# PVFS Tuning

PVFS Development Team

September 2, 2026

## Contents

**1 Introduction** **2**

**2 Cluster Partitioning** **2**

**3 Storage** **2**
3.1 Server Configuration . . . 2
3.2 Local File System . . . 2
3.3 Disk Synchronization . . . 2
3.4 Metadata . . . 3
3.4.1 Coalescing . . . 3
3.5 Data . . . 3

**4 Networks** **3**
4.1 Network Independent . . . 3
4.2 TCP . . . 3
4.2.1 Kernel Parameters . . . 3
4.2.2 Socket Buffer Sizes . . . 3
4.2.3 Listening Backlog (?) . . . 3
4.3 Infiniband . . . 3
4.4 Myrinet Express . . . 3

**5 VFS Layer** **3**
5.1 Maximum I/O Size . . . 3
5.2 Workload Specifics . . . 3

**6 Number of Datafiles** **3**

**7 Distributions** **4**
7.1 Simple Stripe . . . 5
7.2 Basic . . . 5
7.3 Two Dimensional Stripe . . . 6
7.4 Variable Strip . . . 7

**8 Workloads** **7**
8.1 Small files . . . 7
8.2 Large Files . . . 7
8.3 Concurrent IO . . . 7

**9 Benchmarking** **7**

**10 References** **8**

# 1 Introduction

The default settings for PVFS (those provided and in the source code and added to the config files by `pvfs2-genconfig`) provide good performance on most systems and for a wide variety of workloads. This document describes system level and PVFS specific parameters that can be tuned to improve performance on specific hardware, or for specific workloads and usage scenarios.

In general performance tuning should begin with the available hardware and system software, to maximize the bandwidth of the network and transfer rates of the storage hardware. From there, PVFS server parameters can be tuned to improve performance of the entire system, especially if specific usage scenarios are targetted. Finally, file system extended attributes and hints can be tweaked by different users to improve individual performance within a system with varying workloads.

Some (especially system level) parameters can be tuned to provide better performance without sacrificing another property of the system. Tuning some parameters though, may have a direct effect on the performance of other usage scenarios, or some other property of the system (such as durability). Our discussion of performance tuning will include the tradeoffs that must be made during the tuning process, but the final decisions are best made by the administrators to determine the optimal setup that meets the needs of their users.

# 2 Cluster Partitioning

For users that have one use case, and a generic cluster, what's the best partition of compute/IO nodes? Is this section needed?

# 3 Storage

## 3.1 Server Configuration

How many IO servers? [1] How many MD servers? Should IO and MD servers be shared?

## 3.2 Local File System

- ext3
- xfs

## 3.3 Disk Synchronization

The easiest way to see an improvement in performance is to set the `TroveSyncMeta` and `TroveSyncData` attributes to "no" in the `<StorageHints>` section. If those attributes are set to "no" then Trove will read and write data from a cache and not the underlying file. Performance will increase greatly, but if the server dies at some point, you could lose data. At this point in PVFS2 development, server

[1] The FAQ already answers this to some degree

crashes are rare outside of hardware failures. PVFS2 developers should probably leave these settings to “yes”. If PVFS2 hosts the only copy of your data, leave these settings to “yes”. Otherwise, give “no” a shot.

Sync or not, metadata, data coalescing
distributed metadata

### 3.4 Metadata

#### 3.4.1 Coalescing

### 3.5 Data

## 4 Networks

### 4.1 Network Independent

1. Unexpected message size
2. Flow Parameters

- buffer size
- count

### 4.2 TCP

#### 4.2.1 Kernel Parameters

#### 4.2.2 Socket Buffer Sizes

#### 4.2.3 Listening Backlog (?)

### 4.3 Infiniband

### 4.4 Myrinet Express

## 5 VFS Layer

### 5.1 Maximum I/O Size

### 5.2 Workload Specifics

## 6 Number of Datafiles

Each file stored on PVFS is broken into smaller parts to be distributed across servers. The metadata (which includes information such as the owner and permissions of the file) is stored in a metadata file on one server. The actual file contents are stored in *datafiles* distributed among multiple servers.

By default, each file in PVFS is made up of N datafiles, where N is the number of servers. In most situations, this is the most efficient number of datafiles to use because it leverages all available resources evenly and allows the load to be distributed. This is especially beneficial for large files accessed in parallel.

