PURESTORAGE® | PERFORCE

WHITE PAPER

DIGITAL ASSET MANAGEMENT

WITH PERFORCE HELIX CORE ON PURE STORAGE®

TABLE OF CONTENTS

INTRODUCTION

Electronic Device Automation (EDA) vendors like Synopsys, Cadence, and Mentor Graphics, and semiconductor design companies including Intel, Qualcomm, and Xilinx generate lots of software assets for silicon-on-chip (SoC) design and development. These software or digital assets may be in the form of binaries for logical and physical design tools or actual intellectual property (IP) blocks used in the circuit design process. A version-control system is required to manage the changes to files and digital assets over time. Many semiconductor companies have followed a waterfall model for software development in which they create a hardware product before the software is available. This leads to high lead times and potential synchronization problems.

Semiconductor companies are transitioning from a sequential development process to an agile and parallel development model in which software development begins before the hardware is available. This allows engineers to test and debug their software on virtual prototypes. There is a huge momentum to “shift left” in the process (as illustrated in Figure 1) to locate bugs as early as possible. The chip-design workflows are now optimized to use iterative subroutines with continuous tools and a categorical approach from design to tapeout, leading to manufacturing for software development internally and for customers.

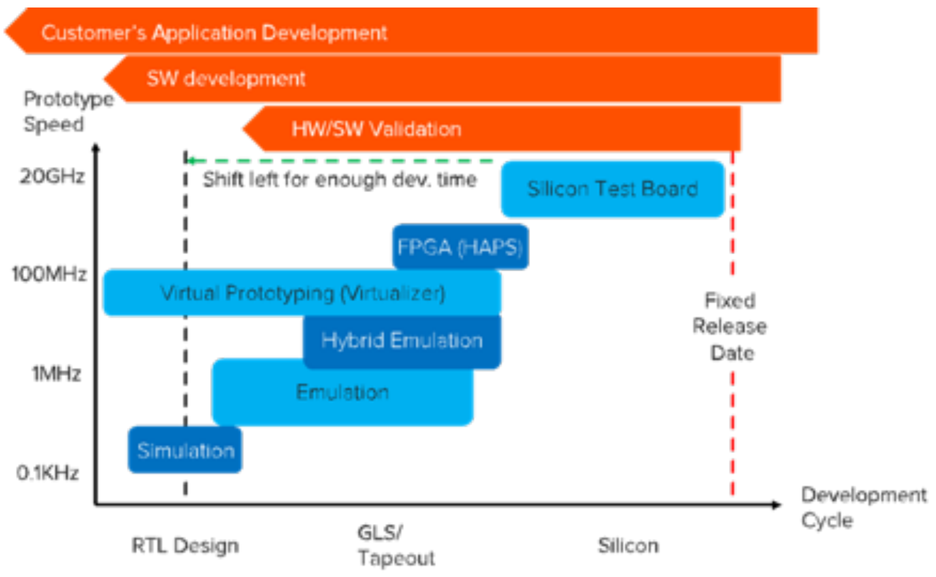


FIGURE 1. Transformation in SW development for EDA – “Shift Left”

TRANSFORMATIONAL CHALLENGES WITH SOFTWARE DEVELOPMENT

Some of the by-products of this transition include exponential data growth, the need for faster processing times, and consolidation of multiple workloads onto a single data platform. The transformation of the traditional design process into a modern implementation has the following changes with respect to data and infrastructure:

- Exponential data growth with respect to capacity and performance at scale
- Elimination of silos for heterogeneous workloads: software development, artificial intelligence, and analytics all need to run on the same data platform
- Speed to complete the software development, logical, and physical EDA jobs
- Increasing storage costs with data growth and data-management complexity

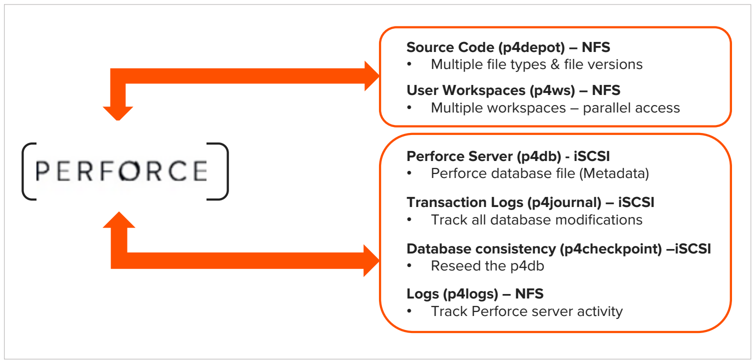
OBJECTIVE

The premise of this paper is to improve the performance of modern and agile software development processes that use Perforce Helix Core as the software control management (SCM) for managing digital assets on Pure Storage at EDA/semiconductor companies. Optimizing the performance and re-seeding the Helix Core server quickly after failures, using the Pure Storage platform and technologies, helps you manage the high number of digital assets and artifacts used during SoC design and development of IP blocks.

PERFORCE HELIX CORE

Helix Core is a popular version-control tool that EDA/semiconductor companies use to manage large data repositories consisting of various versions of digital assets and large files associated with IPs and the analog/digital chip design process. Helix Core Server manages all files irrespective of type, size, and quantity. Perforce Helix integrates into the entire workflow from design to tapeout. It connects all custom tooling through APIs, SDKs, and out-of-the-box integrations like Pure Storage for data recovery from failures.

A typical distributed Perforce Helix architecture consists of some common components widely used in EDA/ semiconductor, gaming, and numerous other industries. Figure 2 lists components and terms used in a Perforce environment.



PERFORCE
Source Code (p4depot) – NFS
• Multiple file types & file versions
User Workspaces (p4ws) – NFS
• Multiple workspaces – parallel access
Perforce Server (p4db) - iSCSI
• Perforce database file (Metadata)
Transaction Logs (p4journal) – iSCSI
• Track all database modifications
Database consistency (p4checkpoint) –iSCSI
• Reseed the p4db
Logs (p4logs) – NFS
• Track Perforce server activity

FIGURE 2. Perforce storage component overview

Various Perforce Helix components make up a federated/distributed repository management system. A single Helix server is sufficient to handle the load for small installations. In larger installations, it's sensible to distribute the work to different types of servers to achieve improved performance and scalability, especially in geographically distributed environments.

