2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

MDDRSPF: A Model Driven Distributed Real-time Stream Processing Framework

Yijun Wen
School of Software
Tsinghua University & Naval Research Academy (NVRA)
Beijing, China
wenyj15@mails.tsinghua. edu.cn

Li Zhang
School of Software
Tsinghua University
Beijing, China
lizhang@mail.tsinghua.edu.cn

Cheng Wang
School of Software
Tsinghua University
Beijing, China
wchengnju@foxmail.com

***Abstract*—Some enterprises have demands for distributed stream processing framework to manage and process diverse business services. But their business is usually user-oriented, the access data sources and outputs are heterogeneous and need to be configurable, and the user-oriented service mode requires the platform have the ability to deploy new algorithm dynamically and provide on-demand services. As we know, there is no system that can meet all these requirements, hence a Model Driven Distributed Real-time Stream Processing Framework is proposed. By building project and task models, the functional model is separated from the implementation platform. The framework support multi-source heterogeneous data input and configurable output, on-demand services based on user's request, and algorithm dynamic loading. Service oriented architecture is deployed to lower the threshold of algorithm development and deployment. Furthermore, the functionality and performance is evaluated with a practical case to verify whether the system can meet the needs of practical application.**

Keywords- Model Driven Architecture, Distributed Real-time Stream Processing Framework, project, task

I. Introduction

Stream processing system has been widely used in many scenarios, such as health monitoring, fault detection, social network, and log processing [1][2]. At present, many machinery construction devices have installed intelligent terminal. The physical parameters collected by internal sensors are sent back to the data center. Stream Processing Frameworks (hereafter called SPFs) are the solutions of management for the distributed applications [1][3][4][5].

Some Small and Micro Enterprises (hereafter called SMEs) engaging in product manufacturing or service have the requirements for distributed platform to manage diverse business services and deploy various distributed applications. However, their technology development focus on the function of equipment monitoring and prognostic health management [6], and don't pay attention to the development of distributed processing system. They expects to have a general stream processing framework with the business model and running platform separated, which can run algorithms themselves on distributed platform and don't need to solve the workload scheduling problem [5], the out-of-order problem [7][8][9], and the sliding window problem [10] in developing the distributed real-time stream processing system.

In addition, the requirements of these SMEs are variable and diverse. For example, in industrial internet, the data structure and data type are complex and diverse, stream processing platform is required to allow the access to multiple heterogeneous data sources and being configurable for multiple results push [11]. With the development of industry 4.0, it is also a trend to provide customized services for users. The platform should have the ability to provide on-demand services based on user's request. Since the change of user's demand would occur at any time, enterprises need to develop new algorithms constantly, and the new algorithm need to be deployed on platform dynamically without affecting the running business.

As we know, there is no distributed stream processing framework could meet all these above-mentioned requirements. So a Model Driven Distributed Real-time Stream Processing Framework (hereafter called MDDRSPF) is proposed. By building the models of project and task, the functional model is separated from implementation platform. The MDDRSPF is able to support configurable multi heterogeneous data sources access and output, provide on-demand services, and deploy new algorithm dynamically without affecting the running business. In addition service oriented architecture is deployed to lower the threshold of algorithm development. The main contributions are as follows.

1) A MDDRSPF is proposed, which realized the separation of business model and running platform in stream processing field. And the advantages of this separation are that enterprises can deploy their algorithms dynamically, run the business logic algorithms themselves on distributed stream processing platform and don't need to care about the implementation details of the platform.

2) The framework adopts Service Oriented Architecture (SOA), encapsulates platform independent computing model into service set through interface definition, which is convenient for developer, and greatly enhances the flexibility.

II. Related Work

At present there are many open source SPFs that can satisfy the need for stream data processing scenarios. And several SPFs are widely used, such as Apache Storm [12], Apache Flink [13], Apache Spark Streaming [14][15], Apache Samza [16][17], Apache Apex [18] and Google Cloud DataFlow [19]. These SPFs greatly promote the application of streaming data processing.

