Improving Perforce Performance by over 10x

Tim Barrett & Shannon Mann, Research In Motion
December 2007

Abstract
Research In Motion (RIM) has completed a 9 month project that resulted in improving the performance of Perforce by over 10x. The Project team worked with Perforce Technical Support, peer companies and internal resources to develop and execute a comprehensive project that addressed infrastructure and application issues. The team surpassed that goal of a 75% improvement in performance and has delivered over 90% improvement in performance.

This paper discusses the major and minor initiatives included in RIM's performance project along with key ways of measuring performance.

Introduction
Research In Motion (RIM) is a world leader in the mobile communications market and has a history of developing breakthrough wireless solutions. RIM's portfolio of award-winning products, services and embedded technologies are used by thousands of organizations around the world and include the BlackBerry(tm) wireless platform, the RIM Wireless Handheld(tm) product line, software development tools, radio-modems and software/hardware licensing agreements.

RIM has a large and rapidly growing Perforce server. In late 2006, the performance of Perforce was a growing concern across the Software organization and a Project Team was assembled with the responsibility to fix all performance issues related to Perforce.

Starting Infrastructure
In January 2007, RIM was operating the following configuration:

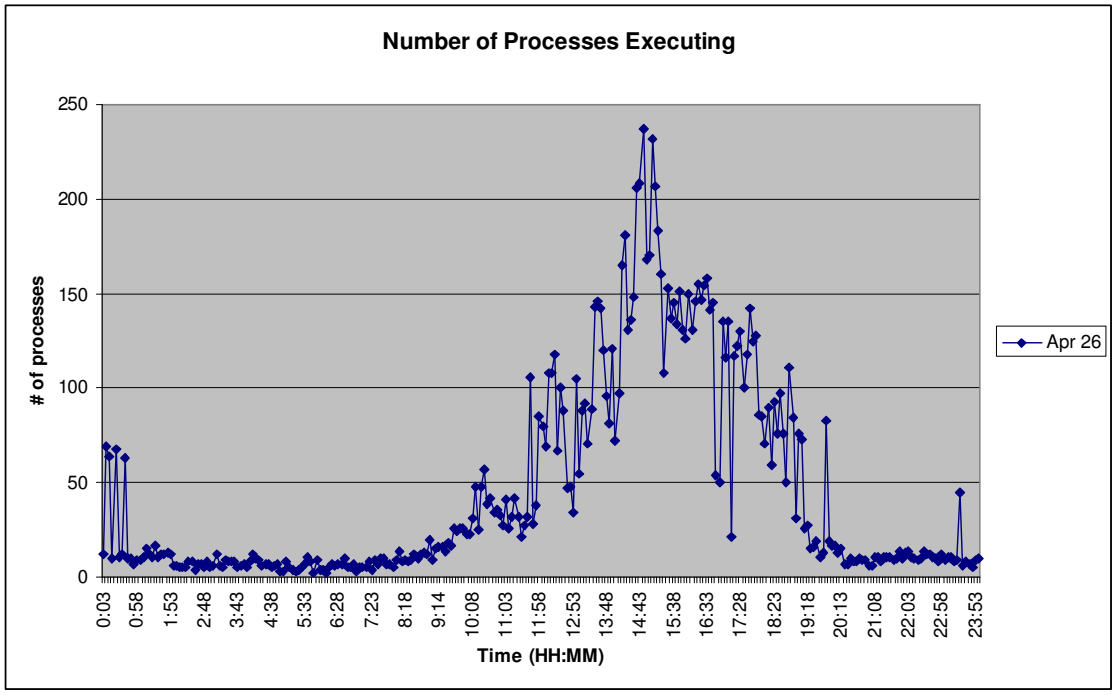
Hardware: SunFire V890with 8 dual SPARC IV (1.35Ghz), RAM = 32GB (12% file cache)
OS: Solaris 9
Software: Perforce server 2005.2, multiple GUI levels (mostly versions of P4WIN)
# of Users: 1,950

Baseline Performance
Before the project began, performance was measured based on the number of processes executing at a given time. Monitoring tools were used to count the number of processes executing every 5 minutes. These tools highlighted performance issues after they had affected the main server.

Table 1: Number of processes executing at a time in Perforce. Measured Mon – Fri, 10:00am – 5:00pm.

