Distributed Software Development with Perforce

Perforce Consulting Guide
Get an overview of Perforce's simple and scalable software version management solution for supporting distributed development teams.
Table of Contents

Overview ................................................................. 1
Introduction .......................................................... 1
Operational Requirements for Distributed Development .......... 1
Software Version Management Architectures for Distributed Development ........................................ 2
  Traditional Client/Server Architecture .......................... 2
  Traditional Replication Architecture ............................ 2
  Centralized Repository with Remote Caching .................. 3
  Distributed Repositories ......................................... 4
  The Perforce Solution .............................................. 5
Perforce Proxy: Implementation of a Cached Architecture ...... 5
Perforce Replicated Servers: Low Overhead Implementation of a Replicated Architecture ..................... 6
P4Sandbox: Hybrid Distributed Solution ........................ 7
Conclusion ................................................................... 8
Learn More ................................................................... 8
  Evaluating Perforce ................................................. 8
  Scheduling a Demo of Perforce ................................. 8
  Professional Services .............................................. 8
Overview

Distributed software development has become pervasive in modern enterprise environments. To support collaboration, a software version management system should feature a common, consistent repository, be tolerant to network outages, and support heterogeneous environments with minimal administrative overhead.

Traditional client/server architectures can be prohibitive in terms of network and server requirements. Replication solutions have been developed to address these issues, but many have significant costs in administrative overhead.

A centralized repository with remote caching, low overhead replication, and an integrated distributed solution supports distributed development more reliably and with more scalability than either traditional client/server or most replication-based solutions.

Introduction

As distributed environments have become the norm for software development, coordinating the efforts of development teams has become one of the biggest challenges faced by the enterprise today. It is hard to imagine any sizable development project managed without an underlying software configuration management tool.

This paper outlines the operational requirements that a software version management tool must meet to support distributed software development and the common approaches adopted by tool vendors over the years, and contrasts these approaches with the architecture implemented by Perforce Software.

Early models of software development were centralized; every developer working on a project worked in the same building, often with offices located along the same hall. Managers were of the belief that “communication breaks down at a hundred feet.” Today, this kind of centralized development is rare. The pervasiveness and increased reliability of broadband internet access enables developers to work in small offices and even at home.

Global teams of thousands of users, combined with newer methodologies such as Agile development, result in processes that are both geographically dispersed and highly interactive, requiring the real-time exchange of source code, documents, and images between team members.

Operational Requirements for Distributed Development

Phone calls, emails, instant messages, zip files—while they are all essential to promote healthy and timely communication amongst distributed teams, an ideal software version management system should enable all team members, regardless of their location, to have real-time access to the repository that houses source code, digital assets, and project-related documents. Programmers in San Francisco require access to assets under development by artists in the UK, while offshore testing, QA, and build teams must be able to provide timely feedback to everyone. The entire system must be configured so as to make efficient use of network resources between the sites, and must not place undue burdens on IT staff, particularly at remote sites.

Consequently, any software version management tool intended for use in a distributed environment must meet four basic requirements:

A consistent view of the repository. The repository should provide a consolidated, real-time view of all relevant assets, their current states, and their development histories to all users. Both the versioned files and the metadata describing change history, job status, who has what files checked out, and so on, must be available to all users and displayed consistently.

Network efficiency and fault tolerance. The connection between users and the repository should consume minimal bandwidth, and the system must gracefully adapt to temporary network outages in order to reduce the effect of the network as a constraint on performance and availability.

Bandwidth efficiency is essential from a usability standpoint so that users do not feel the need to circumvent software version management procedures due to the amount of time required to check in or check out files. Network outages should be minimally disruptive; users must be able to work productively during an outage, and it should be easy for users to reconcile changes made offline when connectivity is restored.

Support for multiplatform environments. It is typical for large organizations to have multiple development groups working in cross-platform environments. Multi-platform environments imply more than cross-platform support. As organizations grow, additional challenges to deployment may also arise from the need to integrate code and data between groups that follow different development methodologies and use different tool sets.
A software version management tool should support all major hardware platforms and operating systems, and interoperate with popular IDEs and whatever third-party tools a team has chosen to implement its best practices.

**Low administrative overhead.** As organizations evolve to support the needs of remote development teams, their software version management tool must remain manageable without imposing undue burdens on system administrators and other IT support staff.