- **Master server or commit server** – This manages metadata and depot or versioned files. The versioned files modified in the respective user workspaces are checked in as part of change lists that are sent to the commit server. An atomic action both updates the metadata and saves the specific versioned file changes in one transaction.
- **Replica servers** – These have a replicated copy of the metadata and the versioned files. The replica servers only serve read operations. The write operations are directed to the commit server.
- **Edge server** – A type of replica server, these maintain local workspace status information, thus offloading the majority of the work to handle normal user commands from the commit server. Edge servers are read-write and offload a lot of I/O from the commit server by serving the data locally to the user workspace in a local site. It is mainly check-in data that is sent to the commit.
- **Depot** – The Helix Core Server stores multiple versions of different file types and sizes in one or more depots (similar to top-level folders in a file system). The P4 database tracks all the versions of files in the depot during the life of the project and beyond. They are immutable in nature. A gaming organization may have a depot for each game in development. A chip-design company may have a single large depot to store its IP.
- **Checkpoint** – These point-in-time snapshots are used to re-create the P4 metadata database from any kind of disaster or data loss. Checkpoint files do not include the contents of the versioned files. Online checkpoints are time consuming to produce depending on the size of the database. They can also disrupt users because the whole database is locked for consistency while the checkpoint is created. Offline checkpoint processes are recommended to avoid user disruption.
- **Journal** – These are transaction logs of all the P4 database modifications since the last checkpoint. Use them to recover metadata since the last checkpoint. The journals are good for auditing things including who deleted a workspace, the active users, who modified labels, and the like. Journals are also the underlying mechanism for replication between commit servers and different replica types.
- **Logs** – The logs record all the commands sent to the Helix core server and any errors logged by the server. They also provide metrics on system performance.

PURE STORAGE

Pure Storage provides FlashArray™ and FlashBlade™ appliances out of many others in the product portfolio.

- The Pure FlashArray//X product is a fault-tolerant, high-availability storage controller with integrated compute, flash storage, client I/O, and expansion capabilities. It's designed to support and enhance the Pure Storage Purity Operating Environment. FlashArray is a block-storage device that supports FCP and iSCSI protocols and improves performance of latency-sensitive applications like databases that don't respond well to remote procedure calls (RPCs) and locking over NFS.

- The Pure FlashBlade array is an advanced scale-out file system that supports NFS and Object Storage on the same platform. It's a true data hub, purpose-built to handle all the workloads generated during software development and chip design. The ability of FlashBlade to efficiently serve all different types of I/O under high concurrency and parallelism makes it an excellent fit for high-performance environments. It enables organizations to consolidate data silos and share data in today's rapidly-evolving, data-first world.

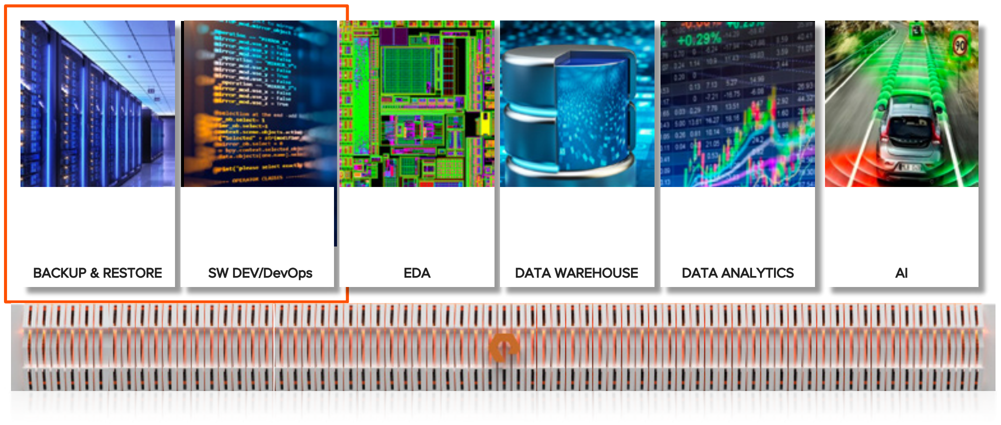


FIGURE 3. Data hub: Heterogeneous EDA/semiconductor workloads at scale on FlashBlade

As illustrated in Figure 3, the data hub on FlashBlade takes the key strengths of each silo and integrates them into a single unified platform with three must-have qualities:

- High IOPs and bandwidth for metadata intensive workloads with native scale-out
- Multi-dimensional performance
- Massively parallel architecture for concurrent workloads like software development during design to tapeout

This paper focuses on optimizing Helix core Server operations on Pure Storage FlashArray and FlashBlade as well as improving data-recovery time for P4 database during disaster recovery.

VALIDATED REFERENCE ARCHITECTURE FOR HELIX CORE ON PURE STORAGE

The "Shift Left" initiative for the agile software development workflows requires the ability to support many parallel operations in a continuous manner. This transformation requires significant speed and agility to run parallel builds at scale with high performance to speed up the software development process as time-to-market is critical.

The local server (compute nodes) storage is the simple way to implement the software development process to store data and manage data. However, with high data growth from the software, hardware, and embedded sensors, local server storage is limited by capacity and performance scalability, along with data manageability like protection and recovery from disasters.

Pure Storage has validated a hybrid architecture to optimize the Perforce Helix version-control system on FlashArray and FlashBlade (Figure 4) to manage all digital assets generated from different EDA workflows and other industry verticals.

The latency-sensitive Helix database is configured on FlashArray over iSCSI/FCP along with the replicas, journals, and checkpoints. Using three replicas (edge servers) with the Helix server provides parallelism and load-balances the read workload when users start to submit code changes and perform other metadata operations at scale. The tests result below validate the improvements with replicas over a single Helix server.

Versioned files in the single or multiple depots and server logs are mounted over NFS from the FlashBlade. The depot file-system from FlashBlade is mounted on the commit and all the replica/edge servers. Use the P4 "sync" command to update multiple user workspaces for various code branches at scale over NFS from FlashBlade. These are stored on the "p4ws" file system. User workspaces on FlashBlade provide improved manageability and parallelism when the number of workspaces scales with developers and code branches.

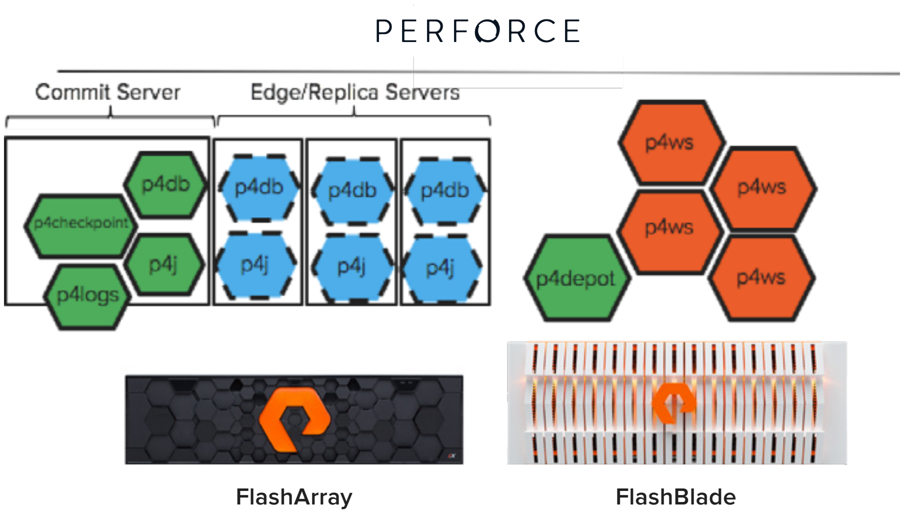


FIGURE 4. Hybrid architecture for Helix core server on FlashArray and FlashBlade

Test Environment

We tested the reference architecture in Figure 4 with Perforce Helix Server version 2018.2. The metadata database was 500GB and the versioned files were 1TB in size. The P4 server had an iSCSI LUN (volume) created on FlashArray with volumes for journals and checkpoints. We mounted the P4 depot and server logs from two file-system shares from the FlashBlade over NFS. We used similar configurations for the P4 replica nodes.

