MooseFS Hardware Guide

v.0.9.1

Best practices for choosing appropriate hardware components
for a MooseFS storage cluster.

Table of contents

Preface

The purpose

Due to the fact, that MooseFS storage cluster may serve many purposes, and the components used for its construction (servers, hard disks, memory cards, processors, network etc.) have a variety of independent parameters such as latency, frequency or capacity, there is no possibility to point out specific configurations that would suit the needs of every client. Also, while configuring a cluster you need to consider the purpose it needs to serve. One should carefully choose the parameters of each component according to the purpose of a cluster. Some selections may be determined by available resources such as network type and capacity or the servers which are already on site. We aim to inform you thoroughly on how to utilize storage in the most efficient manner.

State of the art

This document provides guidelines for selecting appropriate equipment for building storage using MooseFS distributed file system. We are making every effort to ensure that the guidelines included here refer to the current hardware configurations (servers, disks, etc.) available in the market. Nevertheless, we are aware of the continuous progress of technology and since we cannot ensure that the document will refer to the latest models everywhere, we have presented universal information which will enable you to select from the available hardware solutions.

No requirements, just tips

MooseFS has some basic requirements mainly for operating systems on which it can be ran but there are no special/minimum hardware requirements. It can run on embedded computers, servers of older generations and the latest machines. Also, it can use traditional hard drives as well as SSD disks. It can use various types of physical networks at different speeds, e.g. Ethernet from 100 Mbps to 200 Gbps or InfiniBand. Therefore, the document does not provide the minimum hardware requirements. In addition to clearly described cases, we refer to standard components which are easily available in the market. For more information, please visit our website.

Component dependencies

When preparing a specific hardware configuration, it is necessary to take into account the mutual dependencies between individual components in the cluster. The idea is to select components and parameters so that another part of the system does not limit their use. The purposefulness of using a specific component in relation to the purpose of using the entire cluster is also important.

MooseFS Architecture

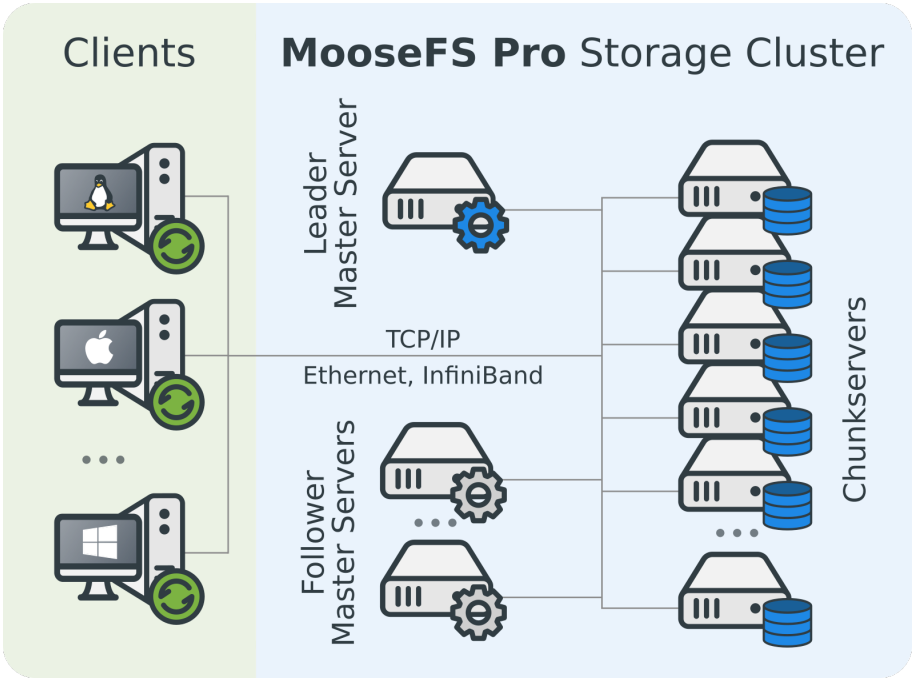

Fig. 1. A basic MooseFS architecture diagram

A MooseFS cluster consists of components of four types: Master Servers, Chunkservers, Metaloggers (metadata backup servers) and client machines.

