Storm @Twitter- Summary

The paper “Storm @Twitter” introduced architecture, working and future work of Storm, an open-source system to process distributed data stream in a fault-tolerant environment. It also shows empirical results by conducting experiments that Storm is resilient to machine failures. Storm provides scalability, resilience, extensibility, efficiency and it is also easy to administer. Storm was created by Nathan Marz at BackType in 2011 and has been improvised at Twitter. Many organizations are using and experimenting Storm, some of them are Yahoo!, Groupon, The Weather Channel, Alibaba, Baidu, and Rocket Fuel.

**Architecture of Storm:**

The architecture of Storm is basically stream of tuples being processed in topologies. Topology is a directed graph with vertices as computations and edges as data flows. There are two types of computation units spouts- pulls data from stream queues and passes them to bolts and bolts- processes tuples and sends the output to other bolts.

Storm runs on distributed cluster such as Mesos. It has Master Node and worker Nodes. The Master node is called Nimbus which distributes and coordinates execution of topology. A Worker Node has one or more worker processes each of which runs a JVM and has one or more executors. Executor has one or more tasks. Each spout/ bolt is associated with a set of tasks running on set of executors across machines in a cluster. Thus intra-spout/bolt parallelism and intra-topology parallelism is possible because of tasks and executors respectively. A supervisor runs on each worker node which communicates with Nimbus. The cluster state is maintained in Zookeeper. Programmer has to specify the number of spouts and bolts to be used. Thus specifying a topology is equivalent to writing a query on a database.

Data is shuffled when it is sent from producer spout/bolt to consumer spout/bolt. There are 5 types of partitioning strategies in storm which are shuffle grouping, fields grouping, all grouping, global grouping and local grouping.

Internally Nimbus is an Apache Thrift service and topology definitions are Thrift objects. Specifying topology as Thrift objects allows one to program in any language. Popularly Twitter uses Summingbird to generate Storm topologies. Summingbird optimizes queries presented to it and can also produce MapReduce job to run on Hadoop. Nimbus uses local disk(s) and Zookeeper to store topology state. It saves user code on local disk and thrift objects on Zookeeper. Nimbus manages topologies and supervisors match-making. It also keeps track of state of all topologies running on the cluster. Nimbus and supervisors deamons are fail-safe and stateless, the coordination between is done by Zookeeper. This is the main reason for Storm’s resilience. The worker nodes keep working even if Nimbus service fails and supervisor restarts worker nodes if they fail. But neither can new topologies be submitted nor the topologies experiencing machine failures be reassigned to other machines if Nimbus is down.

Supervisor spawns workers based on assignments received from Nimbus. It spawns three threads: Main thread, event manager thread and process manger thread. Then it schedules recurring timer events which are hear beat event (every 15 secs): reports to Nimbus that supervisor is alive, synchronize supervisor event (every 10 secs): checks for topology reassignments; and synchronized process event (every 3 secs): manages worker processes running on same node as supervisor and also classifies workers as valid, timed out, not started or disallowed.

Worker has two threads: worker receive thread which places incoming tuples into appropriate in queue by checking the destination task identifier; and worker send thread examines task destination identifier for tuples in the global transfer queue and sends it to next worker. Executor which is running on every worker process has two threads: user logic thread which processes tuples by running actual tasks(spouts/bolts) and places it outgoing tuples in out queue of associated executor; and executor send thread checks task destination identifier of tuples from out queue and places it on global transfer queue or directly into the in queue of the destination task.

Storm has two processing semantics guarantees- “at least once” and “at most once” semantics. The first semantics is made possible by introducing an acker bolt which keeps track of every tuple emitted by a spout. Storm randomly generates 64-bit message id for every tuple and new id is generated for every new tuple being produces in the topology. When the tuple finally leaves the topology, a backflow mechanism tracks back to the spout which generated the tuple and retires it. For this semantic data source must “hold” a tuple until the spout receives a positive ack from acker and replays it if ack or fail message is not received within specified time. Such behavior is exhibited by Kestrel. Kafka queues work based on check points in Zookeeper. When a spout instance fails and restarts, it starts processing tuples from the last “checkpoint” state that is recorded in Zookeeper. The second semantic works when acker is disabled, thus a tuple is either processed once or not at all.

**Working of Storm:**

Storm has hundreds of topologies running on cluster with hundreds of servers at Twitter. Many TB of data flows in these topologies producing billions of tuples. Storm is resilient to failures and continues to work even if nimbus is down. According to the paper p99 processing latency of tuples is close to 1ms and 99.9% of the time the cluster is available in their 6 months’ observation. Storm provides rich Visualization by collecting logs, for which each topology has a metrics bolt. The metric bolt reports metrics (System metrics and topology metrics) such as CPU utilization, network utilization, memory usage on heap, number of tuple acks/fails per minute, average latency for processing a tuple. These metrics reported to Scribe which routes data to persistence key and value store. A dashboard related to a topology displays how the topology is behaving.

As already described Zookeeper is used to maintain state information. The configuration of Zookeeper with storm is crucial. At twitter three different configurations where tested. First by running storm with existing Zookeeper cluster, but the client zookeeper could support exceeded pretty quickly. Second, by using dedicated hardware for Zookeeper on the cluster, in this case the number of workers processes and topologies on storm cluster improved but a limit of 300 workers per cluster was hit quickly. This was because the zknode corresponding to worker process must receive heartbeat that it is alive otherwise Nimbus would reschedule that worker process assuming it is not alive. In the third configuration, the hardware configuration for Zookeeper was increased, and thus it scaled to approximately 1200 workers. Finally, after great analysis of logs they found out that the performance was affected by kafkaSpout code which performed 67% of the write to Zookeeper. Thus they used the fourth configuration in which they changed KafkaSpout code to write its state to a key-value store improvise the scalability of storm.

There were some concerns about the storm adding overhead to java code which was run on cluster without storm. Experiments were conducted which mitigated this concern.

The parameter “topology.max.spout.pending” which limits the number of tuples that are on the topology which have not been acked or failed can be configured in the yaml file. This also prevents storm from getting overloaded. But the problem here is about setting the parameter precisely so that the topology doesn’t starve or is not overloaded. To solve this problem, auto-tuning algorithm is implemented which adjusts this parameter based on the metrics called “progress”. This metric indicates how much data is successfully processed by the spout task.

Empirical evaluation was conducted which evaluated the resiliency and efficiency of storm. Initially storm was run on 16 physical machines with initial set of tasks for spouts (200), distributorbolts (200), usercountbolts (300) and aggregatorbolts (20). For every 15 minutes 3 machines were brought down and performance of storm was captured. As expected the results showed that storm could quickly recover from failures by stabilizing itself on fewer machines in all cases which indicates that storm is resilient and efficient.

**Future Work on storm:**

The future work is to include optimizing topologies statically and re-optimizing dynamically on runtime. Another improvement to storm is to add “exactly once” semantics. In addition, they aim to improve visualization tools, reliability of parts such as Nimbus to make it more fault -tolerant, to provide more integration of storm and Hadoop, etc.