We applied the following recommended P4 settings to the server configuration (via the **p4 configure** command) prior to running the file-system tests:

```
p4 configure set:
track=1 (aids metrics reporting)
filesys.bufsize: 1048576 (configure)
lbr.bufsize: 1048576 (configure)
db.peeking: 2 (configure)
db.reorg.disable=1 (configure)
net.parallel.max: 10 (configure)
net.parallel.threads: 10 (configure)
net.tcpsize: 1048576 (configure)
net.backlog: 2048 (configure)
```

Note: The net.parallel settings may vary in different environments.

All the Linux nodes were bare metal machines running on CentOS 7.5. Each node had 48 cores with 256GB of RAM. Two 40GB Ethernet ports were bonded to a single 80GB using LACP that connected to the FlashArrary and the FlashBlade through the back-end core switch. Centos 7.x /RHEL 7.x does not perform very well with write workloads when the physical servers have more than 8GB of RAM.

Reference to set the following parameters to all the P4 nodes (server, replicas, and clients) with > 8GB of RAM for optimized write performance:

```
echo 0 > /proc/sys/vm/dirty_ratio
echo 0 > /proc/sys/vm/dirty_background_ratio
echo 629145600 > /proc/sys/vm/dirty_bytes
echo 314572800 > /proc/sys/vm/dirty_background_bytes
```

For the Perforce database on FlashArray, these settings are recommended for optimized performance.

Helix Core Test Details

- The tests were written using Python and the Locust load testing framework.
- Bash scripts running Ansible commands were used to distribute files and configuration information and execute commands on remote hosts. Ansible was also used to copy log files back to the main server for analysis.
- The files were randomly generated and checked in to the server. A regular structure of //depot/some/long/path/NN/NN/<random filename>.txt or .bin, where NN was 00 – 79 at both levels.
- The number of files in each bucket was randomly generated but fairly similar:

```
00/... 1684 files 10.9G bytes
01/... 2036 files 12.9G bytes
```

```
02/... 1975 files 14.9G bytes
03/... 1995 files 14.6G bytes
```

- The contents of each workspace was 50 percent of a bucket, so roughly 7GB on average.

The Locust framework was used with two equally weighted tasks to be run 50 times in total. Each test runner did the following:

- Randomly selected which server to connect to (commit or one of the edge servers)
- Generated a unique workspace name (including the server and machine for ease of analysis)
- Ran **p4 sync** on the workspace – typically downloading all files in the workspace the first time
- Repeat 50 times one of update/report:
  - Update actions task:
    - Randomly select up to 40 files and run **add/edit/delete** actions on them (note that edits were weighted roughly four times as frequently as adds/deletes). For add or edit, the content would be generated or modified.
    - Submit those files.
    - Re-sync the workspace, thus bringing down any changes made by other test runners.
  - Reporting actions task:
    - Run reporting commands such as **p4 describe/changes/fstat/filelog** on randomly selected files.

At the end of every test, various results were collected and analyzed by a bash script that also used Ansible to collect data. This creates a new folder under a run directory into which it copied basic test configuration data:

- Test script yaml config file
- Server configuration data
- Monitoring output such as CPU load average and network usage
- Server logs from commit and edge servers – these were then converted into a SQLite database for querying as to number and duration of commands etc
- Logs from each client machine (mainly to review in case of errors)

Finally, various SQL reporting commands were run against the database to produce results together with the key indicators (e.g., number of threads).

TEST VARIABLES

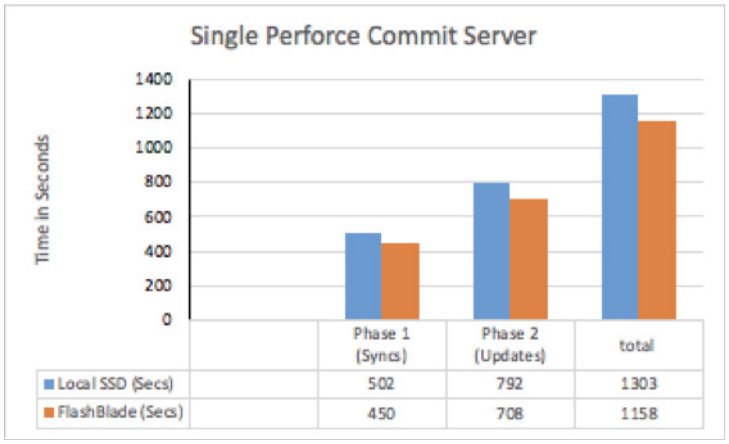
The following variables were tested in various combinations:

- Number of servers (either just single commit server, or commit server plus one edge server, or commit server plus three edge servers)
- Number of parallel threads in use

Important: Sync sufficient data to exceed server memory to avoid file-system caching effects.

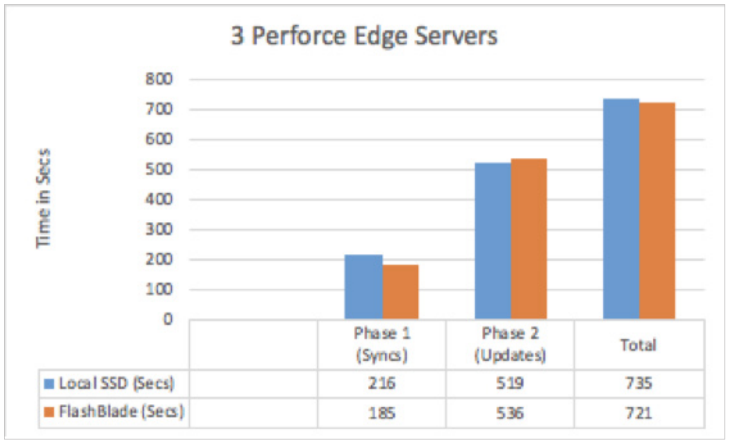
Test Results

Single P4 commit server. Lower is better.



	Phase 1 (Syncs)	Phase 2 (Updates)	total
🟦 Local SSD (Secs)	502	792	1303
🟧 FlashBlade (Secs)	450	708	1158

Three P4 edge servers. Lower is better.



	Phase 1 (Syncs)	Phase 2 (Updates)	Total
🟦 Local SSD (Secs)	216	519	735
🟧 FlashBlade (Secs)	185	536	721

Refer to the Appendix for more details on the commands run during Phase 1 and Phase 2 respectively.

