Perforce Standardisation at Citrix

Coping with Change in a Growing Global Organisation

Jason Leonard & Lee Leggett, Citrix Systems

Abstract

This white paper describes the Perforce standard environment (PSE) created at Citrix Systems to aid and simplify the management and administration of Perforce instances.
Introduction
The purpose of this white paper is to describe the Perforce standard environment (PSE) created at Citrix Systems to aid and simplify the management and administration of Perforce instances. It will cover the historical setup employed at Citrix for more than 10 years as well as the new implementation linked to the development of the PSE. This will be followed by a description of the syncing and building processes employed by Citrix, in part driven by some of the complexities discussed in the past implementation. Then we will fully describe the PSE. The paper concludes with a look at all the future improvements planned for the PSE and the general Perforce implementation at Citrix.

Citrix Perforce History
Citrix has been a customer of Perforce for more than a decade. As a result, many of the early practices and recommendations have been followed with little deviation. As Perforce has grown, best-practice recommendations have naturally evolved as well. New practices and ways of thinking, however, sometimes can meet with a lot of resistance in an established environment. How Perforce was implemented and run changed very little at Citrix. New Perforce instances continued to be created and product dependencies between these different instances magnified exponentially. The result was management problems for our administrators and frustration from our end users.

Example of Hardware Implementation
This example has been taken from one of the Citrix offices. It starts by defining an old hardware implementation, describes what problems were encountered, and ends with a new implementation, PSE, that is currently in use. The PSE was created for the new implementation (as will be described later).

Previous Implementation
This hardware specification was in use until around 2010.

Physical Server

- Rack-mounted server
- Windows Server 2003
- 4 GB RAM
- 350 GB HDD organised in a RAID 5 array

Perforce Instance Configuration

- 7 Perforce master instances running on local hard disk drive (HDD):
  - 5 of which linked via an authorisation server
  - 1 of which used external authorisation via Active Directory LDAP; specifically used as a test bed for Perforce version updates and trigger scripts
- 22 Perforce proxy (p4p) instances pointing to other Citrix Perforce instances at other sites. All hosted on local HDD.
- Total licensed user count of nearly 2,000, around 150 local heavy users including the automated build system in both the United Kingdom and United States
Performance
The following sync, branch, and resolve examples are all based on a sample 5 GB area. Some sync time examples are given below:

<table>
<thead>
<tr>
<th>Remote site</th>
<th>Local site</th>
<th>Remote site using proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 hours</td>
<td>45 mins</td>
<td>55 mins</td>
</tr>
</tbody>
</table>

Average p4 branch time: 2 mins
Average p4 resolve time: 3 mins
Sequential read to disk:
Sequential write to disk:
Longest checkpoint: 1.5 hours
Longest verify: 4 hours

Problems Encountered with the Previous Implementation
There were several issues with the old implementation encountered at multiple sites.

Perforce Server Downtime
This was largely due to checkpoints and other database intensive commands. With the size of the Perforce instances at certain sites, checkpoints could easily last 16 hours or more. This meant that certain instances were down for an entire day at the weekend. Although this is potentially acceptable during the middle of a project, it quickly becomes intolerable as the release date approaches. In some sites, checkpoints were fast enough to run every day; the longest checkpoint took around 1.5 hours. However, once the company expanded to include development sites in Australia, China, India, Japan and the U.S. West Coast, it became impossible to select a checkpoint time that didn’t affect a development team somewhere.

System Stability
The Perforce instances were running on a 32 bit OS (Windows Server 2003), which meant that any one process had a 2 Gb memory limit. With the number of commands being run against certain servers, the p4d process was reaching this limit, which caused subsequent commands to stall or fail completely. This problem was simply getting more severe as time progressed.

Disaster Recovery
Regular checkpoints were performed and tape backups of these as well as all the versioned files were kept. However, with the hardware available, test restores were not a regular occurrence.

