Twitter Heron

SHAHIDHYA RAMACHANDRAN[1]

[1] *School of Informatics and Computing, Bloomington, IN 47408, U.S.A.*
[*] *Corresponding authors: shahrama@iu.edu*

Technology paper 1, February 27, 2017

This paper summarizes the usage, architecture, features, performance and use cases of Twitter Heron. Comparison between the capabilities of Twitter Heron and its predecessor Twitter Storm has also been mentioned. Other streaming technologies which provide varied capabilities have been listed.

Keywords: Twitter Heron, Stream data, Real-time data Processing

https://github.com/shah0112/sp17-i524/paper1/S17-IR-2027/report.pdf

1. INTRODUCTION

With the increase in the volume and diversity of data being generated in Twitter, the need to process these huge amounts of data in real-time along with an uprise in the number of use cases for real-time analytics, led to the necessity of having a real-time streaming system in place that could process large amounts of data per minute; have sub-second latency and predictable behavior at scale; have high data accuracy in case of failure, ability to handle temporary traffic spikes and pipeline congestions; be easy to debug; and simple to deploy in a shared infrastructure as mentioned in [1].

Twitter initially introduced 'Storm' which served as the real-time data processing system and was later replaced with Heron that had enhanced capabilities and was architecturally more advanced than Storm. Since extension of Storm would require redesigning and switching to a new platform may lead to prolonged migration periods,Twitter decided to reuse existing components of Storm and made Heron fully backward compatible with Storm. Twitter Heron is a real-time analytics platform that was developed at Twitter for distributed streaming processing [2]. It was introduced at 'SIGMOD 2015' to overcome the shortcomings of Twitter Storm.Heron was open sourced under the permissive Apache v2.0 license.

2. ARCHITECTURE

Heron's architecture is a process-based system instead of the thread-based system used in Storm. It is implemented using standard languages like Java, C++ and Python for efficiency, maintainability, and easier community adoption since it is open sourced. Heron allows for deployment in clusters by integrating with open source schedulers like Apache Mesos, Apache Aurora, Apache REEF, Slurm.

2.1. Data Model

The data model and API for Heron and Storm are same in order to maintain the backward compatibility. Heron contains topologies which is a directed acyclic graph of spouts and bolts. Spouts generate the tuples that are given as input to the topology. Bolts are used for the computation. In the Heron topology the number of tasks for every spout and bolt is specified. This decides the the degree of parallelism and grouping of the data as partitions when it moves through the spout and the bolt tasks. Heron's tuple processing semantics are similar to that of Storm: At most once – A tuple is not processed more than once. Some tuples may not be processes even once by the topology.At least once – Every tuple is processed at least once. Some tuples may be processed more than once.[3]

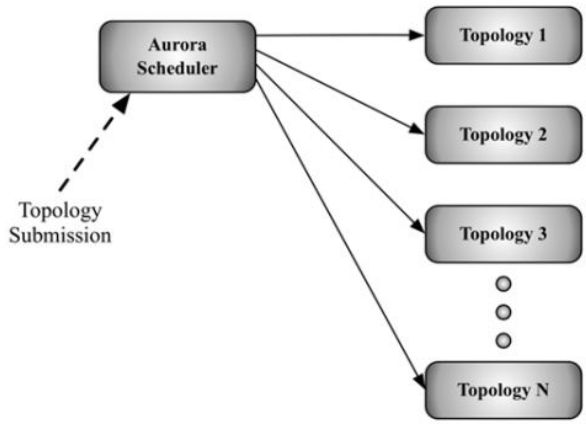

Fig. 1. Heron Architecture

2.2. Overview

The architecture for Heron is shown in Figure 1. Heron API(spouts/bolts programming) is used to create and deploy topologies to the Aurora scheduler through a command line specific to Heron. As shown in Figure 2, each topology consists of several containers, Topology Master is run by the first container,each of the other containers run Stream Manager, Metrics Manager and Heron Instances. Instances are the spouts/bolts in which the user code is run. Multiple containers can be launched on a single node. Based on the resource availability these containers are allocated and scheduled by Aurora. The metadata for the topology which includes user information(who ever launched the job), time of launch, and the execution details are kept in Zookeeper. Protocol buffers are used for communication within the different processes.