A. Stream Data Processing Frameworks Based on SPFs

There are many frameworks base on SPFs have been developed to cater to the application needs of enterprises. For example, T. Sirisakdiwan [11] proposed a spark streaming framework for multiple heterogeneous data stream processing. The framework allows multiple heterogeneous data stream processing being deployed in a single spark application; can reduce coding redundancy and the deployment workloads, and solve the problem of inefficient job queueing in multi-stream applications. J. Dhaouadi [20] introduced an architecture based on the open source data stream processing frameworks. The architecture provides convenient way for better management of customer services and faster adaptation to different types of events. The study makes it more convenient to compute accurate results with streaming processing where events are of different types and manners. The researcher evaluated the proposed architecture on apache flink and the results were promising. W. A. Tanujaya [21] thought that developers have to write a lot of codes in developing stream processing applications even for a simple functionality. So they presented a configurable data stream application framework, in which a Domain Specific Language (DSL) for defining and configuring the data stream application was introduced. Most configurations related to these functionalities can be easily translated with the DSL without the need to develop the code. Then the researcher used two case to demonstrate that the framework can help developers with reducing the amount of codes and developing data stream processing applications more rapidly. In addition, some other studies have solved the problem respectively about distributed count-based sliding window [22], elastic streaming processing engine [23], performance analysis of message queuing integration [24] and so on. However, these research have solved parts of the requirements of enterprise, there are no package of solutions and general framework. When enterprises make use of these technologies, they need to integrate according to their own needs and do a lot of development work.

B. Model Driven Architecture in SPFs

Model Driven Architecture (MDA) is developed by OMG to separate business function model from the specific platform implementation model. MDA emphasizes portability, interoperability and reusability. The system is constructed by establishing Platform Independent Model (PIM) and Platform Specific Model (PSM). PIM describes the functions and behaviors of the system independent of the specific implementation platform technology, and PSM includes the implementation details related to the specific implementation platform. MDA form multiple mapping rules for different platforms, and then transforms PIM into PSM through mapping rules and auxiliary tools [25].

There are many researches on the application of MDA, but the research for MDA in SPFs is quite few, Atanasovski [26] used MDA to implement an e-health system to improve development efficiency and decrease the development costs. Cang-hong Jin [28] introduced a model driven framework for processing massive mobile stream data named FastFlow, which support complex operators, heterogeneous outputs, extensible computing model, and real-time algorithm deployment. Furthermore, FastFlow includes optimizers to reorganize the execution topology for batch query to reduce resource cost rather than executing each query independently. However, FastFlow is aimed at SQL-like queries, and provides single source queries only, which can't support the algorithm developed by the enterprises themselves. Hence there are no distributed stream data processing system or other commercial products that can meet all the proposed requirements of introduction, and no related papers were found.

III. Design and Implementation of MDDRSPF

A. Overall Framework of MDDRSPF

The Overall framework of MDDRSPF is demonstrated in Figure 1, which is divided into two parts, design-time (grey part) and run-time (yellow part).

- Design-time

The design-time is responsible for the modeling, customization and management of the model, which includes model customization management module and data source module. Model customization management module provides users with the ability of flexible customization and management of models. Data source module provides users with data source configuration interfaces, so that users can customize and maintain data sources conveniently.

- Run-time

The run-time is responsible for the model loading, data loading, and system operation, which includes real-time processing management engine, open source SPFs, message queue, universal application framework, storage platform and other functional modules. Real-time processing management engine is responsible for submitting and publishing the model of design-time to SPFs, while maintaining the synchronization of model data. When users customize the model or modify the model, the model data will be sent to the real-time processing management engine, and the real-time processing management engine will synchronize the changed data to each computing node. Universal application framework is a program that runs real-time stream processing tasks, filters and computing the data and pushes the calculation results, which obtains model data through real-time processing management engine when initializing.

The above design-time and run-time definitions could clearly define the development and sales mode in enterprises through project to task, and effectively standardize the development process of real-time stream processing. The system adopts service oriented architecture (SOA), and encapsulates platform independent computing model into service set. For model developer, it provides visualization user-interface for customizing mode, and algorithm template for developing algorithms. For real-time stream processing service vendors, it provides API interfaces for task management, so that the users can determine the status of tasks, such as suspended status, running status and invalid status.

B. Real-time Stream Processing Model

The main idea of model-driven method is to abstract the platform independent model that can describe the business function and is regardless of the implementation platform.

