Test Summary - DeepSpeech

Environments & Configurations & Setup

Docker image:

```
harbor.ops.veritone.com/challenges/deepspeech
```

DeepSpeech Version:

```
TensorFlow: v2.3.0-6-g23ad988
DeepSpeech: v0.9.3-0-gf2e9c85
```

Docker Resource setting:

```
CPU: 4
Memory: 2GB
Swap: 1GB
Disk Image Size: 59.6GB
No limits set
DeepSpeech is the only container running in the dockerbox.
```

Sample File:

`audio1.wav, audio2.wav, audio3.wav, audio4.wav, audio5.wav`

Third-party library:

`jiwer2.2.0(`https://pypi.org/project/jiwer/`): Calculate WER between expected and actual result`

SetUp:

1. Pull docker image
2. Start the container with entrypoint /bin/bash to keep the container alive
3. Set CPU gather interval to 1s

TearDown:

1. Stop the container
2. Process test run raw data to csv files

Feature Testing

Test Case 1: Happy Path - process 1 audio - audio got transcripted && if pass rate 80%
Test Result: See is_pass column for individual result

audioName	size	cpu	process_time	expected_accurancy	actual_accurancy	is_pass	is_processed
audio1.wav	1556696	117.8066667	7.084916115	0.8	0.555555556	FALSE	TRUE
audio2.wav	11917312	122.87	48.9436748	0.8	1	TRUE	TRUE
audio3.wav	8944318	125.5472	75.57108402	0.8	0.993788819	TRUE	TRUE

audio4.wav	8454138	123.8316667	25.42660928	0.8	0.619047619	FALSE	TRUE
audio5.wav	10532390	126.1106667	30.88063693	0.8	0.105263158	FALSE	TRUE

Test Case 2: Process 5 audios sequentially - Compare on corresponding audio result from test case 1 - No significant change

Test Result: Pass

audioName	size	cpu	process_time	expected_accuracy	actual_accuracy	is_pass	is_processed
audio1.wav	1556696	123.765	7.923945189	0.8	0.5555555556	FALSE	TRUE
audio2.wav	11917312	125.4379167	48.44889593	0.8	1	TRUE	TRUE
audio3.wav	8944318	125.5472	75.57108402	0.8	0.993788819	TRUE	TRUE
audio4.wav	8454138	123.7818519	26.38932109	0.8	0.619047619	FALSE	TRUE
audio5.wav	10532390	125.4414286	36.93647909	0.8	0.105263158	FALSE	TRUE

average cpu	average process_time	average accuracy
125.18	39.05394506	0.6547310305

Test Case 3: Process 5 audios parallelly - Compare on corresponding audio result from test case 1 - No change

Test Result: Pass for accuracy rate, Fail for process time. Processing time increases 123%

audioName	size	cpu	process_time	expected_accuracy	actual_accuracy	is_pass	is_processed
audio1.wav	1556696	403.0768182	21.99604607	0.8	0.5555555556	FALSE	TRUE
audio2.wav	11917312	346.9966957	115.5190122	0.8	1	TRUE	TRUE
audio3.wav	8944318	303.1638462	144.6561861	0.8	0.993788819	TRUE	TRUE
audio4.wav	8454138	398.697857	70.3710160	0.8	0.61904761	FALSE	TRUE

		1	3		9		
audio5.wav	10532390	392.7270238	84.82768893	0.8	0.105263158	FALSE	TRUE

average cpu	average process_time	average accuracy
303.1638462	87.47398987	0.6547310305

Negative Testing

Test Case 1: Audio file cannot be found - Correct error/warnings displayed (manual test case)
Test Result: PASS
Explicte error displays:

```
FileNotFoundError: [Errno 2] No such file or directory: 'audio7.wav'
```