	Average	Median	Maximum	Minimum
Apr 17 – 20	60	49	217	17
Apr 23 – 27	60	53	227	11
Two week Average	60	51	222	14

Diagram 1: Number of processes executing at a time in Perforce. Measured every 5 minutes on April 26, 2007.



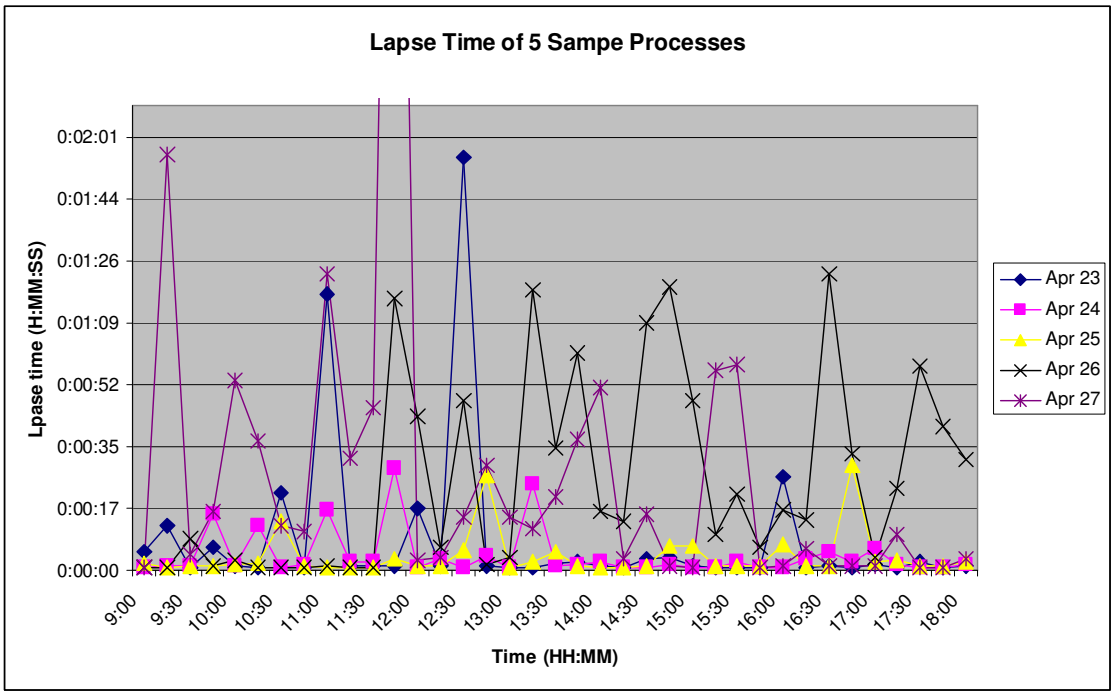
The Project Team developed a method to measure server response times that was more indicative of a developer experience. The team built a script to execute 5 processes every 15 minutes. The lapse time from the start of the first process to the end of the last process was recorded and tracked. This method of measuring performance still highlighted issues after they had affected the main server, but, it also provided a method to measure the scale of the impact. The 5 processes executed were:

- p4 info > /dev/null
- p4 print -q //depot/swdocs/pub/hasty_guides/usage.pdf > /dev/null
- p4 fstat //depot/swdocs/pub/hasty_guides/usage.pdf > /dev/null
- p4 dirs //depot/vendor/* > /dev/null
- p4 sync -f //depot/admin/announcements/... > /dev/null

Table 2: Average weekly lapse times of the 5 test processes. Measured from 9:00am to 6:00pm EST, Mon-Fri

	Average	Median	Maximum	Minimum
Apr 3 – 5	4.7s	1.4s	49.0s	0.9s
Apr 9 – 13	8.7s	3.2s	89.1s	0.9s
Apr 17 – 20	18.1s	5.0s	209.6s	0.7s
Apr 23 – 27	14.4s	5.9s	123.2s	0.7s
Monthly Average	12.2s	4.1s	125.3s	0.8s

Diagram 2: Individual lapse times of the 5 test processes for the week of April 23 – 27



Performance Targets

While the mandate for the Project Team was clear, “Improve performance for the developers”, the team needed concrete performance targets. After considering the scale and distribution of the historical performance issues, the targets were set as:

> Deliver a 75% improvement in performance measured using the average lapse times of the 5 test processes Monday to Friday, 9:00am to 6:00pm EST.

In other words, the average lapse times of the 5 test processes must be below 4 seconds, adjusted for growth, in order for the project to be considered a success.

