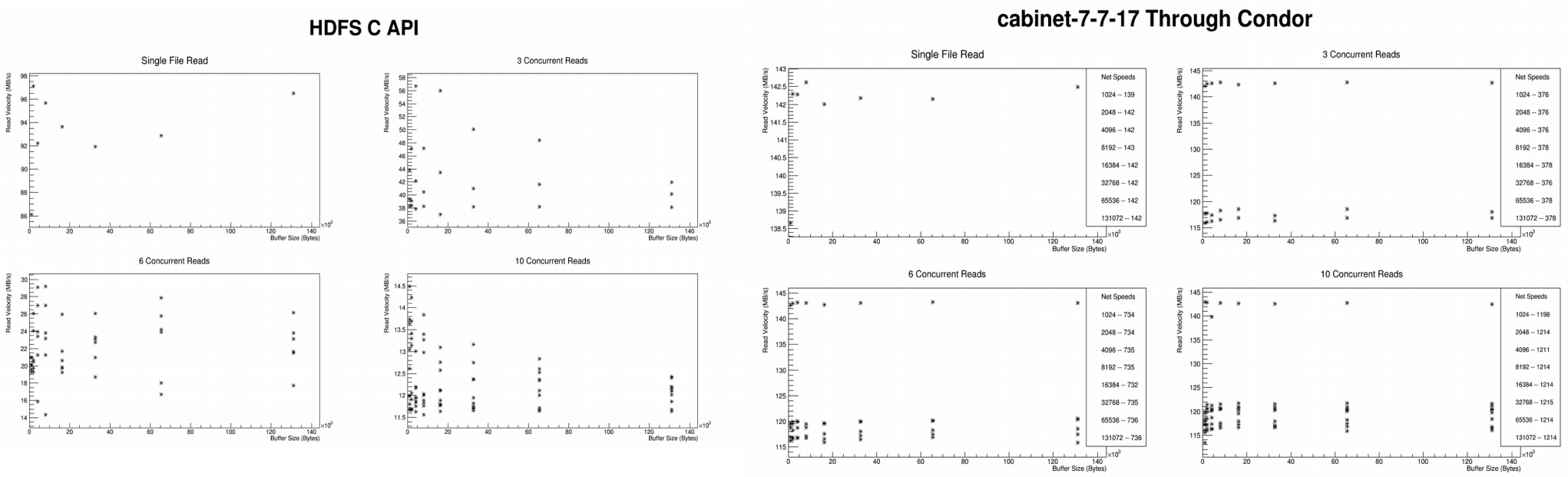
Smartsubmit Benchmarking

Outline

- Rationale
- Test specs
- Timing histograms for Batch Submission and Smartsubmit
- Further testing with the IOTime variable
- Conclusions

Rationale for Smartsubmit

Comparison of Read Speeds of Multiple Files When Files are Stored on HDFS vs Independent Local Disks.



The plots above show the original rationale for smartsubmit. On the right, it is seen that reading 10 local files, in this case on cabinet-7-7-17, could produce a read velocity nearly 10 times faster than reading 10 files through hadoop (left). We extrapolate that if babymaking is an IO bound process, then there should a significant speedup in the time it takes to make babies by reading files from local disks rather than Hadoop. This is the goal of smartsubmit, to make that easy and as transparent as possible to the end users submitting babymaking jobs.

Test Specifications

What Does Smartsubmit Do?

Smartsubmit is a software package for managing the location of miniAOD files outside of the hadoop filesystem. It also allows users to submit jobs through condor that are guaranteed to land on machines hosting the miniAOD files that the user is interested in running over. A typical SS workflow involves loading files into the system, then asking SS to make jobs that run over files tagged with some sample name.

Smartsubmit has been endowed with about 150 condor slots on 150 disks available across 13 machines to store datafiles in the testing period.

Stopbabymaker Testsing

As a first pass, smartsubmit was tested on the stopbabymaker.

The test described on the next page is a comparison of the time taken to run the stopbabymaker on remote cabinets between the normal batch submission and smartsubmit batch submission. In the smartsubmit case, each instance of the stopbabymaker was run over approximately 4 files (varied between 3-4 per machine), in convensional batch submission, each file is run over a single sample file.