OBSERVATIONS

Despite the high specification of the network cards (two bonded 40Gbps), most tests quickly showed that the network is the main bottleneck. So, there is little point running tests with huge data-set sizes when smaller data sets show that the steady state of maximum network bandwidth has been reached.

There were some positive takeaways from the tests:

- There was a 14% improvement for p4 sync operations while creating user workspaces on FlashBlade over NFS compared to local SSD. We used the three edge servers compared to a single P4db commit server. The three edge servers provided load distribution and better handling of metadata operations.

- There was about an 11% improvement in job-completion time for various P4 operations like sync, submit, filelog, and fstat in user workspaces provisioned on FlashBlade compared to local SSD storage. We also noticed that FlashBlade can have similar job completion as the local SSD storage. This variation is primarily due to job concurrency. FlashBlade has a better ability to handle concurrent jobs compared to local SSD storage at scale.
- Provisioning user workspaces on FlashBlade over NFS provides better capacity, performance scalability, and manageability.
- The compression feature in FlashBlade demonstrated up to 1.4:1 space savings during the test. Storage efficiency is one of the most important requirements, along with scalable performance, for better ROI.

The Helix Core server will typically store a compressed copy of every revision in the p4depot filesystem. (Text files can also be stored as deltas.) If the file is already compressed (such as .zip, .mp4, or .docx), store it without compression since there's no point wasting CPU to try to compress an already compressed file.

The percentage of compression achieved will vary according to the different file formats an organization uses. Text files typically compress well to a small percentage of their original size, whereas binary files may or may not compress well.

When a file is synced to a workspace, the server reads it from the p4depot filesystem (compressed) and sends it in compressed format to the client machine over the network. The client un-compresses it and writes it to disk. New files submitted by clients are compressed client-side and sent to the server, which can just write them to disk. This distributes the CPU load to the clients to improve scalability and maximizes network bandwidth.

Although FlashBlade can compress files automatically, it's not a major benefit to the Helix Core server. However, it is a useful space saving for client workspaces.

DATA PROTECTION AND RECOVERY FROM FAILURE

Backup of the P4 database and the depot are important to protect all the digital assets in the organization and reduce interruption to business operations. Data protection and recovery failure depend on two objectives:

- Recovery Point Objective (RPO): Amount of data loss. This determines the frequency of taking the P4 checkpoints on FlashArray and the snapshots for the depot file-systems on FlashBlade.
- Recovery Time Objective (RTO): The amount of time required to recover from failure. Longer recovery time leads to disruption to business operations.

Re-seeding the P4 server after failures can be time consuming and disruptive. As mentioned earlier, you can use checkpoints to recreate the P4 database. (This doesn't apply to version files that reside inside the depot; these should be backed up using standard mechanisms.) The snapshots taken from FlashBlade provide file-system consistency and can quickly recover versioned files from failure and corruption.

PERFORCE HELIX SERVER BACKUP AND CHECKPOINT PRINCIPLES

As previously noted, the Helix Server maintains a metadata database consisting of many (binary) db.* files in the $P4ROOT directory. All updates to the metadata are logged in a journal file (in text format with markers for every atomic transaction).

In order to back up the metadata of a running Helix server, you must have all the information in the db.* files—thus a consistent copy of them. The key word is consistent: If you attempt to just copy the db.* files while the server is running, you risk an inconsistent copy. (If files db.A, db.B, and db.C are updated for a single atomic transaction, it is important not to copy them when the transaction has been written to db.A but not yet to db.C.)

Thus, the most basic backup method is:

1. Stop the Helix Server
2. Copy the db.* files (which can be hundreds of GB or more)
3. Start the Helix Server

The obvious drawback to this method is that stopping and starting the server causes downtime for users. Can we achieve this while avoiding server downtime? If Pure Storage FlashArray and/or FlashBlade is in the P4 environment, then you can accomplish backup and restore non-disruptively and quickly.

PERFORCE HELIX CHECKPOINTS

It is a best practice to regularly checkpoint the server, which means writing a single checkpoint file (in the same format as the journal, thus text) that contains a logically consistent copy of all the db.* files. This checkpoint file is easy to back up and restore to another server installation (which could also be a replica), and it also has uses in terms of backup/restore across different operating systems.

If you run the checkpoint command while the server is running, the process will lock all the db.* files to ensure the logically consistent snapshot is possible. This process could take hours to complete, thus impacting users and meaning you can't do it during normal working hours.

As part of the checkpoint process, the server will rotate the journal. It basically renames the live journal to a sequentially numbered backup journal file, and then recreates the live one. This rotation typically takes only a second or two. It's a simple action and doesn't require copying lots of data (even if the journal is many gigabytes).

Advantage of Perforce Helix Checkpoint

Every checkpoint is a consistent snapshot of the database at a single point in time. It's possible to recreate a database by replaying all the journals in order since the database was created. However, this quickly becomes infeasible as the total size of journals increases. Instead, it's good practice to recover from a recent checkpoint and any journals since that checkpoint.

Checkpoints are also the mechanism by which replicas of various types are seeded/created (including edge servers). Filtered checkpoints are also possible, reducing the amount of data when appropriate. Once created, the replica is updated continuously via journal replication.

For good disaster recovery it is important that you regularly create and backup checkpoints. Depending on the size of the database, the checkpoint process may take time. Recovering takes longer. There are two ways to ease the process of taking checkpoints:

- Offline checkpoint without impacting the production database
- Pure Storage array-level snapshot for Perforce Helix database

Offline Checkpoints

This is a standard process that you can do with any type of storage to reduce a potential outage caused by database locking during a checkpoint. The mechanism is:

1. Create an initial consistent copy (offline_db) of the production database (root containing db.*). This is typically done from a checkpoint or immediately following a journal rotation.
2. Rotate the live journal (which takes seconds).
3. Replay rotated journals from production into the offline database. They're numbered sequentially.
4. Checkpoint the offline database. This may still take hours, but doesn't lock the production database during this process.
5. Optional: Remove the offline db.* files and recreate them from the newly created checkpoint. This defragments the db.* files and tests the checkpoint.

The one downside of offline checkpointing is that you require twice the storage of your db.*, which is normally stored on faster (and higher cost) SSD or equivalent.

The Server Deployment Package (SDP) is a setup of best practices implemented via scripts which implements:

- Standard (but configurable) layouts
- Offline checkpoints
- Regular admin jobs (crontab) for log rotation
- Standard triggers to be implemented
- Multiple instances on the same server if required
- Easy ways to implement topologies with multiple replicas, edge servers, backups, etc.

The SDP has evolved over a number of years and can be implemented on different storage technologies while still providing a robust Helix core architecture with disaster recovery and high availability covered.