Project Priorities

The Project Team engaged Perforce Technical Support early in the project life cycle to help identify potential sources of improvement. In addition, the Project Team engaged other large users of Perforce looking for solutions implemented at their organizations. The expectation of the Project Team was that, while RIM was a unique organization, the Perforce performance problems were not unique and were experienced by other organizations. The Project Team decided to take the early direction from other people who experienced and dealt with the performance issues while investigating our own environment for unique issues. The list of potential solutions was:

1. Stop commands that are known to cause problems
2. Upgrade RAM from 32GB to 64GB
3. Upgrade server version from 2005.2 to 2006.2
4. Move from Sparc/Solaris to Opteron/Linux
5. Move from SAN storage to DAS storage
6. Optimize the Protection table
7. Upgrade Client GUI’s to current level
8. Implement a RamSan for the metadata

The Project Team set priorities based on expected improvement and elapsed time required to implement each solution.

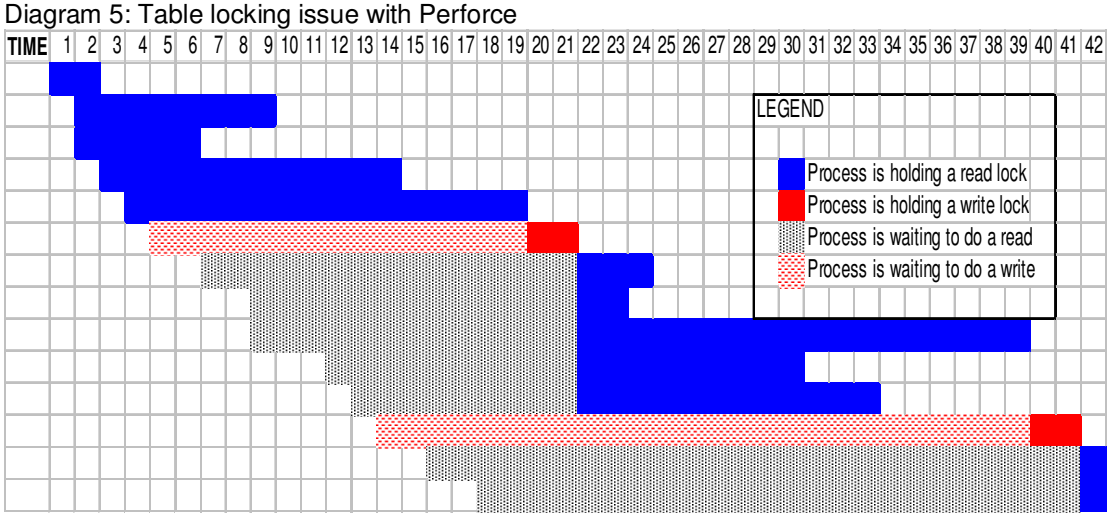
Performance Analysis

The causes of performance issues with Perforce is well known and has been discussed at previous user conferences. The RIM Project Team spent time throughout the project to understand the nature of the database locking problem in order to be able to duplicate the problem in a test environment.

The nature of the problem is the inter-table locking that occurs between read commands and write commands. The problem is not related to a single command, but the interplay of multiple commands. When a command is reading information from a table, it holds a shared lock which allows other reads to also access the table. If a command needs to write to a table, the command holds an exclusive lock so that no other command can access the table at the same time. The database locking problem occurs when a long read command is followed by a write command. The read command places a shared lock on the table. The write command must wait for the read to finish before it can have its exclusive lock on the table. Since Perforce requests locks for commands in the order they are received, all other commands must wait for both the read command to finish and the write command to finish.

The example below depicts the locking issue with one table. However, Perforce has over 40 tables and most commands utilize multiple tables, some commands requiring all locks before proceeding. In a worst case scenario, a long read command blocks a write command that is holding exclusive locks on multiple tables. This combination of locks is the essence of the cross-table lockjam problem that can effectively turn Perforce into a single-threaded application.

