**Regression Prevention System for Performance Test**

**1. Overview of the Features**

We develop this system to run Performance test on each Check-in from SNAP, so that regressions can be found early.

Our RPS system has the following components: Build Monitor, Test Scheduler, WTT controller, RPS website, Failure & Regression analyzer, and RPS database. It’s a database driven system that each component runs independent and communicate indirectly through the database. We have brief summary for the basic features of our RPS system:

**1)** The Build Monitor (running as a Windows Service so it can be easily stopped and automatically restart upon machine reboot) will monitor the SNAP build for each check in, detect what kind of changes in each check in (product change, test change, etc.), cache the build that satisfied our cache rules (for example, we may specify that we only cache the build that has product change), and insert a pending test job in the database for each build.

During caching, if it finds the source is zipped, it can unzip it automatically.

**2)** The Test Scheduler (running as a Windows Service so it can be easily stopped and automatically restart upon machine reboot) will monitor the test runs in the database.

If there is pending runs and there is no test running, it will pick up the next pending run based on our test scheduling rule (for example, we may specify run high priority tests first, then run normal priority tests, and skip some normal priority tests when there are too many pending tests, and we can also specify that it always runs test on the latest cached build and skip others).

It will then schedule the test run in WTT (on the dedicated lab cell for RPS) via Performance Lab Runner (PLR, a command line tool that user by Performance Test to run test in WTT).

It will automatically choose if the lab machine will be imaged or not before the test run according to the imaging rules we specified (for example, we may specify to imaging the machine after 5 consecutive runs and if the number of pending test runs is less than 6).

**3)** The RPS website can display the detail information of test run for each build, filtering the test runs history based on the filter rules the user specifies. The user can also view the change list for each build, the test result for each build and the regression information and graphs for each build.

It provides UI interfaces to cancel pending runs, re-run test runs, change the test run priority.

And it also provides UI interfaces for user to view/change the configuration data (like cache filter rules, test run rules, imaging rules, etc) for build monitor and test scheduler.

It enforces the security by using roles: every one can view the website, but only users in admin role can modify the configuration data and change test runs.

**4)** The RPS system has log and warning functionalities, which will create .log and .err files for the build monitor and test scheduler, and send email to warn if there is error or specific event happens (for example, a test is hang there for a long time).

**5)** The Regression Analyzer will check if the build causes any regressions after the test is done, and send email with the regression information

**6)** The test results for each test run will be summarized and displayed in the indiperf website just like the test results for our daily lab runs.

**2. Design Overview**



The RPS system has five major components and a database. Each component runs mostly independently and has very little dependencies to other components. ***The database is the central point for communication and storing test running status***. Each component can notify other components of any changes by modify/insert values in database, and each component can get changes/notifications/messages by retrieving values from the database.

This design minimizes the dependencies among components, and stores all the running/changing status in database. It’s easy for maintaining (for example, program crash recovery) and extending (for example, to cancel or re-run a test can end up as just modifying some values in database).

**Build monitor:**

1. Build monitor search on SNAP to get the next build numbers (given a current build number), based on some constraints: build is finished; it has either test or product code changes, etc. If it can’t find the next build number, it will keep asking for it until it gets one.

2. Then it caches the build onto the File Server.

3. After cache is done, it inserts a pending test job row into the database, and continues on step 1.

**Test Scheduler:**

1. Test Scheduler will running independently from Build Monitor, it will keep checking if there is test job in “Running” status. If not, it will find the next “Pending” test job (based on some priority rules) inserted by Build Monitor.

2. Test Scheduler copies the tplans/tsources from File Server for the specific build to indigo-perf-cc where the Perf Lab Runner(PLR) is running, and then schedule the test (using PLR) on WTT: install/setup product codes, run test, post test operations.

3. It then changes the specific test job running status to be “Running” in the database, and continue on Step 1.

**WTT controller:**

1. After test is done, the WTT controller will change the specific test job running status to “Done” in the database (by a WTT job).

