Extending Perforce Scalability Using Job Content Synchronization

Shannon Mann, Research In Motion
December 2010

Abstract

Research in Motion (RIM) has completed an 8 month project to split our main software depot. A hidden difficulty of a split of a large depot is the impact to the bug tracking system. Job Content Synchronization (JCS) allows multiple depots to share a single link with the bug tracking system, eliminating needed changes and significantly reducing split preparation time.

This paper discusses the split project and the use of JCS to reduce the complexity of changes to the bug tracking system integration.

Introduction

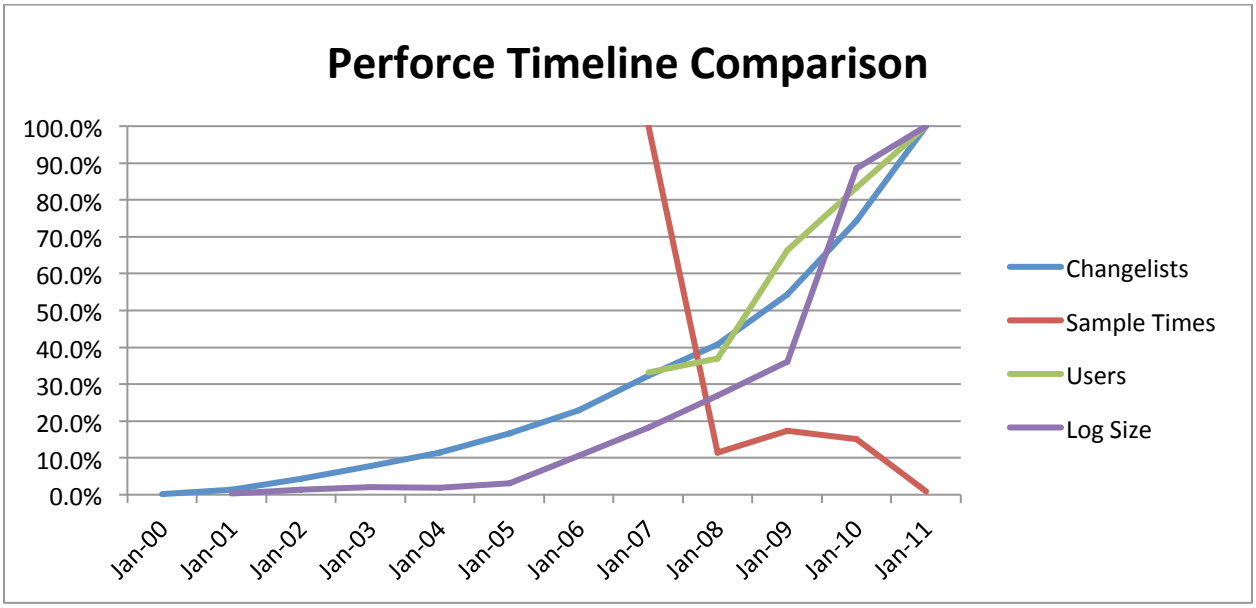
Research In Motion (RIM) is a world leader in the mobile communications market and has a history of developing breakthrough wireless solutions. RIM's portfolio of award-winning products, services and embedded technologies are used by thousands of organizations around the world and include the BlackBerry™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

Our largest depot has had a series of performance problems since 2005, and in response, we've had several projects over the years to continually improve performance. In 2006, we decided to split the depot, however, code organization at the time prevented splitting – other measures were taken that improved performance by over 10x (see 'Improving Perforce Performance by over 10x', 2008 Perforce European Users Conference for details). The decision to split was revisited in March 2010 with one group at RIM committing to the split which we completed in November 2010.

History

Perforce has been used at RIM since November 1999. Although application performance wasn't as spectacular as Perforce 2010.2, the early depot did not suffer many issues. Starting late 2005, several automated and continuous integration build systems were implemented. Soon after these new automation systems were implemented serious slowdowns appeared. By late 2006, performance during the day was so severe that a single file change sometimes took two hours. With Perforce guidance, RIM decided to split the depot, and a project in early 2007 was launched to

accomplish this goal. Investigations during this project showed that source architecture interconnections at RIM prevented a split at that time. The project team sought alternatives, and again, Perforce was helpful with some suggested hardware improvements. These changes dramatically improved Perforce performance, however, these improvements were only temporary. Increased user base and expanded automation were guaranteed to hit our performance again.



