SageMaker Debugger Profiling Report

SageMaker Debugger auto generated this report. You can generate similar reports on all supported training jobs. The report provides summary of training job, system resource usage statistics, framework metrics, rules summary, and detailed analysis from each rule. The graphs and tables are interactive.

Legal disclaimer: This report and any recommendations are provided for informational purposes only and are not definitive. You are responsible for making your own independent assessment of the information.

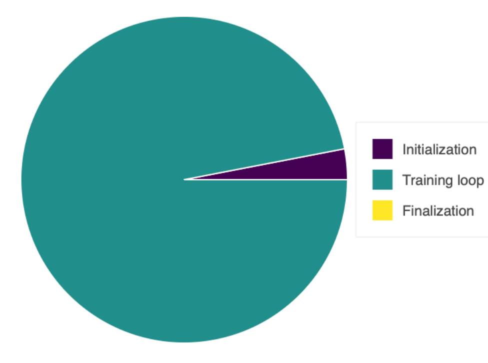
In [4]:

```python
# Parameters
processing_job_arn = "arn:aws:sagemaker:us-east-1:393972968491:processing-job/pytorch-training-2023-04-0-profilerrepor
t-9411d2c2"
```

Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 04/09/2023 at 04:35:24 and ran for 2495 seconds.

#		Job Statistics
0	Start time	04:35:24 04/09/2023
1	End time	05:16:59 04/09/2023
2	Job duration	2495 seconds
3	Training loop start	04:36:41 04/09/2023
4	Training loop end	05:16:58 04/09/2023
5	Training loop duration	2416 seconds
6	Initialization time	77 seconds
7	Finalization time	1 seconds
8	Initialization	3 %
9	Training loop	96 %
10	Finalization	0 %



System usage statistics

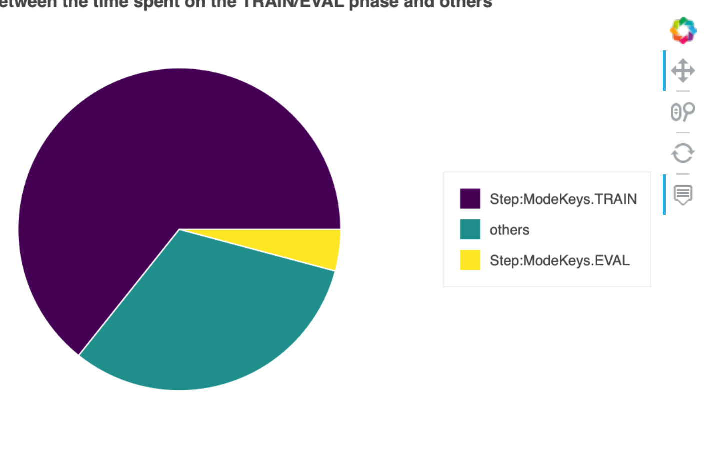
The following table shows statistics of resource utilization per worker (node), such as the total CPU and GPU utilization, and the memory utilization on CPU and GPU. The table also includes the total I/O wait time and the total amount of data sent or received in bytes. The table shows min and max values as well as p99, p90 and p50 percentiles.

#	node	metric	unit	max	p99	p95	p50	min
0	algo-1	Network	bytes	89753157.27	0	0	0	0
1	algo-1	CPU	percentage	77.89	63.59	54.27	49.83	0
2	algo-1	CPU memory	percentage	39.61	26.31	25.53	21.6	1.74
3	algo-1	I/O	percentage	25.57	8.73	0.25	0	0