### Software Version Management Architectures for Distributed Development

There have been four schools of thought when it comes to designing software configuration management systems for distributed development:

- **Traditional client/server.** Setting up only one instance of the repository, to which all local and remote users connect.
- **Traditional replication.** Setting up multiple repositories at various geographical locations, and keeping the repositories in sync by means of either real-time data replication or regularly-scheduled reconciliation processes. Every remote site requires its own repository.
- **Centralized repository with remote caching.** Setting up only one instance of the repository, to which all local and remote users connect. Additionally, caching proxies are deployed at remote sites to distribute bandwidth and processing load. Users at remote sites connect to the central server through the caching proxies.
- **Distributed repositories.** Each user has her own repository on her local workstation. Each repository is independent, although certain repositories are used as important or master repositories by convention. Collaboration is coordinated and procedures designate when changes are pushed upstream.

Each approach has its own benefits; they are discussed next.

### Traditional Client/Server Architecture

The traditional client/server approach requires that every user, regardless of geographical location, connect to a centralized server that maintains the repository.

Data consistency and ease of administration are typically not major issues, as there is only one repository, but the network can become a major performance bottleneck, both in terms of fault-tolerance and in terms of bandwidth requirements that grow with every new user added. As the number of users grows, the number of (potential or actual) simultaneous network connections to the central server grows accordingly, exposing further server-side bottlenecks in the form of available RAM, computational power, and I/O throughput capabilities.

### Traditional Replication Architecture

The first attempts to address the fundamental limitations of the traditional client/server architecture came in the form of replication solutions. Since then, replication has become a popular approach to making assets available to a global enterprise. Replication involves setting up multiple repositories across the network and keeping each repository in sync with the others by means of real-time or batched synchronization processes (see Figure 1).

Replication solves the scalability issues associated with the traditional client/server approach by bringing data closer to the users. Because failure of one node does not prevent users of other nodes from working, and because failure of a node's network connectivity does not prevent users of the isolated node from being able to work, replication also increases network fault tolerance.

The implicit trade-off in a replication system is that it’s no longer a small exercise to ensure that all users have a consistent view of the repository - every change made by every user must be propagated to all other repositories, whether in real time, in batch mode, or through a manual reconciliation process.

Real-time data replication is a non-trivial problem that requires careful implementation and administration. Almost by definition, data synchronization demands uncompromising connectivity between participating nodes on the network. Any disruption during transactional updates between nodes can introduce inconsistencies between the nodes. To ensure a consistent view for all users, however, updates must be frequent. The requirement that all sites have high availability often results in higher additional costs in terms of hardware and system administration.
From a configuration management standpoint, the fact that each node on the network operates independently means that conflicts can easily arise—two users can modify their own repository's copy of a file independently, but when their respective repositories try to replicate the changes to each other, which change (that is, which repository's copy of the data) should prevail? Every change to every repository requires a two-phase commit process. Attempting to automate the two-phase commit on a real-time basis results in high hardware and network availability requirements outlined above; users must still wait to see whether or not their changes are successfully propagated to other servers. Attempting to manually manage the two-phase commit process scales poorly; each change may potentially require manual reconciliation against multiple repositories.

Administrative overhead tends to increase linearly as the number of nodes on the network grows. For each node added, the requirements for system administration in the form of backups, security, and license management increases. Similarly, there is an increased need for configuration management administration, particularly when it comes to defining and implementing the two-phase commit processes by which conflicts between repositories are managed.

Centralized Repository with Remote Caching

A centralized repository with remote caching proxies attempts to combine the best features of the traditional client/server and replication approaches. A central server responsible for managing a master copy of all assets and metadata ensures data consistency, but processing and bandwidth requirements are offloaded to proxy servers that locally (and transparently) cache copies of the versioned assets at remote sites (see Figure 2).

The proxy servers are transparent to users; users at the remote sites configure their client software to connect to the proxy, rather than to the central server. By querying the proxy server (which in turn queries the central server), remote users can obtain real-time information on project status from the central server.

When a remote user requests a file, any revisions of files cached by the proxy server are delivered directly to the user with minimal WAN traffic. If the revision is not cached, the proxy requests the file from central server. One copy of the file is transmitted over the WAN, regardless of how many users are working at the remote site, conserving both bandwidth usage and I/O load on the central server. When a remote user submits a change...
to the repository, the proxy server forwards the change to the central server, and any merging operations are performed just as though the user were physically located at the central site.

Because files are automatically cached upon request, the cache is self-maintaining; there is no need for backups at the remote sites. Even in the event of complete hardware failure of a proxy server, the cache on the new proxy server will begin to be refilled upon the first user request, greatly reducing downtime. Because only the central server requires backups, administrative workload at the remote site can also be reduced.

