Table of Contents

2013 HATS Pileup Exercises

Introduction

The goal of this exercise is to introduce the tools that are used for studying pileup and understanding the information used in calculating the pileup distributions and passing them along to the Collaboration. Advanced tools, like calculating pileup for prescaled triggers, will also be explored.

The Exercise is divided into a number of Steps. The color scheme of the Exercise is as follows:

- Commands will be embedded in a grey box, e.g.

```
ls -ltr
```

- Configuration files (full or snippets) will be embedded in a blue box, e.g.

```
datasetpath            = /Jet/Run2010B-Nov4ReReco_v1/AOD
```

- ROOT macros and codes will be embedded in a pink box, e.g.

```
  const TString var     = "ST";
```

- Output and screen printouts will be embedded in a green box, e.g.

```
 Number of events with ST > 2000 and N>= 3 is 10099. Acceptance = 0.842075
```

- Important messages will be highlighted in red

Other Useful Information

The basics of what the pileup distributions look like in Data and MC, and how the distributions in MC are generated are available at the PileupTwiki linked from the 2011 PVT and this link to the 2012 PPD web pages. A tutorial for how to do pileup reweighting can be found in the CMSDAS pages (click here), since it has been presented as a Short Exercise since 2012.

Getting Started

First, a set of introductory slides can be found (linked to this TWiki page) here:

- Pileup_Overview_HATS13.pdf: Intro slides

Next, we are going to examine what goes in to the pileup calculation. What you need:

- You should set up a `CMSSSW_5_3_11_patch4` working area. Follow these steps once you have properly set up your CMS environment variables:

```
cmsrel CMSSW_5_3_11_patch4
cd CMSSW_5_3_11_patch4/src
cmsenv
(need to use Git now:) git cms-addpkg RecoLuminosity/LumiDB
(compile!) scram b
```

This package contains all of the luminosity query and pileup calculation scripts. If you haven't set up Git yet, you should take the time to do that. Check out the links here. Particularly important is setting up your ssh keys so that you can check out packages from the repository.

- A JSON file with good luminosity sections from the 2012 running:

- For today, we'll use the one linked from the bottom of this page: Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt (also found in `/uscms_data/d2/mikeh/work/HATS/CMSSW_5_3_11_patch4/src` or on `lxplus` in `~mikeh/public` if you want to copy it from there.) You need this in your working area.

Calculating Luminosity for Pileup Work

You may already be familiar with the process of calculating the luminosity covered by your analysis. Here, we are going deeper into the luminosity information because we need more details in order to calculate the pileup. This tutorial leads you through the steps outlined on the TWiki page describing how the luminosity information is used for the pileup calculation, which can be found here. Along the way, we'll take a look at some interesting features.

As an input to the pileup calculation, the instantaneous bunch-by-bunch luminosities and the delivered and recorded luminosity for each LumiSection are necessary. These are obtained by doing a Luminosity DataBase query using the script `lumiCalc2.py` provided in `RecoLuminosity/LumiDB` which you just dowloaded. The bunch-by-bunch luminosities are ***required*** here, because it is the luminosity of each collision that determines how many additional interactions can occur each time the beams cross. Note that, at present, the luminosity can only be provided bunch-by-bunch by the HF detector. While it is fast, it has known non-linearities and aging propensities that need to be monitored and corrected. We will address this later in this exercise.

The luminosity database query can look like:

```
lumiCalc2.py lumibylsXing --xingMinLum 0.1 -b stable -i Cert_190456-191202_8TeV_22Jan2013ReReco_C
```

Giving the script the argument `lumibylsXing` causes it to produce a (very large) .csv file containing all of the needed luminosity info for each bunch for each LumiSection, including the bunch-by-bunch luminosities. You should ***never*** do this for calculating the luminosity for a given analysis. You don't need the luminosity information at this level of detail, and it places a huge burden on the luminosity database.

Note the parameter `--xingMinLum`: this determines which bunch crossings will be ignored because their instantaneous luminosity is too low. The units here are 10^30/cm^2/s (Hz/ub). Here, we have set it to a low value just to see what sort of junk is lurking in the luminosity measurements.