Graph note: Values for changelists range from 5700-3.5 million, for sample times from .2 seconds to 22 seconds, for users from 1750-5275 and for log size from .4 million-145 million. To graph such diverse ranges, each data point was taken as a fraction of the largest value for that set and plotted as a percent. This allows relative changes to be compared for diverse data. Plotting using four separate Y-axes would have produced the same graphs. Data for sample times and users doesn't exist prior to January 2007.

In March 2010, a large group of RIM stakeholders met with Perforce to discuss whether to split and, if so, which split approach to use. Discussions confirmed that recent source architecture changes now allowed a split for some groups. Several alternative split methods were considered – of those, splitting using a full depot copy and using protections for separation was finally chosen and a project to implement initiated.

Split Solution Overview

The selected split approach involves four stages:

a. Duplication and Separation: Create a duplicate copy of all data. Use protections table entries to partition each copy into separate unique depot trees.
b. Clientspec cleanup: On each copy, delete any clientspecs that contain only despot trees that exist on the opposite depot.

c. Offline obliterates: Do obliterates of the no-longer-needed depot tree on a copy of each depot and replay the results.
d. Clean up depot files: Find a solution to identify and remove no-longer-needed archives (still a research topic).

Duplication is done using a SAN flashsnap to copy real-time data to a duplicate disk. At split time, the depot is brought down, the flashsnap broken apart and the copy mounted on a separate machine. The protections table for each creates a 'virtual' separation, each depot having an exclusive portion of the original depot tree. Although Perforce Admin can see everything, as far as the users are concerned, two entirely separate depot trees are evident on each depot. After the split, user confusion over where a particular depot sub-tree is located is quickly and permanently resolved. Moreover, if mistakes are made in the directory placement, a simple protections table change allows a sub-tree to migrate from one depot to the other.

Clientspecs that are no longer meaningful (containing only paths that are no longer visible to the users on that depot) are deleted, producing each depot with a mostly-separated set of clientspecs. A clientspec will still appear on both depots in the rare case of a clientspec with paths on both depots. Removing unnecessary clientspecs helps reduce the size of the db.have table giving both depots an additional boost in performance.

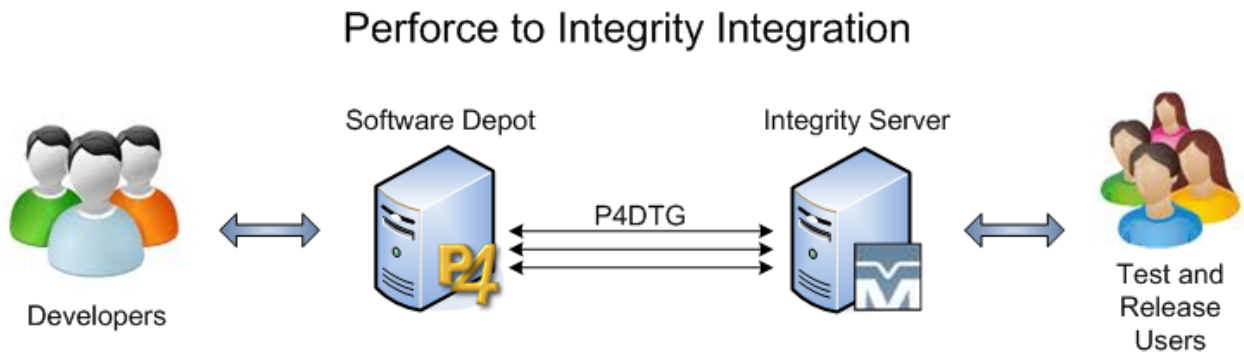
Offline obliterates are done for each new depot. A copy of a depot is created for this purpose on a non-production machine. On this copy, the obliterate commands are run to remove the no-longer-needed depot trees producing a journal that can be replayed to enact the obliteration in production. Given obliterates are long and costly operations, being able to do this operation while production is unaffected is key. Even if the operation takes weeks to complete on the non-production machine, the result is equivalent to doing the obliteration on production while the actual impact is minimized. The depot files associated with the obliterated files are NOT eliminated with this process.

The last stage, depot cleanup, is still a research topic aimed at reducing our disk footprint as not being able to remove no-longer-needed archives makes this split technique very expensive.