1. Managing servers (Leader Master Server, Master Server Followers) – machines managing the whole file system, storing metadata for every file and folder. Leader Master Server is always the central point of a cluster. Leader Master Server directs clients to appropriate data servers (Chunkservers) for read/write operations. It leads all inter-servers data manipulations (replicating, auto-healing, balancing etc.). Master Server neither reads nor sends any file data (chunks). Each client and data server must know the IP/DNS address of the Master Server. Clients learn from the Master Server addresses of Chunkservers (data servers) for sending or receiving chunks of data. Master Server Followers are Highly Available online backup copies of Leader Master Server (available with HA configurations).
2. Data servers (Chunkservers) – These are machines that store files' data (chunks). They serve data for clients and synchronize among themselves when necessary.
3. Metadata backup server(s) (Metaloggers) – There can be any number of such servers, all of which store metadata changelogs and periodically download the main metadata file. In MooseFS Community, any Metalogger can be easily set up as a new Master Server in case of main Master Server failure.
4. Client computers that use (mount) the filesystem – There can be any number of such machines contacting Master Servers (to receive and modify file metadata) and Chunkservers (to exchange actual file data) using the MooseFS TCP/IP network protocol. Linux/FreeBSD/OS X machines use the `mfsmount` process based on the FUSE mechanism (Filesystem in USErspace) so that MooseFS is available on every operating system supported by FUSE. Also, Windows machines (both desktops and servers) use the native Windows `mfs4win` driver.

All components communicate to each other using the MooseFS TCP/IP based protocol. The most common network type used for MooseFS clusters is Ethernet but other types supporting TCP/IP stack may be used.

Operating systems requirements

This document is all about hardware requirements but a few words on OS requirements may also assist in making right decisions on hardware.

It's always recommended to use up-to-date operating system as it helps in maintaining security, stability and compatibility. For example, some MooseFS 3.0+ features will not work with old FUSE versions. Please refer to MooseFS website for an up to date list of supported operating systems.

Servers

The MooseFS server side (Master Servers, Chunkservers) may be used on virtually any modern POSIX-compliant operating system. This is the list of platforms with officially supported repositories:

- Ubuntu 12/14/16/18,
- Debian 6/7/8/9,
- RHEL/CentOS 6/7,
- FreeBSD 9.3/10/11,
- MacOS X 10.9+,
- Raspberry Pi 3.

In case binary files are unavailable for some platforms, one can compile MooseFS from the source code.

Clients

The client side for Linux/FreeBSD family of systems requires FUSE. MS Windows client driver supports Windows Server from 2008 R2 SP1 and MS Windows from 7 SP1.

Master Servers (Leader, Followers) and Metaloggers requirements

The Leader Master Server is the central point of a cluster. It is responsible for storing metadata of the file system (file names, attributes, folders structure etc.) in the in-memory database. Any change in metadata is immediately logged to its local disk (changelog). There is one Master Server but there may be many so-called Metaloggers. The Metalogger server just logs all metadata changes to its local disk and may be turned into Master Server manually.

As the Metalogger simply gathers asynchronous online metadata backups from Leader Master Server – the hardware requirements are not higher than that for the Master Server itself as it needs about the same disk space. The Metalogger should have at least the same amount of disk space (especially the free space in `/var/lib/mfs`) as the main Master Server. If one would like to use the Metalogger as the Master Server in case of the main Master's failure, the Metalogger machine should have at least the same amount of RAM as the main Master Server and CPU operating at the same (or at least similar) frequency.

Whereas for the High Availability configuration, there is one Master Server but there may be many so-called Follower Master Servers (or Followers). A follower is an asynchronous, online "backup" Master Server that follows all metadata changes, keeps its own copy of the in-memory database and may be elected as a new Leader Master Server by majority of Chunkservers in case Leader Master Server is down. Hardware requirements for Leader Master Servers and Follower Master Servers are the same.

The Master Server may save the entire metadata database to local disk every hour (no HA edition) or once a day (HA edition), however Follower saves its database every hour.

It is always recommended to use either Follower Master Servers (see HA configuration) or Metaloggers to backup Leader Master Server's metadata. Losing a file system's metadata information usually leads to losing data in a cluster. Although possible, it is extremely difficult to recover files from chunks without metadata information. So, here one can use both Follower Master Servers and Metaloggers but usually it is unreasonable as Follower Master Server covers Metalogger tasks and much more.

Memory

There should be sufficient memory in the Leader Master Server (Master Server Followers, Metaloggers) as it needs to store all metadata in the in-memory database. Each file or folder (inode) metadata takes about 300-350 bytes. The size of the metadata database neither depends on the cluster capacity nor on files sizes, rather it is proportional to the number of files. Therefore, for best results one should estimate the maximum number of files and folders in a cluster and adjust the Master Server memory accordingly.

