Design of Big Data Processing System Based on Stream Computing Storm

Bao Ling
Leshan vocational and Technical College, Leshan, Sichuan 614013, China
35720392@qq.com

Keywords: Big Data; Real-time Streaming; Stream Computing

Abstract: Aiming at the real-time, volatile, and disorderly characteristics of big data streams, this paper takes optimizing Storm stream computing as a cornerstone, and proposes a big data real-time stream processing system based on the Storm stream computing engine. The system can not only realize data collection, data buffering and streaming calculations, but also meet the fast processing needs of condition monitoring anomaly detection and data analysis, as well increase the component throughput of the sliding window and improve the real-time processing efficiency of condition monitoring anomaly detection. This study lay the foundation for cloud manufacturing platform, it has a practical significance.

1. Introduction

With the development of emerging technologies such as virtualization, cloud computing, and the Internet of Things, human society has quietly entered the era of big data [1]. Data warehouse systems, data analysis platforms, cluster management tools, machine learning algorithm libraries, distributed coordination services and so on provide convenience for the analysis and processing of massive data. However, Map Reduce, a programming model suitable for batch processing, exhibits high latency and slow response when responding to high-rate data processing requirements, which leads to its limitations in the field of real-time data analysis and processing. The streaming computing system with the goal of "high real-time computing" provides support for real-time analysis and processing to increase speed and efficiency [2]. But these systems still have obvious shortcomings in many aspects such as scalability, system fault tolerance, state consistency, load balancing, and data throughput. It is an urgent problem that how to build a big data streaming computing system with low latency, high throughput and continuous and reliable operation. This article takes the optimized streaming computing system storm as the core, trying to build a big data real-time analysis processing system with the storm streaming computing. This system provides some references for building a big data streaming computing system with low latency, high throughput and reliable operation, in addition that makes up for the current lack of research results on big data streaming computing.

2. Optimization of Storm streaming computing

2.1 Dynamic progressive backpressure strategy

In this paper, a dynamic stepwise backpressure strategy is used to dynamically adjust the rate of Tuple flowing through the component [3]. This strategy adjusts the rate at which the upstream sends data to the task according to its current load.

Input: Ti, QML, system //Topology Tasks collection, maximum queue threshold, system preset value

Output: Back pressure or cancel the back pressure //Perform back pressure or cancel the back pressure

1. Get sensitivity for system; //Get the preset back pressure sensitivity
2. Get loadFactor for system; //Get the preset Task queue load factor
3. FOR task j in Ti DO // Traverse Task
4. Get queueLength, Ui of task j; //Get the queue length of task j, upstream Task collection  
5. Get timeSchedule of task j; //Get task j task handle  
6. IF queueLength>loadFactor* QML AND //Trigger the highest water level, send a back pressure message  
   7. timeSchedule==null THEN  
5. promise=timeSchedule. schedule({send a back pressure signal to Ui}, sensitivity);  
9. IF promise. resolve THEN //The back pressure signal is sent successfully  
10. cancel the timeSchedule of task j; //clear the task handle  
11. ELSE IF promise. reject THEN //Failed to send back pressure signal  
12. add taskj to Ti; //task j join Ti  
13. END IF  
14. ELSE IF queueLength<(1-loadFactory)*QML AND //Trigger the lowest water level, cancel a back pressure  
15. last sensitivity*2^4 THEN  
16. send a back pressure cancel signal to Ui;  
17. END IF  
18. END FOR  

The implementation algorithm of dynamic gradual backpressure begins to backpressure upstream when the task input queue load reaches the maximum threshold. The upstream Task that receives the backpressure signal will slow down the rate of sending data downstream, and the sending rate will slow down to the current speed v'. Therefore, the downstream Task load can be slowed down, but this may cause the upstream queue to be blocked, so the upstream may continue to back pressure to its upstream. In the worst case, the back pressure will not end until the topological Spout. Each time the Task is back pressured, the rate will drop to v' of the current speed. This article uses a timer (timeSchedule) and sets the timer sensitivity. The process is shown in Figure 1. Continuous is an empirical value after many experiments. The value adopted in this article is 16 (2^4). A slight change to the continuous value has no effect on the back-pressure effect.

![Fig.1 flow chart of dynamic stepwise backpressure](image-url)
2.2 Parallel data reflow method

Ether Tuple loss or Tuple processing time exceeding the prescribed processing time of the Topology (determined by configuration) will cause Tuple replay, and frequent replay of Tuple is one of the main reasons for Topology overload [4]. This article adopts the method of parallel data reflow, the data being read has no "hanging" state, and the simplified state machine is shown in Figure 2. In order to ensure the reliability of data, a multi-parallel topic replication algorithm is embedded in Kafka. In the implementation, Kafka Topic is split into three parallel logical topics, as the Eden area, the From area and the To area, where Eden is the original queue area, From and To are the replication areas. The replication algorithm process is shown in Figure 4.