Diagram 5: Table locking issue with Perforce



The Project Team created a test scenario that duplicates the table locking issue. This test has enabled the Project Team to duplicate the major performance problem RIM experiences to determine if changes address the issue. The test developed by RIM is:

- Execute 10 submits of 10,000 files each in parallel
- At the same time, execute a spinning integrate of 10,000 files
- An at the same time, execute a spinning FSTAT of all files

Major Initiatives: Memory Upgrade

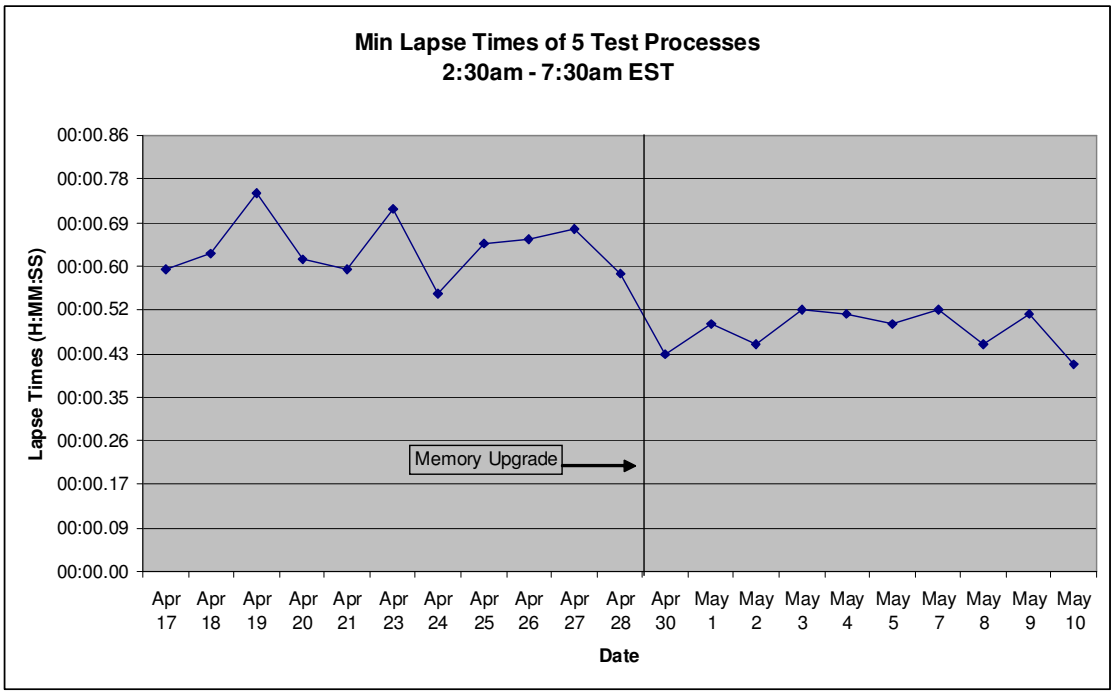
RIM was operating servers with 32GB's of RAM. Based on advice from Perforce and from another peer company, it was determined that RIM required at least 64GB of RAM. The calculation used by Perforce was:

1. RAM should be 32MB per user or
2. RAM should be 1.5KB per file

The Project Team executed a series of tests, first with 32GB of RAM, followed by 64GB of RAM. In addition, different levels of file cache (setting segmapsize from 12% to 50%) was tested to determine if any performance improvement existed with higher file cache amounts. The testing results showed an average speed improvement of 11% with 64GB/segmapsize =50% and that file cache changes did make a big difference in performance.

The memory was upgraded in Production on April 29, 2007. The performance improvement in production was immediately apparent during the early morning hours when little activity occurred on the Perforce server. The average lapse times from 2:30am to 7:00am dropped from 1.2seconds to 1.1seconds, an improvement of 8%. Minimum lapse times of the 5 test processes also dropped from 0.73 seconds to 0.55 seconds, an improvement of 24%. Unfortunately, the user count increased significantly at the same time and the improvement was not enough to compensate for the increase in volume.

Diagram 4: Minimum lapse times of the 5 test processes from2:30am to 7:30am before and after the memory upgrade



Major Initiatives: Server upgrade from 2005.2 to 2006.2

RIM had started experiencing performance issues shortly after upgrading to server version 2005.2 in May 2006. The suspicion was that the degradation of performance related to the upgrade of the server version along with other RIM incurred changes within our development process. Again, on the advice from Perforce, the Project team evaluated the server version

2006.2. The major performance improvements included in the upgrade were changes to p4 integrate (version 2006.1) and p4 submit (version 2006.2).

In a QA environment, a series of commands were executed multiple times and durations were analyzed to compare the speed of the two server versions. Of the nine types of commands tested, 6 were noticeably faster, but three commands were actually slower with the new version of the server software. The Project team worked with Perforce and was unable to determine why these three commands executed consistently slower with version 2006.2.