Framework metrics summary

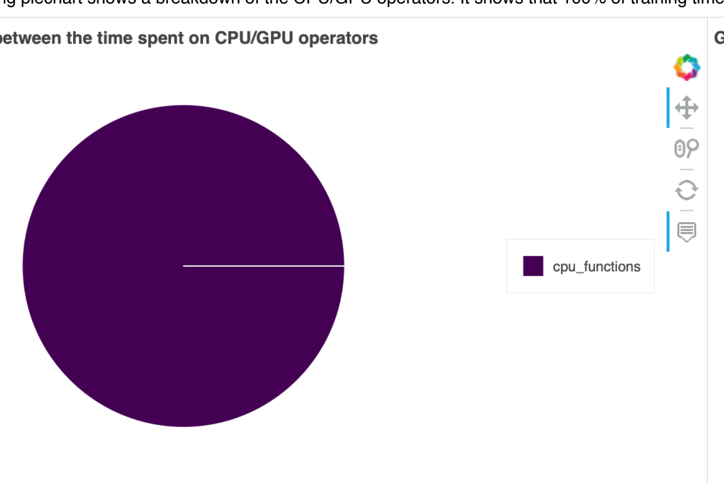
The following two pie charts show the time spent on the TRAIN phase, the EVAL phase, and others. The 'others' includes the time spent between steps (after one step has f the next step has started). Ideally, most of the training time should be spent on the TRAIN and EVAL phases. If TRAIN/EVAL were not specified in the training script, steps w GLOBAL. Your training job spent quite a significant amount of time (31.56%) in phase "others". You should check what is happening in between the steps.

The ratio between the time spent on the TRAIN/EVAL phase and others



The following piechart shows a breakdown of the CPU/GPU operators. It shows that 100% of training time was spent on executing the "cpu_functions" operator.

The ratio between the time spent on CPU/GPU operators

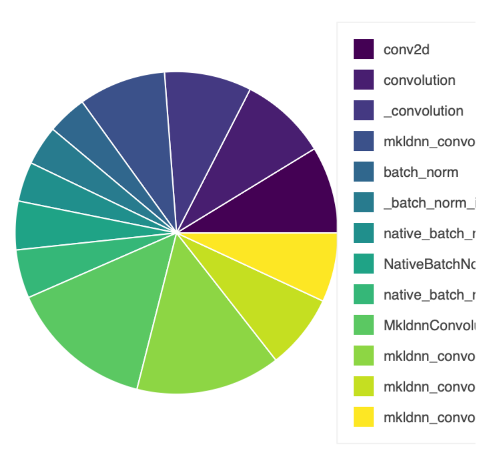


General framework operations

Overview: CPU operators

The following table shows a list of operators that ran on the CPUs. The most expensive operator on the CPUs was "MkldnnConvolutionBackward" with 14 %.

#	Percentage	Cumulative time in microseconds	CPU operator
0	14.49	35294817	MkldnnConvolutionBackward
1	14.49	35287933	mkldnn_convolution_backward
2	8.73	21271957	conv2d
3	8.73	21269758	convolution
4	8.73	21267869	_convolution
5	8.73	21257826	mkldnn_convolution
6	7.52	18314822	mkldnn_convolution_backward_in
7	6.94	16890968	mkldnn_convolution_backward_w
8	4.87	11858858	NativeBatchNormBackward
9	4.87	11847931	native_batch_norm_backward
10	3.97	9659031	batch_norm
11	3.97	9656603	_batch_norm_impl_index
12	3.96	9652706	native_batch_norm



Rules summary

The following table shows a profiling summary of the Debugger built-in rules. The table is sorted by the rules that triggered the most frequently. During your training job, the StepOutlier rule was the most frequently triggered. It processed 259 datapoints and was triggered 13 times.

	Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
StepOutlier	Detects outliers in step duration. The step duration for forward and backward pass should be roughly the same throughout the training. If there are significant outliers, it may indicate a system stall or bottleneck issues.	Check if there are any bottlenecks (CPU, I/O) correlated to the step outliers.	13	259	threshold:3 mode:None n_outliers:10 stddev:3
LoadBalancing	Detects workload balancing issues across GPUs. Workload imbalance can occur in training jobs with data parallelism. The gradients are accumulated on a primary GPU, and this GPU might be overused with regard to other GPUs, resulting in reducing the efficiency of data parallelization.	Choose a different distributed training strategy or a different distributed training framework.	0	0	threshold:0.2 patience:1000
LowGPUUtilization	Checks if the GPU utilization is low or fluctuating. This can happen due to bottlenecks, blocking calls for synchronizations, or a small batch size.	Check if there are bottlenecks, minimize blocking calls, change distributed training strategy, or increase the batch size.	0	0	threshold_p95:70 threshold_p5:10 window:500 patience:1000
CPUBottleneck	Checks if the CPU utilization is high and the GPU utilization is low. It might indicate CPU bottlenecks, where the GPUs are waiting for data to arrive from the CPUs. The rule evaluates the CPU and GPU utilization rates, and triggers the issue if the time spent on the CPU bottlenecks exceeds a threshold percent of the total training time. The default threshold is 50 percent.	Consider increasing the number of data loaders or applying data pre-fetching.	0	4992	threshold:50 cpu_threshold:90 gpu_threshold:10 patience:1000
GPUMemoryIncrease	Measures the average GPU memory footprint and triggers if there is a large increase.	Choose a larger instance type with more memory if footprint is close to maximum available memory.	0	0	increase:5 patience:1000 window:10
IOBottleneck	Checks if the data I/O wait time is high and the GPU utilization is low. It might indicate IO bottlenecks where GPU is waiting for data to arrive from storage. The rule evaluates the I/O and GPU utilization rates and triggers the issue if the time spent on the IO bottlenecks exceeds a threshold percent of the total training time. The default threshold is 50 percent.	Pre-fetch data or choose different file formats, such as binary formats that improve I/O performance.	0	4992	threshold:50 io_threshold:50 gpu_threshold:10 patience:1000
BatchSize	Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	0	4991	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:1000 window:500
MaxInitializationTime	Checks if the time spent on initialization exceeds a threshold percent of the total training time. The rule waits until the first step of training loop starts. The initialization can take longer if downloading the entire dataset from Amazon S3 in File mode. The default threshold is 20 minutes.	Initialization takes too long. If using File mode, consider switching to Pipe mode in case you are using TensorFlow framework.	0	259	threshold:20
Dataloader	Checks how many data loaders are running in parallel and whether the total number is equal the number of available CPU cores. The rule triggers if number is much smaller or larger than the number of available cores. If too small, it might lead to low GPU utilization. If too large, it might impact other compute intensive operations on CPU.	Change the number of data loader processes.	0	10	min_threshold:70 max_threshold:200

Analyzing the training loop

Step duration analysis

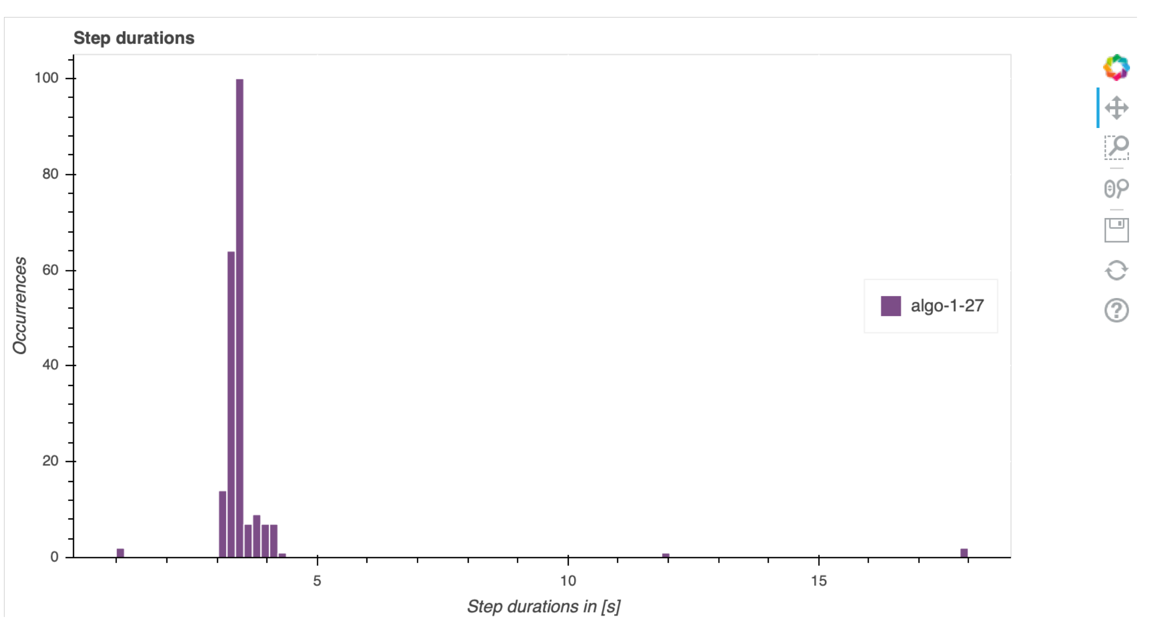
The StepOutlier rule measures step durations and checks for outliers. The rule returns True if duration is larger than 3 times the standard deviation. The rule also takes the parameter mode, that specifies whether steps from training or validation phase should be checked. In your processing job mode was specified as None. Typically the first step is taking significantly more time and to avoid the rule triggering immediately, one can use n_outliers to specify the number of outliers to ignore. n_outliers was set to 10. The rule analysed 259 datapoints and triggered 13 times.