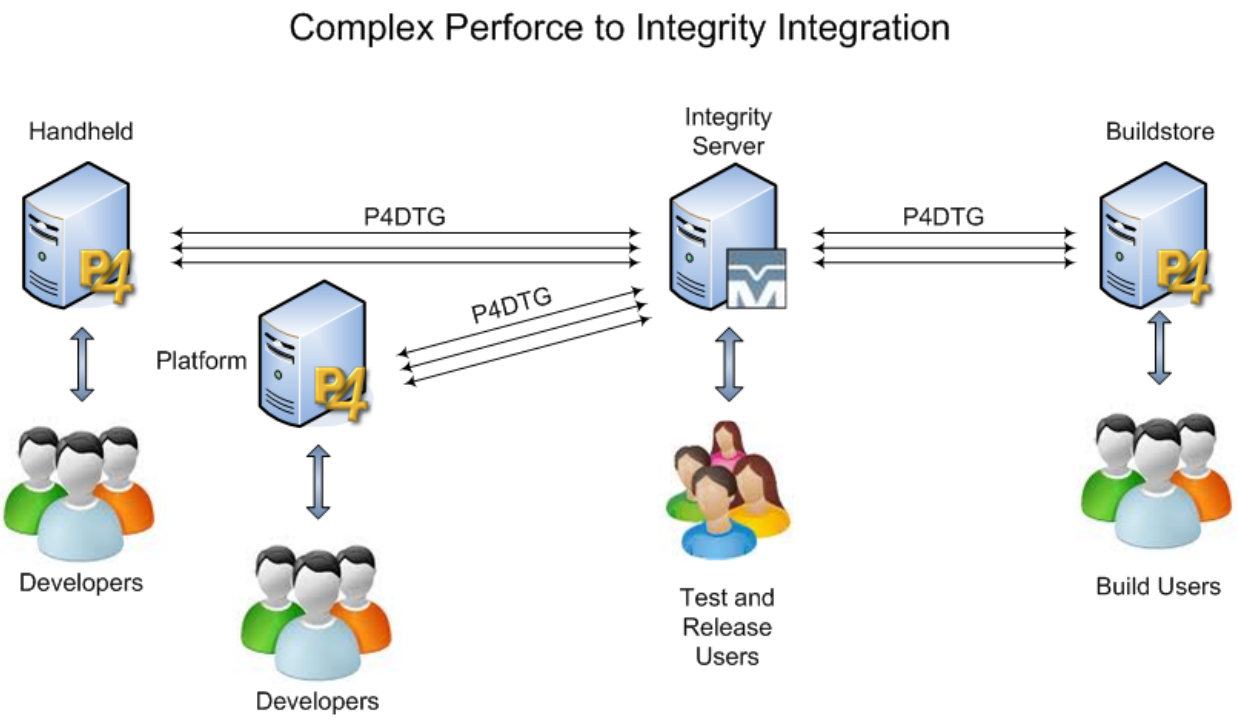
Integration Impacts

Although a split improves performance of each depot by splitting the user load and doubling the availability of db table locks, what about the tools that integrate with Perforce? Some tools will only connect to one depot post-split – largely, these tools will be unaffected by the split save possible reconfiguration to point to a new depot. Others