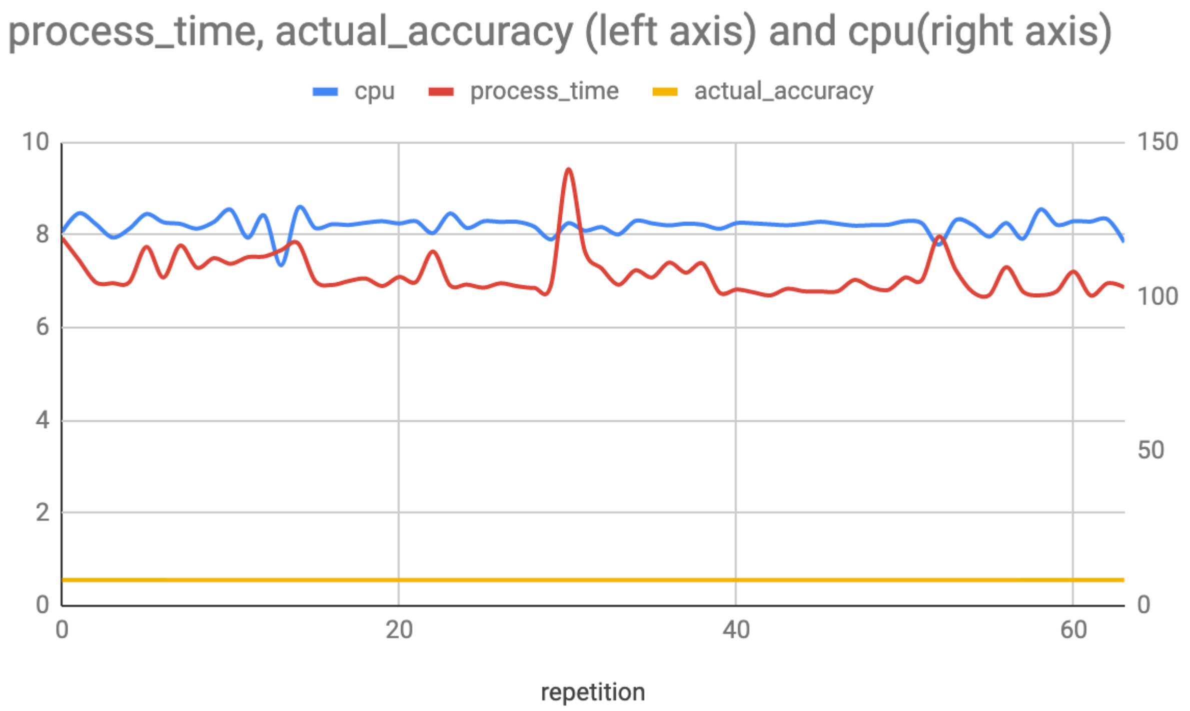
Test Case 2: Audio file isn’t parsed in - Correct error/warnings displayed (manual test case)
Test Result: PASS
Explicite error displays:

```
deepspeech: error: the following arguments are required: --audio
```

Performance Testing

Test Case 1: Load - Run one audio file repeatedly for multiple times - Processing time, cpu utilization and accuracy rate are keeping consistent
Limit to the laptop capability, set the replication number to 64
Test Result: **PASS**

average cpu	average process_time	average accuracy
123.1784802	7.144031584	0.5555555556



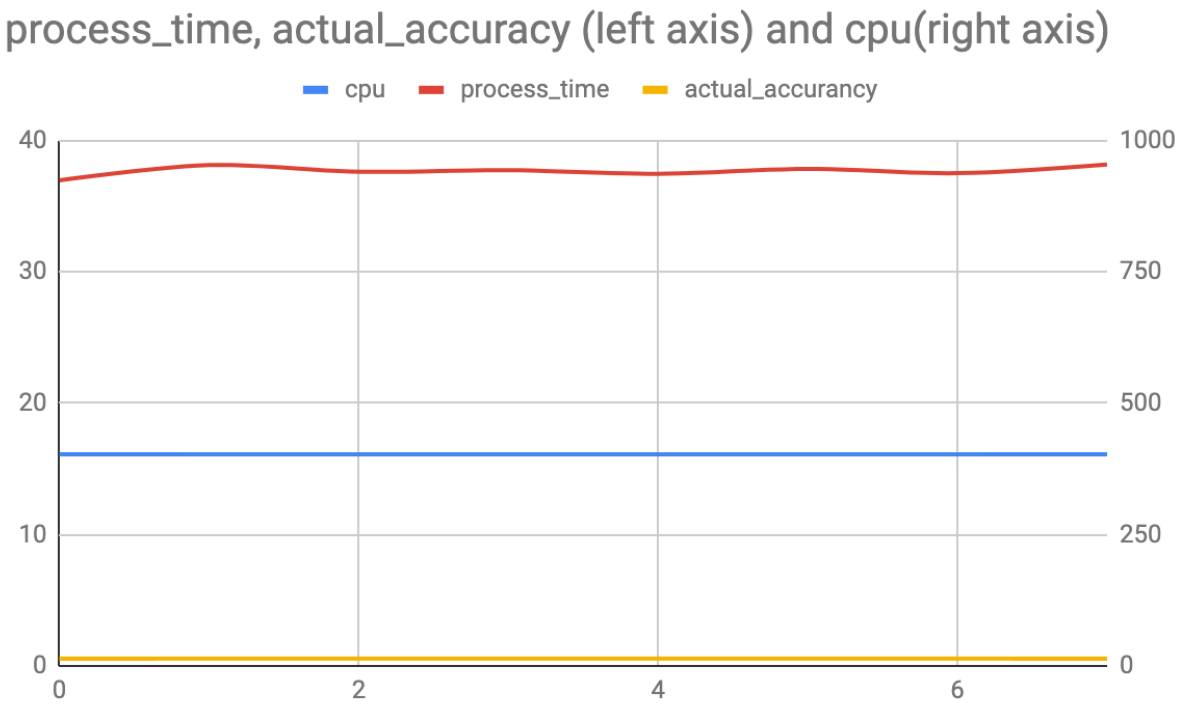
Test Case 2: Load - Start multiple threads upload the same audio parallelly