- Project

A project is a general stream processing functional model to describe a class of business. The project is defined in P(Project_ID, List_Of_Windows, Config, Output, Algorithms). Project_ID is the project unique ID, List_Of_Windows refers to the window list, which is a collection of all windows and

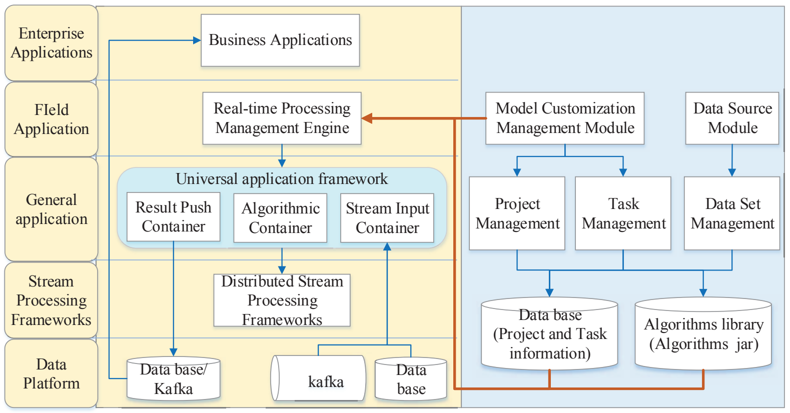

Figure 1. Overall framework of our proposed MDDRSPF, which is divided into two parts, design-time (grey part) and run-time (yellow part). Design time is responsible for the design, customization and management of the model, while run-time is responsible for the model loading, data loading, and system operation

data cache required by the project. Config refers to the input configuration source of the project, Output refers to the routing that the results are pushed, Algorithm refers to the real-time processing algorithm. Project status machine is shown in Figure 2. Developers can create and edit projects, modify the attribute of the project.

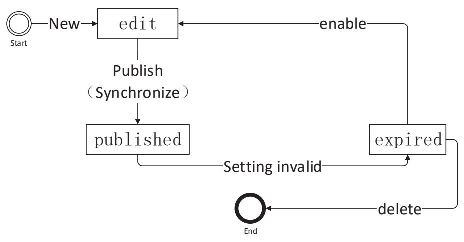

Figure 2. Project status machine

- Task

Project model does not contain device information, but monitoring data usually comes from a device. Task is a stream processing service instance that can run on a specific SPFs, and it inherit from project by binding with device, task configuration parameters and other information.

Tasks is defined as T(TaskId, Task_Config, List_Of_Device, start_time, end_time). TaskId is the task unique ID, List_Of_Device refers to the device list bound with the task, TaskConfig refers to the configuration parameters, start_time and end_time refer to the validity periods of the task. The task is subscribed by users.

The task status machine is shown in Figure 3, the status include edit status, suspended status, running status and invalid status. When the task is created, it is in editing status. After the task has been defined, it can be published and the task turns into the suspended status. The suspended status indicates that the task has taken effect and starts to access stream data or cache data. However, if the subscribers does not subscribe the task and the real-time processing is not actually executed. When subscribers request the service, the task turns into running status, triggers computing and pushes result to the subscribers. When the task exceeds its validity period, it turns into the invalid status.

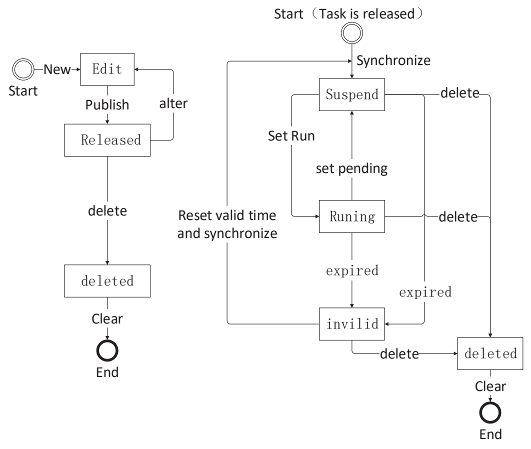

Figure 3. Task status machine

C. *Model Customization Role and Process*

The roles involved in development and customization process are enterprise model developer and data subscription consumer.