Complexity & 24/7/365 Support
With little standardisation between sites, misconfigurations were common. A significant part of the time spent solving a Perforce issue became learning how a particular Perforce instance was configured rather than resolving the problem.

Perforce Knowledge
With a distributed part-time administration team, the level of Perforce experience varied wildly. This meant that advanced administration of the Perforce instances became very difficult. Issues raised included: What options do I pass to the checkpoint command? How do I do a restore? What happens if I set this p4 configurable? Citrix needed a way to simplify the
Perforce administration experience for less experienced users without losing some of the in-depth technical knowledge from the more experienced administrators.

**Performance**

Citrix has a global distributed workforce accessing Perforce instances and syncing files from one geographic location to another. Users would often complain of slow client application and sync times. A classic example of this comes from the way Citrix stores its toolset. Citrix has a common set of build tools placed in Perforce for compiling most Citrix products. This toolset has grown to exceed 30 GB of data, and is currently held in one of our U.S. sites. Users from any other geos syncing all of these tools could lose around 3.5 hours waiting for the sync to complete. The use of proxies has helped reduce this problem dramatically, but the issue still remains for the administrator. With most users required to sync all the tools before they set to work, the ‘have’ tables on this Perforce instance grow very large. Given the 32-bit OS problem, we end up with a memory swapping issue causing increasingly bad performance.

**Revised Implementation**

The following specification is in use today in one of the U.K. offices:

**Physical Server**

- Rack-mounted server
- Windows Server 2008 R2
- 16 GB RAM
- 450 GB HDD organised in a RAID 5 array (due to limited spindles). Two separate partitions, one for the journal and the rest (430 GB) for Perforce metadata of the local p4d processes.
- A further 800 GB is connected via iSCSi from a SAN device and contains the version files

**Perforce Instance Configuration**

- 7 Perforce master instances running on local HDD
  - 5 of which linked via an authorisation server
  - 1 of which used external authorisation via Active Directory LDAP, also used as a test bed for Perforce version updates and trigger scripts
- 22 Perforce proxy (p4p) instances pointing at other Citrix Perforce master instances. The proxy cache files are hosted on the SAN storage device.
- Total licensed user count of nearly 2,000, around 150 local heavy users including the automated build system in both the United Kingdom and United States

**Performance**

The following sync, branch, and resolve examples are all based on a 5-Gb sample area. Some sync time examples are given below:

Remote site | Local site | Remote site using proxy
---|---|---
3 Hours | 30 mins | 35 mins

Average p4 branch time: 20 seconds
Average p4 resolve time: 30 seconds
Longest checkpoint: 45 mins
Longest verify: 1 hour 40 mins

As these examples illustrate, the improvement in sync times is modest but the improvement in other database intensive commands such as resolves and verifies is massive. The overall stability of the system has also been greatly improved with a marked decline in Perforce problems reported by end users.

**Users’ Interaction with Perforce**

Over the years some interesting solutions to this Perforce instance explosion have surfaced. The next few sections describe the problem and some of the efforts to solve it.

When something multiplies exponentially without any control, it causes massive knock-on effects for whatever environment it is multiplying into. In terms of Perforce, we are talking about groups of isolated individuals who, with the best intentions, decide to put their own Perforce server into production in a company where working in silos was the norm.

Over time our company ethos has evolved, and a big push towards product integration has started to break down these silos.

Fortunately a core group of Perforce server owners stayed in contact from the outset and had begun to implement some process over the Citrix Perforce architecture. These individuals put together an idea based around how one might view Perforce from a high-level perspective, by only having one piece of information that uniquely identifies the server instance.

**Perforce Mesh Network**

Usually a user needs two pieces of information in order to connect to an instance—the hostname of the server that runs the instance and the port number on that machine. The default port number for Perforce is 1666, but this can be easily changed. What if the port number was the unique component? This would mean that a user could identify the instance with only a port number. But what of the hostname? This question becomes more interesting when we think about another Perforce technology, the proxy.