With the features that FlashArray offers, there are opportunities to enhance the SDP to take advantage of options—including snapshots and volume copying—to achieve what it currently does even quicker. However, integrating Pure Storage REST APIs into the SDP is not within the scope of this paper.

The following section outlines the process of taking array-level application snapshots on FlashArray and FlashBlade to backup and restore the Perforce Helix database a lot faster compared to the traditional P4 checkpoints.

BACKUP AND RESTORE OF P4 DATABASE ON PURE STORAGE

This section elaborates the automated process of creating offline checkpoints to eliminate any disruption to the users, plus reseeding the P4 database quickly from corruption using a Pure Storage utility called Snap-to-NFS and RESTFul APIs. The Snap-to-NFS utility makes a snapshot portable.

The portable snapshot consists of metadata and the data required for restoring a volume. You can replicate the portable data blocks and metadata to any NFS target. Snap-to-NFS is installed and setup by Pure Storage Support. The following are sample scripts using REST APIs to perform the backup and recovery process of the Perforce Helix database:

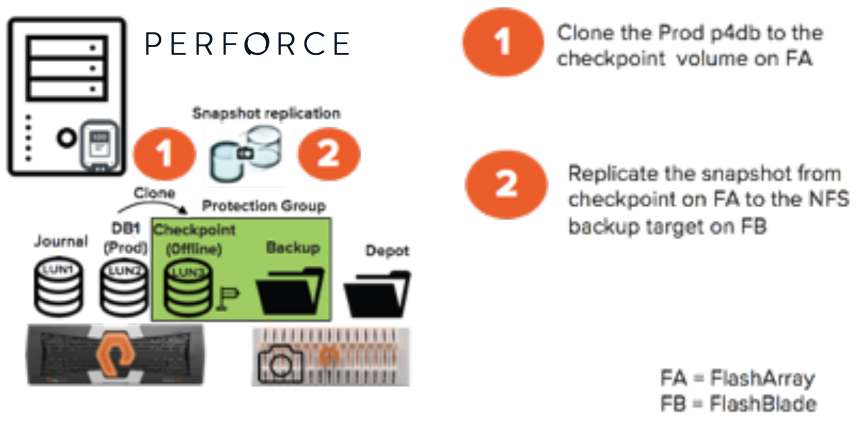
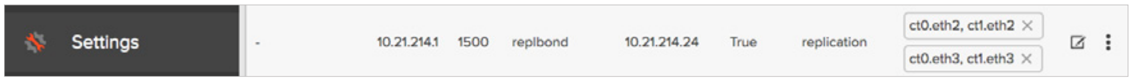


FIGURE 5. The P4 database backup process

Prerequisites for Snap-to-NFS

The following prerequisites are a one-time configuration to set up Snap-to-NFS before the backup and recovery process.

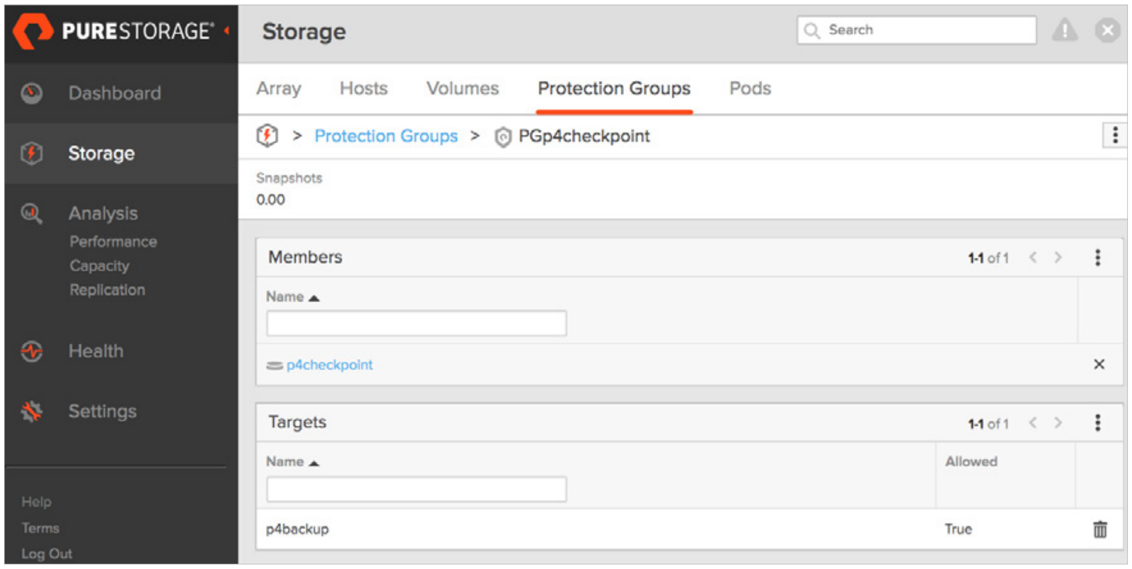
On FlashArray (X70 in this case), ports eth2 and eth3 are the default replication ports. A replibond is created on the above-mentioned replication ports to offload the snapshots. It is recommended to have the replibond and the backup target—"p4backup" on the FlashBlade—in the same network.



-	10.21.214.1	1500	replbond	10.21.214.24	True	replication	ct0.eth2, ct1.eth2 ct0.eth3, ct1.eth3

FIGURE 6. Replibond configured on FlashArray

A protection group—PGp4checkpoint is created with the source volume—p4checkpoint volume on FlashArray and the target p4backup file-system on FlashBlade.



Snapshots
0.00

Members 1-1 of 1

Name ▲
p4checkpoint

Targets 1-1 of 1

Name ▲	Allowed
p4backup	True

FIGURE 7. Protection group setup for the source and target volumes

The following is the operation sequence for the backup and recovery process for the checkpoints and for re-seeding the P4 database. The entire process is automated using RESTFul APIs.

P4 DATABASE BACKUP

1. Rotate the Helix Server journal (as is standard practice).
2. Copy/clone the "p4db" (database) to "p4checkpoint" volume (sample test environment) for the baseline. This needs to be done while the db.* files are locked (for consistency). You can do this with the "p4d -c <snapshot command>" feature of the Helix Server, which locks tables, executes the command, and releases the lock.

3. A new snapshot for the p4checkpoint volume is replicated to "p4backup" file system on FlashBlade using Snap-to-NFS. The snapshot contains all the p4 database files along with the checkpoint file.

Note: With the application consistent snapshot at the array level, a P4 checkpoint may not be entirely required. But Perforce recommends that you have a checkpoint as a second-level protection.

OBSERVATIONS

The testing was performed on the production P4 database volume - p4d. The size of the p4d volume is 550GB and the database size was 350GB.

