baohez2(Baohe Zhang) and kechenl3(Kechen Lu)

CS425 MP4 Report Group 29

Part I System Design:

The Crane System is a Apache Storm-like distributed realtime computation system. It can run a topology that contains both stateful or stateless spouts and bolts. The system can save the snapshots periodically, which means once a machine failure occurs, the system can continue the job from the nearest snapshot, and not necessarily to restart the entire job.

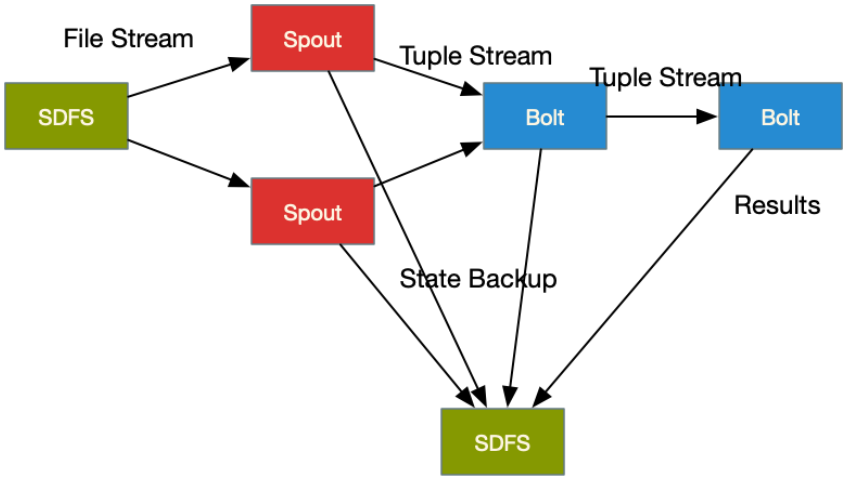


Fig. 1 Crane system streaming processing flow with SDFS and Bolt/Spout

In our Crane System, we have one driver node (equivalent to master node in Storm) and several supervisor nodes. A node running as a driver node can also work as a supervisor node. The driver and supervisors are running as daemon processes in each machine.

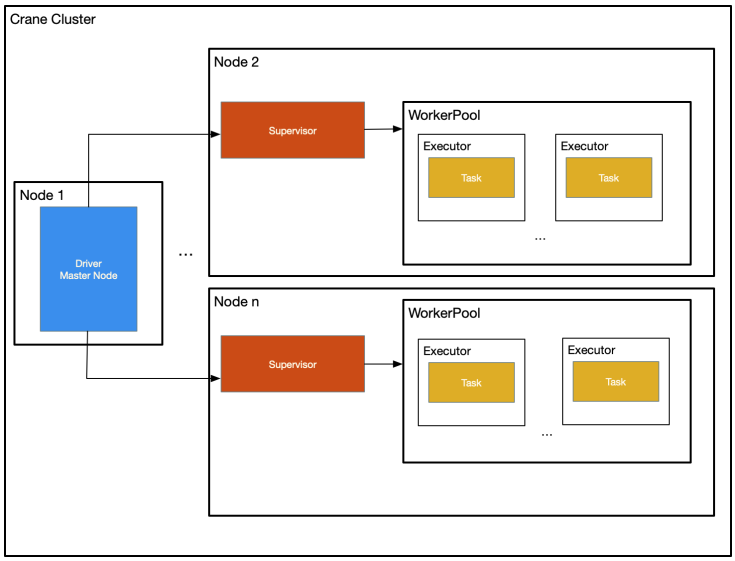


Fig. 2 Crane system streaming processing system architecture desgin

The main responsibility of the driver is to handle the job submitted by the user and schedule the specific tasks (the spout or bolt workers) for each supervisor. It should analyze the topology and distribute the tasks to each supervisor as balanced as

possible. The driver also control the snapshot saving and failure restore. The driver will periodically ask each supervisor to save snapshot and once a failure occurs, the driver will recompute and redistribute the tasks to the existing supervisors. Our inter-machine communication based on publisher-subscriber pattern. As for driver-supervisor communication, the driver is a publisher and each supervisor is a subscriber that subscribe the driver. The driver-supervisor communication channel is a two-way communication channel, which means the driver can send control signal to the supervisors through its publish board and supervisor can respond to this control signal or initiate a request though its subscriber board.