Step durations on node algo-1-27:

The following table is a summary of the statistics of step durations measured on node algo-1-27. The rule has analyzed the step duration from Step:ModeKeys.TRAIN phase. The average step duration on node algo-1-27 was 5.3s. The rule detected 14 outliers, where step duration was larger than 3 times the standard deviation of 6.85s

	mean	max	p99	p95	p50	min
Step Durations in [s]	5.30	33.27	31.54	30.97	3.40	0.99

The following histogram shows the step durations measured on the different nodes. You can turn on or turn off the visualization of histograms by selecting or unselecting the labels in the legend.



GPU utilization analysis

Usage per GPU

The LowGPUUtilization rule checks for a low and fluctuating GPU usage. If the GPU usage is consistently low, it might be caused by bottlenecks or a small batch size. If usage is heavily fluctuating, it can be due to bottlenecks or blocking calls. The rule computed the 95th and 5th percentile of GPU utilization on 500 continuous datapoints and found 0 cases where p95 was above 70% and p5 was below 10%. If p95 is high and p5 is low, it might indicate that the GPU usage is highly fluctuating. If both values are very low, it would mean that the machine is underutilized. During initialization, the GPU usage is likely zero, so the rule skipped the first 1000 data points. The rule analysed 0 datapoints and triggered 0 times.

Workload balancing

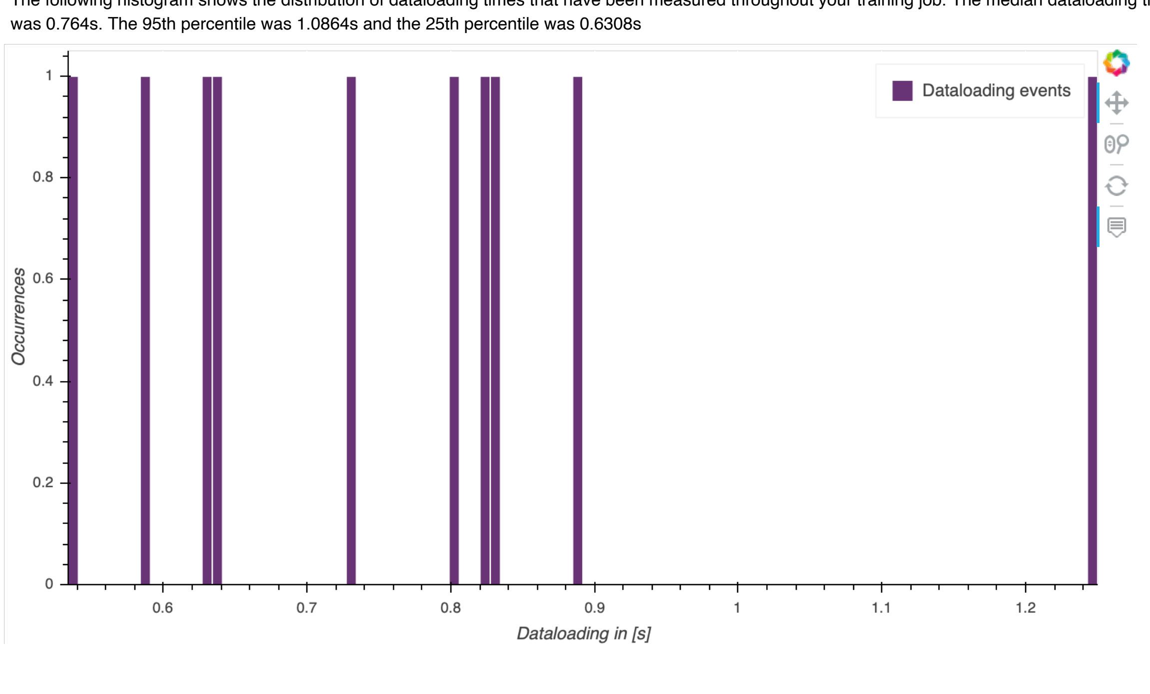
The LoadBalancing rule helps to detect issues in workload balancing between multiple GPUs. It computes a histogram of GPU utilization values for each GPU and compares then the similarity between histograms. The rule checked if the distance of histograms is larger than the threshold of 0.2. During initialization utilization is likely zero, so the rule skipped the first 1000 data points.