The caching approach achieves most of the benefits of the traditional client/server approach (chiefly those arising from providing all users a consistent view of a single authoritative repository), and by offloading processing and I/O load to the proxy servers, it does so in a much more scalable manner. The implicit choice (between the high availability requirements of automated replication processes and the administrative requirements of manual conflict resolution) inherent in traditional replication solutions is obviated. The caching approach saves as much bandwidth as possible, and does so at a lower base hardware requirement and with less administrative effort.

Distributed Repositories

Distributed repositories neutralize any potential problems due to network lag. Each user operates independently, and communication between repositories only occurs on demand. Distributed repositories also offer compelling workflow features. The ability to work in isolation gives developers freedom to work outside the bounds imposed by the central repository (local branching), and choose which parts of their work to share.

The distributed architecture is a paradigm shift, and presents several challenges. Security and access control are more difficult to manage. Backup procedures are more complex, and there is more risk of important work being left on a local repository. By definition, a consistent view of the repository is not available to most users unless they actively maintain that view in their local repository. Working on a large, complex project may require coordination between several repositories, as each repository is normally small enough to be managed well on a local workstation. And finally, teams must more carefully manage collaboration, establishing procedures that dictate when and how to share changes.
Recall that the designation of a master repository is purely a matter of convention; that repository could be at any location (see Figure 3).

The Perforce Solution

Perforce deploys a unique federated server architecture to provide a complete solution for remote and distributed development.

Perforce Proxy: Implementation of a Cached Architecture

Perforce Software has adopted the approach of a centralized server ("Perforce Server") with remote caching proxies ("Perforce Proxy") as the model for one part of its distributed development solution. Many large organizations have successfully deployed the Perforce Proxy in support of their distributed development teams. The observed benefits have been as follows:

Data consistency. The combination of the Perforce Server and the Perforce Proxy provide real-time access to all users without remote sites having to replicate data to other nodes on the network. All users, whether connected directly to the Perforce Server or through the proxy, have access to the same repository of data at all times.

Network-friendly architecture. Perforce uses TCP/IP as the communication link between the Perforce Server, Perforce Proxy servers, and Perforce client software. The choice of TCP/IP makes deployment simple across WANs, VPNs, and through firewalls. End-to-end data transmission
can be compressed by the Perforce Server to boost performance over low-bandwidth links, regardless of an organization's choice of encryption solution. Because there is no requirement for the proxy servers to communicate with each other, and because files are transferred between the central server and the remote sites on demand, bandwidth use is kept to a minimum and brief network outages may never be noticed by end users.

If network connectivity is disrupted for a longer period of time, users can continue working on their local set of files, or enjoy a local repository provided by P4Sandbox (see P4Sandbox: Hybrid Distributed Solution). When connectivity is restored, users only need to reconcile their changes with changes made to the single central repository during the outage. Because there is only one authoritative copy of the file on the central repository, resolving conflicts after network outages is simpler in Perforce than it would be in solutions that require a two-phase commit process.

**Deployment versatility.** Perforce Server, Perforce Proxy, and Perforce client software is supported on a wide variety of hardware platforms and operating systems, and is interoperable with a wide range of IDEs, bug tracking systems, and other third-party development tools.

**Low administrative overhead.** Because the Perforce Proxy is transparent to end users; imposes lower reliability, availability, and scalability hardware requirements on remote sites than replication solutions; and requires neither backups nor license administration, administrative costs at both the central site and at all remote sites are kept to a bare minimum.

**Perforce Replicated Servers: Low Overhead Implementation of a Replicated Architecture**

As a second piece of its distributed development solution, Perforce Software offers replicated copies (“replicas”) of the centralized server. The replicas have a full copy of desired server metadata and versioned files, and can be used for any operation. Replicas maintain a read-only copy of central server data, with a limited subset of data managed locally, and can service read-only requests independently. Read-only operations account for a substantial percentage of typical user activity.

Perforce replicas can be tuned to support several behaviors, including disaster recovery, end-user activity, and read-only build server requests.
Perforce replicas offer several advantages:

**Improved local performance.** Users at remote sites will experience faster performance, as all read-only operations will be serviced directly by a local replica, with write operations proxied to the central server. All network communication to the central repository is avoided for read-only operations.

Automated processes such as continuous builds can pose a heavy load on software version management servers. Using another replica behavior, replicas can service a read-only process while still allowing the process to maintain some local metadata (e.g. workspace state information). No load is placed on the central server, allowing greater use of automated solutions.