The Perforce proxy is a piece of Perforce technology that will redirect users’ commands to the master instance, but will cache a local copy of any file data that might travel across the connection for speed improvements for later sync requests.

![Figure 1: All ports available on all servers](image-url)
Suppose we have two machines and each runs a Perforce server instance (see Figure 1). Suppose each of these machines is situated in a different country and assume that some sort of WAN connects them. Each of the instances has a unique port number, but we also run a Perforce proxy on each of the machines making the missing port available. Now, it doesn’t matter which hostname the user employs, the instance is still accessible. Of course, there is a small performance improvement if the user uses a server that is located nearby.

If we now expand this idea into a multi-instance, multi-server, multi-location environment, we reveal a mesh network of Perforce services (see Figure 2). The users need only know their local Perforce server hostname and then provide whichever port they wish to connect to.

At Citrix, how we number the ports is important but mostly from an administration point of view. We use a fairly easy scheme to identify which site the instance is at and some small indication of the usage. Each port uses 4 digits, just like the default port number for Perforce, but the first digit describes the geographic location. For example, 1 = United Kingdom, 2 = West America, 3 = East America, 4 = Australia, 5 = China, and so on. To the user it’s all transparent, but from an administrator perspective it serves as a reminder.

**Multi-Port Problems**

Eventually, no matter how hard you try to keep products built out of one port, through mergers and internal reorganisations you will find that products will build out of multiple ports. For Citrix, it didn’t take long before this started to happen. Unfortunately it causes a cascade effect on tools and systems. One example of this is the build system.

We modified our build system to control the multi-port issue. But this small change led to some interesting build numbers (e.g., 112233#443322). Normally the users would know which port their product source code was in; with only one changelist number this was easy. But now we have a changelist for each of the ports the product source is located on. In this example we see one at change 112233, and the other at change 443322. To decode the combined build number, more information is needed—the port numbers and the order in which they appear in the build number. So by adding the following port ordering string—1666#2666—we can match
up the changelist and the port.

What happens now if a developer makes a change to the code that affects multiple ports? This is where things can get complicated. It’s up to the developer to figure out where the source code came from and separate the change on the ports that the code change affects. This has caused lots of frustration in the past and continues to do so.

**Solutions**

Two years ago, at the San Francisco 2011 Perforce user conference, a colleague of ours presented "Creating a World-Class Build System, and Getting It Right. It covered in-house techniques Citrix has developed to fulfill the engineering build requirements. The next evolution of this over the years has been a rebranding and consolidation exercise to present our developers with a standard end-to-end build system called “Solera”. It continues to be internally developed and has the following five parts:

- Solera Sync
- Solera Build
- Solera Controller
- Solera Release
- Solera Layout

Each part is very distinct and covers a specific area of the build system. Because our focus here is on Perforce at Citrix, we will describe only Sync and Controller.

**Solera Sync**

Solera Sync tries to reduce the complexity of multi-port syncing by providing a way for us to describe (using configuration files) what a product component requires in the way of inputs for it to build successfully. The inputs are typically source code but could as easily be SDKs and tools, including compilers.

Each of the product components has a unique name usually made up of the component’s name and the branch. For example, the Solera Sync mainline code could have a component name of solerasync_main. For users to obtain the correct build environment and source code for this component, they would simply instruct Solera Sync to sync ‘solerasync_main’.

An interesting side effect of doing this syncing is that we have the opportunity to insert some extra information into folders about where the files come from. We make use of P4CONFIG files so that if you were using the p4 command-line, you could easily submit files without having to remember which port the source code came from.

Solera Sync also helps with determining the best way to obtain the inputs required. The Perforce server hostnames are site specific. Therefore with a little knowledge of the Citrix internal network, its subnets, and geo time zones, we can determine the correct Perforce server to use for maximum performance.

**Solera Controller**

This part of Solera is at the heart of the build system and is the automated continuous integration (CI) engine.