Dataloading analysis

The number of dataloader workers can greatly affect the overall performance of your training job. The rule analyzed the number of dataloading processes that have been running in parallel on the training instance and compares it against the total number of cores. The rule checked if the number of processes is smaller than 70% or larger than 200% the total number of cores. Having too few dataloader workers can slowdown data preprocessing and lead to GPU underutilization. Having too many dataloader workers may hurt the overall performance if you are running other compute intensive tasks on the CPU. The rule analysed 10 datapoints and triggered 0 times.

Your training instance provided 8 CPU cores, however your training job only ran on average 1 dataloader workers in parallel. We recommend you to increase the number of dataloader workers. Using pinned memory also improves performance because it enables fast data transfer to CUDA-enabled GPUs. The rule detected that your training job was not using pinned memory. In case of using PyTorch Dataloader, you can enable this by setting pin_memory=True.

The following histogram shows the distribution of dataloading times that have been measured throughout your training job. The median dataloading time was 0.764s. The 95th percentile was 1.0864s and the 25th percentile was 0.6308s



Batch size

The BatchSize rule helps to detect if GPU is underutilized because of the batch size being too small. To detect this the rule analyzes the GPU memory footprint, CPU and GPU utilization. The rule checked if the 95th percentile of CPU utilization is below cpu_threshold_p95 of 70%, the 95th percentile of GPU utilization is below gpu_threshold_p95 of 70% and the 95th percentile of memory footprint below gpu_memory_threshold_p95 of 70%. In your training job this happened 0 times. The rule skipped the first 1000 datapoints. The rule computed the percentiles over window size of 500 continuous datapoints. The rule analysed 4991 datapoints and triggered 0 times.

CPU bottlenecks

The CPUBottleneck rule checked when the CPU utilization was above cpu_threshold of 90% and GPU utilization was below gpu_threshold of 10%. During initialization utilization is likely to be zero, so the rule skipped the first 1000 datapoints. With this configuration the rule found 0 CPU bottlenecks which is 0% of the total time. This is below the threshold of 50% The rule analysed 4992 data points and triggered 0 times.

I/O bottlenecks

The IOBottleneck rule checked when I/O wait time was above io_threshold of 50% and GPU utilization was below gpu_threshold of 10. During initialization utilization is likely to be zero, so the rule skipped the first 1000 datapoints. With this configuration the rule found 0 I/O bottlenecks which is 0% of the total time. This is below the threshold of 50%. The rule analysed 4992 datapoints and triggered 0 times.

GPU memory

The GPUMemoryIncrease rule helps to detect large increase in memory usage on GPUs. The rule checked if the moving average of memory increased by more than 5.0%. So if the moving average increased for instance from 10% to 16.0%, the rule would have triggered. During initialization utilization is likely 0, so the rule skipped the first 1000 datapoints. The moving average was computed on a window size of 10 continuous datapoints. The rule detected 0 violations where the moving average between previous and current time window increased by more than 5.0%. The rule analysed 0 datapoints and triggered 0 times.