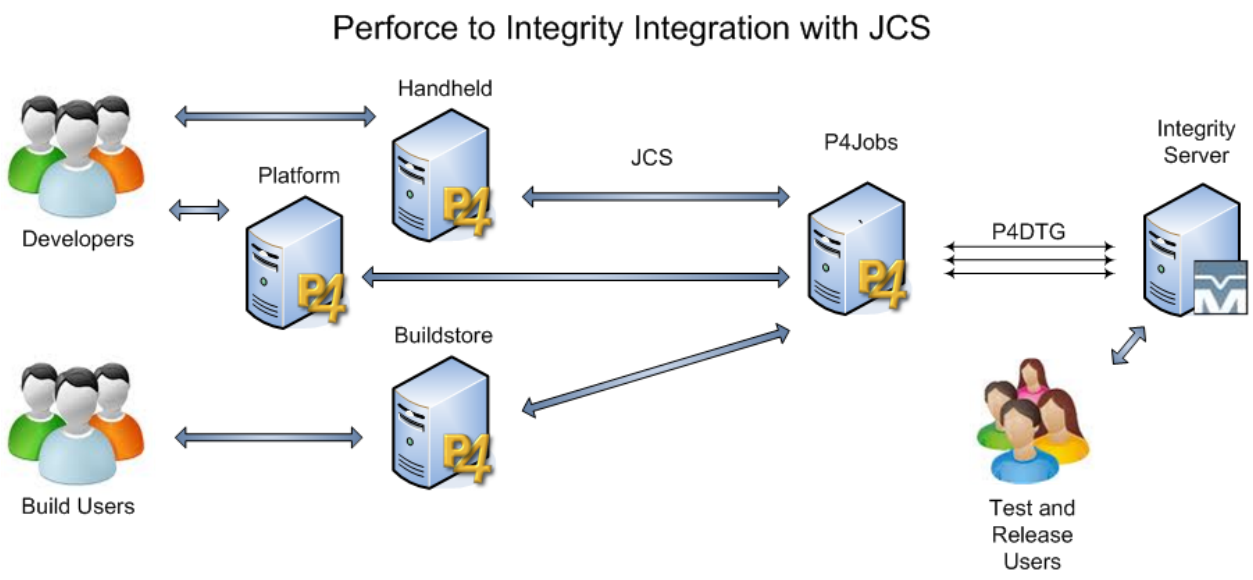
are broadly impacted as they will need to talk to both depots post-split. The split project team opened discussions with all Perforce-integrating tool owners to help them assess the impacts the split would have on their processes and tools. Most of these discussions were quite fruitful and were able to identify needed changes and potential impacts in a few short weeks. After six weeks of protracted discussions a solution for RIM's bug-tracking system, MKS Integrity, was still outstanding. At the time, the integration existed as three P4DTG-based integrations, although a fourth was being implemented and a fifth was planned.



Integrity and Perforce are tightly integrated with Perforce jobs triggering post-submission testing and release processes. Although each of those integration links could be duplicated to each split depot, there were performance issues on Integrity and possibly some on each Perforce depot. As we planned further depot splits the impacts would not scale linearly – they would multiply. Worse, in order to make Integrity work properly, the application would need to be able to 'reassign' an issue between the depots – we would need to make Integrity 'aware' of multiple depots. As there are dozens of integrations with Integrity, the integration impact would grow tremendously. Lastly, items in Integrity automatically create their associated Perforce jobs. As there would now be multiple depots for this creation, either all depots would get a copy of the job (increasing the impact again), or the user would have to select which depot the job should appear, usually an impossible task for the user. These issues indicated that a bug-tracking-centric approach was not the right solution as the approach was introducing more problems than solving - another solution was desperately needed. The issues with the integration to Integrity put the split effort in serious jeopardy.



A brainstorm session to identify an alternative solution produced an abstraction where all activity between Perforce depots and Integrity were considered one virtual process. In order to preserve the existing Perforce-to-Integrity integration, a Perforce depot would have to serve as the middle point, a jobs-only depot. The next step recognized that job information from the production depots would need to be transferred and kept up-to-date by some process. This process became Job Content Synchronization (JCS) which would handle copying appropriate information between the production depots and the jobs-only depot. A trigger would request job information on an as-demanded basis and a separate process would keep those jobs up-to-date. This approach minimized the amount of job data that would need to be maintained, as only jobs used by users would be copied and synced to each depot.



Additional discussions suggested that JCS would be best implemented using P4DTG as the updating engine; each production depot would act as the source control system and the jobs depot would act as the defect tracking system. This approach keeps application-domain information within the application suite. In late May 2010, a description of the proposed JCS solution was sent to Perforce Support to get their input as to whether this was a good solution, whether alterations were needed or whether a different solution would be best.

After two weeks, Perforce support requested a meeting to discuss the proposed JCS solution. During the meeting they explained that they had spent the previous two weeks developing a reference implementation much to our surprise and pleasure. To the basic process they added another addition: P4Broker to redirect 'p4 jobs' commands to the jobs depot, making the whole system invisible to the user. With that alteration, the showstopper problem had found its lynchpin solution.