Ever since Citrix has been considering the cloud and what that means for its technology, a group of build engineers has debated the merits of viewing the controller as a cloud controlling
technology. They have considered how to decouple systems from source control and challenged the ideas of fixed infrastructure machines in favour of a rich and flexible system that is almost organic in nature.

This is how we view the next generation of CI engines, and with the help of the virtualisation technology Citrix has built up over the years and its talented engineers, we believe that this vision is our future.

Solera Controller builds on the ideas of Solera Sync and therefore gains the simplicity of syncing our products. However, it must still keep some control over the syncing process because the controller needs to keep a track of what inputs it used in the construction of any of our builds for reproducibility reasons.

**Reporting Services**
A number of Citrix tools can extract data from our source and build servers and display it in a variety of different ways. Historically it’s been hard to truly visualise how our products are built, particularly when they are made up of smaller components, SDKs, and libraries that could be built in many other geographies and build systems. If changes go into one of the SDKs, testers need to know when they can test the product for the fix.

‘Sniff’, one of our newest engineering tools, was developed by a Citrix engineer for this very purpose and has quickly become one of the handiest tools in our engineers’ toolboxes. It collects data from all of our Perforce instances, collates it with the data from our build systems, and pulls in any extra metadata from the various control files we have dotted around. It allows any engineer to pull up and drill down on any of these items. It can even draw diagrams that show how a change to one component gets pulled into other components and eventually bubbles up until it’s on one of our DVDs. For a test engineer this tool has helped to keep focus and ensure effort isn’t wasted.

**Citrix Perforce Standard Environment (PSE)**
The PSE was created to solve a myriad of problems plaguing the implementation and administration of Perforce at Citrix.

Over time the company has grown to incorporate other sites that own Perforce servers. This led to the need for a common environment that everyone understood and let non-advanced Perforce administrators easily perform operations on servers.

A major driving factor for needing to solve the administration problem came about because of the loss of Perforce knowledge within a key team. This team was seen as a thought leader when it came to Perforce, particularly one individual who had been using Perforce since its inception. It had developed many scripts using advanced ideas and techniques that quickly became unsupportable. A new set of admins made up of mostly beginners or intermediates attempted to pick up the pieces, but quickly the decision was made to start afresh with a system that all administrators could understand and use effectively and confidently.

After reading lots of white papers and information on the Perforce website, a team set about creating an administration environment that fitted Perforce for Citrix. And so the Citrix Perforce Standard Environment (PSE) was born.

**Overview of the PSE**
The PSE is fundamentally a set of scripts and configuration files supporting the running of multiple Perforce instances on a single machine.
The PSE defines three types of Perforce instances:

1. A “Root”: This would be a standard p4d Perforce instance. Sometimes called a master.
2. A “Proxy”: This is a standard p4p (Perforce proxy) instance pointing at a “Root”.
3. A “Replica”: This is a p4d instance that is configured as a replica of a “Root”.

The PSE also can support a “multi-version” environment. This means that each Perforce instance controlled by the PSE can be running a different version of the Perforce software. For example, one could be at 2012.2 while another is running at 2011.1. Because of the large number of Perforce instances in Citrix, the ability to upgrade one piece at a time is a necessity. This certainly does not mean that Citrix should be running several different Perforce versions at once; it simply means that upgrades can be rolled out and tested in a structured way with the ultimate objective of all the Perforce instances at least at one location being the same Perforce version.

**PSE Configuration Files**

The PSE has two key configurations files. The first, config.txt, describes how the machine is configured, where instance artefacts are to be stored, as well as defaults values for certain actions. The second, site.txt, describes which instances are to be serviced by the machine and how they are to be run.

**Config.txt**

Figure 3 presents an example of the type of information this file contains.
Figure 3: config.txt example

BinBasePath = C:\Perforce
JournalBasePath = D:
LogBasePath = D:
MetadataBasePath = E:
VersionBasePath = F:

These paths describe the base paths of each of the artefacts required by the Perforce software. This allows the flexibility to define different types of storage for the artefacts according to their needs. For example, the metadata is best on very fast access drives, while the journal is best on a sequential write optimised file system.

