Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 3, No. 2, 2023

A Distributed Streaming Computing Model Management System

Shangyu Zhao[1], Ping Lin[2]

[1] Wenzhou Polytechnic, Wenzhou, China
[2] Zhejiang Industry & Trade Vocational College, Wenzhou, China

Abstract: IoT big data has many dimensions, many business types and many customized requirements, and each business needs to be implemented using a stream computing model with a wide variety of stream computing models. This paper proposes a management method for streaming computing models, which can dynamically manage streaming computing models and achieve hot loading of streaming computing models. This makes it easier to manage computational models in a streaming computing cluster.

Keywords: Distributed system; Streaming compute; Class loader.

1. Introduction

With the rapid development of IoT big data, more and more front-end collection devices are producing huge amounts of IoT data all the time [1]. This IoT data flows like water into the data centre or business system, which then processes the IoT data. The data centre or business system uses a stream computing model to process the IoT big data, but there are many dimensions of IoT big data, many types of business and many customized requirements, each business needs to use a stream computing model to implement, and there are many kinds of stream computing models. The current common streaming computing engines, such as Flink and Spark Streaming [3], mostly adopt the manual online approach to install the streaming computing models on the computing nodes of data centres or business systems [4], and there is no effective management and scheduling method for streaming computing models, which makes the development and management of streaming computing models difficult [5]. In this paper, we propose a management method for streaming computing models, which enables hot loading of computing models by means of custom classloader, and design a distributed management mechanism that can manage computing models in a distributed system, and computing models can be synchronized up and down in each computing node in the distributed system.

2. System Architecture

The model management architecture as a whole consists of interface, model library, messaging module, model loading module and redis database, and the overall architecture is shown in the Figure 1.

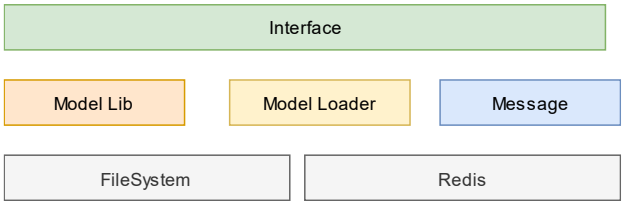

Figure 1. Overall system architecture

2.1. Streaming Computing Model

Generally, streaming computational models are developed by JVM based programming languages such as Java and Scala, and executable computational models consist of mutated Jar files and their configuration files. In this paper, the targeted computational model refers to the compiled application developed by a JVM based language. Managed computational models include model files, model configurations, model metadata.

2.2. Interface

All operations on the streaming model are submitted via the external interface. The interfaces can be divided into 6 types according to their function: add models, update models, delete models, online models, offline models and query models.

The cluster has a centreless design, where the user sends a request to any node in the cluster, and the request is broadcast and forwarded throughout the cluster.

2.2.1. Add Models

add a new model file to the model library, and after the model passes the verification, call the broadcast module to notify other nodes to download the added model from the model library.

2.2.2. Update Models

Update the existing model in the current system, update the file in the model library, and after the model passes the verification, call the broadcast module to notify other nodes to download the updated model from the model library.

2.2.3. Delete Models

Delete m Delete the existing model in the current system, after the model passes the verification, research the broadcast module, each node in the cluster delete the model in the model library.

2.2.4. Online Models

Bring models that are already in the model library online, load the models in memory so that they can start processing streaming data.

2.2.5. Offline Models

Offline models: take models that are currently online offline so that they no longer process streaming data.

2.2.6. Query Models

Query the current model's metadata and status in the system.

With the above 6 types of interfaces, users can complete a unified style of management of stream computing models, or develop UI interfaces based on the interfaces for more convenient management of streaming computing models.

2.3. Model Library

As a lightweight streaming model management system, distributed file systems such as HDFS [6] and Ceph[7] are not used as model repositories, cause these systems would greatly increase the complexity of the system.

When using the local file system as the model repository storage, when adding models, updating models, deleting models and other operations, only when all nodes in the cluster have finished adding, changing and deleting files in the model repository, the model operation is considered successful and the model information is updated.

When a node is started, there may be a situation that the model version is inconsistent with other nodes in the cluster. In this case, to recover the model data from other nodes synchronously, find the currently existing computational model and its latest version and the node where it is located in the database, and if the latest version is inconsistent with the local version, download the latest version of the model from the node where the latest version of the model is located.

