CS 425 MP4

Jiayi Luo (jiayil5)
Zhoutong Jiang (zjiang30)

For this MP, we build our model based on UDP (for failure detection) and TCP (for stream transfer). The failure detection (which is based on SWIM algorithm) from MP2 is directly used here, with parameters tuned. For the stream transfer, we use TCP, which is used instead of UDP considering the data loss.

1. DESIGN:

1.1 Architecture introduction

Overall, our Crane's architecture follows typical Storm system. We have two masters in total, one in serve and another one standby. When the master fails, the standby master will immediately join the group and role as a new master. Except the two masters, all the other nodes are worker nodes with different functionalities. Users can submit task from any of the worker node. For terminology clarification, a tuple is the message lives between two nodes. It originates from the spout and flows through the topology. Tuples may vary after passing bolt node.

1.2 Topology

Our storm-like system has a master that will handle the task assignment of our system, Master will receive the task requirement from the client in the format of YAML and generate a topology for the task. To maximize the utilization of workers in our topology, we first distribute the loads (tasks) to each worker based on the number of layers. The workers will have the functionality including one spout, one client, and multiple bolts. Each layer will have approximately equal number of workers, in which case we try to minimize the longest processing time. Also, at each stage, the outputs of workers will be sent to its children using the round robin algorithm to distribute the load as much as possible.

1.3 Fault-tolerance design

On the node level, the failure detection is based on MP2 SWIM implementation. More details of this part are already discussed in our report for MP2. If a master is done, we will require the standby master to take the place of master and ask the client to resend the request. The disruption of other nodes will only influence our system in the data transmission level.

On the data transmission level, we select TCP to guarantee the reliability of the data transmission during the stream transmission phase. Each member will initial TCP connection before the message was sent. Additionally, we also have a task queue to collect the tuple(s) and task(s) if the parent worker fail to Dial the child process. Every worker will try to finish its task (dequeuer every task in their queue).

1.4 Functionality supported

We support four major functions, (1) filter based on key, (2) transformation of data, (3) join with static database, (4) word count of files. Multiple tasks can be chained to process more difficult tasks. Our algorithm is also designed in the way that more functions can be easily incorporated.

1.5 Failure Recovery

Crane system supports 1). one master failure and 2). two simultaneous failure of workers. When one node detects one failure, it will report to master, and master will start reorganizing subroutine. Under reorganizing subroutine, there are two cases (1), If the failure happens to worker nodes which have other alive nodes in the corresponding layer - for instance, three workers A, B, C do "filter" concurrently, then A and B fail – master won't ask to restart the whole system (including the topology and task). Those remaining jobs that have been assigned to A and B will be handed over to C to avoid restarting the whole task again. 2). When failure happens to the master, or worker nodes with no more other workers remaining on the same layer, the master will ask to restart the whole system (including the topology and task).

2. EVALUATION

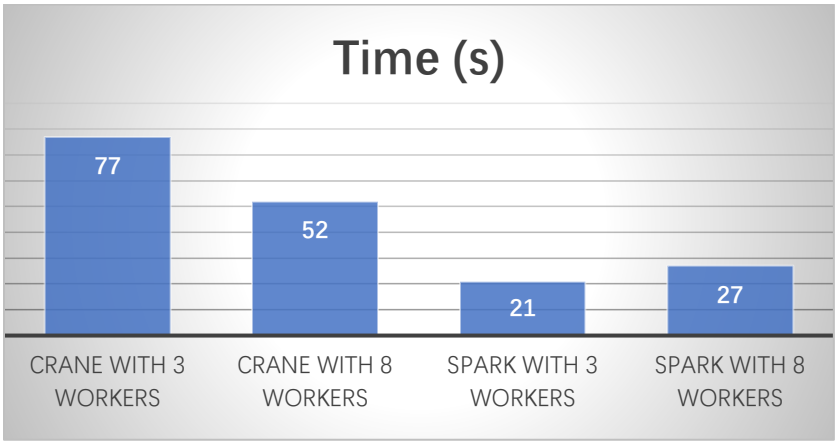

Fig 1

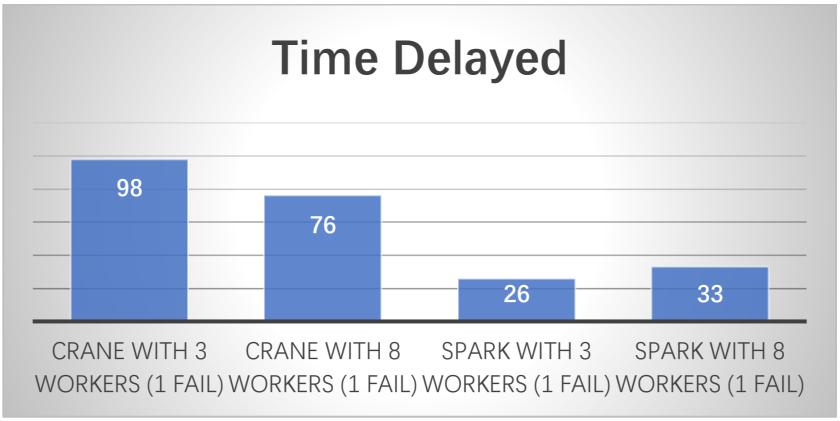

Fig 2

From Figure 1 and Figure 2, we can find that: 1) both Crane and Spark can improve their performance with more workers joining in the group. However, the overhead of adding one new worker is big, thus the improvement is subtle. This probably is caused by the latency of network transmission. Adding one new worker means at least two more TCP connections are required for several existing nodes. 2) Crane is slower than Spark based on the time performance, this is probably because some non-optimized operation in Crane's system. 3) The failure (if the remained jobs can be handled by its siblings) won't affect the ongoing task much since its siblings will voluntarily pick up the remained tasks.

Fig 3

From Figure 3 (3 workers for both systems), we can find that the computation times of both Crane and Spark are increasing with the increase of input file size. The trend is almost linear, O(n), which is not desirable. The overhead of the initial costs of both systems are almost the same. Nevertheless, with the increasing of the file size, the performance of Spark system appears to be more and more desirable. This might results from the latency and lateness of network in Crane system. We expect the performance of Crane system might be improved if we were able to improve the level of parallelism in some work, in stead of using round robin stream processing, which might generate lots of traffic in network.