For example, 150 million of files/folders requires approx. 42 GiB of RAM for storing metadata plus Master Server operating system overhead.

CPU

Since Master Server is a single-threaded process, we would recommend to use modern processors with high clock and low number of cores, e.g.:

- Intel® Xeon® Gold 5122, 3.70 GHz,
- Intel® Xeon® Platinum 8156, 3.70 GHz.

A good starting point to help in the selection of CPU for Master Server would be the single-thread performance rating published by CPU Benchmark: www.cpubenchmark.net/singleThread.html

It's recommended to disable the hyper-threading CPU feature for Master Servers. Additionally, disabling CPU power management in BIOS (or enable mode like "maximum performance") may have positive impact on efficiency.

Disks

Master Server logs each operation to its local disk but occasionally it also dumps the whole metadata database to its local disk. It is recommended to have redundant local storage (e.g. RAID 1 or RAID 1+0) in the Leader and Follower Master Servers.

The size of incremental logs depends on the number of operations per hour. Length (in hours) of this incremental log is configurable. For example, the space of 20 GiB should be enough for storing information for 25 million files and for changelogs to be kept for up to 50 hours. One can calculate the minimum amount of recommended disk space using the following formula:

```
DISK_SPACE = RAM *(BACK_META_KEEP_PREVIOUS+2) + (BACK_LOGS+1)[GiB]
```

where:

• `RAM` – amount of RAM used by Master Server process [GiB],

• BACK_LOGS – number of metadata change log files, default is 50 (from /etc/mfs/mfsmaster.cfg),

• BACK_META_KEEP_PREVIOUS – number of older metadata files to be kept (default is 1) (also from /etc/mfs/mfsmaster.cfg).

If default values from /etc/mfs/mfsmaster.cfg are used, it is RAM*3+51. For 128 GiB of RAM used by Master Server process, one should reserve for /var/lib/mfs: 128*3 + 51 = 435 GiB minimum disk space.

For configurations where Master Servers share the same machine with Chunkservers it is important to ensure the Master Server has dedicated access to its local disk. Otherwise it is possible that Chunkserver's heavy I/O disk operations slows down Master Server disk operations further causing Master Server slowdowns and finally leading to the slowdown of the entire storage cluster.

Networking

Master Server neither sends nor receives actual file data, it just deals with metadata which is usually much smaller in amount than actual file data. As it is at the central point of the cluster, each client and each Chunkserver exchanges data with the same Master Server. Even for clusters with HA configurations there is always just one Leader Master Server. This leads to a significant number of small network I/O operations. Please note, neither Chunkservers nor clients connect to Followers and Metaloggers.

Master Server may use two separate network interfaces: the first one for serving clients and the second one for communicating with other servers (which is not LACP).

Chunkservers requirements

The Chunkserver's main duty is to transfer data: read from disk & send to network and receive from network & write to disk. It doesn't perform a lot of calculations or memory operations. For EC configurations, Chunkservers are also responsible for calculating erasure codes. However, the parity calculation algorithm is extremely fast (up to 5GB/s with average hardware) and should not affect Chunkserver resources.

Chunkservers store data in the form of "chunks", on local disks. Each chunk is a file ranging from 64 kiB to 64 MiB plus 8 kiB chunk header. Files longer than 64 MiB are divided into many chunks and spread across many Chunkservers. This is why files smaller than 64 kiB occupy minimum 64+8=72 kiB of disk space.

A new data server can be connected to the system at any point in time and the new capacity is used immediately.

Memory

During normal operations the Chunkserver keeps information about its chunks in memory. Each chunk takes approximately 150-200 bytes. Additionally, approximately 350 MiB (Virtual) / 200 MiB (Resident) of memory should be available for the processes.

The required memory size depends on a desired maximum number of chunks on a server. The maximum number of chunks on a server (in terms of how many chunks the server disks may store) may be calculated roughly from the Chunkserver's total available (for MooseFS) disk space divided by average chunk size. Average chunk size depends on the average size of files. When storing files much larger than 64 MiB it should be about 64 MiB and for smaller files, average chunk size will be between 72 kiB and 64 MiB.

Example for memory requirement calculation:

- Chunkserver has 20 * 6TiB disks available for storing data,
- Users store files of approx. size ~100MiB each,
- It makes ~50MiB average chunk size (avg. each file is kept in 2 chunks)
- Memory required: (20*6TiB / 50MiB) * 170B + ~350MiB = ~408MiB + 350MiB = ~758MiB

The Chunkserver may use the remaining available memory for caching data as it boosts its performance. From our experience a typical memory size for the Chunkserver is 8-12GiB.