1. Cloning the p4d (production DB) to the p4checkpoint (staging backup of P4 db.* files) was instantaneous.
2. An offline P4 checkpoint on the p4checkpoint volume took 110 minutes. As mentioned, you can avoid this checkpoint time by increasing the frequency of array-level snapshots. Snapshots on FlashArray take minimum space and are instant.
3. There was significant data reduction in the p4checkpoint volume, with 9:1 deduplication.
4. Replication of the p4checkpoint snapshot to the backup target volume (p4backup) completed in under 3 minutes. Replication time depends on the size of the P4 database.
5. When the Perforce Helix checkpoint was migrated to a different server; rsync took 32 mins.

DISASTER RECOVERY

In a disaster-recovery scenario, the Perforce Helix checkpoints may be on a different server and often in a different data center. Ideally, you'd do disaster recovery using a Perforce Helix core Standby Replica in non-mandatory mode. However, with the Perforce Helix core storage components on shared infrastructure like FlashArray and FlashBlade, the metadata and logs/journal on FlashArray should be regularly snapshotted and copied to FlashBlade using Snap-to-NFS. In case of disaster, you can restore the snapshot to any location, including the original. This process eliminates additional rsync time spent to move the P4 checkpoints.

The P4 database recovery depends upon the latest P4 checkpoint and journal logs. This is time consuming. The snapshot backed up to "p4backup" on the FlashBlade contains the P4 database files and the last P4 checkpoint.

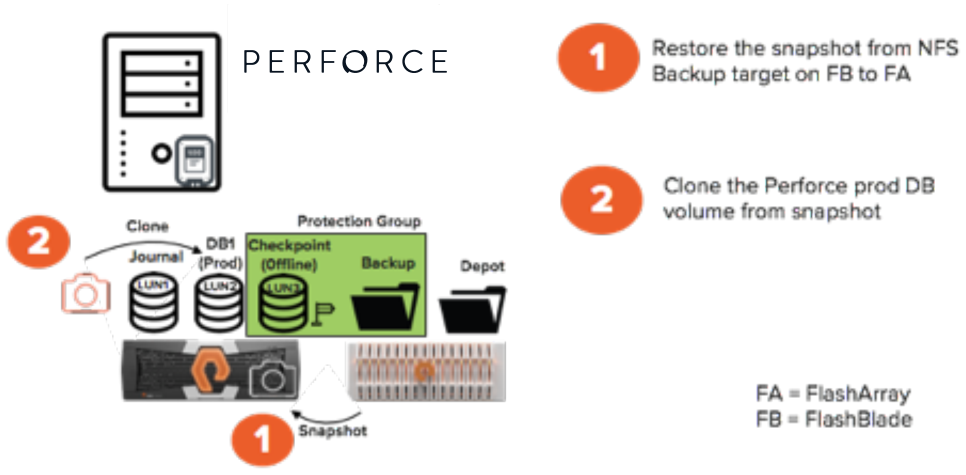


FIGURE 8. P4 database-recovery process

1. The snapshot is recovered from the backup target “p4backup” to FlashArray using Snap-to-NFS.
2. The recovered snapshot is used to overwrite one of the target volumes (p4d in this example) on FlashArray. You can use the snapshot to recover the p4 database files by overwriting the original p4db volume or any other volume that will host the new P4 database.
3. Overwriting the db.* files and restarting the Perforce Helix DB is complete under 5 seconds.

OBSERVATIONS

Re-hydrating the snapshot on a target volume allows Helix Core to rebuild the database using files when the snapshot was last taken.

- Rebuilding the 350GB database using P4 checkpoint file took 90mins.
- Snap-to-NFS was able to reseed the P4 database in under 1 minute.

HIGH-AVAILABILITY (HA) FAILOVER

For a failover situation, we assume that the same FlashArray is mounted on two different servers and contains metadata and journal/log volumes. If the main server dies—or we want to upgrade the operating system, for example—we usually want to fail over to the backup. It’s important that you deploy the Helix Server on both machines using SDP, as discussed above. It’s possible to use a P4 Mandatory Standby Replica and implement standard “p4 failover” actions.

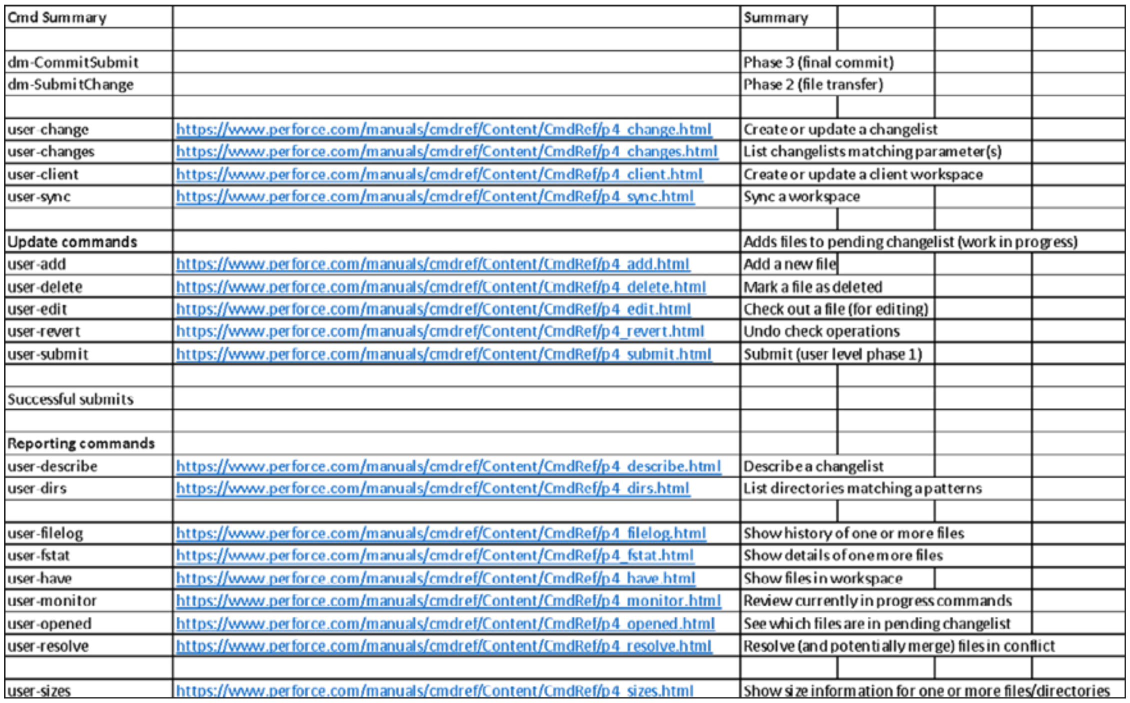
Alternatively, since the data is already on FlashArray, you can snapshot it regularly and replicate to the appropriate backup target quickly using Snap-to-NFS. This capability was not previously possible. As Pure Storage snapshots contain metadata and data, you can provision new replica or edge servers quickly for HA and load balancing traffic.

CONCLUSION

FlashArray and FlashBlade provide a service-level offering for latency-sensitive workloads like databases, and improved throughput for workloads that are high in metadata and read/write operations, respectively. Perforce Helix Core benefits from this hybrid architecture by providing better file-system performance at scale to end users who constantly check code in and out for their workspaces, run private, integrate and nightly builds pre and post code summits on FlashBlade.

