Perforce Tunables

Michael Shields
Performance Lab Manager
Perforce Software

March, 2010

Abstract

Tunables have been available in Perforce since the 2008.2 release. With help from Perforce, tunables can be used to modify some performance characteristics and limited changes to functionality. Though not strictly supported, Perforce would like to understand the effects of some of the tunables in customer environments.

This paper discusses the importance of working with Perforce when considering the use of tunables. Also discussed is the mechanics of using tunables with the Perforce Server and clients. A strategy for implementing tunables is suggested. The tunables currently available are surveyed, detailing which might have an effect under what circumstances, and which should rarely be used and why.

1 Introduction

Since the first releases of Perforce, there have been significant changes to both hardware and operating systems. Hardware typical of the mid-1990s included 167MHz UltraSPARC I or 166MHz Pentium processors, with perhaps a gigabyte or two of physical memory. Solaris 2.5 and Windows NT Server 3.51 were popular operating systems, and the Linux 2.0 kernel was first released in 1996. The hardware of the mid-1990s seems slow and small compared to current hardware consisting of several multi-core 3GHz Xeons or Opterons with tens or hundreds of gigabytes of physical memory. The performance of the Linux 2.6 kernel makes it a popular operating system on which to run today's largest Perforce Servers.

Perforce products have certainly evolved and improved since their first releases. Every Perforce release usually includes new features, enhancements, and performance improvements. New products, such as the Perforce Visual Client, have also been introduced.

Perforce development has been mindful that not all Perforce customers are running on the latest hardware or the best-performing operating systems. But other customers continually improve their Perforce infrastructure as hardware advances or operating systems improve. Perforce tunables were introduced as a mechanism to fine-tune Perforce for the infrastructure within which it runs. For example, if the Perforce Server is running on a machine that has a large amount of physical memory and effective filesystem caching, performance might be improved by increasing the size allocated to caches within the Perforce server, perhaps at the expense of slightly more physical I/O.

Perforce tunables first appeared in the 2008.2 release. Since their introduction, each release has added a few more tunables. Some of the added tunables, such as the dm.shelve.maxfiles and dm.shelve.maxsize tunables added to the 2009.2 release, were added with a new feature so that Perforce administrators could throttle usage of the new feature if necessary. The tunables discussed in this paper are as of the 2009.2 release.

Each tunable provides a value, such as a number of bytes or seconds, which is used in the code of one or more Perforce products. If the value for a tunable is not specified, the value used is the tunable's default value, which is generally the value that was used in the code prior to the 2008.2 release. Each tunable has a minimum and maximum used to limit the value for a tunable if it is specified. If the value specified for a tunable is less than the tunable's minimum value or greater than the tunable's maximum value, the specified value is silently adjusted to the minimum or maximum value. And some tunables also have an increment. If the value specified for those tunables are not evenly-divisible by the increment, the specified value is silently adjusted up to the next increment.

For example, the default value for the net.tcpsize tunable is 32KB. If not specified, a value of 32,768 is used. If a value for the net.tcpsize tunable is specified, the value is limited by the minimum and maximum values for the net.tcpsize tunable, which are 1KB and 4MB. If necessary, the value specified is either silently adjusted up to the minimum or down to the maximum. The net.tcpsize tunable also has an increment of 1KB. If, for example, the value specified for the net.tcpsize tunable was 65,535, it would be silently adjusted up to 65,536.

The tunables are generally applicable to the Perforce Server. Since some of the code used in the Perforce Server is also used in other products, some of the tunables are applicable to other products as well. If a tunable discussed in this paper is applicable to other products, it will be mentioned in the discussion of the tunable.

2 Working with Perforce

Customers should work with Perforce if considering tunables for changing Perforce behavior. Perforce might have additional information that could help customers considering a particular tunable. Based upon other customers working with the same tunable, Perforce might be able to suggest realistic values and expectations. But more importantly, Perforce might be aware of potential negative consequences resulting from using a tunable in a particular scenario. Or perhaps Perforce can suggest something other than a tunable to better address an issue.

Perforce also benefits from working with customers that are considering tunables. As customers try different values for tunables in their environment, Perforce gets a better understanding of what values for which tunables produce different results under varied conditions. And if a particular value for a tunable results in consistently better behavior for the majority of sites that have worked with the tunable, Perforce might want to change the default value of that tunable. Changing the default value based upon customer feedback might benefit a significant number of Perforce customers.