**Data consistency.** Replicas provide real-time access to all users; replicas can be configured to pull data regularly from the central repository, or pull on demand. All user have access to the same repository of data at all times.

**Network-friendly architecture.** Perforce uses TCP/IP as the communication link between the Perforce Server, replicas, and Perforce client software. The choice of TCP/IP makes deployment simple across WANs, VPNs, and through firewalls. End-to-end data transmission can be compressed by the Perforce Server to boost performance over low-bandwidth links, regardless of an organization's choice of encryption solution.

Because there is no requirement for the replicas to communicate with each other, and because replicas will fetch data on demand if necessary, bandwidth use is kept to a minimum and brief network outages may never be noticed by end users.

If network connectivity is disrupted for a longer period of time, users can continue working on their local set of files, or enjoy a local repository provided by P4Sandbox (see P4Sandbox: Hybrid Distributed Solution). When connectivity is restored, users only need to reconcile their changes with changes made to the single central repository during the outage.

Because there is only one authoritative copy of the file on the central repository, resolving conflicts after network outages is simpler in Perforce than it would be in solutions that require a two-phase commit process.

**Deployment versatility.** Perforce Server, replicas, and Perforce client software are supported on a wide variety of hardware platforms and operating systems, and are interoperable with a wide range of IDEs, bug tracking systems, and other third-party development tools.

**Low administrative overhead.** Replicas are transparent to end users. Configuring and maintaining Perforce replicas imposes lower administrative overhead than most replication solutions, as there is still only one canonical set of files in the central repository.

Replicated servers provide a simple solution for backup and disaster recovery purposes as part of a comprehensive backup and recovery strategy.

---

**P4Sandbox: Hybrid Distributed Solution**

As the final component of its distributed development solution, Perforce Software has P4Sandbox, a distributed component that offers connection independent versioning and private local workflow options. P4Sandbox offers several compelling benefits:

**Connection independent versioning.** Users of P4Sandbox have a completely local copy of a slice of the central repository. They can use a normal Perforce client program to access all normal features, with no connection to the central repository. Because a substantial portion of work is now done on a local repository, the load on the central repository is decreased.

**Data consistency.** Collaboration still occurs in the central repository. P4Sandbox also makes exchanging data with the central repository easy and semi-automated.

**Deployment versatility.** Perforce Server, P4Sandbox, and Perforce client software are supported on a wide variety of hardware platforms and operating systems, and are interoperable with a wide range of IDEs, bug tracking systems, and other third-party development tools.

**Low administrative overhead.** P4Sandbox imposes no administrative burden, and puts very little overhead on the users. P4Sandbox is tightly integrated with the central repository.

**Improved productivity.** Distributed (and other) developers see improved productivity aside from performance gains. The private local workflows supported by P4Sandbox allow for more flexible local work independent of the central repository, supported by just-in-time branching and fast context switching.
Conclusion

Perforce Software provides simple and scalable solutions for supporting distributed software development teams. The Perforce Proxy, replicated servers, and P4Sandbox offer different approaches to distributed development, and can be combined to match the needs of any environment.

The Perforce Proxy has been successfully deployed by hundreds of companies around the world to help thousands of users to collaborate in real-time. Providing a consistent view of the repository, making judicious use of network resources, and supporting a wide range of platforms, the Perforce Proxy is simpler to implement and administer than most alternative solutions.

Replicated Perforce servers go a step further, providing a completely local copy of the repository for fast read-only access, with minimal extra administrative cost.

P4Sandbox offers a unique combination of local and distributed features tightly integrated with a strong central server. P4Sandbox requires no extra administrative overhead and minimal overhead for users. Distributed users can work independently of the central repository most of the time, and also enjoy compelling workflow improvements, while retaining the consistency and scalability of the central repository.

The Perforce Proxy, replicated servers, and P4Sandbox are available for use at no additional licensing cost.

Learn More

Evaluating Perforce

More than 400,000 users at 5,500 companies rely on Perforce for enterprise version management. Perforce encourages prospective customers to judge for themselves during a typical 45-day trial evaluation. Free technical support is included with your evaluation. Get started: perforce.com/trial

Scheduling a Demo of Perforce

To learn more about Perforce, schedule an interactive demo tailored to your requirements: perforce.com/product/demos

Professional Services

Perforce Consulting has experience assisting customers with optimizing their Perforce deployments. For more information, visit: perforce.com/consulting