Adaptive Scheduling Based CPU/IO in Storm

Xinli Zhang

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China

zhangxinli638945@163.com

Keywords: Big Data, Adaptive Scheduling, Storm.

Abstract. Storm is a real-time streaming of Big data processing field tools. But the task of topology will even divided to the executors in worker node, which as default scheduler. It neither consider internal solt transmission delay time, nor consider transmission delay between nodes, but does not consider the task of local characteristics. The first two issues has already been resolved, this paper focuses on local characteristics and solve the task. Experimental results show that optimized of scheduling strategy optimize the default Scheduling policy in the comprehensive aspects.

Introduction

Big data processing is divided into two models: one is a batch mode, the representative system is Hadoop, HPCC, Spark, etc., which data source is static. Another is the stream processing model, the representative system is the S4 system Yahoo, Twitter's Strom. Strom has a great advantage in dealing with real-time data. Not only Tiwtter, but also many domestic companies such as taobao, 360, are in one way or a particular business is using Strom.

Storm is a complex event processing engine, the distributed result of which can real-time processing of high data traffic. Officially because of these characteristics, Strom quickly conquered these big companies like Twitter, Groupon and so on. Strom cluster can run a topological structure composed of running nodes. Its two components spout (produced by the event) and bolt (implement the processing logic). Events generated by the spout forming a data stream can be transmitted in different bolts.

In this paper we target to design and implement of IScheduler to to replace default scheduler. Compared to default storm, the Iproved_Scheduler in this paper aim at sloving the delay time between inter nodes and sloving the local characteristics, such as CPU/IO Data. But as default scheduler, their all purpose is improving their performance

Problem and Algorithm

Problems in default scheduler.

IScheduler is the scheduler of the storm, which is responsible for allocating available resources of Topology to the current cluster. storm offers three Scheduler in storm:

1. EvenScheduler：If slot is sufficient，it can ensure that all topology of the task is evenly distributed throughout all nodes in this cluster. In the slot is not sufficient, all the task of topology will assign to the only slot.
2. DefaultScheduler: As EvenScheduler, before the distribution, it will release of other Topology unneeded resources first, then call EvenScheduler.
3. IScheduler: It specify a separate machine resources they need for certain Topology. This interface is reserved in storm. According to their needs,they can write the corresponding scheduling algorithm.

EvenScheduler as default scheduling algorithms of storm, and its role is the average allocation of resources. Because there is no consideration the time delay between the node, between inside solt of nodes, and also does not solve the problem of local data. Therefore, this article proposes scheduling algorithms to solve these problems.

The basic idea of the algorithm.

We use N ={ n_i } to represent a cluster Worker nodes, each node has { S_i }available solts. There are T={ t_i } topology in several clusters. Each topology has W={ w_i } a worker. And each topology t_i contains C_i components ,which connected with each other. Each module contains two parameters (i) the number of executors (ii) the number of tasks. Thus, The topology that contains the number of the above worker is min(W_i , E_i)。 . So a cluster that contains an amount of worker $\sum_{i=1}^{t} \min(E_i, W_i)$. A scheduler can be used to meet there is sufficient available S, so $\sum_{t=1}^{t} S_i \geq \sum_{i=1}^{t} \min(E_i, W_i)$ 。

We use ε ={ $e_{i,j}$ } to represent each worker above executors. $R_{i,k,j}$ Represented by network load from $R_{i,k,j}$ $e_{i,k}$ to $e_{k,j}$. By representing each executor is the kind of intensive actuators (0,1,2). Meanwhile β denoted by weight of each task.

Adaptive Scheduling Algorithm Based CPU/IO.

In the first phase, for each topology, the pairs of communicating executors are iterated in descending order by rate of exchanged tuples. For each of these pairs, if both the executors have not been assigned yet, then they get assigned to the worker that is the least loaded at that moment. Otherwise, the set Π is built by putting the least loaded worker together with the workers where either executor of the pair is assigned. Π can contain three elements at most: the least loaded and the two where the executors in the pair are currently assigned. All the possible assignments of these executors to these workers are checked to find the best one,that is the assignment that produces the lowest inter-worker traffic. At most, there can be 9 distinct possible assignments to check.

This phase is amied at computing an allocation $\xi \rightarrow$W,which is form executors to worker,and anthor allocation W→N ,whch maps from workers to nodes and satify the contraints:

$$\forall k = 1 \ldots N \quad \sum_{\substack{A_2(A_1(e_{i,j}))=n_k \\ i=1\ldots T;\, j=1\ldots E_i}} L_{i,j} \leq B_k$$

Similarly, in the second phase the pairs of communicating workers are iterated in descending order by rate of exchanged tuples. For each pair, if both have not been allocated to any node yet, then the least loaded node is chosen to host them.If any or both have already been assigned to some other nodes, the set is built using these nodes and the least loaded one. All the possible allocations of the two workers to the nodes in are examined to find the one that generates the minimum inter-node traffic. Again, there are at most 9 distinct allocations to consider.