Given the number of tunables and their range of values, it is not possible for Perforce to test every combination of tunable values. There is therefore some uncertainty when using tunables with values other than their default values. Because of this uncertainty, it is not possible for Perforce to fully support the use of tunables. To be precise, tunables are unsupported and can be changed at any time. But since customers can benefit from using tunables, Perforce would like to help as much as we can.

Customers considering tunables can work with either Perforce Technical Support as part of a current support agreement, or Perforce Consulting as part of an active consulting engagement. Both the Technical Support and Consulting organizations have experience with tunables and have access to other Perforce staff that can also help. For tunables related to performance, Technical Support and Consulting have access to the Perforce Performance Lab organization, which can provide additional assistance and is certainly interested in the results from changing the values of the performance-related tunables.

3 Mechanics of Using Tunables

There are several methods for specifying Perforce tunables depending upon the product. One method is specifying **`tunable-name=value`** in a **`-v`** argument on the command line. Multiple tunables can be specified in multiple **`-v`** arguments on the same command line. For example:

```
p4d ... -v dbopen.nofsync=1 -v dm.batch.domains=1000 ...
```

Some Perforce products use the value of a specific environment variable or registry key as an extension of the command line. The value of the environment variable or registry key can include one or more tunables specified in one or more **`-v`** arguments. For example, the Perforce Proxy uses the value of the P4POPTIONS environment variable or registry key as an extension of the command line:

```
export P4POPTIONS="... -v lbr.bufsize=16384 -v net.tcpsize=262144 ..."
```

Perforce tunables can be specified where some **`name=value`** debugging options are specified. Some **`name=value`** debugging options are specified in the value of specific Perforce debugging environment variables and registry keys. Multiple tunables can be specified in the value of the same debugging environment variables and registry keys. For example (note the absence of the **`-v`** flags):

```
export P4DEBUG="... db.isalive=20000 dm.isalive=100000 ..."
```

Using Perforce products built as of 2009.2/226798 or later, tunables can be specified in a P4CONFIG file (or more precisely, specified in the file named in the value of the P4CONFIG environment variable or registry key, if the file exists in the current working directory or its parents, recursively). Multiple tunables can be specified in the P4CONFIG file, one tunable per line. For example, a P4CONFIG file might include (again, note the absence of the **`-v`** flags):

```
...
filesys.binaryscan=16384
```

```
filesys.bufsize=8192
...
```

The following table shows the methods for specifying tunables in some of the 2009.2 Perforce products. If there is more than one method for a particular product, the methods are listed from highest precedence to lowest. That is, if a particular tunable is specified using more than one method, the value specified in the method with the highest precedence will be used.

Product	Methods for Specifying Tunables
Perforce Server	P4DEBUG environment variable or registry key specified as **`tunable-name=value`** ... -v command line arguments
Perforce Visual Client	P4CONFIG file
Perforce Command-Line Client	P4CONFIG file -v command line arguments
Perforce Proxy	P4DEBUG environment variable or registry key specified as **`tunable-name=value`** ... P4POPTIONS environment variable or registry key specified as **`-v tunable-name=value`** ... -v command line arguments
Perforce Web Client	P4CONFIG file -v command line arguments
Perforce FTP Plug-in	-v command line arguments P4FTPDEBUG environment variable or registry key specified as **`tunable-name=value`** ... P4CONFIG file

Perforce C/C++ API	`#include "debug.h"`
	`...`
	`p4debug.SetLevel( "tunable-name=value" );`
	P4CONFIG file

Values for Perforce tunables can be specified with either a “K” or “M” (uppercase or lowercase) suffix. A suffix of “K” multiplies the value by either 1,000 (10^3) or 1,024 (2^10). A suffix of “M” multiplies the value by either 1,000,000 (10^6) or 1,048,576 (2^20). Whether powers of ten or powers of two is used is dependent upon the tunable for which the suffix is used. In general, tunables that specify some number of bytes will use powers of two. For all other tunables, powers of ten will generally be used.

Perforce administrators can see the tunables and their values specified for the Perforce Server by using the **`p4 tunables`** command. Administrators can see all of the Perforce Server’s tunables and their values by using the **`p4 tunables -a`** command.

4 Implementing Tunables

Because of the uncertainty of using Perforce tunables with values other than their default values, implementing tunables in a production installation should be careful and methodical. The method suggested here should minimize the risk of implementing tunables in production.