2.3. Topology Master

As the name suggests the topology master is responsible for managing the entire topology and provides a consistent view of the topology throughout the process. A given topology can have only one TM and this is ensured by creating an ephemeral key in the Zoo keeper that can be easily identified by all processes run in that topology.

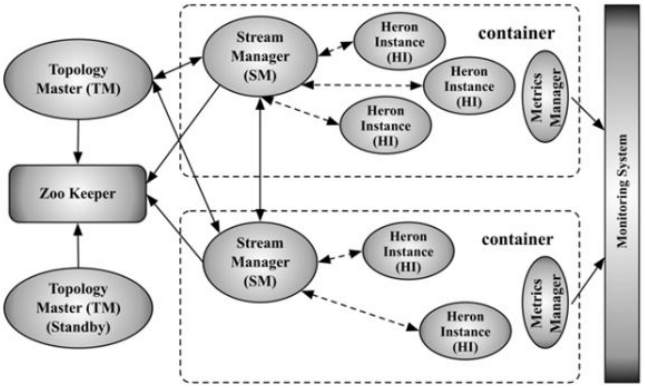

Fig. 2. Heron Topology Architecture

2.4. Stream Manager

Stream Manager(SM) is used to route tuples that are sent and received by the Heron Instances. Heron uses a 'Backpressure' mechanism to control the flow of data into the system. If in a pipeline, all the stages do not process at the same rate then this may lead to tuples being dropped in order to clear the queued up buffers. This in turn will lead to loss of computational efficiency. Heron uses Spout Backpressure mechanism where the SM clamps the local spout to reduce the inflow of data. To maintain the processing rate across all SMs, the SM that is noticing slowed down performance sends a 'Start Backpressure' message to the other SMs. Once the other SMs receive this message they also clamp down their spouts until they get the 'Stop Backpressure' message from the SM that initialized the Backpressure.[4]

2.5. Heron Instance

Each Heron Instance(HI) executes a single task of the spout/bolt. Heron uses the two-threaded approach that employs a Gateway thread and a task execution thread as shown in the Figure 3. The Gateway thread is responsible for controlling all the communication and data movement in and out from the HI. It maintains TCP connections to the local SM and the metrics manager and receives the incoming tuples from the local SM. These tuples are sent to the Task Execution thread for processing. The task execution thread runs the user code. It evokes the 'execute' method in case of a bolt and the 'nextTuple' method in case of a spout.

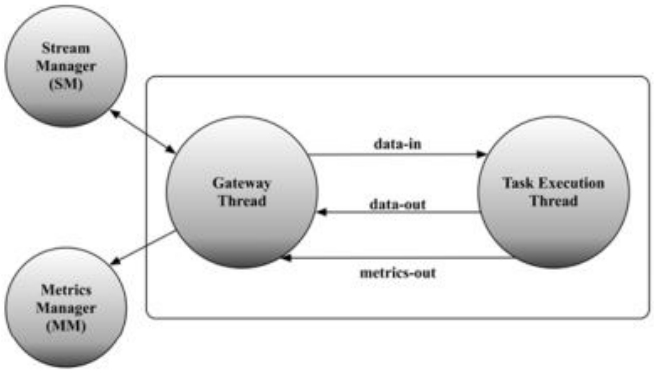

Fig. 3. Heron Instance

2.6. Metrics Manager

The Metrics manager exports the system metrics and user metrics for the topologies. There is one metrics manager for each container, to which the Stream Manager and Heron Instances report their metrics.