CPU

Since Chunkserver daemon is a multi-threaded process, it is recommended to use multi-core processors. Chunkserver would usually utilize equivalent of ~1 core of the CPU for cluster related operations.

Utilization of CPU may increase due to erasure codes calculations (for EC cluster configurations). However, the utilization increase factor depends on the EC configuration, mainly on the amount of data written with erasure codes. EC calculations are not performed during normal read operations but is calculated only during write operations and while repairing corrupted chunks.

Disks

It is recommended to connect disks as JBODs and each disk should be formatted with POSIX compliant file system and mounted separately to the operating system (e.g. as `/mnt/chunk01`, `/mnt/chunk02`, ...). The file system recommended for local disks is XFS.

Moreover, it is not recommended to use RAID controllers in the Chunkservers as it is MooseFS's role to keep data redundant and safe. There are at least two reasons for not using underlying RAID controllers:

- MooseFS has a mechanism of checking if the hard disk is in a good condition or not. MooseFS can discover broken disks, replicate data and mark such disks as damaged. The situation is different with RAID: MooseFS algorithms do not support RAIDs state checking, therefore corrupted RAID arrays may be falsely reported as healthy.
- The other aspect is the time of replication. Let's assume there is a replication goal set to 2 for the whole MooseFS instance. If one 2 TiB drive breaks, the replication (from another copy) will take about 20-60 minutes. However, if one big RAID (e.g. 36 TiB) becomes corrupted, replication can take even 12-18 hours. So, until the replication process is finished, some of data is in danger, because there is only one valid copy and if another disk or RAID fails during that time, some of data may be irrevocably lost. Thus, the longer replication period puts data in greater danger.

How many disks per Chunkserver?

Theoretically, there is no limit on the number of disks in a Chunkserver apart from hardware limits: chassis size, cooling or disk controller limits. Appropriate number of disks in a chunk server should be determined taking into consideration the way storage is to be used and the hardware parameters. For better overall cluster performance, it is recommended to have more Chunkservers with smaller number of disks:

- as each Chunkserver has its limitations for serving client I/O requests - more Chunkservers may serve more requests for many clients in parallel,
- usually either a network bandwidth limit or a controller bandwidth limit is reached when there are too many (e.g. fast SSD) disks in one server,

One may consider putting more disks in a Chunkserver when either a cluster has to keep "colder" data (more files but used infrequently) or when just a few clients are using the cluster. If just a few (or just one) clients are accessing the cluster, an increase in the number of Chunkservers will not increase the speed beyond a certain level as it is limited by the client's network bandwidth and its machine performance.

There are other factors also regarding Chunkservers/disks ratio which should be taken into account. For example, Erasure Coding requires at least 8+2n Chunkservers in a cluster so for desired cluster size the minimum number of disks per Chunkserver will be: cluster_size / ((8+2n) * disk_size).

SSDs vs. HDDs

SSD disks are getting more and more attention these days as they are faster (both latency and throughput) than spinning drives and they are getting cheaper each year. There is a trend for filling up storage clusters with SSD drives. This is possible with MooseFS, where one may build SSD-only cluster or mix SSDs with HDDs thereby building a kind of tiered storage solution.

Building such a mixed solution is easy with so-called storage classes of MooseFS. If there are Chunkservers containing disks of a single type, appropriate storage classes may be used for grouping Chunkservers and assigning policies for storing data on certain groups. Such as files not used for a long period may be automatically moved from SSD- to HDD-only Chunkservers. It is expected that SSD-only Chunkservers may be faster but more expensive and HDD-only Chunkservers may be slower but cheaper.

It is important to note that too many SSD disks in a single server may not speed up Chunkservers as expected. This is because of the constraint of either a disk controller bandwidth limit or a network bandwidth limit. Utilizing the full transfer speed of SSD drives is possible when the controller and network speed is greater than the transfer speed of a disk multiplied by the number of disks. One should also refer to the controller specification to check its limits on operating many disks at once.

Mixing SSDs and HDDs for the same Chunkserver process is not recommended. Chunkserver doesn't differentiate SSDs from HDDs. Neither storage classes don't support singe disk assignment nor Chunkserver algorithms use particular SSD disks for data caching. If – for any reason – there have to be both disk types in one Chunkserver it's possible to run separate Chunkserver processes on the same machine. Each process should have assigned a separate set of disks: SSDs and HDDs respectively. Thus, each process is available for a cluster as separate Chunkserver and storage classes may be applied.