PathSep = \\n
This allows the paths formed by the PSE scripts to support different platform conventions.
Perforce Standardisation at Citrix

P4Roots = p4roots

P4Proxy = p4proxy

P4Replica = p4replica
For each instance type supported by the PSE, a corresponding folder is created under each of the base paths. This means that when viewing the folders with a file browser, it is clear what the instance type is.

Under each of these instance type folders, port number folders are created containing the actual artefact files for the instance.

For example, a path labelled E:\p4roots\1666 would contain the metadata or database files for port 1666, which is a root or master instance.

P4Progs = bin
This path is concatenated to BinBasePath to form a path that describes where the p4 executables downloaded from Perforce.com will be stored.

Licenses = license

LicenseFiles = license.10.30.*.*
The “Licenses” path is concatenated to BinBasePath to form a path that describes where the license files for the Perforce server live. The “LicenseFiles” path describes which of the license files to use. This allows slightly better control of license files requested from Perforce.

NagiosServer = *********

NagiosPort = ******
Nagios is used to monitor the Perforce machines. However, to monitor a scheduled process, Nagios recommends the use of passive checks. This means that once either a checkpoint or a verify is complete, the script will contact the Nagios server to supply the result.

Checkpoint = online
The “Checkpoint” value simply controls what the default implementation for checkpoints is—that is, whether checkpoints happen live (online) or on a replica server (offline). This can be overridden in the site.txt file on a per instance basis.

RollOverExtension = _log.txt.gz

RollOverToKeep = 5
The PSE keeps log files of upkeep tasks such as checkpoints and verifies. The rollover values control what file extension to add to previously run log files and how many of these logs to keep.

CheckpointSchedule = Sun|Mon|Tue|Wed|Thu|Fri|Sat#1#01:00:00

VerifySchedule = Sun#1#03:00:00
The final items control on what days and times checkpoints and verifies occur. So in this example, checkpoints occur every day at 1 a.m. and verifies are run each Sunday at 3 a.m.

Site.txt
This file controls configuration of the particular Perforce instances that run on the machine.
Each instance includes the port number, the version of the Perforce software to use, and the logging level to use. Both proxies and replicas always have a pointer to their corresponding root or master instance, which could be on the same or different machine. Roots have the optional ability to point to an authorisation instance. Overrides are used to change default values for particular features (see Figure 4).
**Port Number**

The port number used by the Perforce software to expose the service to users must be unique. It is also used when executing PSE scripts to identify which port to perform operations on.

**Perforce Version**

This field is used to identify the Perforce version to use when running the Perforce software. No provision is made for patched Perforce software.

**Type**

The type identifies how the PSE will treat the port when executing certain scripts. Currently this field can take on one of the following values: “root”, “proxy”, “replica”.

- Root ports use p4d and enable scheduled tasks for checkpoints and verifies.
- Proxy ports use p4p and disable most port management scripts that are meaningless.
- Replica ports use p4d as in roots, but don’t add checkpoint or verify schedules.

**Auth Port**

This is specifically for root ports and specifies the location of the authorisation port that p4d should use when authenticating users checking permissions and group membership.
Proxy Port
This specifies the port for the proxy server.

Master Port
This is specifically for replica ports and indicates the port that the replica server is to pull metadata and/or version files from. It also provides more convenience when using the restore script to restore port metadata from a checkpoint on another machine also running the PSE.

Log Level
This field allows the administrator to control the amount of logging provided by the Perforce software. The logging is written out to the log path defined in the config.txt.

Overrides
This field gives the administrator more control of exactly how the PSE will run the port, by changing the configuration of the features provided—for example, offline checkpoints and named configuration (P4NAME).