Table 3: Average durations in QA to compare lapse times of the server version 2005.2 and 2006.2.

	Version 2005.2	Version 2006.2	Variance
DIRS	329s	64s	80.5%
FILES	165s	161s	2.4%
INTEGED	4347s	4121s	5.2%
LABELS	95s	85s	10.5%
SUBMIT	7335s	2246s	69.4%
SYNC	850s	666s	21.6%
CHANGES	176s	185s	-4.9%
FSTAT	252s	267s	-5.7%
OPENED	**38s**	**44s**	-14.1%

The upgrade in production was completed on May 20,2007 and the result was immediately noticed by the Developers. Locking issues that used to affect the main depot for over 5 minutes at a time were eliminated with the most significant issues locking the main server for no more than 2 minutes. Below are two charts that track the daytime lapse time of the 5 test processes the week before the upgrade and the second week after the upgrade

Diagram 4: Individual lapse times of the 5 test process before the server 2006.2 upgrade (May 14, 2007 – May 18, 2007)

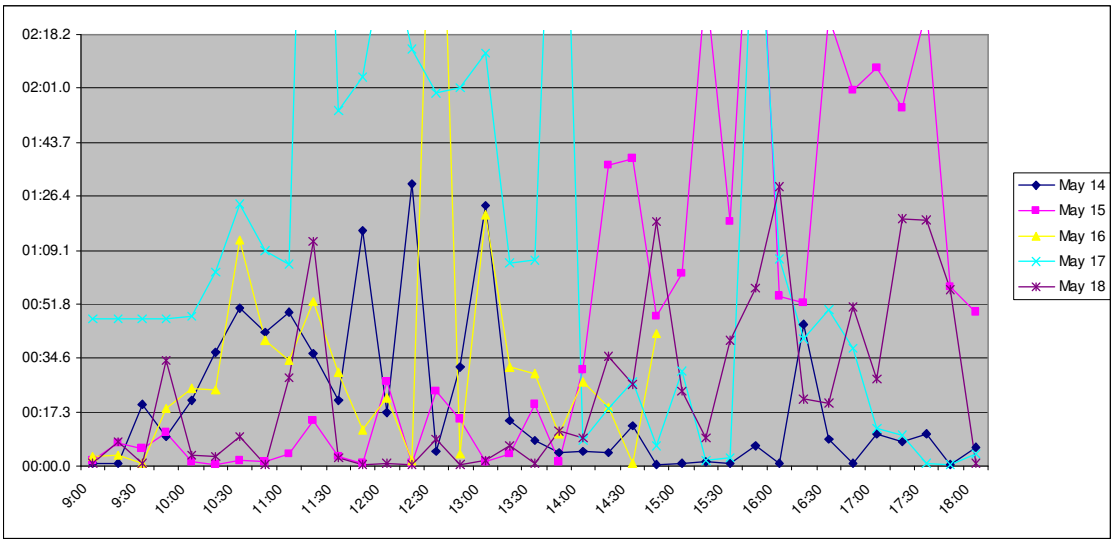
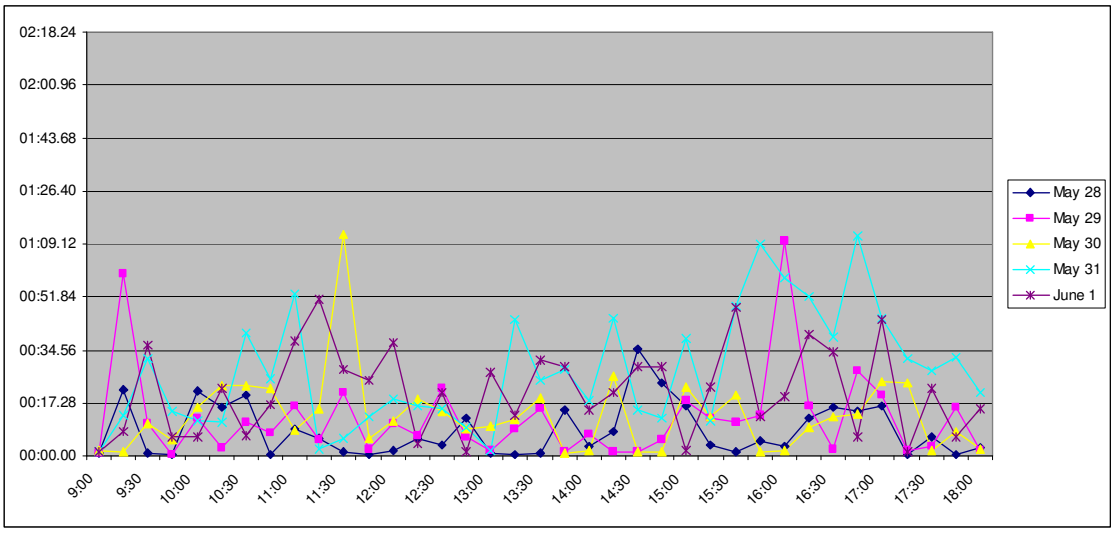