Due to its low latency SSD disks may boost a cluster when many random I/O operations are expected e.g. storing SQL database files, home directories, web assets etc.

Pre-fetch & read-ahead

MooseFS's read-ahead and pre-fetch algorithms make HDD-only Chunkservers very effective. They pre-fetch a single chunk data when a request regarding a chunk appears. It makes data required by a client available in Chunkserver's memory prior to the I/O operation requesting it. MooseFS client may even read in advance following chunks - when data stream is read sequentially. These algorithms alleviate the problem of a slower random data access and slower data transfer for HDD disks. Data caching and read-ahead/pre-fetch algorithms only use RAM for storing cached data. Once again, these algorithms do not use SSDs.

Pre-fetch and read-ahead algorithms enhances the HDD disks clusters performance especially, for stream-like cluster access patterns e.g. storing and serving video files, logging etc.

Networking

Chunkserver sends and receives a lot of data. Its networking interface is essential for robust operations. Always keep in mind that at any given point in time, each Chunkserver serves many clients simultaneously and each client connects to many Chunkservers even when dealing with one (large enough) file. In addition, each Chunkserver sends and receives data from other Chunkservers thereby balancing operations and auto repairs.

Chunkserver may use two separated network interfaces: the first one for serving clients and the second one for server to server communication (which is not LACP).

Networking requirements

MooseFS requires TCP/IP stack as a networking protocol and the underlying network is not important. MooseFS has been successfully tested using Ethernet and InfiniBand (IP-over-IB).

Ethernet

It is recommended to set up jumbo-frames (MTU=9000).

With larger number of Chunkservers, network switches should be connected either through an optical fiber or use aggregated links.

It is recommended to use at least 1 Gbps networks although MooseFS performs better with faster networks. Configurations with many clients and Chunkservers (even when speed of disks are not taken into consideration) may easily saturate 10 Gbps and faster networks as the communication is performed on many-to-many mode. For instance, even 8 Chunkservers of moderate performance are able to saturate a client's endpoint 40 Gbps network.

LACP

To enable redundancy of the network connections (no-SPoF) it is recommended to use two switches, setting LACP between them and connecting each machine to both of them.

Cluster configurations

There are a few typical cluster configurations described below.

High Availability and No Single Point of Failure

Each edition of MooseFS keeps data safe as data is spread across many Chunkservers and it is kept redundant. However, there is also a higher level of cluster data availability which allows uninterrupted access of data. High Availability means that data is not just safe but also easily accessible by the storage clients.

To achieve HA configuration user has to:

1. install at least 3 Chunkservers (if not using erasure codes) or at least 10 Chunkservers (if using Erasure Codes),
2. install the Leader Master Server and at least 1 Follower Master Server,
3. set up replication goal greater than 1 for all data in a cluster or EC with at least 1 parity sum (EC "@1"),
4. install the HA-aware (client side) `mfsmount` daemon,
5. set up LACP or other means for network HA.

Minimal number of 3 Chunkservers are required due to the automated Leader Master Server election mechanism. The election process is designed in such a way that it prevents a possible cluster split-brain scenario. As the minimal odd number greater than 2 (required for redundancy) is 3 – so this is the minimal number of Chunkservers required.

Another reason for using at least 3 Chunkservers is to keep replication goal at safe level (at least 2) even in case of failure of one Chunkserver. With 3 Chunkservers, when one of them goes down, data is still accessible and it may be replicated to the 2nd available Chunkserver. In case when only 1 Chunkserver is available, MooseFS cluster waits for another one (in order to elect a new Leader Master Server) and is not able to perform any operations: data may be safe but is inaccessible.

One may use Erasure Coding with HA configuration. Such a case requires more Chunkservers as each chunk is divided into many "parts". Please refer to Erasure Coding configuration described below.

Installing Metaloggers for HA configuration is not necessary as Follower Master Servers take care of metadata backups. User may install more than one Follower Master Server to get higher degree of cluster availability.

The HA-aware (client side) `mfsmount` daemon assures constant file system access for client applications – even during automatic Leader Master Server election. It is important to notice that with HA configuration all pending client-side I/O operations are not interrupted, they may be just sustained for a short period of time (usually less than a few second).

Using external gateways (SMB, NFS, etc.) in front of MooseFS HA cluster requires configuring HA for these gateways independently. However, it may not be possible due to protocol or implementation limitations. MooseFS protocol supports HA for all client-cluster communication.