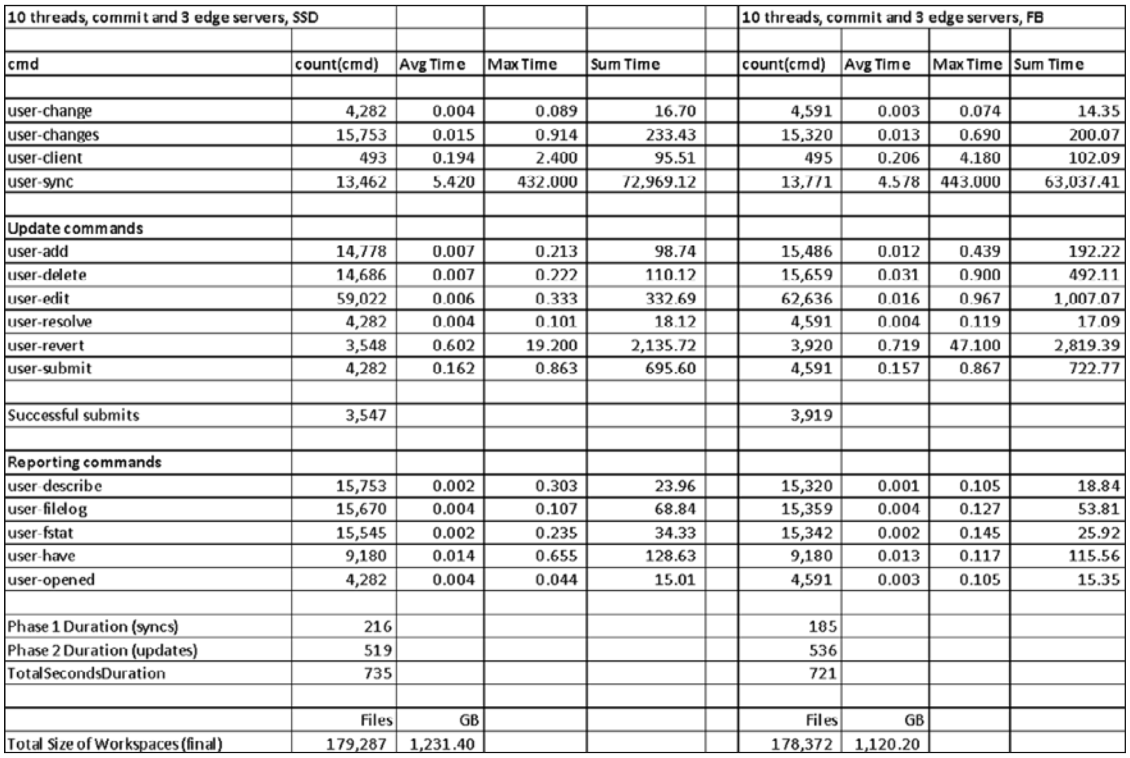
Rapid data recovery to reseed the P4 database from any disaster is equally important for business continuity.
The FlashArray snapshot FlashArray provides an application consistency that can reseed the p4 database and the depot from data corruption or loss a lot faster than traditional methods. Snap-to-NFS provides a unique way to protect and recover P4 checkpoints to recover from failures.

APPENDIX



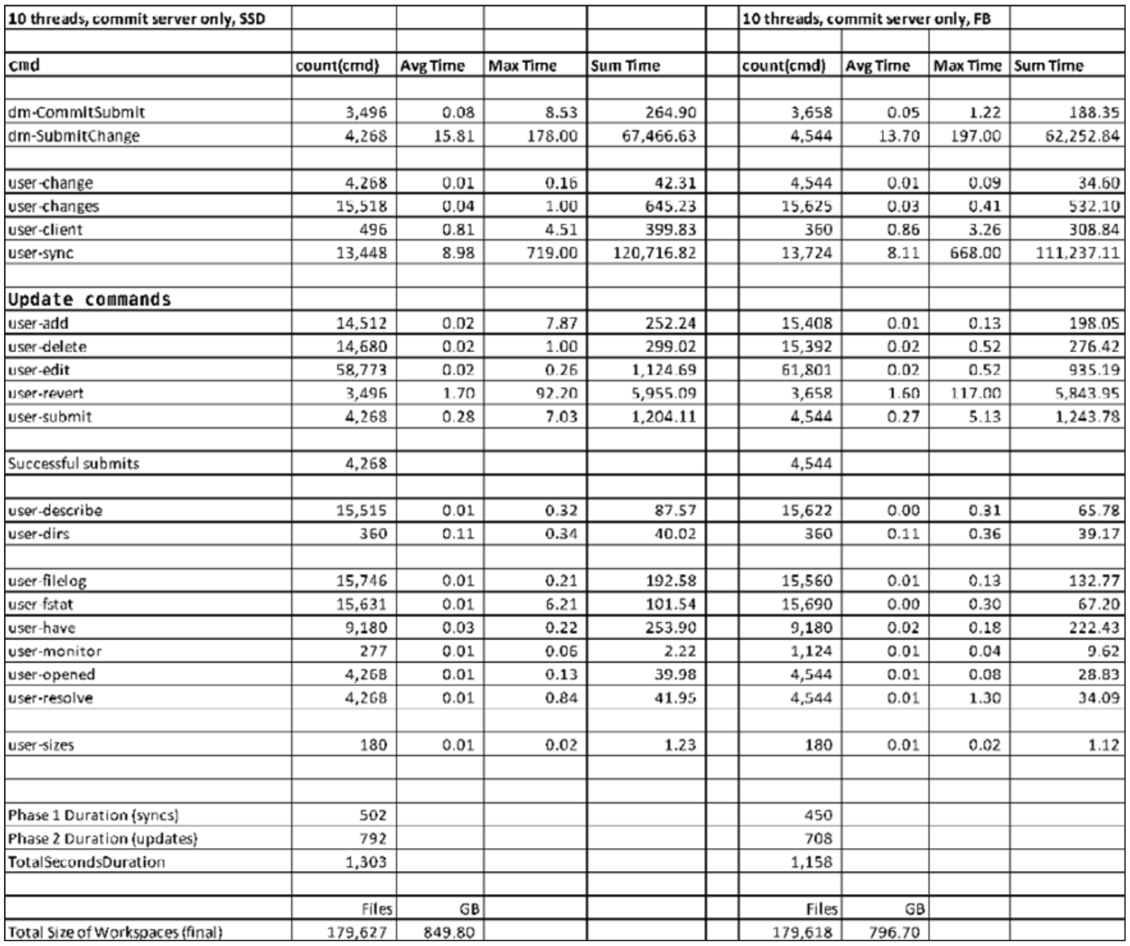
Cmd Summary		Summary			
dm-CommitSubmit		Phase 3 (final commit)			
dm-SubmitChange		Phase 2 (file transfer)			
user-change	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_change.html	Create or update a changelist			
user-changes	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_changes.html	List changelists matching parameter(s)			
user-client	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_client.html	Create or update a client workspace			
user-sync	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_sync.html	Sync a workspace			
Update commands		Adds files to pending changelist (work in progress)			
user-add	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_add.html	Add a new file			
user-delete	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_delete.html	Mark a file as deleted			
user-edit	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_edit.html	Check out a file (for editing)			
user-revert	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_revert.html	Undo check operations			
user-submit	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_submit.html	Submit (user level phase 1)			
Successful submits					
Reporting commands					
user-describe	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_describe.html	Describe a changelist			
user-dirs	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_dirs.html	List directories matching a patterns			
user-filelog	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_filelog.html	Show history of one or more files			
user-fstat	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_fstat.html	Show details of one more files			
user-have	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_have.html	Show files in workspace			
user-monitor	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_monitor.html	Review currently in progress commands			
user-opened	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_opened.html	See which files are in pending changelist			
user-resolve	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_resolve.html	Resolve (and potentially merge) files in conflict			
user-sizes	https://www.perforce.com/manuals/cmdref/Content/CmdRef/p4_sizes.html	Show size information for one or more files/directories			