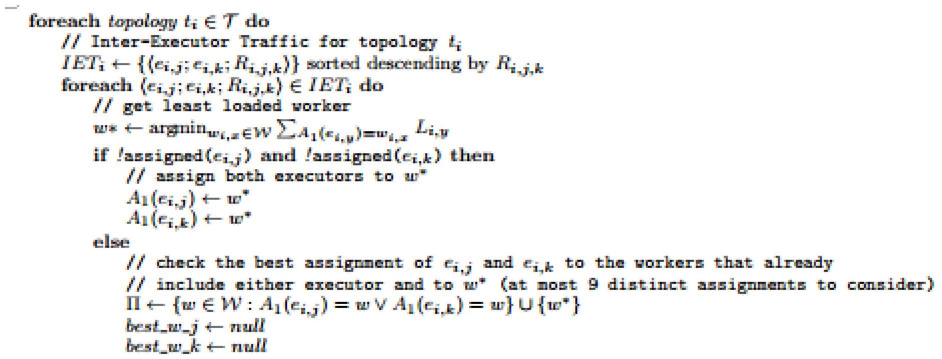
The max number of worker node of topology can satify:

$$\forall i = 1 \ldots T$$

$$|\{w \in W : A_i(e_{i,j}) = w,\, j = 1 \ldots E_i\}|$$

The iner-node tarffic satify:

$$\min \sum_{\substack{j,k:A_2(A_i(e_{i,j})) \neq A_2(A_1(e_{i,k})) \\ i=1\ldots T,\, j,k=1\ldots E_i}} R_{i,j,k}$$



```
foreach topology $t_i \in T$ do
    // Inter-Executor Traffic for topology $t_i$
    $IET_i \leftarrow \{\langle e_{i,j}; e_{i,k}; R_{i,j,k} \rangle\}$ sorted descending by $R_{i,j,k}$
    foreach $\langle e_{i,j}; e_{i,k}; R_{i,j,k} \rangle \in IET_i$ do
        // get least loaded worker
        $w* \leftarrow \mathrm{argmin}_{w_{i,x} \in W} \sum_{A_1(e_{i,y})=w_{i,x}} L_{i,y}$
        if !assigned($e_{i,j}$) and !assigned($e_{i,k}$) then
            // assign both executors to $w^*$
            $A_1(e_{i,j}) \leftarrow w^*$
            $A_1(e_{i,k}) \leftarrow w^*$
        else
            // check the best assignment of $e_{i,j}$ and $e_{i,k}$ to the workers that already
            // include either executor and to $w^*$ (at most 9 distinct assignments to consider)
            $\Pi \leftarrow \{w \in W : A_1(e_{i,j}) = w \vee A_1(e_{i,k}) = w\} \cup \{w^*\}$
            $best\_w\_j \leftarrow null$
            $best\_w\_k \leftarrow null$
            $best\_ist \leftarrow MAX\_INT$
            foreach $(w\_j, w\_k) \in \Pi^2$ do
                $A_1(e_{i,j}) \leftarrow w\_j$
                $A_1(e_{i,k}) \leftarrow w\_k$
                $ist \leftarrow \sum_{x,y:A_1(e_{i,x}) \neq A_1(e_{i,y})} R_{i,x,y}$
                if $ist < best\_ist$ then
                    $best\_ist \leftarrow ist$
                    $best\_w\_j \leftarrow w\_j$
                    $best\_w\_k \leftarrow w\_k$
                end
            end
            $A_1(e_{i,j}) \leftarrow best\_w\_j$
            $A_1(e_{i,k}) \leftarrow best\_w\_k$
        end
    end
end
```

Evaluation.

Using a host as a Nimbus and Zookeeper (CPU 2 × 2.4G, memory 2G, hard disk 500G)
8 host for the worker node, each worker node has 5 solts.
100M of lan
Cluster Installation Environment: Java1.7 + Zookeeper -3.4.5 -cdh4.3.0 + Kafka + storm-0.9.1 + MySQL5.1.69 + HBase-0.94.3.

