G-Storm: GPU-Aware Scheduling in Storm

Cheng-Hao Huang, Yi-Ren Chen, Che-Rung Lee, National Tsing Hua University, Taiwan

Introduction

Storm is a popular solution for cloud stream processing [1]. A Strom cluster contains one master node and multiple slave/worker nodes. Storm starts up different management daemon: *Nimbus* and *Supervisor* on the master node and worker nodes, then uses Zookeeper to coordinate cluster.

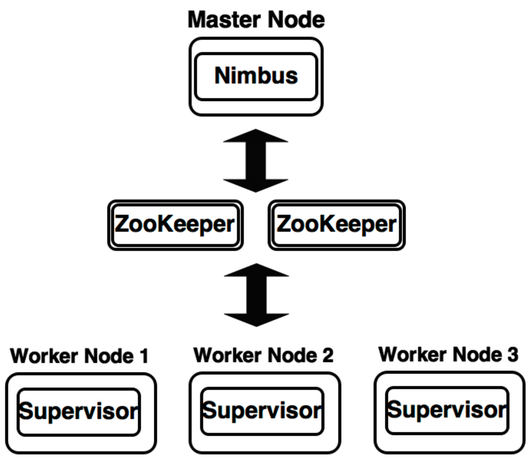


Figure 1, Storm cluster

Storm uses DAG to describe an application with alias *topology* in Storm, like Figure 2. A node in topology can be Spout or Bolt, acting as stream source or stream processor individually. Every edge in topology defines the way how stream flow between Spout/Bolt.

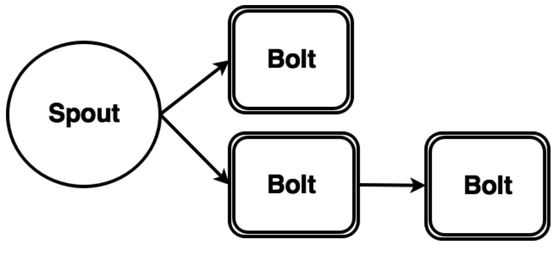


Figure 2, Storm topology

Storm uses *round robin* policy to assign topology on worker nodes, which makes sure balanced loading on each worker node in homogenous environments. However it does not perform well in heterogeneous environments.

We wish to give a better scheduling for heterogeneous cluster like some worker nodes with GPGPU, as shown in Figure 3.

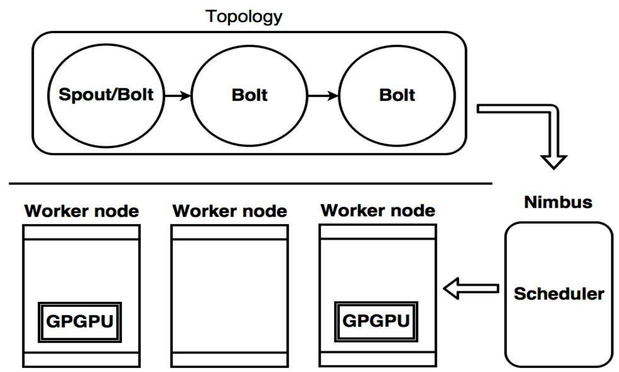


Figure 3, Scenario

Methodology

To register GPGPU on every worker node, we add Metadata for every Supervisor configuration:

Device1:	K20	NVIDIA K20
Device2:	none	no GPU
Device3:	K20	NVIDIA K20

To specify the topology components that can use GPGPU to speed up computation, we add additional *Tag* to topology component:

GPU_Bolt1	bolt can use GPU to speed up
Bolt1	normal bolt

Algorithm – G-Storm

G-Storm algorithm filters out topology components set E_g that can use GPGPU and worker nodes with GPGPU.

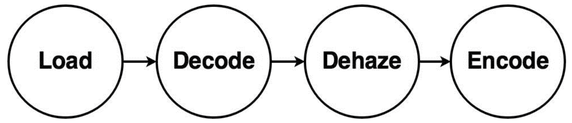
G-Storm evaluates capacity of GPU by define computation factor ***GPower*** for each GPU.

We can assign topologies by the following procedure:

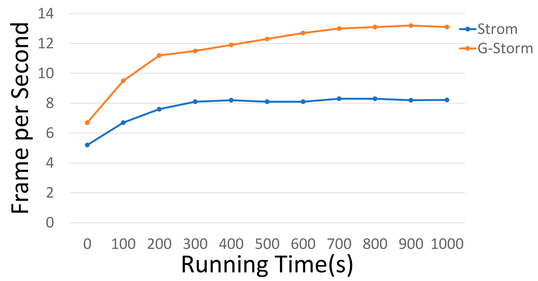
1. assign every element of E_g to the worker node with max value of GPower + available worker process, update GPower value of the worker node.
2. assign normal topology component by round-robin policy.

Evaluation

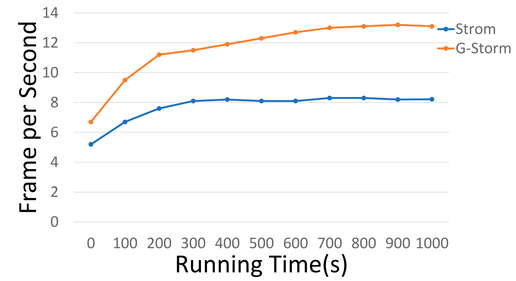
We implemented a benchmark using image dehazing method [2] like Figure 4(a) and Figure 4(b), and choses video with 1280*720 and 1920*1080 resolution to evaluate lightweight and heavyweight performance. The results are shown in Figure 4(a) and Figure 4(b).



Load reads in period of video, then send them to ***Decode.***
Decode uses *FFmpeg* API decoding video to raw data.
Dehaze removes image haze with/without GPGPU.
Encode encodes raw data, and then stores it to local disk.



(a), 1280*720 resolution video



(b), 1920*1080 resolution video

Figure 4, Benchmark

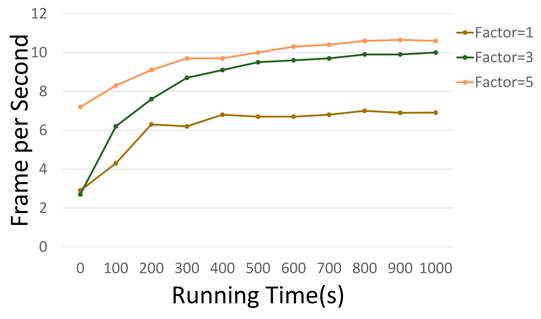


Figure 5, GPower Factor influence

To measure the influence of GPower evaluation , we gave different GPower values and used FullHD videos.

Figure 5 shows Factor 1 underrates the capacity of GPGPU while Factor 5 overrates the capacity. Hence the performance is lower than Factor 3 that moderately gives loading to worker node with GPGPU.

Conclusion

With this work , we provided a scheduling algorithm for Storm to effectively use GPGPU in the stream processing. It achieves 2x improvement compared to Storm.

Reference

[1] Apache storm. https://storm.apache.org/.
[2] Kaiming He, J. S. (2010). Single Image Haze Removal Using Dark Channel Prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (pp. 2341 - 2353).