Customers considering tunables can use a test Perforce installation to begin researching the effects of a tunable. While the test installation can be small and contrived toward understanding the tunable, it's best if the test installation is a snapshot of the entire production installation. The test installation should be on machines separate from the production installation. Customers with limited resources might be able to use a smaller test installation for their research, but the test installation should not be on production machines. Though using a smaller test installation might provide substantial insight, the best way to understand the effects of a tunable on a production installation is to research the tunable in an environment as close to production as practical.

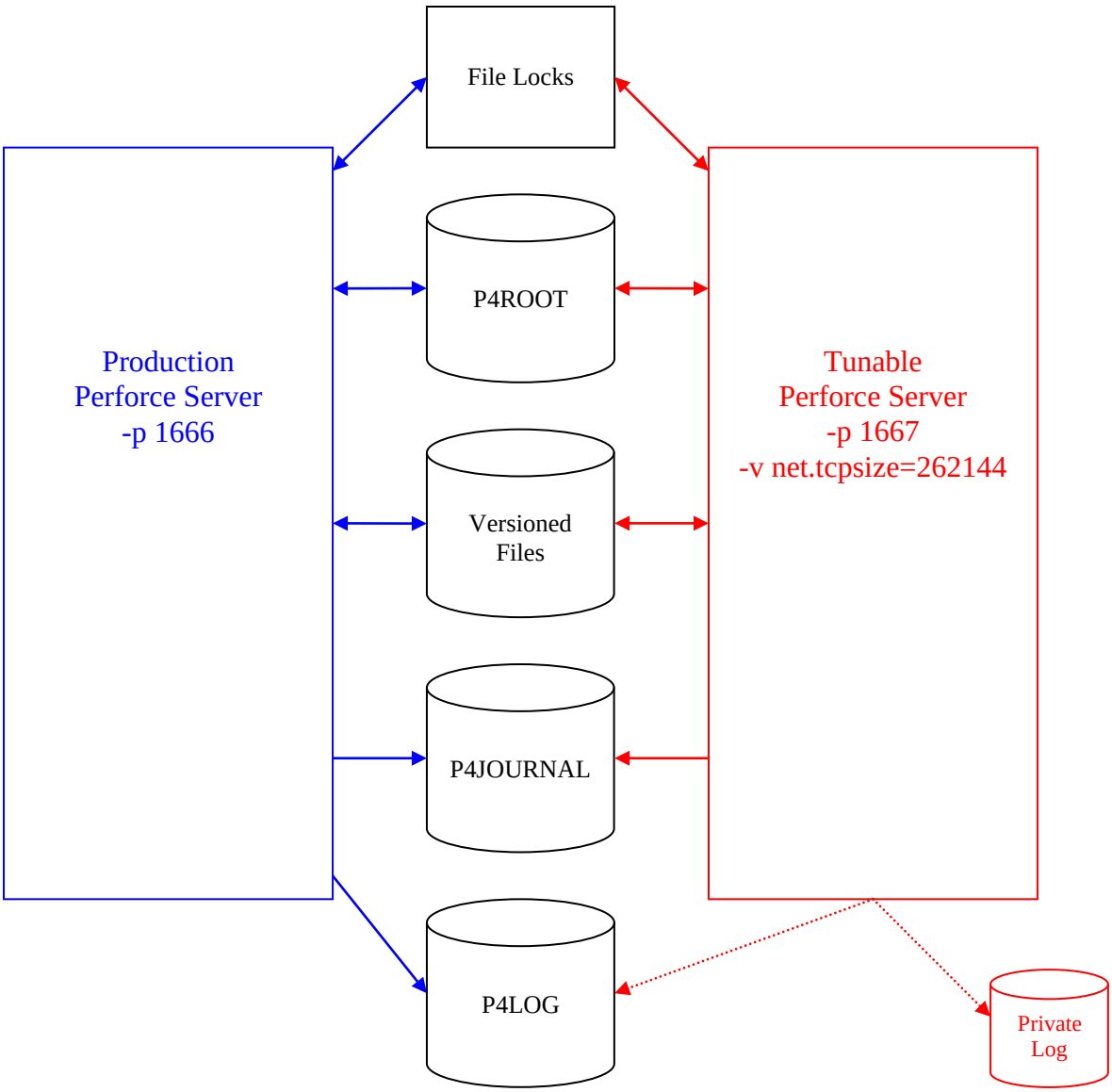
After adequately testing a tunable, customers should contact Perforce before starting deployment of the tunable into their production installation. Even though Perforce might have been contacted prior to testing, it's prudent to do so again prior to deploying into production. Perforce Technical Support or Consulting might have recent information that should be considered before deploying into production.