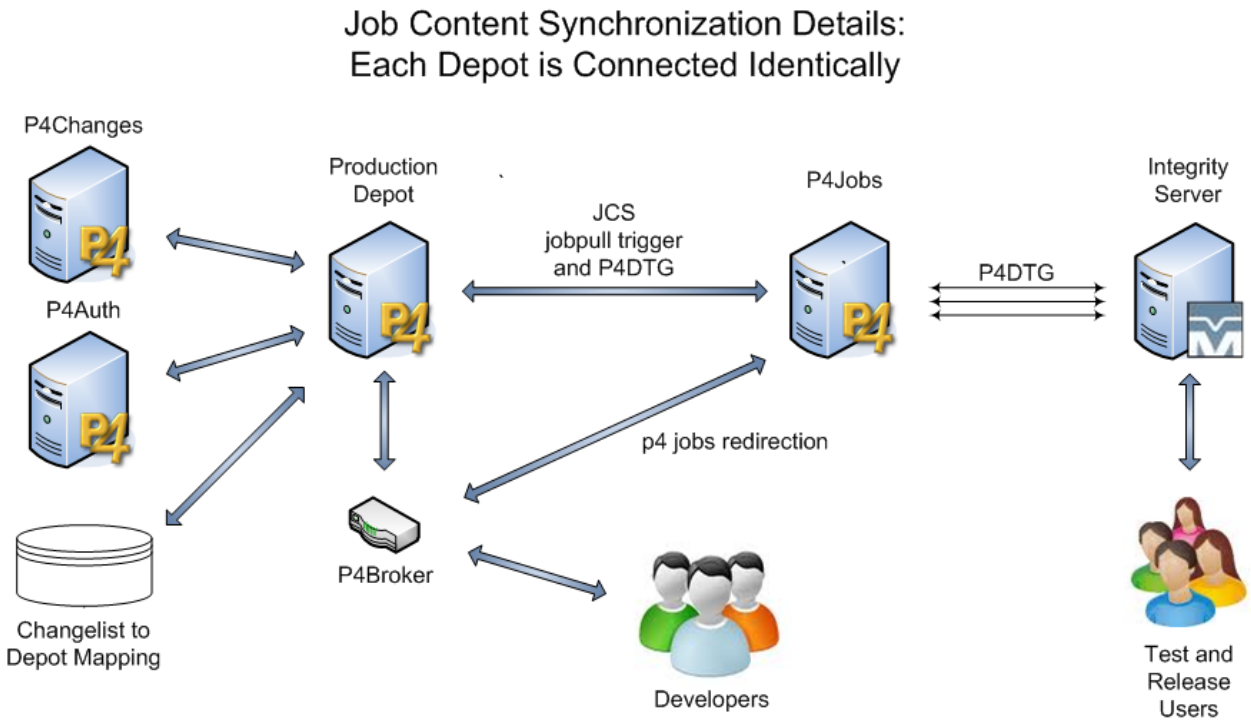
The Rest of the Recipe

In order to support the needs of JCS and the various impacted tools and integrations, further facilities were needed:

a) P4Auth – supports users connecting to multiple depots with one login
b) P4Change – makes changelist numbers unique amongst connected depots
c) P4Jobs – the interconnecting depot separating JCS from the bug tracking system
d) P4Brokers – needed to redirect all 'p4 jobs' commands to the P4Jobs depot
e) Changelist to depot mapping DB – maps a changelist to a particular depot

These extra facilities add significant complexity to the Perforce environment, however, the benefits have been many. Adding new depots to the environment involves simple

configuration taking minutes. Adding more bug-tracking instances (and even different types of bug-trackers) is also straight forward and supportable. Moreover, all additions involve linear growth at all levels – only the specific connections are involved in increasing complexity keeping growth permanently linear, a vast improvement over the solutions initially considered.



Pre-Implementation Testing

In order to test the integrations and the split protections, two duplicate copies were created on a test machine with test versions of the different protections tables, complete with broker, P4Auth, P4Change and P4Jobs depots. A test P4Jobs-to-Integrity integration was created, and the JCS links between the split depots and P4Jobs were also created. This environment allowed our users and tool developers to test the split protections and to test how well their tools handled the split. As well, our build release team was able to test many builds against the new protections as the changed protections may very well have split a build into both depots.