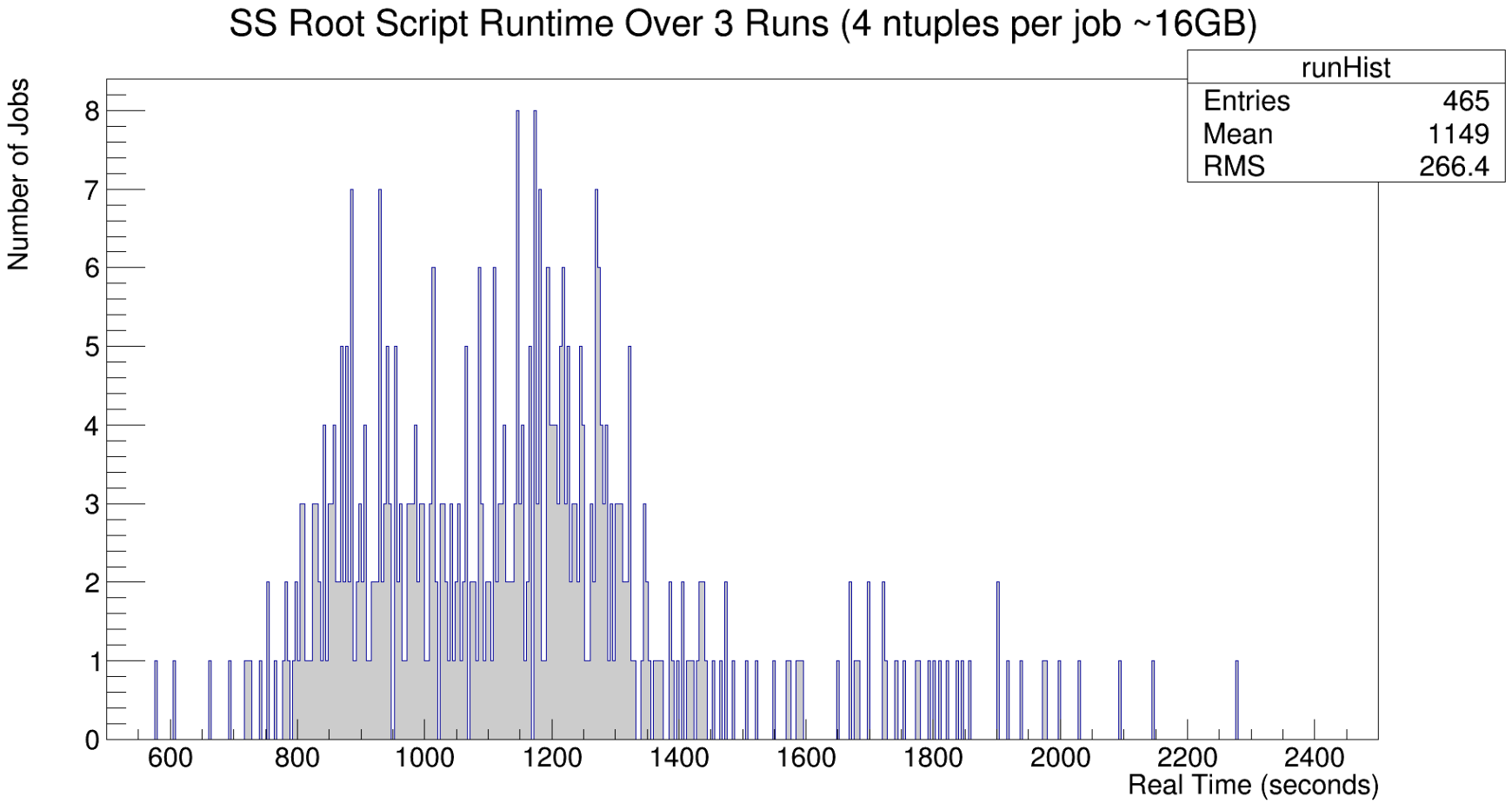
We expect that smartsubmit should allow for the babymaker to process data much faster as there will be no network bottleneck shown on page 3.

Test Specifications

In the first test, we take a look at the time to run the executable ‘runBabyMaker’ on smartsubmit vs. conventional batch submission. This test was run on a modified version of the cmstas/StopAnalysis committed on Nov 23 2015. The version used is here <- (link). Smartsubmit uses the same code, but replaces the submission through the batch/ folder with smartsubmit’s code.