2. And after test is done, the WTT controller will also call Failure & Regression Analyzer (by a WTT job) to analyze the test results.

**RPS website:**

1. Display the test run history and status.

2. Allow user to view and modify configuration data.

**Failure & Regression Analyzer:**

1. Analyze the test result after test is done.

2. Set the “result” attribute of the test job row in the database to be “success”, “fail”, or “regression”, and send out email.

**3. Builder Monitor**



1) Get next build

a. First get all the builds dir in SNAP

b. Filter out the builds whose drop time is older than current build

c. Filter out unfinished/unsuccessful build by check the existence and content of specific files in SNAP

d. Filter out builds without product change by check the content in the checkin summary file

e. Sort the remaining builds by last write time.

f. If “only cache latest build” is specified, return the last one in the sorted list.

g. If “only cache latest build” is not specified, check the value of “skip cache rule”, and skip the builds from the start of the sorted list, and return the next build.

2) Cache build

a. cache build is done by Utility.CacheManager.

b. it first create mutex files in target dir to make sure the cache is thread safe.

c. then it uses Utility.FileDirHandling.Copy to recursively copy build to the file server, only copies a file if that file doesn’t exist in target or the file in target is older.

d. release mutex when copying is done.

e. retry the cache 2 more times when it failed at some point.

f. if the build is zipped version in SNAP, unzip the build in file server after it’s cached, using Utility.FileHandling.ZipUtil.unzipZipList, and delete the zipped version in file server to save space.

3) Insert test jobs

a. use Utility.DBoperation.insertTestRun to insert a pending test run row in RPS\_testrun table in database, with status “pending” and priority as normal priority 3. The checkin number (change list number) will also be set in the database for the build.

**4. Test Scheduler**



1) Get next run

a. check if there is still test running. If there is, sleep a while and try again. If a test is running too long, send the “test hang” email for warn.

b. when there is no test running, try to get the next run.

c. first, get the next pending run with high priority, i.e. the priority higher than 3, if there is some, return the next pending run with highest priority and oldest creation date (highest priority first, then creation date).

d. if there is no pending run with priority higher than 3, try to get the normal (priority 3) pending run. Sort the priority 3 test run by creation time. If “choose latest run” rule is specified, return the latest pending run with priority 3. If not, skip the specific number of test run according to “skip test run” value, and return the next pending run. The skipped run will be set as “cancelled” in database.

e. if there is only pending test run with priority lower than 3, get the next pending run sorting by priority and creation time.

2) Copy test

a. first, use Utility.FileHandling.FileDirHandling.deleteDir to delete the old test in indigo-perf-cc.

b. then, use Utility.CacheManager.copyTest to copy the tplan/tsources to indigo-perf-cc.

3) Scheduler test

a. set the start time for the test, it will be used to check if the test running too long.

b. find out how many consecutive runs without imaging and how many pending test runs, send out warning email if there are too many pending runs. And setup imaging if the number of consecutive runs is larger than a specifice value and the number of pending runs is less than a specific value.

c. scheduler the test through a scheduler scripts (.cmd file), who will use PLR to schedule test.

d. set the approiate field value for the scheduled test in database: set status as “running”, set labcell, starttime, ifImage field.