Another note: during data collection, the default is to calculate the pileup using the `DCSONLY` set of JSON files. This is, by definition, a super-set of the luminosity sections available for certification. That way, pileup information is available for any possible luminosity section with real integrated luminosity.

So, you could run the script using the command given above, but don't. (This takes a while, even for this small JSON file. Don't even think of doing this for the whole dataset.) Just copy the output file `lumi_2012a.csv` from `/uscms_data/d2/mikeh/work/HATS/CMSSW_5_3_11_patch4/src` or from `~mikeh/public` on `lxplus` to your working area. (Or just use the full file path in the commands below.)

Calculating Luminosity Distributions for Pileup

Next, we need to make a JSON format file containing the appropriate pileup information for each LumiSection. This step is usually done centrally for all of CMS and need only be done once unless either the luminosity corrections (the pixel luminosity is added, for example) or the list of LumiSections in the DCSONLY sample (never has happened) changes. A few words on the necessary information that must be stored in the JSON file. We must have a way of calculating the number of expected pileup interactions, so we need the single-collision instantaneous luminosity averaged over all of the bunches colliding during each LumiSection. In order to normalise between LumiSections, we also need the total integrated luminosity for a given LumiSection. Finally, we need some idea of the spread of the individual bunch luminosities so that we can reproduce the tails of the distribution. So, for each LumiSection, we also store the rms of the individual

bunch-to-bunch luminosities.

The production of the pileup JSON file is done by a script called `estimatePileup_makeJSON.py`, also found in `RecoLuminosity/LumiDB`. The command looks like:

```
estimatePileup_makeJSON.py --csvInput lumi_2012a.csv  pileup_JSON.txt
```

The pileup JSON file contains one entry for each LumiSection with four values (LS, integrated luminosity (/ub), rms bunch luminosity (/ub/Xing), average bunch instantaneous luminosity (/ub/Xing)), which makes it a relatively large file. Note that NO assumption is made as to the minbias cross section here, allowing for flexibility later. A sample entry for a random run looks like this:

```
"160577": [[14,1.2182e+01,2.2707e-06,6.7377e-05],[15,3.3884e+01,2.4204e-06,6.7367e-05],...]
```

Run the script as listed above. Have a look at the pileup_JSON.txt file, just to see what is in it. You have just been through the process of making the pileup JSON file that all of the analysts use for pileup reweighting.

(Note: if you get a whole bunch of "Format has changed errors", copy the version of `estimatePileup_makeJSON.py` in `/uscms_data/d2/mikeh/work/HATS/CMSSW_5_3_11_patch4/src` or on `lxplus` in `~mikeh/public` to the `RecoLuminosity/LumiDB/scripts` directory in your working area. You will have to explicitly run that version of the script, so replace `estimatePileup_makeJSON.py` with `./RecoLuminosity/LumiDB/scripts/estimatePileup_makeJSON.py` in the command above. This is what happens when the pileup world doesn't have good communication with the people writing the luminosity scripts.)

Calculating Pileup Distributions

For the user, the important issue is how to access the pileup information you just calculated. All that is required is a file in standard JSON format listing the LumiSections included in a user's analysis (so, we will re-use `Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt`).. Then, a script called `pileupCalc.py` in `RecoLuminosity/LumiDB` can be used to create the histogram of the pileup distribution corresponding exactly to the LumiSections used in the analysis. This is the input needed for the pileup reweighting tools.

First, run this calculation:

```
pileupCalc.py -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt --inputLu
```

The following explanations of the use of this script have been excerpted from the Pileup TWiki:

Begin Excerpt

There are a number of important arguments and options on display here. (All of the arguments can be seen by typing `pileupCalc.py --help`.)