To help minimize the risk, most tunables applicable to the Perforce Server can be deployed into production in a way that directly affects only a limited number of users. Rather than applying the tunable to the production Perforce Server, the tunable can be applied to a new instance of the Perforce Server that shares most of its resources with the production Perforce Server. With the new instance of the Perforce Server listening on a different port, only users connecting to the new instance are directly affected by the tunable.

The new instance of the Perforce Server must share most of its resources with the production Perforce Server. These include the db.* files, the live journal, and the versioned files. And perhaps most important, the new instance of the Perforce Server must share file locks with the production Perforce Server. Since data synchronization amongst Perforce Server processes is managed by file locks on the db.* files and the live journal, data integrity depends on sharing file locks between the Perforce Servers. Sharing file locks necessitates that the new instance run on the same machine as the production Perforce Server. (On some operating systems, file locks acquired by the Perforce Server cannot be shared between machines. And on other operating systems that can share file locks acquired by the Perforce Server between machines, the resulting performance hit makes it impractical to do so.)

Sharing the Perforce Server's log between the new instance and the production Perforce Server is optional. Sharing the log provides an orderly record of events that have occurred on both the new instance and the production Perforce Server. On the other hand, separating the logs can make it easier to monitor the effects of the tunable on those users connected to the new instance.

The following figure shows the resources shared between a new instance of the Perforce Server on which a tunable is specified and the production Perforce Server.



After the desired behavior has been seen with the tunable deployed in the new instance of the Perforce Server, there are at least two options going forward. If the behavior resulting from the tunable would benefit all production users, the tunable can be specified in the production Perforce Server. The users that had been connecting to the new instance would then connect to the production Perforce Server, and the new instance is no longer needed.

On the other hand, if the behavior resulting from the tunable would not necessarily benefit all production users, the new instance of the Perforce Server can remain as part of the production installation. The new instance would continue serving just those users that benefit from the tunable. As an example, perhaps the tunables specified in the new instance favor users connecting though a Perforce Proxy over a high-latency connection. These same tunables might actually degrade performance for locally-connected users. Local users could continue to use the production Perforce Server without these tunables. Remote users would benefit from the tunables used in the new instance of the Perforce Server to which their Perforce Proxy is connected.

It's certainly possible to have many Perforce Server instances running on the same machine, sharing the same resources, and listening on different ports. Each Perforce Server instance could be configured with a set of tunables optimized for a particular application, usage pattern, or group of users. Connecting to the different ports then results in the customized behavior resulting from the tunables specified on each Perforce Server instance.

5 Survey of Tunables

The behavior of each tunable discussed in this section is as of the 2009.2 release.

db.isalive

Default: 10,000 rows Minimum: 1 row Maximum: 2^31-1 rows

As rows are scanned from a db.* file during a command's compute phase, the Perforce Server periodically checks if the command has held a lock on any db.* file longer than the number of milliseconds specified by the MaxLockTime value in the group definitions applicable to the user running the command. This check is made every db.isalive rows scanned from each db.* file. If the command has held a lock on any db.* file longer than the applicable MaxLockTime value, their command is terminated.

If the user belongs to multiple groups, the server computes their MaxLockTime value as the maximum of the MaxLockTime value for all the groups of which the user is a member (removing the limit if it encounters a setting of **`unlimited`**, but ignoring any settings still at the default value of **`unset`**). If the MaxLockTime value is still at the default value of **`unset`** for all the groups of which the user is a member or the user is not in any groups, their MaxLockTime value is **`unset`**. If the MaxLockTime value applicable to the user is either **`unset`** or **`unlimited`**, their commands will not be terminated as a result of this check.