2.7. Failure Handling

When TM dies, the failed process is restarted in the container and the TM gets its state from the Zookeeper. The standby TM becomes the master and the master becomes the standby. When an SM fails, it rediscovers the TM to initiate a connection and get the physical plan to check if there are any changes in its state. When an HI dies, it is restarted within the container and contacts the local SM to get the physical plan.[5]

3. FEATURES

- API compatible with Storm: Back compatibility with Twitter Storm reduced migration time.
- Task-Isolation: Every task runs in process-level isolation, making it easy to debug/profile
- Use of main stream languages: C++, Java, Python for efficiency, maintainability, and easier community adoption
- Support for backpressure: dynamically adjusts the rate of data flow in a topology during run-time, to ensure data accuracy
- Batching of tuples: Amortizing the cost of transferring tuples
- Efficiency: Reduce resource consumption by 2-5x and Heron latency is 5-15x lower than Storm's latency. [6]

4. USE CASES

Apart from being the primary streaming Engine of Twitter, Heron is being used for ETL, model enhancement, anomaly/fraud detection, IoT/IoE applications, embedded systems, VR/AR, advertisement bidding, financial, security, and social media[2]. Microsoft deployed Heron on a YARN scheduler that was implemented with the Apache REEF framework.[7]

5. PERFORMANCE COMPARISON

The performance of Heron was empirically compared with Storm using word count topology. This topology counts the distinct words in a stream generated from a set of 150,000 words. According to [8] all experiments were run on machines with dual Intel Xeon E5645@2.4GHZ CPUs, each consisting of 12 physical cores with hyper-threading enabled, 72GB of main memory, and 500GB of disk space. There are no out-of-memory (OOM) crashes (or any other failure due to resource starvation during scheduling), or long repetitive GC cycles. The Storm topologies were run in isolation, i.e. no process besides the kernel, Mesos slaves, and metric exporter daemons were run in the system. Heron was run in a shared cluster, with Linux "cgroups" isolation. As shown in Figure 4,though the topology throughput increases linearly for both Storm and Heron, the throughput is 10–14x higher for Heron than that of Storm. The end-to-end latency, shown in Figure 5, increases far more gradually for Heron than it does for Storm. Heron latency is 5-15x lower than Storm's latency.Twitter reported in [6] that with Heron, numerous topologies that aggregate data every second were able to achieve sub-second latencies and Heron was able to achieve the same with less resource consumption than Storm.

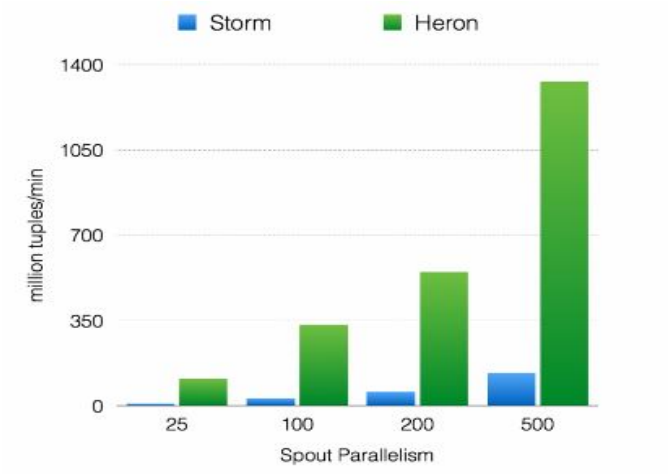

Fig. 4. Throughput with acknowledgements enabled

6. OTHER STREAMING ENGINES

- Apache Flink: This streaming dataflow engine is programmed in Scala/Java/Python API. Flink offers batch processing, fault management, automated memory management, windowing features, etc [9].
- Apache Spark: It is a batch processing engine that emulates streaming via microbatching. It incorporates Scala, Java API, Python and R library [9].