The supervisor is a commander of a machine. It can start, control and terminate the spout or bolt workers running in its machine. It will listen to the driver's control signal and send the corresponding control signal to the workers. The supervisor is also the executor of the SDFS client, it will use SDFS client to retrieve the necessary file, including topology files, process files and worker state files for its workers from the SDFS servers running in all machines. Each bolt worker or spout worker is a goroutine of the supervisor process, and the supervisor communicate with its workers through Go channel. And we use the go plugin package to do the executable files submission with available SDFS. And each worker would only need to call the designated symbol which originating from the user's topology submission.

The worker is a goroutine that created by the supervisor process. Each worker has an independent network address in the system, so it can communicate with every other worker through our publisher-subscriber communication model. There are two types of worker, either spout worker or bolt worker. The responsibility of a spout worker is to generate the tuples that will be processed and forward the tuples to the specific bolt workers. The main responsibility of the bolt worker is to collect the tuples from its previous workers (either spout worker or bolt worker) and process the tuples by some specific function that defined by the user. The result of the processed tuples can be either forwarded to the next hops or stored in local file system.

As for the inter-worker communication mechanism, each worker has a publisher and several subscribers. The worker send out its results to other workers through its publisher channel, and each worker subscriber the workers that will produce the tuple it interested. For example, in a word count topology, we have sentence generate spout, word split bolt and word count bolt. The word split bolt workers should subscribe the sentence generate spout workers' publisher chanel, and the word count bolt workers should subscribe the word split bolt workers' publisher channel. In this case, each upstream worker can pass the processed tuples to the downstream worker efficiently and reliably.

Inside the bolt worker, we have executors. User can define the number executors for each bolt worker, the executor is the unit that calls the user-defined process function.

All executors can execute the process function concurrently. The bolt worker is responsible for distributing the tuples to its executors and collect the results from its executors.

Our bolt can be stateful, which means we will save the state of each bolt workers, the state will be saved as the executors' variables and those variables will be accessed or changed every time the executors execute the process function.

We implement the snapshot saving and restore mechanism. That is every time when an error occurs (machine failure for example), the driver can send control signal to every alive supervisors and rolling back the entire system to a previous snapshot. To implement this function, each workers will serialize and put its state information into the SDFS when snapshot saving occurs, and get and deserialize its corresponding state information file from the SDFS when system restore occurs. Both snapshot saving signal and system restore signal are sent by the driver.

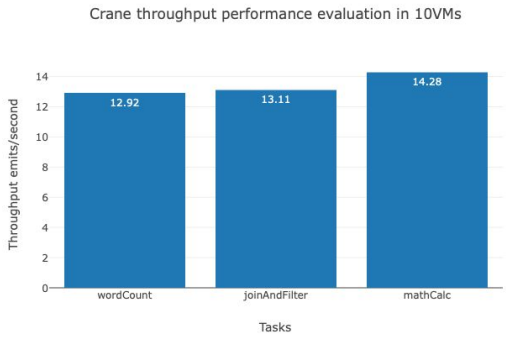
Part II Evaluation:

To evaluate our system, we write and run three real applications, including word count, tuples join and filter, and math calculation.

In our stateful word count example, the spout generate (word) tuple and the bolt output the (word, word_frequency) tuple.

In our stateful join and filter example, one spout generate (id, gender) tuple, another spout generate (id, age) tuple, the bolt merge the tuple with the same id and filter the tuple that contains age > 20, output the (id, gender, age) tuple.

In our stateless math calculation example, the spout generate the random integers tuple (integer), and 1st bolt multiply the tuple and output (tuple[0] * 2), the 2st bolt divide the tuple by 2 and output (tuple[0] / 2).



The evaluation indicates that we still have the bottleneck in our Crane streaming system. We tried to deploy the Spark but failed at the report writing. Our Crane system has the main issues in the bolt tasks and it seems remaining the stable tuple output at 13 emits/second.