Diagram 5: Individual lapse times of the 5 test processes after the server 2006.2 upgrade (May 28, 2007 - June 1, 2007)



Major Initiative: Opteron Update

RIM was utilizing SunFire V890 servers with Solaris 9 and approached the recommendation by Perforce to switch to Opteron/Linux with suspicion. At the 2007 User Conference in Las Vegas, the Project Team confirmed with many other large users of Perforce that Opteron/Linux combination was superior to Sparc/Solaris combination in terms of Perforce performance. The Project Team proceeded to compare the following hardware/OS configurations:

Current Hardware:
SunFire V890 with 8 dual SPARC IV (1.35 Ghz) CPU, 64GB RAM, Solaris 9

New Hardware:
SunFire x4600, 8xAMD Opteron model 8220 processor (2.86Ghz dual-core), 128 GB RAM, Linux (RHEL4, update5)

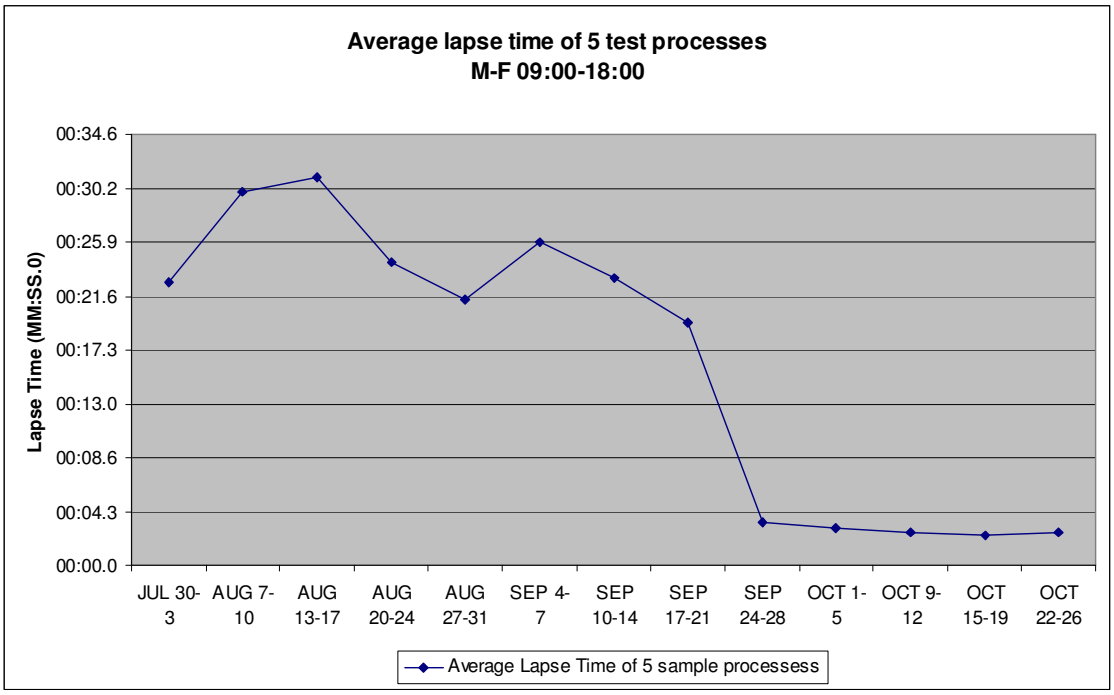
The test results in the QA environment confirmed the information the Project Team had received from Perforce and other large users of Perforce. In the QA environment, Opterons/Linux out-performed Sparc/Solaris and the performance improvements ranged from 62% to 98%.