- enterprise model developer

Figure 4 shows the process of creating a project. What the project developer needs to do when creating a project is to fill in the project name, set the size of the cache window, select the monitoring conditions, configuration parameters, buffer area, interim-status, output route, and the real-time calculation algorithm. The development of project is mainly triggered by the external “business requirement”.

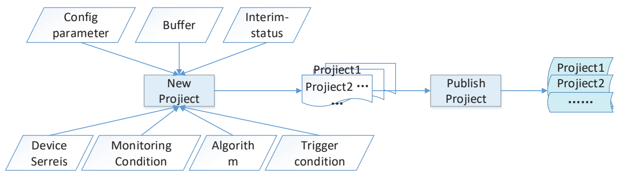

Figure 4. Project creation process

- Data subscription consumer

The data subscription consumer is a client of the framework, who can create and publish the real-time processing task based on the project. Customizing a task includes selecting projects, selecting devices, setting the start time and end time of the task, and publishing the task. After user submits the subscription, the task takes effect and the system automatically add the new task to the active task list.

With the project-task management mode, the general processing logic which can satisfy the same requirement in the similar process can be developed. For example, in order to monitor whether the vehicles are overloaded, most of them have requirement of load calculation. After the load calculation algorithm is developed and the project is created, real-time processing tasks can be created for each vehicle by reusing the load calculation project. According to actual application scenarios of customers, the start time, end time and computing resources can be customized individually. Task creating process is shown in Figure 5.

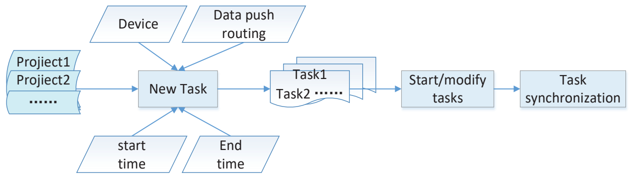

Figure 5. Task creation process

After the task is established, the calculation is triggered by data packet and task status. When a data packet arrives and the task instance is in running status, a task computing is automatically executed.

D. Data Structure and Synchronization Method

We have defined the model of project and task, and described the customization process of the model, but the customized model data is only static resource. How to load and synchronize model resource to distributed processing platform is the problem in the running process.

1) Memory management

a) Resource manager

Model data is stored and managed through resource manager, which is divided into central resource manager and local resource manager. The central resource manager is embedded in real-time processing management engine, and all user-modified information about the model is immediately synchronized to the central resource manager. The local resource managers are distributed in each physical node of the SPFs.

b) Internal data structure of resource manager

Model data is organized in the form of Map in resource manager, and can be quickly retrieved by key value. There are three kinds of maps maintained in resource manager, project Map, task Map and device-task relationship Map.

- Project map

Project list is expressed as Map <Project ID, Project object>, in which Project ID is the key of Map in order to quickly retrieve project object. The project object model is <Project ID, Project Name, Input List, Status List, Push Way List, Algorithmic Name, Configuration List>.

- Task map

Task list is represented as Map<Task ID, Task object>. Task ID is used as the key to query task object quickly. Task object model is <Task ID, Task Name, Project ID, Device ID, Start Time, End Time, Configuration List>.

- Device-task relationship map

DeviceToTask, a device-task association list, is denoted as Map <device ID, task ID>. Task ID is retrieved by device ID. Device-task relationship Map is an inverted index of task object. It can quickly query the corresponding task model through device ID.

2) Model loading and synchronizing method

In order to reduce the load of network, the local resource managers cache the required model resources, so the local resource manager needs to synchronize with the central resource manager. The synchronization process is as follows.

a) Central resource manager initialization

When the central resource manager initializes, all model resources are fetched from the model customization management module, and stored in memory, waiting for the query of the local resource manager.

b) Local resource manager initialization

Local resource managers are one-to-one with physical nodes, and the physical nodes cache the model resource on local. The local resource managers send the physical node IP address to the central resource manager, the central resource manager adds the new IP address to the local resource managers IP list, the central resource manager sends resources to local resource managers, and the local resource manager caches the required model resources locally.

c) Local resource query

The local physical nodes send resource query request to the local resource manager; The local resource managers retrieve the request resource in the local cache, and directly return the resources if the resources are found. If the resources are not cached locally, a query request is sent to the central resource manager.

d) Model modification synchronization process

There are many cases of model modification, such as modifying model information, or task expiration. All model modifications trigger the following synchronization process. When model is modified, the modification information will be submitted to the central resource manager immediately, the central resource manager checks the IP list of local resource managers and broadcasts model modification information to all local resource managers. Then local resource managers check whether the resources are cached locally and update the resources.