2.3 Unaware topology replacement mechanism

The core idea of unaware topology replacement is to collect the number of Tuples that fail due to timeouts in the Topology within multiple time windows. We calculate the mean E(M) of failed Tuples and the weighted dispersion rate S^2, and choose whether to replace it with a new Topology according to the mean and dispersion rate [5]. The calculation formula (1) is as follows, where W represent the weight, and the weight is related to the time window. The longer the distance from the current time window, the lower the weight.

\[
E(M) = \frac{\sum_{i=1}^{n} X_i}{n} \quad S^2 = \frac{\sum_{i=1}^{n} W_i \times (X_i - E(M)) ^2}{n}
\]  

The larger the S^2 value, the more unstable the current Topology. The larger the S^2 value, the more unstable the current Topology. If the number of failures in the current time window is greater than E(M), and S^2 is greater than the maximum threshold max(C), then expansion is performed; if the number of failures in the current time window is less than E(M), and S^2 is greater than the maximum threshold max(C), which means that the current Topology may have excessive resources, and it will be scaled down.

\[
newindex = hash&(newcap - 1) = \begin{cases} 
hash((nowcap \times 2) - 1) & \text{Expansion} \\
hash((nowcap + 2) - 1) & \text{Shrink}
\end{cases}
\]
3. System design based on Storm streaming engine

3.1 System logical structure design

This article is based on the background of a major scientific and technological project about a cloud manufacturing service platform for the elevator industry alliance in a province undertaken by the laboratory, and focuses on the big data stream processing chain. Research on key computing technologies, design and implement real-time monitoring systems for cloud manufacturing platforms, and provide real-time monitoring services for cloud manufacturing platforms. Figure 4 shows the application architecture of Storm-based big data streaming computing in the cloud manufacturing platform. The data generated during the operation of the cloud manufacturing platform is collected and preprocessed by the streaming data layered collection strategy. Flume provides both data collection and bridging functions, and then performs data calculations and stores the calculation results in the persistence layer.

3.2 System business process design

The data flow architecture of the cloud manufacturing platform real-time monitoring system is shown in Figure 5. The data sources include Apache Tomcat logs, Mysql database and Ganglia. This article obtains important performance measurement by Ganglia, such as cluster CPU, disk, memory, and network bandwidth. The data source flows into the system through the streaming data layered collection strategy, and is stored in HDFS and Kafka respectively. Kafka acts as a cache to provide a real-time and stable data stream for the streaming computing engine. The calculated data is stored in the data persistence layer, which is mainly HBase and Mysql. Finally, the front end obtains relevant data by querying the persistence layer and displays the monitoring page.

Fig. 4 Application framework of storm-based big data streaming computing in cloud manufacturing platform

Fig. 5 Cloud manufacturing platform real-time monitoring system data flow architecture

Figure 6 describes the event flow involving data collection and storage in the application example: (1) The cloud manufacturing platform generates a piece of data during its operation; (2) The agent layer of the streaming data hierarchical collection strategy obtains the data and performs corresponding cleaning with filtering before transmitting it to the collection layer; (3) The collection layer processes the header information in the data and sends it to Kafka and HDFS. Kafka is used to cache data and provide real-time data streams for the streaming computing engine. The instance settings only save the last 7 days of data, that is, the data will be automatically cleaned up.
after 7 days of caching in Kafka, and HDFS is used to permanently store data records.

![Fig.6 Data collection and storage event stream](image)

Figure 6 describes the two event streams involved in data calculation and display. The first is the real-time monitoring system front-end display request event flow of the cloud manufacturing platform. The second is the event stream in which the Storm stream computing engine processes data and stores the results in DB (Database, DB).

![Fig.7 Data calculation and display data stream](image)

4. Application examples

The experiment was carried out in the form of test data grouping. Each group of 200 monitoring samples was designed, and a total of 15 groups were used for comparison experiments. The time consumption in each group was recorded as the average value within the group. The time consumption is shown in Table 1:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional monitoring system</td>
<td>65</td>
<td>99</td>
<td>84</td>
<td>119</td>
<td>93</td>
<td>75</td>
<td>133</td>
<td>59</td>
<td>72</td>
<td>41</td>
<td>129</td>
<td>83</td>
<td>68</td>
<td>41</td>
<td>79.57</td>
</tr>
<tr>
<td>The monitoring system proposed in this article</td>
<td>23</td>
<td>42</td>
<td>31</td>
<td>37</td>
<td>50</td>
<td>39</td>
<td>61</td>
<td>27</td>
<td>29</td>
<td>26</td>
<td>48</td>
<td>34</td>
<td>28</td>
<td>34</td>
<td>36.36</td>
</tr>
</tbody>
</table>

It can be intuitively compared from the experiment that the monitoring method adopted by this
system during the data processing, storage, and query request period is about half of the traditional HTML-based method. In addition, the monitoring data switching time of this method is less than 2s, which meets the real-time requirements of web-based configuration software.

References