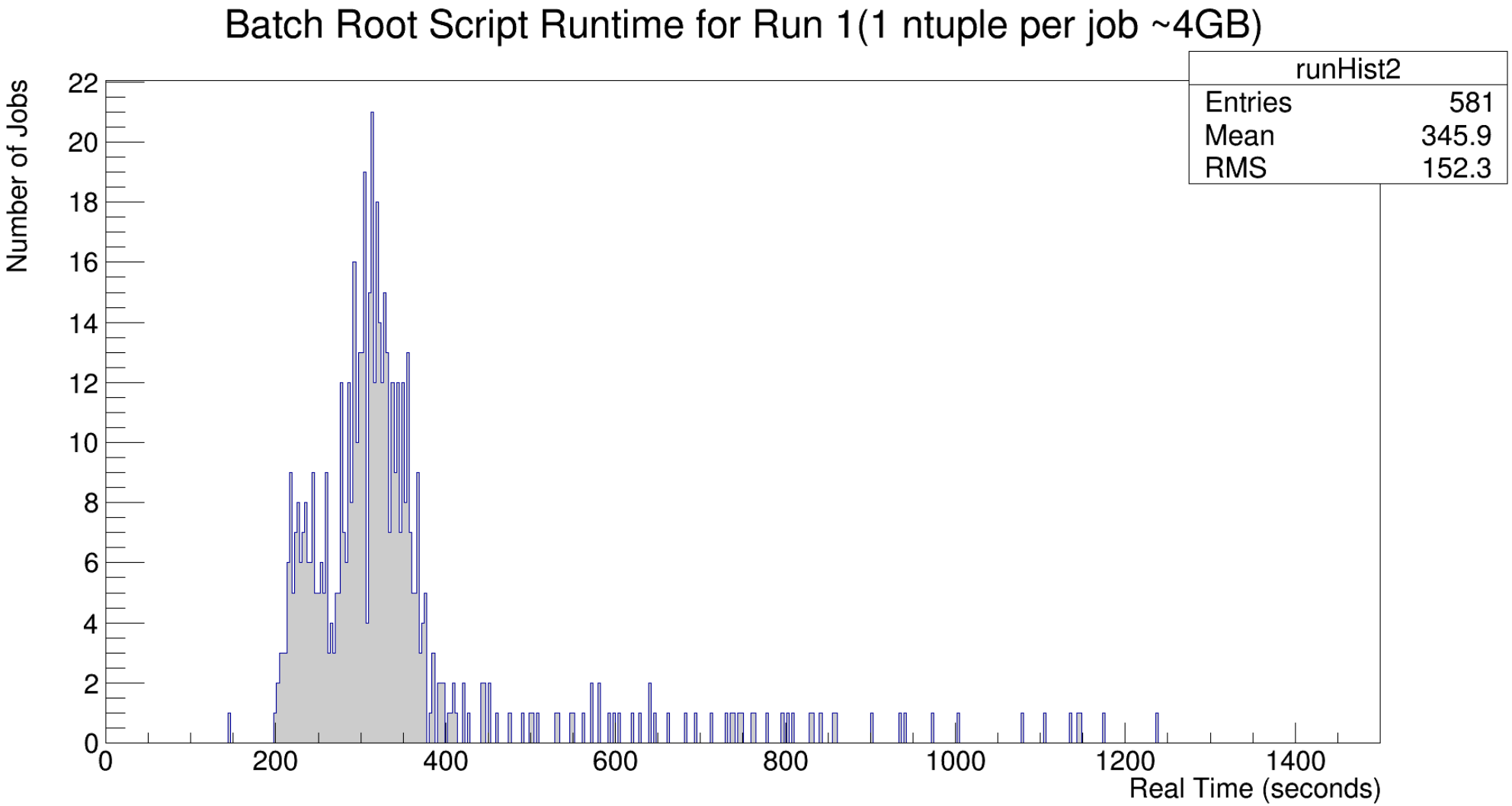
Input Sample	run2_25ns_MiniAODv2/DYJetsToLL_M-50_TuneCUETP8M1_13TeV-madgraphMLM-pythia8_RunIISpring15MiniAODv2-74X_mcRun2_asymptotic_v2-v1/V07-04-11/
Sample Stats	~2.4 TB, ~4GB per file, 581 files
Description	The executable runBabyMaker is run on every file in the sample. runBabyMaker is timed with the ‘time (1)’ command. **For smartsubmit, runBabyMaker is run over 4 files** at a time, whereas for **conventional batch submission, each file gets its own instance**. On the next page, we show the ‘real time’ output of the time command for smartsubmit jobs for 3 runs through this sample. On the page after that we show the output for conventional batch, keep in mind the times for smartsubmit have an implicit factor of 4 compared to conventional batch submission.

Total Dataset for SS Runs



The dataset above comprises of 3 full baby making runs through the sample (notice 600 * ¾ = 450). We have a mean runtime of 1149 seconds, which corresponds to a data processing velocity for smartsubmit of about 14 MB/sec on the cabinets.

Batch Submission



A mean read time of 345.9 seconds corresponds to a processing velocity of about 11.6 MB/sec. These are fairly minimal gains in time, explored more on page 10. On the next slide we'll try and analyze why this is occurring.