2.4. Message Module

When an operation is performed on a model, the operation request is first sent to any node, and then notified to the other nodes in the cluster through a broadcast and subscription mechanism. As shown in Figure 2, Using the broadcast subscription mechanism of redis, the model information, operation type and node information is broadcast to all nodes. The subscribers receive the information about these model operations and perform the corresponding operations. Upon success, the node that received the request is notified that the operation was successful when all nodes in the cluster have been executed.

The same broadcast and subscription is used when going up or down the model, to synchronise information up and down the line throughout the cluster environment.

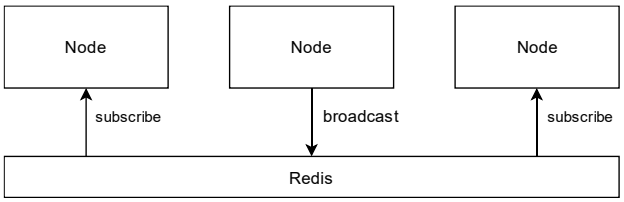

Figure 2. Broadcast and subscribe via redis

2.5. Model Loading Module

When online a model, model loading module is required to load the model into the compute server. The model management system proposed in this paper uses the ClassLoader technology, a custom ClassLoader inherited from the system ClassLoader in Java [8], which enables hot loading of jar files. As shown in Figure 3, The models are grouped together and each group has its own ClassLoader, so that even if the class names and variable names of the models in different groups are the same, they do not affect each other, thus achieving decoupling between models.

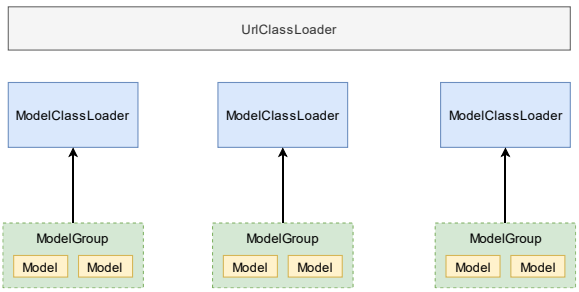

Figure 3. Grouped class loader instance

2.6. Redis Database

Redis is the only external component required by the stream computing model management system to store system metadata, model metadata, broadcasts and subscriptions.

3. Workflow

3.1. Initialization

As shown in Figure 4, an initialisation process is designed in this paper, whereby a newly online node can quickly synchronise models from other nodes and the system can still provide services normally during the synchronisation process. After initialisation, the new node is ready to provide services and the models in the model library of that node are consistent with other nodes in the cluster.

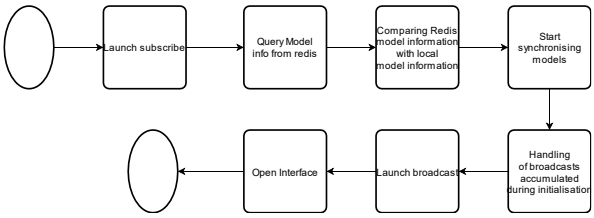

Figure 4. Initialisation process

3.2. Add Models

As shown in Figure 5, when adding a model, the node that receives the add request sends a broadcast to notify the other nodes in the cluster, and all other nodes download the new model file. When a node has a problem or waits for a timeout, the whole model adding process fails and all nodes roll back. After initialisation, the new on-line node can provide services and the models in the model library of that node are consistent with the other nodes in the cluster.

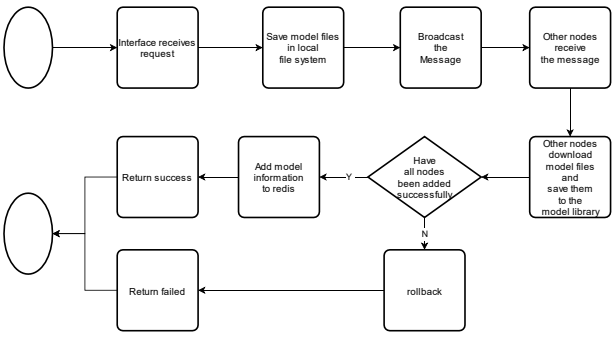

Figure 5. Add models process

3.3. Update Models

As shown in Figure 6, when updating a model, a check must first be performed to see if the model already exists and is out of the offline state, if the model does not exist or is in the online state, the model cannot be updated. As with adding

a model, the node that receives the add request sends a broadcast to notify the other nodes in the cluster, and all other nodes download the new model file. When a node has a problem or waits for a timeout, the entire model add process fails and all nodes roll back. When all nodes are successful, the model information in redis is updated and success is returned.