Examples
The configuration file in Figure 4 shows that instance 2266 is a master port version 2011.1, which doesn't have an authorisation port and is run at log level 0. Instance 2244 is also version 2011.1 at log level 1, but it does use an authorisation instance. Instance 1279 is version 2012.2, also a master instance, but it uses an override and overrides the online checkpoint set in config.txt and performs an offline checkpoint instead using instance 1279 on server Chfofflineserver.

Using the PSE Scripts
The following instructions demonstrate the PSE scripts. They start by configuring PSE for a new port, then go through the steps to enable, run, and finally perform other operations on the port.

Once two configuration files have been populated and a starting Perforce version has been downloaded, it's possible to create a Perforce instance using the scripts that come as part of the PSE. A walk-through of this process follows.

Configuring PSE for a New Port
The site.txt needs to be edited to include the new Perforce instance to be run:

<table>
<thead>
<tr>
<th>Port</th>
<th>Version</th>
<th>Type</th>
<th>AuthPort</th>
<th>LogLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>2211</td>
<td>2012.2</td>
<td>root</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Set Up and Run the Port
As a first step, ensure that the latest hotfix of the required Perforce version is on the server:

download.pl 2012.2

Or

download.pl 2211

Once this is available, the admin then needs to run schedule.pl for the specified instance. This will create the Windows scheduled tasks to run the port, checkpoint, and verify. Note that the PSE actually takes a copy of the downloaded p4d.exe and renames it by appending the port number. This allows the administrator to better identify which p4d.exe corresponds to which
port in the Task Manager processes list. For example, for Perforce instance 2211, the p4d executable would be named p4d-2211.exe:

```
schedule.pl 2211
```

Next the Perforce instance needs the windows firewall opened so that users can access it:

```
firewall.pl 2211
```

Now the port can be started. We can use the schedule script again, but this time instructing it to run the schedule, not create it:

```
schedule.pl 2211 --run
```

A new Perforce instance is now running on port 2211 and is available to users.

**Stopping the Port**

If an administrator needs to stop access to a Perforce instance, then rather than stopping the port and trying to run it on “localhost:port”, the firewall can just be closed on that port while keeping the Perforce instance running:

```
firewall.pl 2211 --delete
```

To remove a Perforce instance, only two commands are needed:

```
schedule.pl 2211 --end
schedule.pl 2211 --delete
```

These commands, however, will not remove the metadata or versioned files from the HDD of the server; the admin would have to manually delete those folders. This functionality hasn’t been added as a deliberate safety measure; making deletion of all Perforce instance data easy was considered too risky.

**Performing Other Port Operations**

If an upgrade of the Perforce instance is required, then the following command can be run:

```
upgrade.pl 2211 2013.1
```

Upgrade.pl performs several functions here. The first step is to p4admin stop the instance. Then a checkpoint is performed; once this is complete the actual upgrade is performed and the new version automatically written into site.txt. Next a checkpoint is taken post-upgrade and if this is successful it will perform a restore of that checkpoint. This step ensures that any large deletion of files/clients are removed from the db.have data table.

In the PSE checkpointing, a Perforce instance is a case of simply running a single command:

```
checkpoint.pl 2211
```

The actual checkpoint mechanism can be configured differently for each port. The checkpoint will either happen “online”, which will momentary lock the database tables, or “offline”, which will perform the checkpoint on a replica of this port and therefore not cause any downtime.

If administrators want to restore from a checkpoint, they have two options: Restore a specific checkpoint or the “latest” one. To restore a specific checkpoint, the administrator simply runs:

```
restore.pl 2211 <full checkpoint filename>
```

To restore the latest checkpoint, simply replace <full checkpoint filename> with “latest”.
To verify a Perforce instance outside of the normal scheduled verify, the following command is needed:

```
verify.pl 2211
```