The protections changes were done using a tagged protections table. Each line in the protections table was tagged with either ‘both’, ‘ENT’ or ‘HH’. A split script converted the tagged file into separate protections tables ready for loading. A separate script loading the generated protections directly into the test depots. Three protections table versions were generated. The first version was a complete copy of everything – essentially the tagged file with the tags removed. The second version was the lines tagged for ‘both’ and ‘HH’ for the ‘handheld’ depot, the third version was the lines

tagged as ‘both’ and ‘ENT’ for the ‘platform’ depot (originally named ‘enterprise’ depot). The first version was compared to the production protections when incorporating recent ‘live’ changes into the test protections while preserving order. Adding those differences to the tagged file and then rebuilding and re-comparing to the original allowed a difficult merge process a straight forward (but tedious) solution. The main advantage to the merge process was that the output proved the merge correct every time, the primary concern. Moreover, a few changes to the split script allowed it to generate a web page that was instantly updated with each change allowing users to see where a directory was destined to appear. Changes that users required were quickly added with the web page confirming the move at the same time. Post-split, the same scripts were used to migrate paths that for various reasons ended on the wrong depot. In this way, users had incorrect paths resolved very quickly, most in the first 24 hours post-split and all shortly after contacting us.

The test environment ran from July through November. Db file and depot file contents were updated in late August and early October. Protections updates from production happened irregularly given the constraints of testing. Each protections update involved between 50 and 300 changes all done by hand by the process noted above.

The test environment and process allowed us to greatly reduce post-split issues, especially with release build systems. Although at great effort, the results make clear not taking that effort would have been disastrous. Many issues were caught months before the split that would have had great impact on ship dates and on development process quality and sanity.

Split Rollout

Given the complexity of the split and the potential for failure, the supporting additions were implemented early. In effect, the problems potentially encountered were removed from the actual split, reducing risk. As well, by implementing early, these additions were further tested in isolation from the actual split, removing possible conflicting reasons for failure at split time. P4Broker was implemented in late September 2010 with P4Auth, P4Change and P4Jobs depots implemented two weeks later.

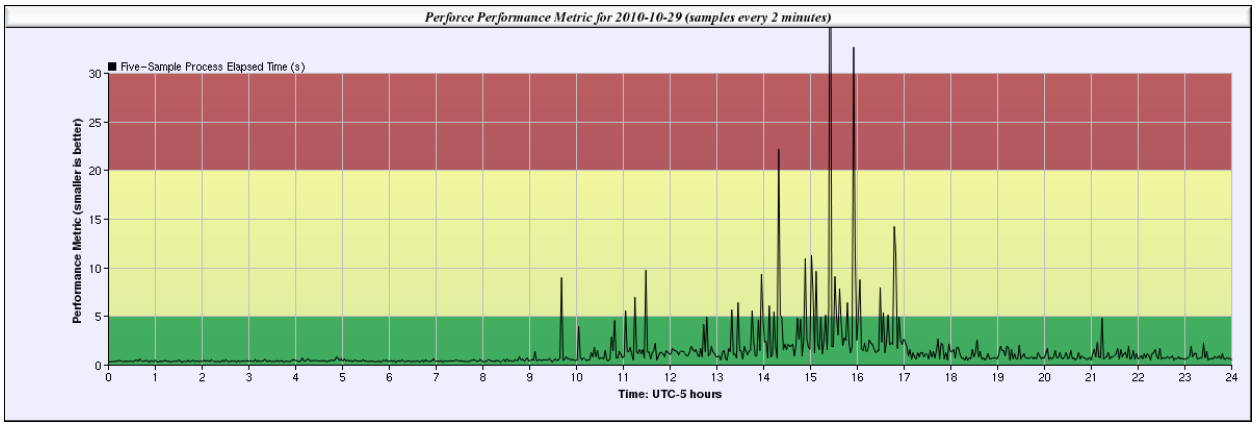
The split was scheduled for late November 2010 after months of negotiating around release schedules and readiness for the split. The split took two hours to complete, with the actual split process taking 30 minutes. In the seven days following the split, 46 messages indicating problems with the split were received from a community exceeding 5000 users. Two-thirds of the messages were due to user confusion as to where their code now resided. One-sixth of the messages were requests to move a path to the opposite depot and the remaining sixth were real issues requiring correction post split.

Over-all, the split has been massively successful with minimal impact, a testament to careful planning and plenty of testing.