Table 4: Lapse times from Sparc/Solaris vs Opteron/Linux tests

Test Type	Sparc/Solaris	Opteron/Linux	Improvement
Lock Test: Created a program to create 20 files, cycle through locking them 2,000,000 times	4m6s	39s	84%
Populate Test: Created script to create 100,000 files (6.4K each). Submitted all files in one changelist	2h14m51s	38m8s	72%
Integrate Test: Integrate files submitted in populate test into a different directory	29m54s	31s	98%
Submit Integrate Test: Submit changelist created in integrate test	1m3s	17s	73%
Rebuild from Checkpoint	4h40m	1h45m	62%
Checkpoint Test	8h31m	1h20m	84%
Average Improvement			79%

The new Opteron servers were implemented into production on Sept 23, 2007 and the result was immediate. For the first time since the project began, Perforce performance was no longer an issue and positive comments were received from many different users. The RIM Perforce environment was able to effectively handle the high volume of traffic without incurring any lengthy lock problems. Below are the average lapse times of the 5 sample processes for the 5 weeks prior and 5 weeks after the upgrade to Opteron/Linux.

Diagram 6: Average weekly daytime duration of the 5 test processes



In addition, the increased speed has allowed a higher number of processes to be executed on a daily basis. Average number of processes completed during a typical business day increased by an average of 8% after the switch to the Opteron CPUs.

Major Initiative: Read-only Replica

Volume of activity on the main Perforce Server at RIM is a growing concern. RIM is adding a significant number of users every few months and the trend is expected to continue into the near future. In addition, explosion of products and new processes has increased the number of builds performed on a regular basis. The Project Team investigated Perforce usage and determined that various build activity executed daily accounted for over 60% of the volume on Perforce. In fact, the top 15 accounts (all build activity) accounted for over 98% of the daily 'p4 sync' commands. The 'p4 sync' command is a top suspect for creating the perfect environment for the database locking issue with Perforce describe in section Performance Analysis (Diagram 3).

Based on advice from a number of large users of Perforce, RIM investigated implementing a read-only replica of the main server to offload all build activity. RIM tested a solution from Perforce (p4 jrep), along with advice from a peer company, and set up a replica server. The server utilizes the 'p4 jrep' script to replay the journal file to the read-only replica. RIM modified the script to avoid copying key tables to the mirror that were needed to control the use of the mirror (such as the DB.HAVE table).

RIM is now in the process of moving all build activity from the main server to the read-only replica server.

Major Initiative: RamSan Solid State Disk

At the 2007 user conference, one company reported success with a RamSan unit for addressing performance concerns for large Perforce customers. Another peer company installed similar technology in the summer of 2007 and also reported significant gains in performance. This information prompted the Project Team to evaluate a RamSan Solid State disk (128GB) unit. The strategy was to test the unit against local disk and our production array. A series of low-level tests were compiled along with the cross-table lockjam test described in the Performance Analysis section of this document. The end result is that, while the RamSan solid state disk showed measurable improvement in the low-level testing, the real world application testing did not show any measurable improvement. It is the opinion of the Project Team that our environment is currently optimized and that the current disk access speed is not a bottleneck. The Project Team has plans to implement a RamSan unit when volumes increase to the point where performance begins to suffer.

Table 5: RamSan Low-Level test results

	Local Disk (SCSI Raid O)	SUNSAN Array 9985	RamSan 128GB
#DD (Disk Dump)	335.974s	36.409s	68.895s
Copy of 24GB file	310.593s	189.469s	177.357s
#IOTMS avg IOPS	620 iops	1261 iops	7288 iops
#Locktest	9.092s	9.350s	8.178s

Table 6: RamSan Real-world testing

	SUNSAN Array 9985	RamSan 128GB	RamSan 128GB – with cache	Local Disk	Local Disk – with cache
Serial Add	46.21s	43.85s	96.07s	52.25s	49.96s
Parrallel Add	1638.08s	1732.95s	1605.71s	1921.14s	1709.06s
Avg FSTAT	2.8s	2.95s	2.56s	3.1s	3.05s
Total	2392.74s	2459.48s	2893.78s	2879.82s	2624.61s

Minor Initiatives:

The Project Team also completed many minor initiatives to address the performance concerns.

"Kill" Orphaned processes
RIM Perforce server had the number of orphaned processes grow every day. These processes were complete from the GUI perspective, but, not from the server perspective and they muddied the performance monitoring results. The Project team created a script that would kill any process that had been executing for over 2 hours. It was not proven if this action improved performance of the main server.