- The only required option is `"--calcMode"` which tells the script which distribution you are making. The two choices are `"true"` and `"observed"`. Some have found this nomenclature confusing, so here is another attempt at an explanation. Given a total inelastic cross section, the average bunch instantaneous luminosity can be directly converted into the expected number of interactions per crossing for this LumiSection. Selecting the `"true"` mode puts this value, and only this value, into the pileup histogram. Hence, in this case the pileup histogram contains the distribution of the mean number of interactions per crossing, which would correspond exactly to the value returned by `PileupSummaryInfo::getTrueNumInteractions()` in the Monte Carlo. Since this is the mean value of the poisson distribution from which the number of interactions in- and out-of-time are generated, no additional information should be required for reweighting if these values are matched in data and

Monte Carlo. On the other hand, selecting the `"observed"` mode causes the script to enter in the histogram a properly-normalized poisson distribution with a mean corresponding to the expected number of interactions per crossing for each LumiSection. Given an expected mean number of interactions, the pileup histogram contains the distribution of the number of interactions one would actually observe given a poisson of that mean. So, this distribution is what one would see if one counted the number of events seen in a given beam crossing (by looking at the number of vertices in data, for example), or using something like `PileupSummaryInfo::getPU_NumInteractions()` in the Monte Carlo. This would be appropriate for pileup reweighting based on in-time-only distributions. Plots of these distributions are shown later in this page.

- The total inelastic cross section is an input argument; the default is set to 73500 ub, but it can and should be modified by the `"--minBiasXsec"` option to set it to the approved value of 68000 (for 2011) or 69300 (for 2012). Note that this command option makes shifting the target distribution for reweighting as simple as regenerating another histogram with a different cross section; all issues with shifting poisson distributions and the like are automatically done correctly. This should make computation of systematic errors much simpler.
- The user also has complete control over the binning of the output histogram. For the `"true"` mode, some users have found that having many bins improves the 3D pileup reweighting technique, so this can be varied. For the `"observed"` mode, since the number of interactions is an integer, it makes sense to have the same number of bins as there are interactions, hence the 50 and 50 in the example above. Smaller bins are allowed in this mode and are properly calculated, however.

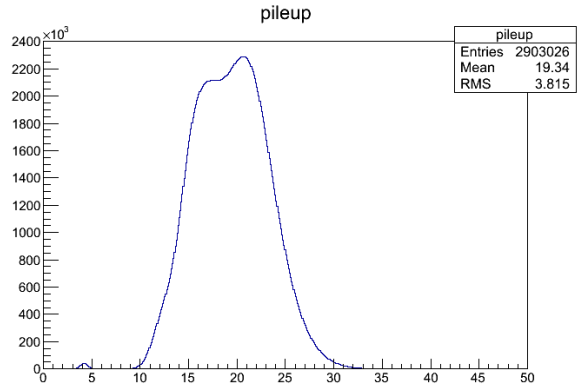
On a more technical note, the rms of the bunch-to-bunch luminosities is used to generate tails on the distributions. Each LumiSection is actually represented by a gaussian centered on the mean number of interactions, and having a width corresponding to the rms of the bunch-by-bunch luminosities, even in `"true"` calculation mode. This is necessary because the distribution in data sees all of the colliding bunches, not just the average. The gaussian is properly convoluted with the poisson distribution for the `"observed"` calculation mode.

End Excerpt

Now, have a look at the pileup plot in `MyDataTruePileupHistogram.root`:

```
root -l MyDataPileupHistogram.root
pileup->Draw();
```

What does it look like? Compare it to this histogram, representing a "correct" calculation:



You should see that the distribution you made has some distortion and portions are shifted to lower luminosity values. (In this particular case the difference is subtle.) This is due to empty bunches which report non-zero luminosity values due to what is called the "afterglow" effect in HF. Basically, light levels in HF do not go instantaneously to zero once it has been hit by all of the interactions in a bunch crossing. So, unless you cut

this afterglow away, you end up with a completely strange estimate of pileup, since the "afterglow" bunches have only a tiny fraction of the luminosity of the colliding bunches. If we had some means of applying a bunch mask based on the collision pattern, we could avoid this altogether, but we don't. It's an interesting effect, actually.

(NOTE: `--xingMinLum 0.3` is required for "normal" 2012 running to remove all of the afterglow effects on non-filled bunches. .)