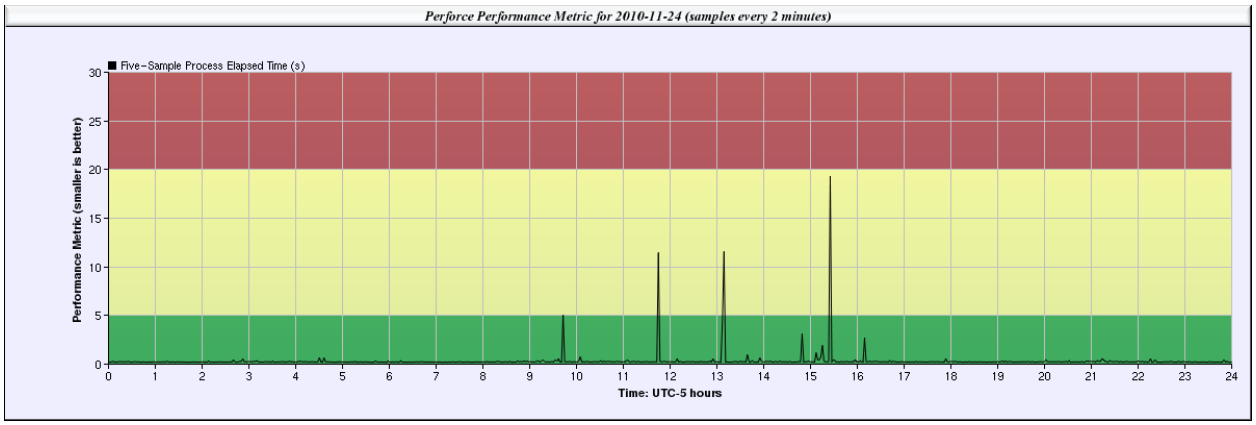
Post-Split Benefits

Perforce performance has improved post split, as expected. The Platform depot has excellent performance while the Handheld depot has generally improved performance with some performance issues.

Software depot – before split



Handheld depot after split



As the charts show, our performance metrics in the weeks leading to the split show significant noise and noticeable area under the graph – indications of performance issues. After the split, Handheld depot performance is much better. The Platform depot graph is not included as it is a dead-flat line hardly distinguishable from the X-axis.

New depot additions now take minutes to add. Since we've split, we've added two depots to the environment, the IT depot and Buildstore. The IT depot is a long-time sibling to the pre-split depot and Buildstore is a binaries-only depot for post-built objects – a clearing house for build support and external vendor binary drops.

The solution has proven very modular. P4Auth and P4Change support is trivial to add to an existing depot or a new depot, opening the way for bug-tracking integration with JCS. Even without using JCS, RIM uses P4Auth and P4Change with an additional acquisition-related depot, simplifying user support. RIM's integration with Code Collaborator, a code review tool, was greatly simplified as instead of copying user and group information into a 'perms' depot, P4Auth provided that facility for free.

P4Broker has provided a method to partially 'pause' Perforce, allowing recovery during performance spikes. Although not preferred, the 'pause' allows more time to develop other solutions for the performance issues .

Lastly, the experience has given our support team confidence to continue splitting. Other splits are planned in the near future (late 2011 or in 2012).

Post-Split Issues

The split approach used by RIM is effective, straight-forward to implement and does give good performance benefits. However, there have been issues. The process RIM followed is extremely expensive with disk as we use SAN disk for both the db files and the depot files. A move to filer is possible for depot files, however, recent experience suggests that there may be performance issues with RIM filer implementation.

The implementation exposed a bug in 'p4 protects' that affects the version of Perforce we used during the split (Perforce 2009.2). The 'p4 protects' command always looks at the local user and group tables, even when P4Auth is in use. RIM used 'p4 replicate' to copy P4Auth tables to the local system, and, for the depot affected, replaced the local user and group tables with a symlink to the replicated tables. Perforce has since corrected this bug in Perforce 2010.1 and later.

RIM found a significant memory leak in the broker that threatened to scuttle the split effort, however, Perforce was able to provide a corrected broker in short order. The memory leak happens only when command redirection is turned on. The latest broker has this memory problem corrected.

Lastly, the handheld depot is experiencing some performance issues. The performance issues are expected as the size of the split was not balanced. Fewer groups could be migrated than preferred, leaving the handheld depot larger than desired.