**Offline Checkpointing**

Checkpointing a Perforce instance that is configured to use an offline checkpoint server is handled differently in the PSE, even though the command is the same. Figure 5 illustrates the process. First, note that the replica port configured to actually perform the checkpoint proper is set to pull metadata from the root port using the “p4 pull” command. The root port also needs to have its “checkpoint” value in the configuration set to the hostname and port number of the replica offline checkpointing server. By executing the PSE checkpoint script as normal, the checkpoint proceeds as follows:

1. The replica port is told to “schedule” the checkpoint, with the standard “p4 admin checkpoint” command.
2. The root port now needs only to rotate the database journal, which causes the replica port to pull over the database changes, detect the rotation, and perform the checkpoint.
3. The script then waits for the MD5 file from the checkpoint to be created; because this is the last file created by the checkpoint process, it is seen as the end of the checkpoint.
4. The checkpoint files are then copied to the root port version file location as they would normally do during an online checkpoint.

![Figure 5: Offline checkpoint procedure](http://www.perforce.com/perforce/doc.current/manuals/p4sag/10_replication.html)

Upgrading an offline checkpointed Perforce instance is the same as the usual upgrade process, except that the offline checkpoint server must be upgraded before the live server. This enables the offline checkpoint server to handle journal entries made in the old or new version. Also when the upgrade of the main instance is performed, the checkpoints that occur as part of the upgrade are all performed online, not offline. This is done to both simplify the upgrade process and give some online checkpoints that can be contrasted with offline ones to

---

1 Configuration details can be found here: [http://www.perforce.com/perforce/doc.current/manuals/p4sag/10_replication.html](http://www.perforce.com/perforce/doc.current/manuals/p4sag/10_replication.html)
ensure that everything is working correctly.

**PSE in Citrix**
The PSE has been in production at the U.K. site for nearly one year, although offline checkpoints have only recently been introduced. The benefits noticed by the U.K.-based Perforce administrators have included faster issue resolution, less downtime in a disaster recovery scenario, and more simplified administration and monitoring. Since the initial phase in the United Kingdom, the PSE has now also been rolled out in the India, China, and U.S. offices. Further rollouts to all other Citrix development sites are planned.

Between the two U.K. offices a disaster recovery event was simulated. One site needed to bring up all the Perforce instances hosted there in the other site. With the use of the SAN replication technology and the PSE, all Perforce instances were restored in an hour. Without the PSE, this would have taken significantly longer.

**Futures**
Recent new features of Perforce have truly opened some interesting paths for us to explore and opportunities for us to innovate. Ultimately we want to address the hard problems facing us in order to get us into better shape for the future.

**Merging Ports**
Since attending the Perforce RoadShow events, we have discussed some interesting ideas around the possibility of merging Perforce ports. Although on the surface, this sounds like an easy task, in reality, it is not. Considerations about other services that use Perforce as an information repository have to be taken into account. They include change review tools, build databases, e-mails, internal technical documentation, and configuration files. Editing all these links would be a massive undertaking, so the merge must be performed in a way that does not invalidate these links.

One way of doing this is to take two Perforce master databases and use the P4Merge tool on them to create a third, combined database. This process is then repeated over and over until the result is one master Perforce server (see Figure 6). Our issue is that we have many systems that point to these Perforce servers (bug tracking, build database, even our syncing tools), so to facilitate this we would have to keep the old ports live but in read-only mode. This situation would remain until a specified amount of time elapsed, at which point the old servers would be backed up and then switched off.
Another way would be to slowly centralise the data by only submitting new projects to a single port. Eventually the data on the other instances will become old and only made available for reference or maintenance.

**Perforce Federated Architecture**

Database replication isn’t a new concept, but recently Perforce has been looking into what it means for the Perforce server. Mostly it’s about addressing the load a company may put on the Perforce server and its associated network. With the help of replication, some of that load can be taken away from the master server and handled by replica servers, and other networks.

Lots of excitement has been generated about the impact federated architecture will have on the design of the Citrix Perforce infrastructure. Ideas include improving site proxies, creating dedicated build farm proxies, and making enhancements to other internal tools that put a heavy load on the Perforce server, such as our reporting services.