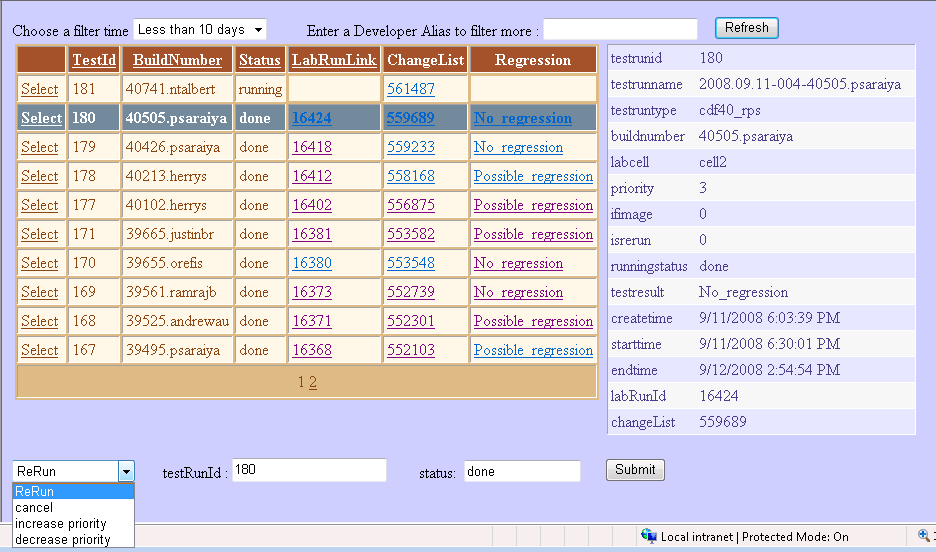
e. set the number of consecutive runs without imaging.

**5. RPS Website**



The website is hosed here: <http://xws-perf-cc/RPSwebNew/>

1) Test Run Manager

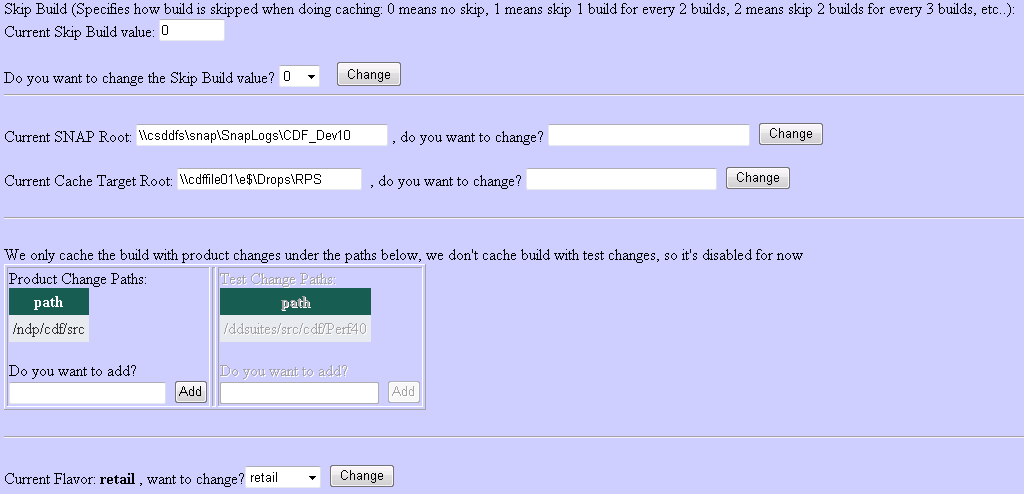


The user can view and filter the test run information, with summary view and detailed view. And the results can be sorted by different columns by clicking the column title. Clicking the LabRunLink for an individual run will bring up the test results for that run. Clicking ChangeList for an individual run will bring up the checkin and changeset information for that build (and for merged build, there will be multiple links, one for each checkin). Clicking the “Regression” will bring up the possible regression page if there are possible regressions, and bring up regression history chart if there are no regressions.

The user can filter out the test run information by specify the filter time and developer’s alias.