If the machine on which the Perforce Server runs has a slow I/O subsystem for the db.* files, it's possible that this check is not made often enough, resulting in commands running longer than the applicable MaxLockTime value. Reducing db.isalive will result in the check being made more often at the expense of additional overhead, but commands should be terminated closer to the applicable MaxLockTime value. On the other hand, for better-performing I/O subsystems, increasing db.isalive might reduce the minimal overhead of this check, perhaps at the expense of commands running slightly longer than the applicable MaxLockTime value.

db.trylock

Default: 3 attempts Minimum: 0 attempts Maximum: 2^31-1 attempts

When acquiring locks on a subset of the db.* files needed for some portion of a command, the Perforce Server first attempts to acquire the needed locks without blocking on an incompatible lock. If unsuccessful, the acquired locks are released and the command waits for the

incompatible lock to be released. (Releasing the acquired locks allows other commands that don't need the same incompatible lock to acquire their locks and continue their tasks.) Once the incompatible lock has been released, the Perforce Server then again attempts to acquire the needed locks without blocking. And again if unsuccessful, the acquired locks are released and the command waits. After db.trylock attempts, the Perforce Server then acquires the needed locks, blocking as necessary while holding locks that might block other commands.

Increasing db.trylock might improve concurrency for some users, probably at the expense of other users running commands that must wait for incompatible locks. Additional attempts might result in additional waits for incompatible locks after releasing other needed locks, only to go back and attempt to acquire the needed locks again. Conversely, decreasing db.trylock might result in fewer waits for incompatible locks at the expense of decreased concurrency for others.

dbarray.putcheck

Default: 4,000 rows | Minimum: 1 row | Maximum: 2^31-1 rows

dbarray.reserve

Default: 4 MB | Minimum: 4 KB | Maximum: 2^31-1 bytes

dbarray.putcheck and dbarray.reserve are part of the Perforce Server's memory allocation mechanism that periodically checks to ensure that enough memory is available for future allocations. The default values for these tunables reflect Perforce development's experience tuning these for various operating systems. If these tunables are changed such that they don't accurately report when future memory allocations might fail, a Perforce Server process might terminate abnormally. Unless instructed by Perforce staff, these tunables should be left at their default values.

dm.changes.thresh1

Default: 50,000 revisions | Minimum: 1 revision | Maximum: 2^31-1 revisions

dm.changes.thresh2

Default: 10,000 revisions | Minimum: 1 revision | Maximum: 2^31-1 revisions

The dm.changes.thresh1 and dm.changes.thresh2 tunables affect when an optimization is used for the **`changes [-mx] <files>`** command, where the **`<files>`** argument includes a path without a revision specifier. The optimization scans db.change (descending by changelist number), and for each submitted change scanned, checks the change's root for the possibility that the change includes a file of interest as specified by the **`<files>`** argument. If there is a possibility, db.revcx is then probed to determine if in fact the change includes a file of interest. If the **`changes`** command uses a **`-mx`** argument (as might be used in some build automation), then the scan of db.change ends immediately after returning the desired number of changes that include a file of interest as specified by the **`<files>`** argument.

Without the optimization, all revisions for the **`<files>`** argument are scanned from db.rev. A list of unique changelist numbers, sorted in descending order, are extracted from the scanned

revisions. For each unique changelist number, db.change is then probed for the changelist information. If a **-mx** argument was used, the probes to db.change end after returning the desired number of changes. But all revisions for the **<files>** argument have already been scanned before the probes of db.change begin.

For **changes [-mx] <files>** commands that might use the above optimization, the scope of the **<files>** argument is first determined. If the scope of the **<files>** argument is wide, that is, consists of dm.changes.thresh2 or more mapped revisions, then the optimization is used. To determine the scope, an initial scan of **<files>** is started using db.rev. The initial scan terminates if all of **<files>** revisions or dm.changes.thresh2 mapped revisions have been scanned. If dm.changes.thresh2 mapped revisions were scanned, then the optimization is used. But the initial scan will also terminate if dm.changes.thresh1 unmapped revisions are scanned before dm.changes.thresh1 mapped revisions. (Revisions can be unmapped in either the protections table or the client view (if **<files>** is expressed in either client or local syntax).) If terminated as a result of the unmapped revisions, the optimization is used if dm.changes.thresh1 is greater than or equal to dm.changes.thresh2.

dm.integ.maxact

Default: 100,000 credits Minimum: 1 credit Maximum: 2^31-1 credits

The dm.integ.maxact tunable is the maximum number of actionable credits pursued for each file by the second generation integration engine. Reducing the value for this tunable might result in fewer positions and scans of db.integed, perhaps at the expense of not finding a fully-credited path. If a fully-credited path does exist but is not found as a result of this tunable set too low, the file might needlessly be opened for integrate and perhaps no content will need to be changed.