We can get back to a "normal" distribution by hacking the `estimatePileup_makeJSON.py` script:

```
cd RecoLuminosity/LumiDB/scripts
(use your favorite editor) estimatePileup_makeJSON.py
```

You will find on line 55 and line 67 things that look like

```
xingInstLumi > 0.1
```

Change these to

```
xingInstLumi > 0.3
```

then save. Now try running `estimatePileup_makeJSON.py` and `pileupCalc.py` again with this modified setting, and look at the resulting pileup distribution. This is a "correct" distribution. Save the pileup_JSON.txt you made here and make sure you don't overwrite it messing around with other studies. (The value of 0.3 was chosen by looking at the luminosity values of each of the bunches and putting a cut that got rid of all of the crazy low values. You can do this study yourself by hacking `estimatePileup_makeJSON.py` to histogram the luminosity values for each bunch. There are instructions below if you wish to try this.)

Hacking the script to get at luminosity values: Show Details... Hide Details...

Hacking the script to get at luminosity values

- edit your version of `estimatePileup_makeJSON.py` in the following way: change line 60 to

```
print xingInstLumi
```

with proper indentation. (And, remove the cut on `xingInstLumi` of 0.3 if you haven't already.) Then, run your version of

```
estimatePileup_makeJSON.py --csvInput lumi_2012a.csv  pileup_JSON.txt > lumilist.txt
```

This will make a giant dump of all of the colliding bunches in `lumi_2012a.csv`.

Next,

```
> root -l
root[0]  TTree *t = new TTree("t", "A SimpleTree");
root[1]  t->ReadFile("lumilist.txt","bunchLumi");
root[2]  t->Draw("bunchLumi");
etc.
```

You can easily see the spike at low "afterglow" values of luminosity. This skews the overall distributions if it is included.

One more thing on Pileup Distributions

(Just to be pendantic) Run this calculation:

```
pileupCalc.py -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt --inputLu
```

Compare the pileup distribution found in `MyDataObservedPileupHistogram.root` with the one you made in `MyDataTruePileupHistogram.root`. (To make the comparison easier, you may want to re-run with `--calcMode true` and `--numPileupBins 50`. Running 500 bins of "observed" takes forever because a poisson distribution must be calculated for every bin (and is also unnecessary, since, by definition, "observed" is an integer).)

Note that the means of the two distributions are exactly the same, but, of course, the "observed" distribution is broader. As explained above, the "observed" distribution represents the number of additional interaction vertices that would actually be present in a given bunch crossing given that the "true" distribution contains the mean number of interactions per crossing. (For each Luminosity section, the observed distribution is constructed by calculating the poisson probability in each bin of number of interactions where the mean of the poisson distribution given by the "true" number of interactions for that lumi section.)

Applying External Corrections for Pileup Calculation

As you all know, the HF is sufficiently non-linear and unstable that alternate methods for measuring the luminosity have been developed. One of these is the Pixel Luminosity determination, derived from counting the occupancy in the pixel tracker for each luminosity section. Clearly, this cannot be used for a bunch-to-bunch measurement of luminosity, or pileup. But, it can give a relative correction by comparing the integrated luminosity for HF vs. the Pixel calculation in each lumi section. Since that is the granularity of our information anyway, we can't really hope to do much better than that. The one complication is that there can be luminosity sections that didn't have enough accumulated data to calculate the Pixel luminosity (or there was some hardware problem with Pixels, so that the information is missing). So, care must be taken to correct those LSs that can be corrected while preserving the information from HF for the other LSs. The values in the pileup JSON file are corrected by simply multiplying by the ratio of the Pixel/HF integrated luminosities for the luminosity section. Note that, since this changes the luminosity *scale*, the instantaneous *and* integrated luminosities are both affected.

This mode of operation is very similar to accounting for the trigger prescales when calculating the pileup information. In this second use case, the ratio of the trigger-exposed luminosity over the total integrated luminosity is used, but only to calculate the relative weight of the different luminosity sections if the prescale is changed. Note that the instantaneous luminosity will not change in this case: the events still have the same amount of pileup.

For applying Pixel luminosity corrections, we first need the Pixel-corrected luminosities for our dataset. These can be obtained using

```
pixelLumiCalc.py lumibyls -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.t
```

For comparison, the same query run by using lumiCalc2.py (and therefore using the HF luminosity) can be found linked to this page at HF_lumi.csv. You might want to look at the first few lines to see the (small) differences.