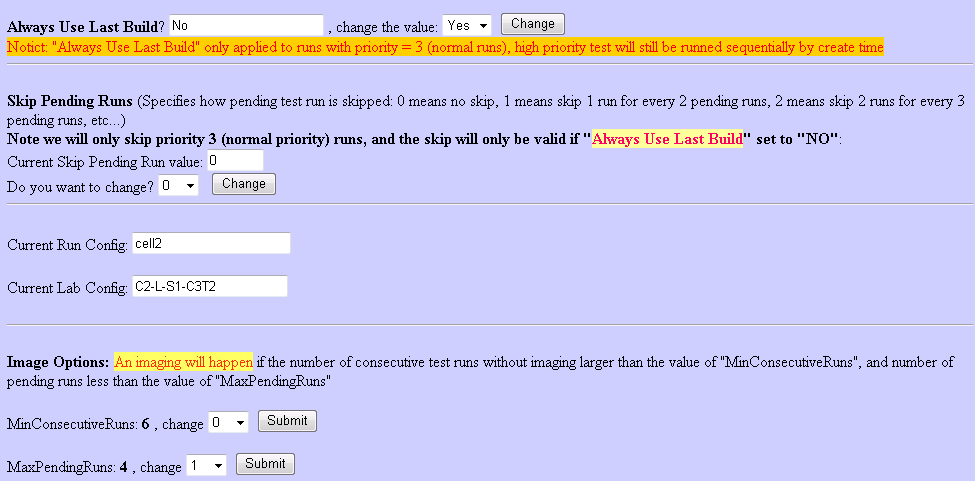
The user with admin access can also manage the test run by re-run, cancel a test run or change the priority of a test run.

2) Build Monitor Configuration



The user with admin access can modify the value of the configuration data for the Build Monitor: 1. Skip build value for build caching); 2. Current SNAP root where build monitor cache from and current target root where the build monitor cache to; 3. The paths for product change (we only cache build with product changes); 4. Which flavor we use for RPS system: retail, free, or checked.

3) Test Scheduler Configuration



The user with admin access can modify the Test Scheduler configuration data. He can choose to always run test on the latest cached build (we only use this when there are too much tests pending, so that we keep running test on most recent one and skipped others).

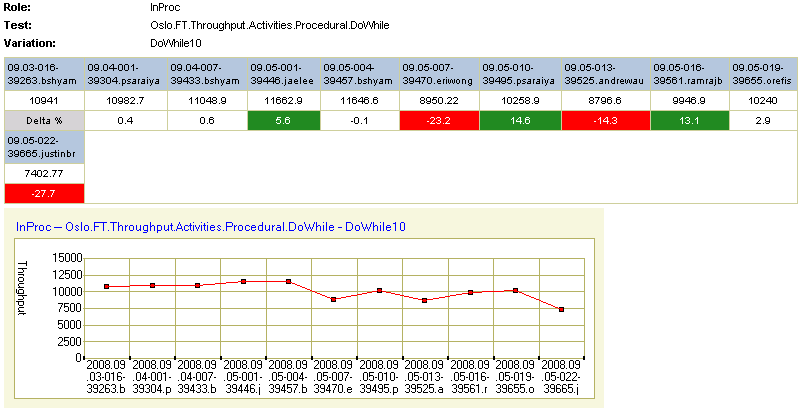
The user with admin access can also specify the number of skip pending runs, this is also only used when too much runs are pending there.

The user can view the labconfig and runconfig but can’t change it, because it’s easy to get things run. We can manually change it in database when needed.

The user with admin access change the imaging options: MinConsecutiveRuns is the number of consecutive runs before the machine gets imaged, and MaxPendingRuns means the lab machine should not be imaged if the number of pending runs larger than that value, because imaging machines takes time.

**6. Regression Analyzer**

When a test is done, the Regression Analyzer will be called to analyze the test data to see if there are possible regressions. If there are, an email will be sent out with the summary and detailed information about regressions, and a link to the graph of the regressions. Below is an example of the regression graph.



**7. RPS Database**

There are 4 tables: RPS\_changepaths, RPS\_config, RPS\_testrun, RPS\_currentvalue, RPS\_permission.