For files with a complex integration history, the random reads of db.integed can be resource intensive. Reducing the positions and scans of db.integed might be advantageous for sites with a Perforce Server running on a machine with limited physical memory or a poorly-performing I/O subsystem. But if the dm.integ.maxact tunable is reduced such that fully-credited paths that exist are not found, users will be impacted by additional files (needlessly) opened for integrate.

dm.isalive

Default: 50,000 rows Minimum: 1 row Maximum: 2^31-1 rows

During a command's compute phase, as rows are scanned from a db.* file or records are put into a results array, the Perforce Server periodically checks if the client application that started the command has dropped the connection and if monitoring is enabled, checks if the command has been marked for termination by an administrator using the **p4 monitor terminate** command. These checks are made every dm.isalive rows scanned from each db.* file and every dm.isalive records put into each results array. If either the connection has been dropped or the command has been marked for termination, the command is terminated as a result of these checks.

A select() call is generally made to check if the client application that started the command has dropped the connection. A probe of the db.monitor table is required to check if the command has been marked for termination by an administrator using the **`p4 monitor terminate`** command. The overhead resulting from these checks can be reduced by increasing dm.isalive, perhaps at the expense of commands running longer after the connection has been dropped or the command has been marked for termination.

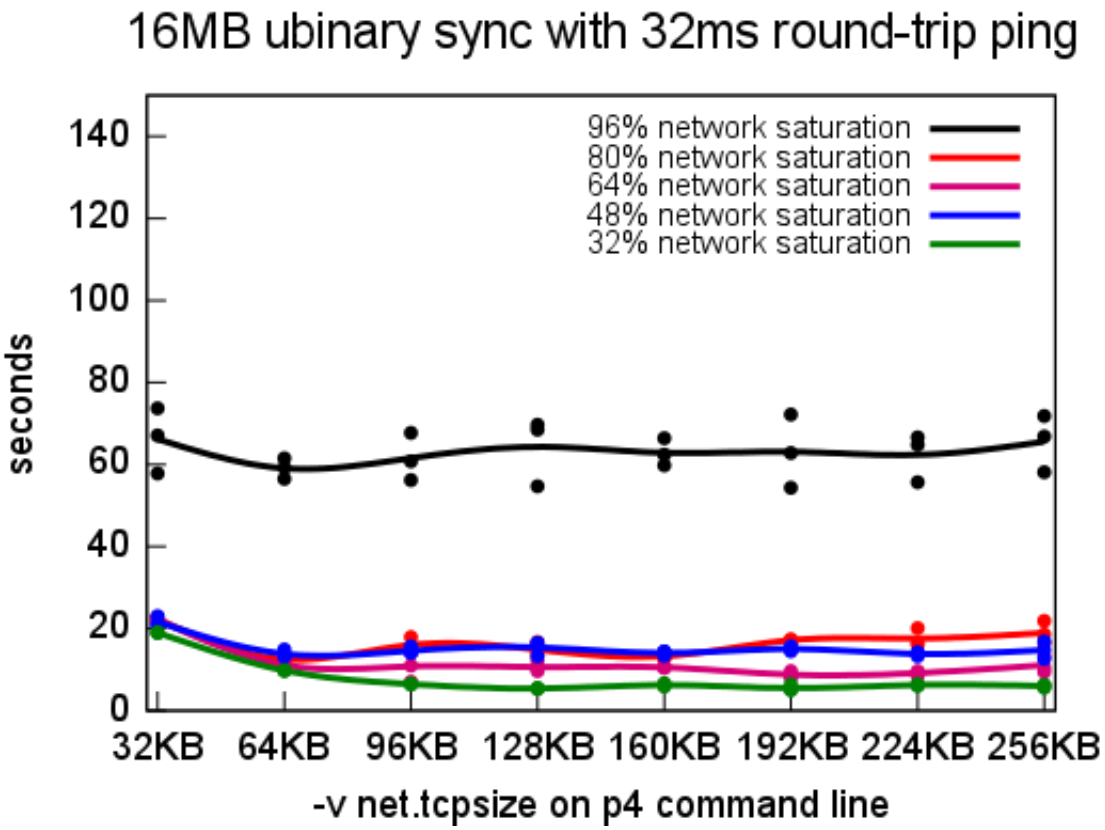
net.tcpsize

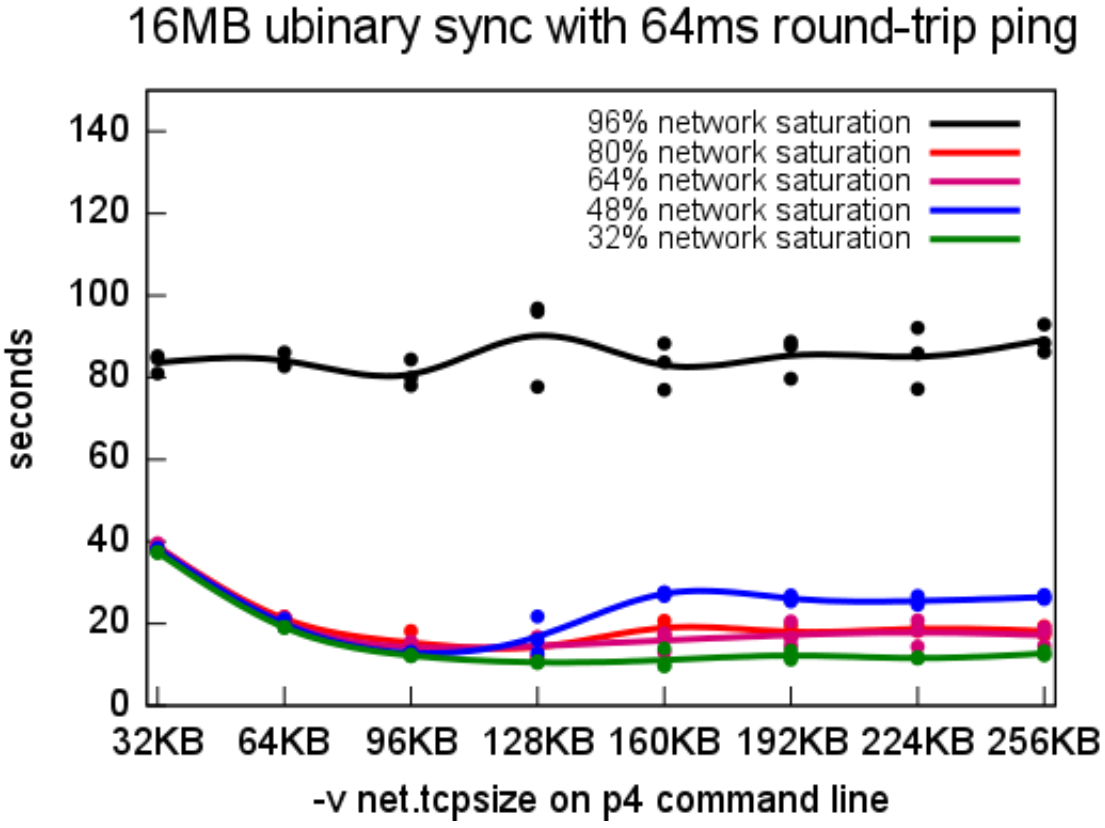
Default: 32KB Minimum: 1 KB Maximum: 4 MB