Future Directions

As always, Perforce performance is an ongoing issue. User base increases and an ever-expanding desire for greater build and data mining automation means the load on Perforce only increases. In 2007, RIM had 1750 licenses and never ran more than 1.5

million commands/day. RIM now has over 6000 licenses and regularly runs more than 4 million commands/day on a single depot/build replica pair. RIM has had peaks of 15 million commands/day during a particularly busy day.

Given user thirst for performance, further splits are likely. RIM is looking at less costly split solutions using new technology - two solutions are under consideration, perfsplit++ by Perforce and a client-side solution under development at RIM.

Cleanup has continued, both to improve performance by reducing db.have table size and by reducing disk overhead.

Data mining in Perforce is a major drain on Perforce performance. The P4toDB solution is being investigated as an alternative, although more investigation of the lack of permissions support is needed.

A hardware refresh is nearing completion, replacing our Sun X4600 servers with IBM x3950 X5 servers containing more cores and more memory. Although hardware improvements are always expected to give mild improvements, initial experience suggests more than normal improvements may be had.

The IO scheduler traditionally used by the DB filesystem is completely fair scheduling (CFS). Given our DB files are on SSD-based SAN, Perforce Support suggested moving to deadline scheduling. Significant testing confirmed that deadline scheduling is superior to no-op scheduling and significantly better than CFS. RIM is now moving to deadline scheduling. This IO scheduler change should relieve the current problems associated with read storms.

Lastly, an upgrade to Perforce 2010.2 is underway, with expected completion before late May. A number of improvements in Perforce 2010.2 are expected to give further performance improvements, including relief from some bugs found along the way.

As always, further performance improvements are sought, considered and implemented where possible. Performance improvements is an ongoing and never-ending process.

Lessons Learned

The split was a long arduous journey which originally began in late 2006. Many new lessons were learned over the last nine months of that journey:

a) Each depot sub-tree in Perforce needs an owner, someone who takes responsibility for that sub-tree. RIM's split was complicated not just for technical reasons, but also due to political issues arising from multiple teams overlapping sets of files. Tree owners can reduce these conflicts and spread the effort to decide where each sub-tree will reside.

b) Early engagement of internal tool owners is critical. RIM's efforts here helped eliminate most of the post-split trauma by getting people who can make the change to integrating tools involved.
c) A full copy of the post-split environment for testing is critical. Providing this environment and keeping it up-to-date allows testing and alterations to happen prior to split, not after in crisis-mode cleanup. Getting people to test their changes against the test environment was a large challenge, but, those efforts proved very important. Only users and groups who didn't test against the test system had significant issues post-split.
d) Protections-based splitting[i] allows extreme flexibility during testing and even after the split. The flexibility and speed of change allows users desires to be implemented quickly and errors to be corrected quickly. Using a tagged protections file as the seed for a protections split process proved very valuable, as it enabled the flexibility to be handled quickly. Having some system in place to make quick changes and corrections will be sorely needed.
e) A system to track users changes was needed and was unavailable. The Perforce split team did not have resources available to develop this facility, so requests for changes had to be handled by-hand. When there were conflicts in requests, helping the conflicting users to connect proved difficult when the first request got buried amongst the significant inflow of email and distanced with time. A simple web page to track requests would have sufficed, but with resource constraints, could not be supported. Take time to implement this first.
f) Perforce was critical to the decisions, the outset planning, correcting two potentially dangerous problems and supporting all stages of the split process. Further, their efforts with the JCS solution placed them ahead of all, turning a proposal into a reference solution in very short time. Before attempting any significant change, discuss it with them. Perforce has long proven capable, knowledgeable, concerned and committed to supporting their product and they do an admirable job. Seek their wisdom. You won't regret it.

[i] Implicit protections grant broad access and reduce access when necessary through exclusions. Explicit protections only grant the exact access you need without needing exclusions. Since May 2010, RIM uses explicit protections as testing showed that implicit protections were 60 times more expensive. The conversion reduced our protections expense by 10 times as we didn't convert all exclusions. Counterintuitively, our protections table line count expanded almost eight times (526 to almost 4000 lines). Explicit protections greatly simplify protections-based splitting as simple division of lines properly separates all paths. Implicit protections can still be used, however, the split depots will perform less well due to increased exclusions.