Secure authentication is of particular interest, and the ability to tie into the active directory to reduce the management overhead of the users’ creation/deletion process is a must.

Administration of the users, groups, and protections is probably the worst part of our administrators’ jobs. By taking advantage of replicated authentication servers, we should be able to centralise the configuration. That would reduce the administration overhead and the pain it causes users when they have to log in to every port they use.

**Perforce Standard Environment (PSE)**

Perforce is constantly improving Perforce software, adding more and more features and tweaking the current ones. Therefore the PSE needs to be an ever-evolving toolset that strives to support key administration features. During its development, it has been pulled in a number of ways to make it fit, and at times maintaining the idea of simplicity has been tricky. Here we offer ideas for extension to the toolset and mention problems we are encountering.
Logging
Gradually as we have seen problems occur with our Perforce deployment running inside of the PSE, we have increased the logging functionality of our scripts. This enables us to capture error conditions that occur and use our existing monitoring servers to receive the alert condition and notify us of the failure.

However, we currently don’t do much in the way of processing the logging output from Perforce itself and therefore find it hard to figure out why something like a hung server went wrong. What we would like to do is couple the log output to a log parsing tool that could give us a clearer idea of the problem the server is experiencing and allow us to take action quickly.

Replicas
Federated Perforce or Perforce replication has only a basic implementation within the PSE. We are able to bring up a port as a replica, but this functionality just limits the abilities of a normal root type port. As administrators, we can modify the Perforce server configuration variables and bring the server up with a particular name to enable a certain setup, but this is rather clunky and adds complexity to using the PSE. Ideally we would like a more fluid and natural way to bring up replica services.

The PSE currently doesn’t support upgrading with a replica. The only way to do this now is to take down the replica, upgrade the master, then replay the new checkpoint into the replica and start it again.

We would like to take advantage of Windows Services for running Perforce, rather than the slightly complicated way of using Windows Scheduler.

Replica servers can be run in a number of different modes; we would like to allow the PSE to support some of the other modes such as smart proxy replica and build farm replica.

The Vision for PSE
Everyone needs an out-of-this-world vision to aim for. We may never reach it, but it allows us to daydream and inspires us to drive on with a project.

The PSE started as a bunch of helper scripts to aid administrators who were less confident with Perforce. Taking this to the next level, we need to start to look at what an administrator needs to know about the current state of the Citrix Perforce architecture. Finding a way to visualise this and log how the system is performing over time will greatly help in making good decisions going forward.

Suppose we had a large-scale system with multiple servers, in multiple locations all running Perforce software, which services users all over the world. What if we had a view onto this system such that we could make changes to the environment easily and quickly? What if this view could show us things like server activity, load, alerts, status of checkpoints, and verifies?

Imagine a scenario where one of the servers was being hit hard by an automated system that had gone astray. It should be relatively simple to isolate the traffic from that Perforce server, or find the user and work with that user to resolve the issue, or even deploy a new replicated smart proxy to deal with the new load.

How about a system that could automatically react to failures by activating hot standby servers? Or maybe even react to a failure that is about to happen?

What if all of this was as simple as a few clicks on a user interface?

This isn’t an impossible vision, and with every version of the PSE, we move closer to this goal.
Situations like upgrading a server with multiple replicas require some synchronisation between the replicas and the master. It’s not going to be long before we connect our servers with software and run the PSE like a distributed application. Providing a view on to this type of application would be a logical next step.

**Conclusion**
The Citrix Perforce architecture certainly isn’t a recommended strategy. For those in a similar situation to Citrix, this white paper offers some ideas and thoughts about how to maintain a working system. For those just starting out on the road to Perforce, here are a few pointers on the right path:

- Ensure you only have one Perforce instance for your company
- Make use of the great replication features of Perforce for your single instance
- Having a dedicated team that rules and controls the evolution of a version control system at a company is important, but doing this from the outset is priceless