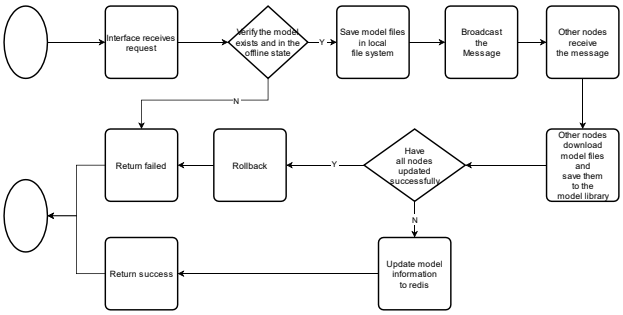

Figure 6. Update models process

3.4. Online Models

As shown in Figure 7, when a model goes live, it needs to be verified that the model exists and is in the offline state, again with a broadcast mechanism to distribute the message and load the model using the model loader described in 2.5.

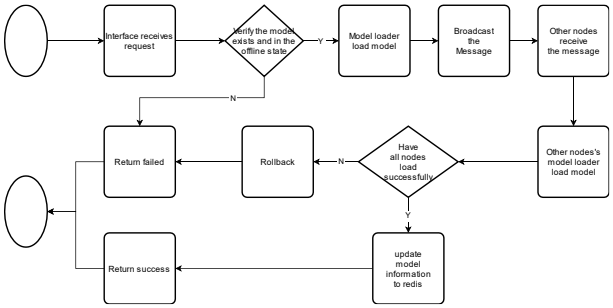

Figure 7. Online models process

3.5. Offline Models

The model offline process is similar to the model online process, where you need to verify that the model exists and is online, and unload the model using the model loader described in 2.5.

3.6. Delete Models

Model deletion and model update processes are similar.

4. Conclusion

This paper proposes a management system for streaming computational models, which enables hot loading of computational models by means of a custom Class Load. In this system, different grouped models are decoupled from each other by using independent loaders. Meanwhile, this paper designs a decentralized distributed model management mechanism, which enables computational models to be added, deleted, modified and up/down synchronously by each computational node in the distributed system through message broadcasting and subscription mechanisms.

Acknowledgements

Artificial Intelligence Academy, Wenzhou Polytechnic.

References

[1] Cai H , Xu B , Jiang L , et al: IoT-Based Big Data Storage Systems in Cloud Computing: Perspectives and Challenges, IEEE Internet of Things Journal, Vol. 4 (2017) No.1, p.75-87.

[2] Wang, Z, Z, et al: IoT-based real-time production logistics synchronization system under smart cloud manufacturing, International Journal of Advanced Manufacturing Technology, 2016.

[3] Cheng D , Yuan C , Zhou X , et al: Adaptive scheduling of parallel jobs in spark streaming, IEEE INFOCOM 2017 - IEEE Conference on Computer Communications (Atlanta, USA, May 1-4, 2017), vol. 29 (2012).

[4] Nabi Z: Pro Spark Streaming: The Zen of Real-Time Analytics Using Apache Spark(Apress, USA 2016).

[5] Upfal, E, and A. Wigderson: How To Share Memory In A Distributed System, 25th Annual Symposium on Foundations of Computer Science(Florida, USA, October 24-26, 1984).

[6] Liu X, Han J, Zhong Y, et al: Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS, IEEE International Conference on Cluster Computing & Workshops. (Beijing, China, September 24-28, 2009).

[7] Weil S A , Brandt S A , Miller E L , et al: Ceph: A Scalable, High-Performance Distributed File System, Symposium on Operating Systems Design & Implementation. USENIX Association (Boston, USA, December 9-11, 2002).

[8] Zhang A , Ji C , Yin Z: Security of mobile code based on redefining JVM's classloader, Computer Engineering, Vol. 4 (2006).