However, there are also cases where it may be helpful to use a different number of datafiles. For example, if you have a set of many small files (a few KB each) that are accessed in serial, then distributing them across all servers increases overhead without gaining any benefit from parallelism. In this case it will most likely perform better if the number of datafiles is set to 1 for each file.

The most straightforward way to change the number of datafiles is by setting extended attributes on a directory. Any new files created in that directory will utilize the specified number of datafiles regardless of how many servers are available. New subdirectories will inherit the same settings. It is also possible to set a default number of datafiles in the configuration file for the entire file system, but this is rarely advisable.

PVFS does not allow the number of datafiles to be changed dynamically for existing files. If you wish to convert an existing file, then you must copy it to a new file with the appropriate datafile setting and then delete the old file.

Use this command to specify the number of datafiles to use within a given directory:

```
$ setfattr -n user.pvfs2.num_dfiles -v 1 /mnt/pvfs2/dir
```

Use these commands to create a new file and confirm the number of datafiles that it is using:

```
$ touch /mnt/pvfs2/dir/foo
$ pvfs2-viewdist -f /mnt/pvfs2/dir/foo
dist_name = simple_stripe
dist_params:
strip_size:65536

Number of datafiles/servers = 1
Server 0 - tcp://localhost:3334, handle: 5223372036854744173
(487d2531626f846d.bstream)
```

# 7 Distributions

A *distribution* is an algorithm that defines how a file's data will be distributed among available servers. PVFS provides an API for implementing arbitrary distributions, but four specific ones are available by default. Each distribution has different performance characteristics that may be helpful depending on your workload.

The most straightforward way to use an alternative distribution is by setting an extended attribute on a specific directory. Any new files created in that directory will utilize the specified distribution and parameters. New subdirectories will inherit the same settings. You may also use the server configuration files to define default distribution settings for the entire file system, but we suggest experimenting at a directory level first.

PVFS does not allow the distribution to be changed dynamically for existing files. If you wish to convert an existing file, then you must copy it to a new file with the appropriate distribution and then delete the old file.

This section describes the four available distributions and gives command line examples of how to use each one.

## 7.1 Simple Stripe

The simple stripe distribution is the default distribution used by PVFS. It dictates that file data will be striped evenly across all available servers in a round robin fashion. This is very similar to RAID 0, except the data is distributed across servers rather than local block devices.

The only tuning parameter within simple stripe is the *strip size*. The strip size determines how much data is stored on one server before switching to the next server. The default value in PVFS is 64 KB. You may want to experiment with this value in order to find a tradeoff between the amount of concurrency achieved with small accesses vs. the amount of data streamed to each server.

```
# to enable simple stripe distribution for a directory:
$ setfattr -n user.pvfs2.dist_name -v simple_stripe /mnt/pvfs2/dir

# to change the strip size to 128 KB:
$ setfattr -n user.pvfs2.dist_params -v strip_size:131072 /mnt/pvfs2/dir

# to create a new file and confirm the distribution:
$ touch /mnt/pvfs2/dir/file
$ pvfs2-viewdist -f /mnt/pvfs2/dir/file
dist_name = simple_stripe
dist_params:
strip_size:131072

Number of datafiles/servers = 4
Server 0 - tcp://localhost:3337, handle: 8223372036854744180 (721f494c589b8474.bstream)
Server 1 - tcp://localhost:3334, handle: 5223372036854744174 (487d2531626f846e.bstream)
Server 2 - tcp://localhost:3335, handle: 6223372036854744178 (565ddbe509d38472.bstream)
Server 3 - tcp://localhost:3336, handle: 7223372036854744180 (643e9298b1378474.bstream)
```

## 7.2 Basic

The basic distribution is mainly included as an example for distribution developers. It performs no striping at all, and instead places all data on one server. The basic distribution overrides the number of datafiles (as shown in the previous section) and only uses one datafile in all cases. There are no tunable parameters.

```
# to enable basic distribution for a directory:
$ setfattr -n user.pvfs2.dist_name -v basic_dist /mnt/pvfs2/dir

# to create a new file and confirm the distribution:
$ touch /mnt/pvfs2/dir/file
$ pvfs2-viewdist -f /mnt/pvfs2/dir/file
dist_name = basic_dist
dist_params:
none
Number of datafiles/servers = 1
Server 0 - tcp://localhost:3334, handle: 5223372036854744172 (487d2531626f846c.bstream)
```

## 7.3 Two Dimensional Stripe

The two dimensional stripe distribution is a variation of the simple stripe distribution that is intended to combat the affects of *incast*. Incast occurs when a client requests a range of data that is striped across several servers, therefore causing the transmission of data from many sources to one destination simultaneously. Some networks or network protocols may perform poorly in this scenario due to switch buffering or congestion avoidance. This problem becomes more significant as more servers are used.