Limit to the laptop capability, set the threads number to 8.

Test Result:

1. Accuracy rate keeps consistent.
2. The average cpu utilization and processing time has been significantly increased by ~300% and ~400%

average cpu	average process_time	average accuracy
402.6327027	37.67524719	0.5555555556



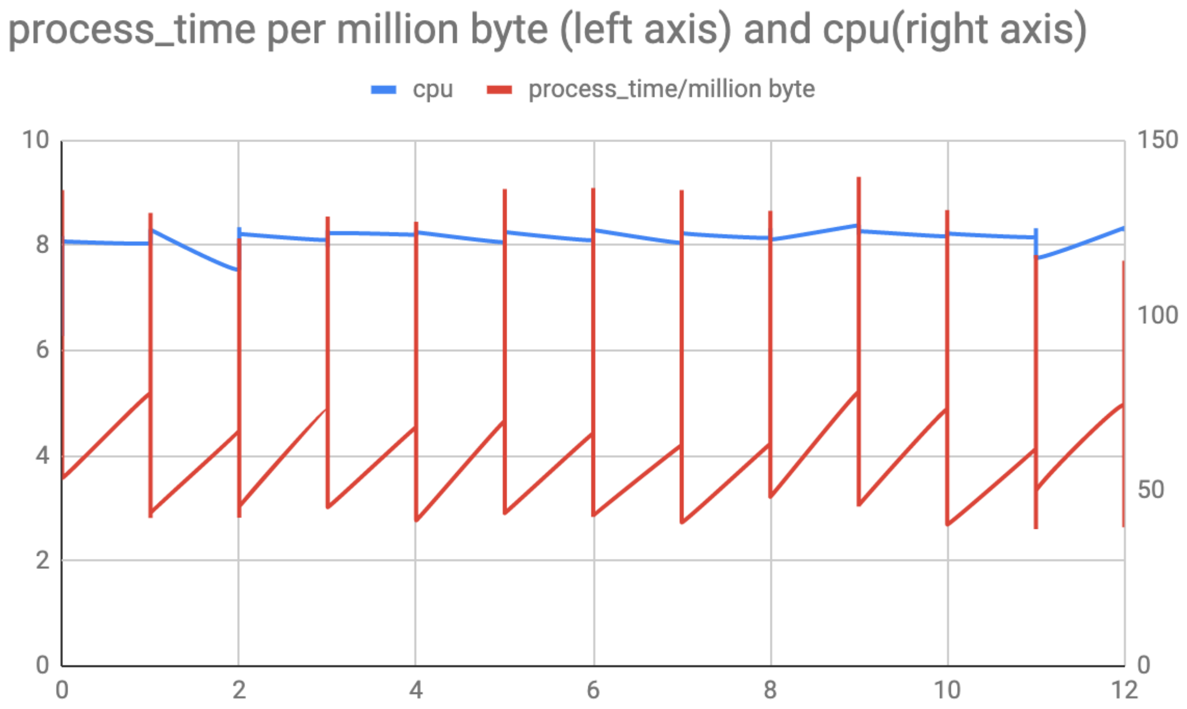
Test Case 3: Load - Run 5 audios repeatedly for multiple times

Limit to the laptop capability, set the replication number to 64/5 ~ 13

Test Result:

1. Accuracy rate keeps consistent. Processing time keeps consistent per file.
2. From processing time/million byte, we can find the processing time relevant to file a lot (Patten over repetition keeps the same).
3. CPU utilization hasn't changed significantly.
4. No failure process

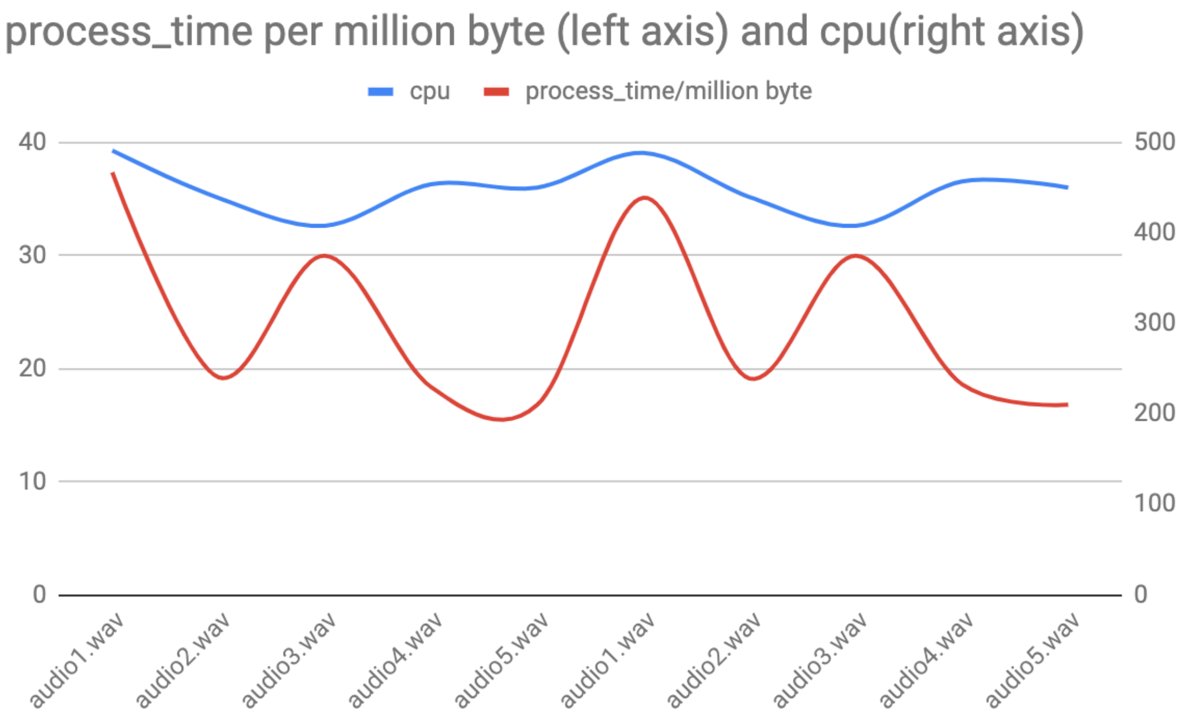
average cpu	average process_time	average accuracy
123.4266801	38.57782697	0.6547310305



Test Case 4: Load - Start multiple threads upload different audios parallelly
Limit to the laptop capability, set the threads number to 8/5~2.
Test Result:

1. Accuracy rate keeps consistent with original rate by individual run
2. Average processing time significantly increased than run videos individually/sequentially
3. CPU utilization rate significantly increased
4. No failure process

average cpu	average process_time	average accuracy
407.8162642	177.2537795	0.6547310305



Future work

1. Add memory usage or other values to estimate in performance testing.
2. Performance Load & Stress testing:
   a. Add data variance to the indices and largest allowance variance threshold.
   b. Stress test: keep increasing the number of thread processing files, when the cpu utilization > threshold, the number of total threads get saved.
   c. Stress test: keep increasing the number of thread processing files, when the accuracy rate < threshold, the number of total threads get saved.
   d. Random sends files with a random number of threads, gathering performance.
3. If accuracy rate drops by time for the same file, need a test case to measure when it may happen.
4. Spiking testing: Given a base line of file processing, suddenly increase the files sent to a large scale, measure all the indices change like cpu utilization increase rate, wer drop rate etc.
5. Volume Testing: Send files to process, increase the file size by binary.
6. Scalability test: Perform testing with scalability settings
7. Network testing: Perform testing with proxy or other network settings.
8. Test cases around audio do not exist in the server, but have to be uploaded from local first. (mimic use case)