1) RPS\_changepaths

CREATE TABLE [dbo].[RPS\_changepaths](

[id] [int] IDENTITY(1,1) NOT NULL,

[type] [varchar](50) NOT NULL,

[path] [varchar](max) NOT NULL

) ON [PRIMARY]

Type: product change or test change

Path: the paths

2) RPS\_config

CREATE TABLE [dbo].[RPS\_config](

[snap\_root] [varchar](50) NOT NULL,

[job\_pattern] [varchar](50) NOT NULL,

[build\_success\_file] [varchar](50) NOT NULL,

[drop\_dir] [varchar](50) NOT NULL,

[build\_change\_file] [varchar](50) NOT NULL,

[cache\_target\_root] [varchar](50) NOT NULL,

[product\_dir] [varchar](50) NOT NULL,

[architect] [varchar](50) NOT NULL,

[flavor] [varchar](50) NOT NULL,

[test\_dir] [varchar](50) NOT NULL,

[test\_sub\_dir] [varchar](50) NOT NULL,

[test\_target\_root] [varchar](50) NOT NULL,

[runConfig] [varchar](50) NOT NULL,

[labConfig] [varchar](50) NOT NULL,

[filter\_old] [int] NOT NULL,

[filter\_skipList] [int] NOT NULL,

[filter\_unfinish] [int] NOT NULL,

[filter\_verifyProductChange] [int] NOT NULL,

[filter\_verifyTestChange] [int] NOT NULL,

[filter\_cacheUseLastBuild] [int] NOT NULL,

[filter\_skipBuild] [int] NOT NULL,

[skipTest\_rule] [int] NOT NULL,

[testUseLastBuild] [int] NOT NULL,

[buildMonitorLogPath] [varchar](50) NOT NULL,

[testSchedulerLogPath] [varchar](50) NOT NULL,

[RPSwebsiteLogPath] [varchar](50) NOT NULL,

[ifZip] [int] NOT NULL,

[maxRuntime] [int] NOT NULL,

[emailPassword] [varchar](50) NOT NULL,

[minRunsNoImaging] [int] NOT NULL,

[maxPendRunsImaging] [int] NOT NULL,

[buildMonitorSleep] [int] NOT NULL,

[testSchedulerSleep] [int] NOT NULL,

[changeSetRegx] [varchar](50) NULL

) ON [PRIMARY]

Those fields contain all the config data for the whole system. Some of them can be changed from the RPS website and others have to be changed manually in the database.

3) RPS\_testrun

CREATE TABLE [dbo].[RPS\_testRun](

[testrunid] [int] IDENTITY(1,1) NOT NULL,

[testrunname] [varchar](50) NULL,

[testruntype] [varchar](50) NULL,

[buildnumber] [varchar](50) NOT NULL,

[labcell] [varchar](50) NULL,

[priority] [smallint] NULL,

[ifimage] [int] NULL CONSTRAINT [DF\_RPS\_testRun\_ifimage] DEFAULT ((0)),

[isrerun] [smallint] NULL,

[runningstatus] [varchar](50) NOT NULL,

[testresult] [varchar](50) NULL,

[createtime] [datetime] NOT NULL,

[starttime] [datetime] NULL,

[endtime] [datetime] NULL,

[labRunId] [int] NULL,

[changeList] [varchar](50) NULL

) ON [PRIMARY]

Those fields contain detailed information for each test run. It connected to the indiperf database by the labRunId field, which is the value of the run\_id of a lab run for indiperf.

4) RPS\_currentvalue

CREATE TABLE [dbo].[RPS\_currentvalue](

[buildnumber] [varchar](50) NOT NULL,

[consecutiveRuns] [int] NOT NULL,

[currentdate] [datetime] NOT NULL,

[currentRunNumber] [smallint] NOT NULL

) ON [PRIMARY]

Buildnumber: the build number for the latest cached build

consecutiveRuns: the current number of consecutive runs without imaging lab machines

currentdate: the start time for the latest test run, it’s used to detect if there is a test hang in lab.

CurrentRunNumber: this value is used to set the runnumber field for each test run. It starts as 1 for each day, and keep increasing for each test in that day.

5) RPS\_permisstion

CREATE TABLE [dbo].[RPS\_permission](

[alias] [varchar](50) NOT NULL,

[role] [varchar](50) NULL

) ON [PRIMARY]

Alias is the alias of the user

Role controls the permission of the user: to make the user an administrator of RPS website, the value of role needs to be “admin”