The two dimensional stripe distribution operates by grouping servers into smaller subsets and striping data within each group multiple times before switching. Three parameters control the grouping and striping:

- strip size: same as in the simple stripe distribution
- number of groups: how many groups to divide the servers into
- factor: how many times to stripe within each group

The common access pattern that benefits from this distribution is the case of N clients operating on one file of size B, where each client is responsible for a contiguous region of size B/N. With simple striping, each client may have to access all servers in order to read its specific B/N byte range. With two dimensional striping (using appropriate parameters), each client only accesses a subset of servers. All servers are still active over the file as a whole so that full bandwidth is still preserved. However, the network traffic patterns limit the amount of incast produced by any one client.

The default incast parameters use a strip size of 64 KB, 2 groups, and a factor of 256.

```
# to enable basic distribution for a directory:
$ setfattr -n user.pvfs2.dist_name -v twod_stripe /mnt/pvfs2/dir

# to change the strip size to 128 KB, the number of groups to 4, and a
# factor of 228:
$ setfattr -n user.pvfs2.dist_params -v strip_size:131072,num_groups:4,group_strip_factor:1

# to create a new file and confirm the distribution:
$ touch /mnt/pvfs2/dir/file
$ pvfs2-viewdist -f /mnt/pvfs2/dir/file
dist_name = twod_stripe
dist_params:
num_groups:4,strip_size:131072,factor:128

Number of datafiles/servers = 4
Server 0 - tcp://localhost:3336, handle: 7223372036854744175
(643e9298b137846f.bstream)
Server 1 - tcp://localhost:3337, handle: 8223372036854744175
(721f494c589b846f.bstream)
Server 2 - tcp://localhost:3334, handle: 5223372036854744167
(487d2531626f8467.bstream)
Server 3 - tcp://localhost:3335, handle: 6223372036854744173
(565ddbe509d3846d.bstream)
```

### 7.4 Variable Strip

Variable strip is similar to simple stripe, except that it allows you to specify a different strip size for each server. For example, you could place the first 64 KB on one server, the next 32 KB on the next server, and then 128 KB on the final server. The striping still round robins once all servers have been filled. This distribution may be useful for applications that have a very specific data format and can take advantage of a correspondingly specific placement of file data to match it.

The only parameter used by the variable strip distribution is the *strips* parameter, which specifies the strip size for each server. The number of datafiles used will not exceed the number of servers listed. For example, if the strips parameter specifies a strip size for three servers, then files using this distribution will have at most three datafiles.

The format of the strips parameter is a list of semicolon separated $< server >:< stripsize >$ pairs. The strip size can be specified with short hand notation, such as "K" for kilobytes or "M" for megabytes.

```
# to enable basic distribution for a directory:
$ setfattr -n user.pvfs2.dist_name -v varstrip_dist /mnt/pvfs2/dir

# to change the strip sizes to match the example above:
$ setfattr -n user.pvfs2.dist_params -v "strips:0:32K;1:64K;2:128K" /mnt/pvfs2/dir

# to create a new file and confirm the distribution:
$ touch /mnt/pvfs2/dir/file
$ pvfs2-viewdist -f /mnt/pvfs2/dir/file
dist_name = varstrip_dist
dist_params:
0:32K;1:64K;2:128K
Number of datafiles/servers = 3
Server 0 - tcp://localhost:3336, handle: 7223372036854744173
(643e9298b137846d.bstream)
Server 1 - tcp://localhost:3334, handle: 5223372036854744165
(487d2531626f8465.bstream)
Server 2 - tcp://localhost:3335, handle: 6223372036854744171
(565ddbe509d3846b.bstream)
```

## 8 Workloads

### 8.1 Small files

### 8.2 Large Files

### 8.3 Concurrent IO

## 9 Benchmarking

- mpi-io-test
- mpi-md-test

# 10 References