Optimizing Performance of Git Fusion and Commons

Michael Shields  Marc Tooley  Ron Cadima

Performance and Diagnostic Lab
Introduction

- **Common themes**
  - Distribute load across multiple instances and cores
  - Use Perforce replicas or proxies when applicable

- **Unique characteristics**
  - Resource utilization (Git Fusion)
  - Java (Commons)
Git Fusion
Marc Tooley
Introduction

- Difficulty of Measurable Quantities
- Consistency of measurements
- Hundreds of factorially-growing variables
- Smaller measurable aspects of use
Overview

- Git Fusion
  - Resources (CPU/Memory/IO/Storage)
  - Network impacts (the latency demon)
- Perforce Server
  - Well-known, fairly well-studied
  - P4D Role in Git Fusion Operations
Overview

- Tuning + where to focus attention
- Increasing limits
- Best practices
- Horizontal scaling with Perforce + Git Fusion
Git Fusion Unified Environment

Clients (git) → P4D → git repos
Git Fusion Performance

- Resources to consider
  - CPU and Memory for compression, translation of data to/from Perforce, SHA1 computation, etc.
Git Fusion and RAM

- Git Fusion and git are strongly IO-bound
- Actual CPU usage is fairly light
- Process memory is light for many operations
I/O, Network and Disk

- Pre-seeding Git Fusion is recommended
- ssh → more chatty
- git pack file calculation and creation*
I/O, Network and Disk

- Expensive to compute on larger datasets
- Much of commit calculated client-side
- git + Git Fusion can benefit from SSD
- Maximize available RAM
  - benefits OS filesystem cache
Storage Requirements

- Each view (per Git Fusion repository) requires git-managed storage for entire view plus history
- git objects, stored directly in Perforce in the raw → minimum of double git object storage
Storage Requirements

- Files themselves committed in separate changelists → versioned files store
- Plus any lightweight unmirrored git branches
Network Latency

- ssh is chatty
- Significant ssh handshake overhead
- Speed of authentication produces latencies, can limit total instantaneous incoming connections
Network Latency

- `git` push should only outweigh this as total fraction if it is a large commit
- Local pack file creation
  - CPU, RAM, IO on client
Perforce Server

- Is NOT a bottleneck
- Individual p4 prints are quick, but mass-printing is time- and memory-consuming
- **significantly** more scalable than git
- Horizontal read-only scaling definitely an option
Increasing Limits

- RAM, then low-latency storage, then CPU
- sshd (MaxStartups 64/128)
- p4d net.backlog (supporting p4 print)
- p4d net.tcpsize (2MB)
Best Practices Summary

- Low-latency drives
- Place Perforce Server and Git Fusion instance on separate machinery
- Git Fusion will consume all IO before its p4d slows
- Liberal use of additional Git Fusion instances
- More liberal use of git-specific clones
Example Git Fusion Farm

- **P4D**
- **Git Fusion**
- **Replica**
- **git**
- **Machine Grouping**
- **Latency**
Commons
Ron Cadima
Outline

- Test environment
- Resource considerations
- Tuning
  - Java, Jetty, and network
- Best practices
Test Environment

Pcomx: Commons Instances
Psimx: 200 highly concurrent users each
Resource Considerations

- Adequate CPU cores for each Commons Instance
  - More cores can support more Commons users
    - Two cores supports 70 highly concurrent users
      - perhaps 700 moderately concurrent users
    - Four cores supports 250 highly concurrent users
      - perhaps 2500 moderately concurrent users
    - Eight cores supports 450 highly concurrent users
      - perhaps 4500 moderately concurrent users
User Response Time for CPU Cores and Memory

Percentage Degradation

Number of Highly Concurrent Users

2 cores 3.5 GB
2 cores 3.7 GB
4 cores 15 GB
8 cores 7 GB
Resource Considerations

- Memory for Java in Commons Instance
  - Memory required for:
    - Large files
    - Long paths
    - Large number of files
    - Large number of users
  - More memory is not always better
    - Can exacerbate garbage collection
    - More system resources consumed
    - Better to use multiple Commons Instances
Resource Considerations

- Network bandwidth and latency
  - 35ms+ latency between users and Commons Instance
    • Deploy another Commons Instance more local to users
  - 45ms+ latency between Commons and Perforce
    • Connect Commons Instance to local Perforce replica
Resource Considerations

- OVA implementation
  - Default of two cores and 3.5GB memory
  - Host machine must have adequate resources for VM
    - Physical CPU sockets and cores
    - Physical memory
  - Use for smaller number of users per Commons Instance
Resource Considerations

- Perforce Server performance
  - Child process/thread for each active Commons client
    - Increased socket usage
Java Tuning

- Ensure CLASSPATH contains minimal entries
- Use --noclassgc to keep class files in JVM
- Heap memory
  - Preallocating might improve performance
    - Set initial (-Xms) and maximum (-Xmx) to same value
    - Minimal 1-2% improvement seen in testing
Java Tuning

Additional JVM tunables

- `-d64` (64-bit machines)
- `-Xconcurrentio`
- `-XX:ConcGCThreads=2`
- `-XX:+OptimizeStringConcat`
- `-XX:NewRatio=2` (perhaps 3)
- `-XX:MaxPermSize=196m` (or more)
- `-XX:+UseFastAccessorMethods`
- `-XX:+UseCompressedStrings`
Jetty Tuning

- **AcceptQueueSize**
  - Set to 300 for active Commons Instances
    - Kernel's somaxconn attribute might need increased
  - Increase if “HTTP ERROR 500” encountered
- **Might need to adjust thread pool**
  - Set minThreads to 50
  - Set maxThreads to 500
Network Tuning

- Assuming adequate memory available
  - net.core.somaxconn=4096
  - net.core.rmem_max=16777216
  - net.core.wmem_max=16777216
  - net.core.netdev_max_backlog=16384
  - net.ipv4.ip_local_port_range=“1024 65535”
  - net.ipv4.tcp_max_syn_backlog=8192
  - net.ipv4.tcp_tw_recycle=1
Best Practices

- Deploy multiple Commons Instances
  - Service larger number of Commons users
  - Reduce impact of Java and thread contention
  - Use commodity hardware or virtual machines
  - Does not significantly impact Perforce Server
    - Compared to same number of users on one Commons Instance
Best Practices

- Connect Commons to Perforce replicas
  - Replicas handle read requests from Commons
  - Service larger number of Commons users
  - Reduce impact of network bandwidth and latency
Best Practices

- Deploy Perforce Server on a separate machine
  - Eliminates contention with Commons for resources
    - e.g. JVM heap memory versus filesystem cache for db.* files
Best Practices

- Use “reasonable” shared spaces
  - Excessive files or versions can impact performance
    - More than 1000 can delay user response
    - Increased memory might be required
  - Avoid excessive users accessing a shared space
Commons Scalability

- 2000 highly concurrent Commons users
  - Approximately 10% performance degradation
    - Multiple Commons Instances, Perforce replicas, etc.
Commons Scalability

Number of Highly Concurrent Users

Perforce Server

Perforce Replica

Perforce Server and Replica

elapsed time (seconds)
Summary

- Git Fusion and Commons Architecture
  - Distribute load across multiple instances
    - Consistent with Distributed Roadmap
  - Utilize Perforce infrastructure as needed
- Performance continues to be important
  - Part of continued customer success
Thank you for using Git Fusion and Commons!

mshields@perforce.com
marc@perforce.com
rcadima@perforce.com