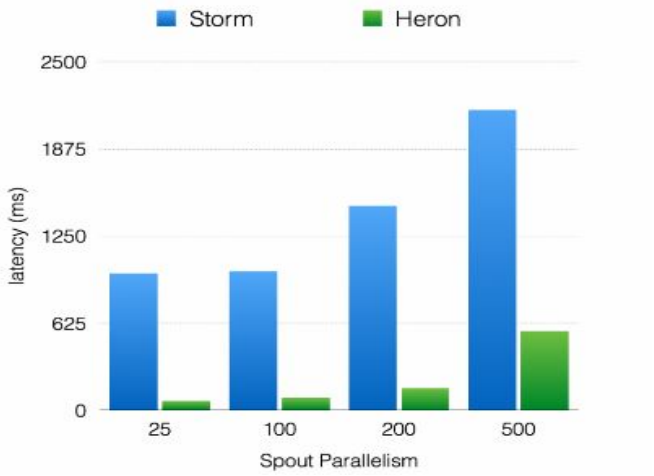

Fig. 5. Latency with acknowledgements enabled

- Apache Storm: It is is designed as a "topology" in the shape of a directed acyclic graph (DAG) with spouts and bolts acting as the graph vertices [10].
- Apache Samza:Apache Samza uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management [11].
- Apache Apex: Apache Apex is a YARN-native platform that unifies stream and batch processing. It is said to be scalable, performant, fault-tolerant, stateful, secure, distributed.
- Apache GearPump: It is an event/message based real time streaming engine. Per initial benchmarks we are able to process 18 million messages per second (message length is 100 bytes) with a 8ms latency on a 4-node cluster [12].
- Amazon Kinesis: It is a platform for streaming data on AWS, to load and analyze streaming data and to build custom streaming data applications for specific needs [13].
- Google MillWheel: Framework at google to implement stream applications. Users specify a directed computation graph and application code for individual nodes, and the system manages persistent state and the continuous flow of records [14].
- Apache Beam: A framework to create pipelines. The pipelines itself will executred on a streaming engine (such as Flink or Spark) [9].

7. CONCLUSION

Heron has been working effectively for large scale real-time data streaming at Twitter. The increase in throughput and reduction in latency of Heron has made it a useful tool for large scale processing. However it is essential to establish the required architecture to leverage the advantages of Heron. The effort and cost of setting up the Heron platform might be useful only for large scale systems.

REFERENCES

[1] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 239. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[2] K. Ramasamy, "Open sourcing twitter heron | twitter blogs," Web Page, May 2016, accessed: 2017-2-09. [Online]. Available: https://blog.twitter.com/2016/open-sourcing-twitter-heron

[3] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 242. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[4] ——, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 243. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[5] ——, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 244. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[6] K. Ramasamy, "Flying faster with twitter heron | twitter blogs," Web Page, Jun. 2015, accessed: 2017-2-09. [Online]. Available: https://blog.twitter.com/2015/flying-faster-with-twitter-heron

[7] Twitter, "Heron documentation - apache hadoop yarn cluster(experimental)," Web Page, Aug. 2015, accessed: 2017-2-19. [Online]. Available: http://twitter.github.io/heron/docs/operators/deployment/schedulers/yarn/

[8] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 247. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[9] S. Papp, "What are the differences between apache spark, storm, samza, flink, beam, apex?" Web Page, May 2016, accessed: 2017-2-19. [Online]. Available: https://www.quora.com/What-are-the-differences-between-streaming-engines

[10] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ACM. Melbourne, Victoria, Australia: ACM New York, NY, USA ©2015, 2015, p. 240. [Online]. Available: http://dl.acm.org/citation.cfm?id=2742788

[11] "Samza," Web Page, accessed: 2017-2-19. [Online]. Available: http://samza.apache.org/

[12] T. A. S. Foundation, "Apache gearpump(incubating) : Overview," Web Page, accessed: 2017-2-19. [Online]. Available: https://gearpump.apache.org/overview.html

[13] A. W. Services, "Amazon kinesis documentation," Web Page, 2016, accessed: 2017-2-19. [Online]. Available: https://aws.amazon.com/documentation/kinesis/

[14] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: Fault-tolerant stream processing at internet scale," in *Very Large Data Bases*, 2013, pp. 734–746, accessed: 2017-2-09.