LACP should be used for network redundancy (as mentioned above) otherwise network becomes a single point of failure.

HA configurations are available for MooseFS 3.x Pro and MooseFS 4.x versions. It is unavailable for MooseFS 1.x, 2.x and 3.x Community Edition.

Cluster with Erasure Coding

Erasure Coding is another way of ensuring data redundancy in a cluster. Instead of keeping several copies of each file, which is disk space inefficient, each chunk of data is divided into parts. There are special, additional parts called "parity stripes" or "erasure codes" which are calculated with a special algorithm from original data. The extra parity stripes allow cluster to recover missing parts of original data when necessary.

One may define up to 9 parity stripes to be calculated for each 8 stripes of original data. It is 8+n type of erasure coding algorithm. All stripes (both original and parity) are always of the same size.

A minimal number of Chunkservers for EC configuration is 8+2n, where n is a desired number of parity stripes to be calculated by a cluster.

It should be remembered that in configurations where erasure coding is used, parity sums are always calculated by Chunkservers. Data written by cluster users' is always kept in several copies in the first step

which is an ordinary, non-EC mechanism of keeping data redundant. It is an independent (from client write operations) process to calculate erasure codes. Once parity codes are calculated keeping copies is no longer necessary, obsolete copies are deleted.

Therefore, there may be an increased demand for CPU power - during the calculation of parity codes (writing) and during data recovery (restoration of damaged data). MooseFS's erasure code calculation algorithm is very efficient with throughput of up to 5 GiBps per Chunkserver thanks to the implementation with XOR operations on large blocks of memory.

Virtualization

MooseFS cluster on virtual machines

It's not recommended (although possible) to run the Leader Master Server as a virtual machine. Virtual machines are known for their periodic slowdowns and lags and they may slow down the entire storage cluster.

It's also not recommended to run Chunkservers as virtual machines. The reason here is that virtual machines usually don't have access to physical disks which slows down data transfer.

Additionally, virtualization adds an extra layer (host operating system) which slows down most I/O and memory operations and adds extra latency overhead on network operations.

MooseFS cluster in a container (LXC)

Although LXC containers are considered to be more efficient than virtual machines the above arguments also apply to containers. It may be useful however to use containers or VMs for testing or evaluating. There are Docker setup scripts available.

Tips & Tricks

There are some tips & tricks for configuring MooseFS storage cluster BELOW.

Disable updateDB feature (Linux only)

UpdateDB is a part of `mlocate` which is simply an indexing system, that keeps a database listing all the files on your server. This database is used by the `locate` command to do searches. *UpdateDB* is not recommended for network distributed filesystems.

To disable *updateDB* feature for MooseFS, add `fuse.mfs` to the `PRUNEFS` variable in the `/etc/updatedb.conf` (it should look similar to this):

PRUNEFS="NFS nfs nfs4 rpc_pipefs afs binfmt_misc proc smbfs autofs iso9660 ncpfs coda devpts ftpfs devfs mfs shfs sysfs cifs lustre tmpfs usbfs udf **fuse.mfs** curlftpfs ecryptfs fusesmb devtmpfs"

Master Server overcommit_memory (Linux only)

If you have an entry similar to the following one in `/var/log/syslog` or `/var/log/`messages:

```
fork error (store data in foreground - it will block master for a while)
```

This would indicate that you are encountering problems with your Master Server, such as timeouts and dropped connections from clients. This happens, because your system does not allow Master process to fork and store its metadata information in the background.

Linux systems use several different algorithms for estimating how much memory every single process needs when it is created. One of these algorithms assumes that if we fork a process, it will need exactly the same amount of memory as its parent. As this algorithm process consumes 24 GiB of memory and a total amount of 40 GiB (32 GiB physical plus 8 GiB virtual) , the forking would always be unsuccessful.

Although in reality, the fork command does not copy the entire memory, only the modified fragments are copied as needed. So since the child process in MooseFS master only reads this memory and dumps it into a file, it is safe to assume that not much of the memory content will change.

Therefore such "careful" estimating algorithm is not needed. The solution is to switch the estimating algorithm the system uses. This can be done one-time by a root command:

```
# echo 1 > /proc/sys/vm/overcommit_memory
```

In order to switch the algorithm permanently, so that it remains the same even after the system has restarted, you need to put the following line into your `/etc/sysctl.conf` file:

```
vm.overcommit_memory=1
```