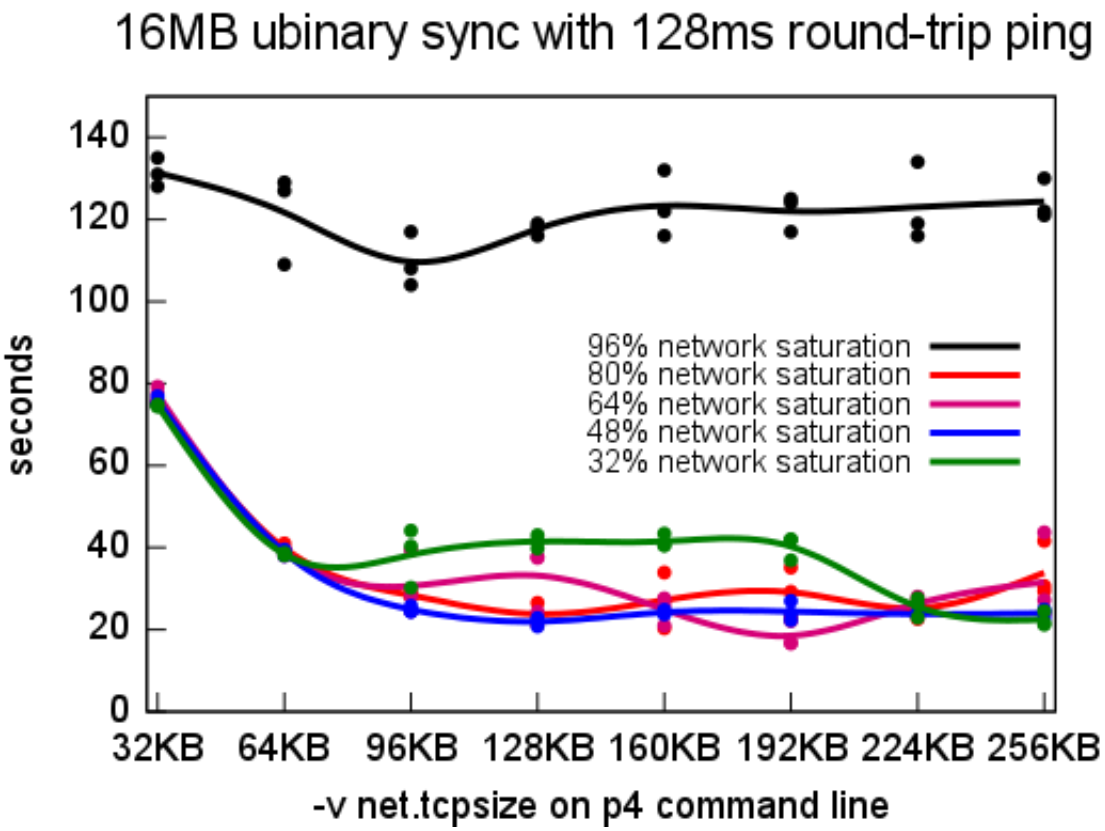
Prior to each socket connection made with the Perforce C++ API, the size of the send and receive buffers for the socket might be increased. The API first determines the size of the send and receive buffers as set by the operating system. If either buffer's size is less than net.tcpsize, the size for the buffer is increased to net.tcpsize (if allowed by the operating system). Since this tunable is implemented in the Perforce C++ API, it can be used with Perforce clients and the Perforce Proxy.

Most operating systems have their own tunables for the sizes of the send and receive buffers for socket connections. The default sizes of the send and receive buffers in some operating systems are small, resulting in decreased throughput. For example, in Windows XP, the default sizes of the send and receive buffers are at most 8KB. (These default sizes can be adjusted in Windows using the DefaultSendWindow and DefaultReceiveWindow registry keys.) The Perforce C++ API increases the size of small send and receive buffers to net.tcpsize, if the operating system allows the increase. Some operating systems have tunables for the maximum sizes of the send and receive buffers, potentially limiting the increase. For example, in Linux, the maximum sizes of the send and receive buffers are limited by the net.core.wmem_max and net.core.rmem_max sysctls, respectively.

Increasing net.tcpsize can result in increased throughput. The following graphs show the performance improvement resulting from increasing net.tcpsize on the Perforce Command-Line Client for higher network latencies to the Perforce Server with varying network saturation levels. Each of the graphs show the number of seconds needed to sync a single 16MB ubinary file as net.tcpsize is increased for the various network conditions. Windows XP SP3 was the client platform, on which the 2009.2/238357 Perforce Command-Line Client was used. Linux 2.6.16 x86_64 was the server platform, on which the 2009.2/238357 Perforce Server was used.







For higher network latencies between the Perforce Command-Line Client and the Perforce Server, the graphs show that for reasonable network saturation levels, throughput increases, up to a point, as the net.tcpsize tunable increases. At the point where throughput no longer increases, there might be a bottleneck elsewhere limiting further performance improvement. For example, perhaps increasing net.tcpsize on the Perforce Server might allow continued increases in throughput as net.tcpsize increases on the Perforce Command-Line Client.

Some deployments of the Perforce Proxy have higher network latencies to the Perforce Server. Increasing net.tcpsize on both the Perforce Proxy and the Perforce Server might improve performance for operations through the Perforce Proxy that deliver file content not already cached. The net.tcpsize tunable might need increased significantly, possibly necessitating an increase of the operating system's tunables for the maximum sizes of the send and receive buffers for socket connections.