So, to do the translation, you will need a copy of the script `pileupReCalc_Lumis.py` which can either be found in `/uscms_data/d2/mikeh/work/HATS/CMSSW_5_3_11_patch4/src` or on `lxplus` in `~mikeh/public`. This sort of thing only gets done once or twice per year, so the script to do it hasn't been put in `RecoLuminosity` yet. Now that you have the pixel luminosities, you can use the file to make corrections and a new pileup JSON file:

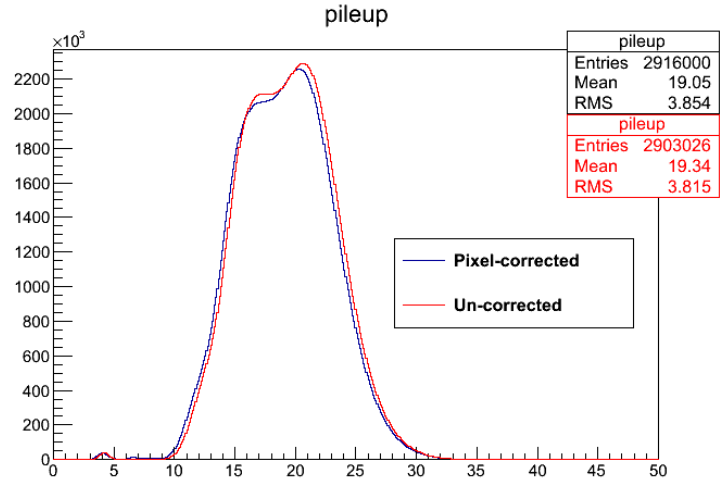
```
./pileupReCalc_Lumis.py -i Pixel_lumi.csv --inputLumiJSON pileup_JSON.txt
```

By default, this script writes its output to a file called `PileupRecalcJSON.txt`. You can change this using the `-o` option.

You can compare the new and old pileup values by doing

```
pileupCalc.py -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt --inputLu
```

Comparison of Pileup with/without Pixel Luminosity Corrections: Show Plot... Hide Plot...



So, the pixel corrections shift the mean value of the instantaneous luminosity down by about 1.5%

Calculating Pileup Distributions for specific HLT Paths

As you may know, you can get the integrated luminosity for a given trigger path (or even ORs) using `pixelLumiCalc.py`. For example:

```
pixelLumiCalc.py lumibyls -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.t
```

should give us the integrated luminosity exposure of this pre-scaled trigger, taking the prescales into account. If the trigger is prescaled by a different amount in different luminosity sections, this will affect the relative amount of data seen by the analysis at different levels of pileup. Therefore, the pileup distribution must be recalculated based on the relative weights of the luminosity sections seen by the trigger, where the weights are calculated using the actual integrated luminosity per lumi section after the prescale is included.

(The output of this command unfortunately isn't really csv-readable. In order to make this work, open the file `PixelHLT_lumi.csv` and change all of the "n/a" to "0.0".)

We can accomplish this using the script `pileupReCalc_HLTpaths.py`, found in `RecLuminosity/LumiDB`.

```
pileupReCalc_HLTpaths.py -i PixelHLT_lumi.csv --inputLumiJSON PileupRecalcJSON.txt -o PileupRecal
```

(Be careful to give an output file, since this might accidentally overwrite the input JSON.)

Now, we can check to see what difference the trigger made to the observed pileup distribution:

```
pileupCalc.py -i Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt --inputLu
```

(Apparently, the answer is: not much!)

Pileup: Conclusion

This has been an exposure to all of the pileup calculating and handling tools used for CMS. Hopefully, this has convinced you that, with appropriate attention to detail, making sense of the pileup spectrum is possible.

-- MichaelHildreth - 06-Aug-2013

- Cert_190456-191202_8TeV_22Jan2013ReReco_Collisions12_JSON_MuonPhys.txt: Golden Muon JSON file for beginning of 2012
- Pileup_Overview_HATS13.pdf: Intro slides