IOTiming

The next test was designed to explore whether the small timing gains seen in smartsubmit were due to problems with the testing methodology or a deeper issue with reading root files in general that necessarily slows down read times.

Root attempts to do a sort of smart read when reading in .root files so that it does not access data which will not be needed. In order to find the time it takes to do this ‘smart read’ (what we will call the IOTime) we ran runBabyMaker on 5 files in the sample mentioned on pg. 4 with the following augmentations:

(1) force the babymaker to read in root files completely with LoadAllBranches tag, but also don’t do any computations with data read, read everything and quit.

(2) read in root files completely with LoadAllBranches tag, do normal analysis. Make babies but read everything.

(3) do normal anaylsis, root does its smart read and makes babies.

We define the following quantities:

Comp Time := (2) – (1) ==> Roughly, the time it takes just to do computations on the data.
IOTime := (3) – Comp Time = Roughly, the time it takes just to do the smart read.

IOTiming

The following data is the result of finding the IOTime when reading from the **local disk**. Three trails are shown, but this data has been corroborated in about 50 runs:

Run	(1)	(2)	(3)	Comp Time	IOTime
1	14m50.718s	20m14.380s	13m27.961s	5m23.662s	8m4.299s
2	14m45.349s	20m9.433s	13m35.337s	5m24.084s	8m11.253s
3	14m41.603s	20m31.633s	13m27.057s	5m50.030s	7m37.027s

Read Velocity: ~24 MB/sec

Here we have the output of the same test when reading from **Hadoop**:

Run	(1)	(2)	(3)	Comp Time	IOTime
1	18m26.903s	23m32.256s	16m34.533s	5m5.353s	11m29.180s
2	17m43.819s	23m5.048s	16m25.700s	5m21.229s	11m4.471s
3	18m4.622s	23m33.325s	17m14.866s	5m28.703s	11m46.163s

Read Velocity: ~19 MB/sec

% read speedup = (IOtime hadoop – IOtime local)/(IOtime hadoop) ~ 30%

% total speedup = (net time hadoop – net time local)/(net time hadoop) ~ 20%

IOTiming

Summary: Through remote submission, we found that the rate of data processing for conventional batch submission was about 20% lower than through smartsubmit. On the local tests, results were in line, with a IO speedup of ~30%, but a net speedup of again about 20% .

Using Those Numbers:

Consider 1 TB of data being processed at 11.6 MB/s vs. 14 MB/s (average from cabinets).

The difference in time between these speeds for a 1TB sample file would be (1/11.6 – 1/14) = 4 hours. The entire run 2 MC+data set is approximately 80 TB. So in terms of computational time, smartsubmit could perhaps save us about 4 hour/TB*80TB=320 condor slot hours per entire batch of babies. **In terms of turnaround time, this equates to a time saved using smartsubmit of 320/(N Avg. Slots) hours**. Where the number of average slots available seems to be ~500 from personal experience.

Given the processing speed of 11.6 MB/sec for babies, we can estimate the total computational time needed to process 80TB as (80TB/11.Mb s^-1) ~2000 hours.

This means that with 2000 slots, we'd be able to process all data on the order of 1 hour (assuming low overhead in condor). Since this number of slots is approximately available, within a factor of 5 or so, **smartsubmit would save us about a fraction of an hour out of a couple hours at best.**

e.g. For 500 slots, we'd take 2000/5=4 hours, and SS would save us 320/500 ~ ½ an hour

Conclusions

Smartsubmit can at most save us fractions of an hour on babymaking, but increases complexity for users of the cluster, making it a questionable endevor.

It is more efficient, due to the low memory footprint of single babymaking jobs to simply add more slots to condor and increase parallelization. With 2000 babymaking slots, we could expect the turnaround time for remaking every baby in the 76x dataset to be on the order of 1 hour.

The resident memory usage of 300 MB corresponds to approximately 3 extra baby making slots per extra GB of memory available that can be added to the cluster. More testing will need to be done to confirm this result, but to first approximation, it would be possible to build the needed 2000 slots into 150 machines by setting aside 14 slots per machine, reserving a total of 4GB of RAM from each box.

Another avenue, not yet fully explored, is changing the basket sizes in our miniAOD files. The going theory is that our baskets are very large, making it so that a lot of data needs to be uncompressed for events that pass cuts. This is our best guess as to the mechanism which is causing babymaking to be essentially CPU limited, rather than IO limited. Further testing will need to occur before we have a handle on the tradeoffs between basket size, net file size, and read velocity.