is another parameter that determines the individual dataset to be written in the ZFS pool, this is the `recordsize` ZFS pool parameter.

Thus another important ZFS pool setting to consider is the `recordsize` value. It defaults to 128K in recent ZFS versions. This value should be adjusted to match the typical I/O operations. For the ZFS pool used by the dedup Index, it was reported that setting the `recordsize` to 8K increased the vacuum performance. In addition, setting the ZFS kernel `zfs_vdev_aggregation_limit_non_rotating` parameter to the same value as `recordsize` highly improved performance.

Note these values are recommended for most SSD disks, but they may vary depending on the SSD model. For example, 16K could fit some SSD models and give a better performance. We recommend to perform I/O benchmark using different settings before the deduplication engine is setup in production.

Regarding the use of ZFS to store dedup containers, as it is not possible to preview the typical dataset because some containers can be much more used than others, it is more probably that a `recordsize` value of 64K or even the 128K default value are sufficient to have a good performance. However, it is strongly important to not allow container files to grow too much and limit the size of containers files to 3TB or 4TB.

7.3 Maximum Container Size

For better performance, it is strongly recommended to not allow container files to grow indefinitely even if the underlying file system support very big files. This can be accomplished by setting the “Maximum Container Size” directive in the Storage resource in the Storage Daemon configuration file. It is recommended to set this directive to a value between 1 TB and 4 TB.

7.4 Vacuum

It is strongly recommended, to keep the deduplication engine healthy, to regularly perform the maintenance tasks triggered by the vacuum.

Make sure to run regularly, in all deduplication engines in your environment, the following tasks: daily prune and truncation of volumes, a simple vacuum daily (it can be run weekly when not too much new chunks are added to the deduplication engine), a dedup vacuum checkindex checkmiss monthly

These maintenance tasks can be scheduled in a job of type Admin and they will help to clean both the deduplication engine index and containers, marking unused entries in the index and chunks in containers, thus allowing the reuse of space. It will contribute to avoid invalid entries in the index to be used by backup jobs in the case of any problems with the server hosting the deduplication engine.

The scrub process is not implemented in Dedup2.

Below are examples of two Admin Jobs that can be used to run a weekly and monthly vacuum.

```
Job {
  Name = "DedupSimpleVacuum_ADMTASK"
  Type = "Admin"
  Client = "bacula-fd"
  Fileset = "LinuxHome"
  Messages = "Standard"
  Pool = "DiskBackup365d"
  Priority = 10
  Runscript {
    Console = "dedup vacuum storage=MyDedup2Storage"
    FailJobOnError = no
    RunsOnClient = no
    RunsWhen = Before
  }
  Schedule = "Vaccum_daily_10AM"
  Storage = "MyDedup2Storage"
}

Schedule {
  Name = "Vaccum_daily_10AM"
  Enabled = yes
  Run = at 10:00
}
```

(continues on next page)

(continued from previous page)

```
Job {
  Name = "DedupDeepVacuum_ADMTASK"
  Type = "Admin"
  Client = "bacula-fd"
  Fileset = "LinuxHome"
  Messages = "Standard"
  Pool = "DiskBackup365d"
  Priority = 10
  Runscript {
    Console = "dedup vacuum checkmiss storage=MyDedup2Storage"
    FailJobOnError = no
    RunsOnClient = no
    RunsWhen = Before
  }
  Schedule = "Vaccum_monthly_secondSunday_11AM"
  Storage = "MyDedup2Storage"
}

Schedule {
  Name = "Vaccum_monthly_secondSunday_11AM"
  Enabled = yes
  Run = 2nd Sun at 11:00
}
```

In an Admin Job, the Fileset and the Pool configured are not used. Thus, you can set up any valid value available in your Bacula Enterprise environment configuration.

7.5 Holepunching

Holepunching doesn't exist anymore in Dedup2. The last part of the *Vacuum* release the *extents* that are left empty but also merge the data that are in *extents* that are nearly empty or half used into new containers. This allows new backup to always write into fully recycled extents, reduce the fragmentation and increase the efficiency of the storage.