Rebuild from Checkpoints
The Project Team completed a series of Rebuild from Checkpoints to rebuild the indexes and improve performance. While RIM was operating server version 2005.2, the occasional Rebuild from Checkpoint provided temporary improvements lasting 2 to 5 business days. However, after

RIM moved to server version 2006.2, no improvement was measured on subsequent Rebuilds from Checkpoints.

Optimize Protection Table

RIM had a protection table that included a number of exclusion entries and wildcards. After the Perforce Conference in Las Vegas, May 2007, the Project Team used the information from Michael Shields presentation to optimize the protection table. The improvements were done a few at a time over the period of two months which made measuring performance improvements from the changes impossible.

Log Parser

To thoroughly understand and manage the use of Perforce, the Project Team developed a log parser along with a reporting database. The Log Parser allows the capture of all relevant information from the log files for reporting and analysis. The database is now used to investigate performance situations, report statistics and monitor user behaviours.

Upgrade GUI versions

The Project Team utilized the new log parser to determine which users were executing old GUI versions. It was determined that over 60% of the users were using versions below our recommended level. Perforce has stated that some of these old versions had known performance problems and were causing some of our issues. The Project Team began a process of identifying users with old GUI versions and worked with these Users to upgrade to the recommended GUI version. An install package was created that allowed Users to easily upgrade Perforce. This package prevents Users from installing P4 EXP and also set key options within Perforce to help optimize the performance.

Limit the scope of Clientspecs

The Project Team determined through various analysis techniques that there were instances of Users “syncing” the entire depot that impacted performance for all of the Users. At these times, the commands placed very long read locks on key tables in the metadata and effectively blocked the rest of the users for up to 20 minutes. The Project Team developed a trigger that now prevents users from mapping the entire depot at one time. Users are now forced to select down at least one layer from the top of the tree to reduce the accidental syncing of the entire depot. Since this trigger has been in place, there have been no instances of the depot being locked for 20 minutes at a time and the trigger prevented 5 Users from syncing the entire depot in just the first few weeks.

Local Proxy Investigation

One team was utilizing a local proxy to improve performance for their group. The Project Team investigated the local proxy and was able to prove that the local proxy not only did not help performance, it actually slowed down performance for the group using the proxy. The only real benefit from the proxy was isolating the activity of these Users from some of the previous volume reporting tools. The group has now stopped using a local proxy. Proxies are still used by all remote locations using Perforce.

Reducing FSTAT

Volume is one of the factors which affect performance. The Project Team analyzed the type of activity looking for excessive use of the depot. Through the analysis, it was determined that some users were polling the depot for changes every second. This meant an excessive amount of FSTAT commands that provide the right conditions for the database locking issue. The Project Team worked with users to change polling to more appropriate times to reduce the load due to the FSTAT command. In addition, changes were implemented into our LAN desk install package to turn off the automatic polling option.

Divide the Depot

The Project team investigated and started the process to split our depot along functional lines. The process involved taking two exact copies of the depot and obliterating alternate part of both depots. The test obliteration process lasted over 15 consecutive days and eventually corrupted the test depot. This option was abandoned for numerous reasons:

1. It was not clear that this alternative would solve all of the performance issues
2. Key functionality and re-use of code would be lost
3. Process of dividing the depot was not stable or efficient
4. Interfacing tools were not able to support multiple depots

Archive Old Version Files

The Project Team investigated archiving files older than a key date in the past. The assumption was if the size of the depot was smaller, performance would be improved. At the 2007 User Conference, this solution was discussed with Perforce and other companies and Perforce expressed interest in addressing the archiving of older files as part of their solution. Therefore, this option was placed on hold to allow time for Perforce to propose a solution.

Conclusion

RIM has successfully addressed the performance concerns with Perforce with the average lapse time of the 5 test processes averaging below 3 seconds (a 90% improvement). At the same time, the Perforce user base has continued to expand and volume has increased. With the ever growing volume, the work required to keep Perforce operating at top speed will never be complete. The Project team is already looking at new initiatives that are in various stages of development that will provide further improvements.

The most important measure for the success of the performance project are the comments received directly from the Developers. Below are a few quotes received after the latest changes which clearly show the success of the performance improvements:

> "Wow, the depot is fast now."
> "I think you're going to have a lot of happy developers - this one included"
> "I've never seen Perforce perform this well! Great job guys!"
> "I was just using Perforce and... it's fast. Really fast. I love it."