TABLE 1. Summary of commands used for performance validation



10 threads, commit and 3 edge servers, SSD						10 threads, commit and 3 edge servers, FB			
cmd	count(cmd)	Avg Time	Max Time	Sum Time		count(cmd)	Avg Time	Max Time	Sum Time
user-change	4,282	0.004	0.089	16.70		4,591	0.003	0.074	14.35
user-changes	15,753	0.015	0.914	233.43		15,320	0.013	0.690	200.07
user-client	493	0.194	2.400	95.51		495	0.206	4.180	102.09
user-sync	13,462	5.420	432.000	72,969.12		13,771	4.578	443.000	63,037.41
Update commands									
user-add	14,778	0.007	0.213	98.74		15,486	0.012	0.439	192.22
user-delete	14,686	0.007	0.222	110.12		15,659	0.031	0.900	492.11
user-edit	59,022	0.006	0.333	332.69		62,636	0.016	0.967	1,007.07
user-resolve	4,282	0.004	0.101	18.12		4,591	0.004	0.119	17.09
user-revert	3,548	0.602	19.200	2,135.72		3,920	0.719	47.100	2,819.39
user-submit	4,282	0.162	0.863	695.60		4,591	0.157	0.867	722.77
Successful submits	3,547					3,919			
Reporting commands									
user-describe	15,753	0.002	0.303	23.96		15,320	0.001	0.105	18.84
user-filelog	15,670	0.004	0.107	68.84		15,359	0.004	0.127	53.81
user-fstat	15,545	0.002	0.235	34.33		15,342	0.002	0.145	25.92
user-have	9,180	0.014	0.655	128.63		9,180	0.013	0.117	115.56
user-opened	4,282	0.004	0.044	15.01		4,591	0.003	0.105	15.35
Phase 1 Duration (syncs)	216					185			
Phase 2 Duration (updates)	519					536			
TotalSecondsDuration	735					721			
	Files	GB				Files	GB		
Total Size of Workspaces (final)	179,287	1,231.40				178,372	1,120.20		

TABLE 2. Local SSD storage vs. FlashBlade with three edge servers



10 threads, commit server only, SSD						10 threads, commit server only, FB			
cmd	count(cmd)	Avg Time	Max Time	Sum Time		count(cmd)	Avg Time	Max Time	Sum Time
dm-CommitSubmit	3,496	0.08	8.53	264.90		3,658	0.05	1.22	188.35
dm-SubmitChange	4,268	15.81	178.00	67,466.63		4,544	13.70	197.00	62,252.84
user-change	4,268	0.01	0.16	42.31		4,544	0.01	0.09	34.60
user-changes	15,518	0.04	1.00	645.23		15,625	0.03	0.41	532.10
user-client	496	0.81	4.51	399.83		360	0.86	3.26	308.84
user-sync	13,448	8.98	719.00	120,716.82		13,724	8.11	668.00	111,237.11
Update commands									
user-add	14,512	0.02	7.87	252.24		15,408	0.01	0.13	198.05
user-delete	14,680	0.02	1.00	299.02		15,392	0.02	0.52	276.42
user-edit	58,773	0.02	0.26	1,124.69		61,801	0.02	0.52	935.19
user-revert	3,496	1.70	92.20	5,955.09		3,658	1.60	117.00	5,843.95
user-submit	4,268	0.28	7.03	1,204.11		4,544	0.27	5.13	1,243.78
Successful submits	4,268					4,544			
user-describe	15,515	0.01	0.32	87.57		15,622	0.00	0.31	65.78
user-dirs	360	0.11	0.34	40.02		360	0.11	0.36	39.17
user-filelog	15,746	0.01	0.21	192.58		15,560	0.01	0.13	132.77
user-fstat	15,631	0.01	6.21	101.54		15,690	0.00	0.30	67.20
user-have	9,180	0.03	0.22	253.90		9,180	0.02	0.18	222.43
user-monitor	277	0.01	0.06	2.22		1,124	0.01	0.04	9.62
user-opened	4,268	0.01	0.13	39.98		4,544	0.01	0.08	28.83
user-resolve	4,268	0.01	0.84	41.95		4,544	0.01	1.30	34.09
user-sizes	180	0.01	0.02	1.23		180	0.01	0.02	1.12
Phase 1 Duration (syncs)	502					450			
Phase 2 Duration (updates)	792					708			
TotalSecondsDuration	1,303					1,158			
	Files	GB				Files	GB		
Total Size of Workspaces (final)	179,627	849.80				179,618	796.70		

TABLE 3. Local SSD storage vs. FlashBlade with a single-commit server

ABOUT THE AUTHORS

Bikash Roy Choudhury

As a Technical Director for DevOps/EDA, Bikash Roy Choudhury is responsible for designing and architecting solutions for DevOps workflows relevant across industry verticals including high tech, financial services, gaming, social media and web-based organizations. He has also worked on validating solutions with Rancher/Kubernetes, GitLab, Jenkins, JFrog Artifactory, IBM Cloud Private and Perforce using RESTful APIs and integrating them with data platforms in private, hybrid, and public clouds. In his current role, Bikash drives integrations with strategic DevOps partners, including Rancher, Mesosphere, Perforce, GitLab and JFrog.

Robert Cowham

Robert Cowham is a Professional Services Consultant at Perforce. He has long specialized in Configuration Management and improving software development practices across enterprises. He is an expert on DevOps and is on the committee (formerly Chair) of the British Computer Society Specialist Group for Change, Configuration and Release Management. In his spare time, he runs a dojo in the Japanese martial art of Aikido.

ps_wp21p_perforce-digital-asset-management_01