1. CPU-Intensive data

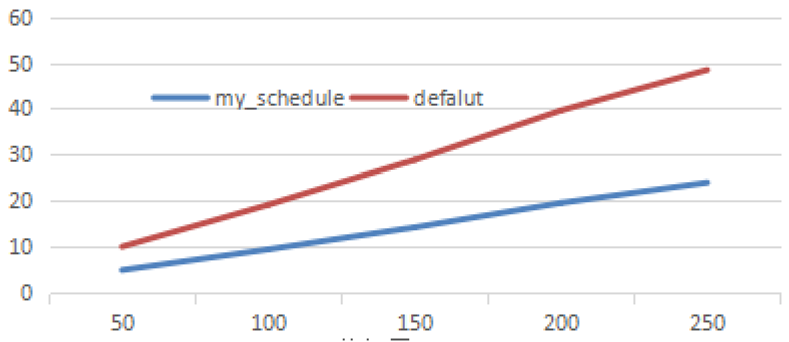


Figure 1 CPU-Intensive

2. IO-Intensive data

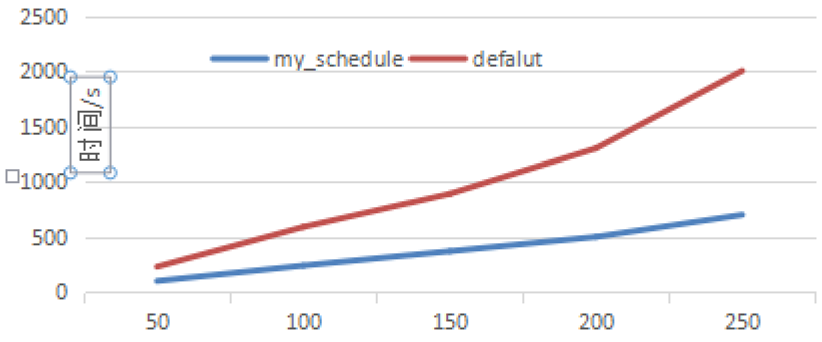


Figure 2 CPU-Intensive

3. Equilibrium data

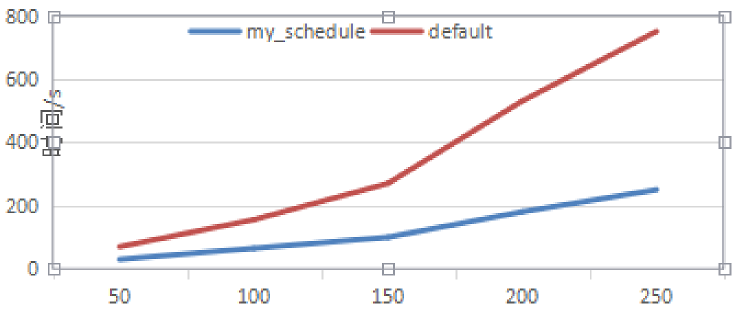


Figure 3 Equilibrium

As can be seen from the above experimental data, when we use CPU-Intensive data, the time between them is similar, but the others the time they use is different. The improved scheduling strategy no matter what kind of data is better than the default scheduling policy.

Related Work

On the basis of the default scheduling algorithms, we propose two adaptive scheduling algorithm. They have in common is the communication between the Executors to be taken into account, but one is off-line, and another is online adaptive scheduling[3].

Re-implement a T-Strom, the main work is in scheduling optimization algorithm, the main approach is the real-time communication between nodes taken into account, it is less to maximize communication between the nodes, in order to protect dynamic to assign tasks, thereby improving the efficiency of scheduling[8].

The main proposed tasks Strom not efficiently processing iteration, which made the Storm system architecture is designed based on an iterative Topology implementation[9].

Summary

Based on the Storm's IScheduler be extended to replace the original Scheduler, not only solved the communication delay between nodes, but also solve the task of the local nature. Experiments show that the efficiency of a significant improvement over the original scheduling policy.

Acknowledgements

This work is supported by NSFC (Grant Nos. 61300181, 61502044), the Fundamental Research Funds for the Central Universities (Grant No. 2015RC23).

References

[1]. Information on: https://blog.twitter.com/2013/new-tweets-per-secondrecord-and-how.

[2]. Sheng X, Tang J, Xiao X, et al. Sensing as a Service: Challenges, Solutions and Future Directions[J]. Sensors Journal IEEE, Vol. 13(2014) No. 10, p.3733 - 3741.

[3]. Aniello L, Baldoni R, Querzoni L. Adaptive Online Scheduling in Storm[C], Proceedings of the 7th ACM international conference on Distributed event-based systems. ACM, 2013,p.207-218.

[4]. Information on: http://storm-project.net.

[5]. A, Verma, G, Dasgupta, T, K, Nayak, P, De, and, R, Kothari. Server Workload Analysis for power Minimization Using Consolidation[D]. US:Proceedings of USENIX Annual Technical Conference, 2009.

[6]. Information on: https://github.com/nathanmarz/storm/wiki/Guaranteeing-messageprocessing.

[7]. Information on: https://github.com/nathanmarz/storm/tree/0.8.2.

[8]. Xu J, Chen Z, Tang J, et al. T-Storm: Traffic-Aware Online Scheduling in Storm[C], 2014 IEEE 34th International Conference on Distributed Computing Systems (ICDCS). IEEE Computer Society, 2014,p.535-544.

[9]. ZhengJie Du, Peng Wang, Yan Huang,et al.An iterative model of Topology based on strom programming[J]. Journal of Chengdu University of Information Technology, Vol. 1 (2014) No. 1,p.47-51.