IV. EVALUATION OF A PRACTICAL CASE

In this section, we will evaluate the functionality and performance of MDDRSPF with a practical case, which has been applied in a logistics vehicle data service provider.

A. *Evaluation Conditions*

The MDDRSPF is evaluated on dell servers cluster, which have installed Ubuntu 14.2 and JDK 8. The physical parameter is shown in Table 1. The Apache Storm 1.1.3 is used as the Distributed Stream Processing Frameworks and the Kafka 0.11 is used as data queue in Fig. 1.

TABLE I. PHYSICAL SERVER CONFIGURATION

CPU	cores	memory	disk size
2.4GHz	28*2	128G	4T

B. *Data Set*

The data set is the actual logistics vehicle historical data of the logistics vehicle data service provider. The total data package number is 100,156,557 which belong to 20,000 logistics vehicles, and every package contains about 40 sensor ID. All the data packages are stored in Kafka 0.11.

C. *Evaluation Method and Results*

1)Model definition at design-time

The customization and management of model and data source in design-time were realized on the DWF platform (see the link for the original Tsinghua University MRO platform, http://mro.thss.tsinghua.edu.cn/). DWF platform is a basic framework for developing distributed information management system, which provides a series of modeling means and extensible interfaces. Through these modeling tools and interfaces, developers can develop application system by model customization. Then the customized model can be transformed into data model recognized by the running platform.

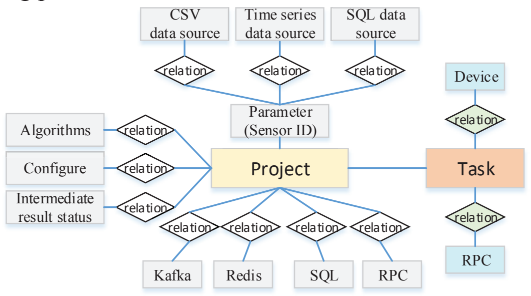

Figure 6. Model definition of the case

The model of the case obtained through the DWF is shown in Figure 6. The whole model takes the project and task as the kernel. The project is bound with the sensor parameter, the intermediate result, the processing algorithm and the configuration parameters. And there are three kinds of data sources, time series data source, CSV data source. SQL data source. The task inherited from project mainly binds the device list.

2) Convenience for algorithm development

MDDRSPF is convenient for algorithm development. In developing algorithm, developers do not need to have the professional knowledge of distributed parallel computing, and just write the algorithm according to the given interface. The algorithm development interface is defined as follows.

```
run(String deviceId, String workStatusId, String value, Long
timestamp,
    Map<String, Map<String, Map<Long, String>>> caches,
    Map<String, Map<String, Map<Long, String>>> middles,
    Map<String, String> configure)

    /** Algorithmic logic implemented by users.*/

    return Tuple(list<<String, String>> push, Map<String, <String,
String, String, Map<Long, String>>> update)
```

In the definition of interface, deviceId refers to device ID, workStatusId refers to sensor ID for triggering calculation, value is the actual working value of the sensor, timestamp refers to time point of triggering calculation, map caches refers to cache window, and the Map middles refers to intermediate result cache window.

3) System running evaluation

a) Multi-source heterogeneous input and output

When customizing the model, the project can bind the input data sources flexibly, as well as the output sources. The relationship between these sources and project is association table as shown in Figure 6. Therefore, the framework can flexibly support multi-source heterogeneous data input and configurable output.

b) Service response time of on-demand service

An important feature of MDDRSPF is to provide on-demand service, which performs real-time processing according to the user's request.

In the status machine of task model, the status include edit status, suspended status, running status and invalid status. When user sets the task to suspended status through the interface, the task will not be executed. When the user sets the task in running status, the task can be performed immediately. We design a task named tasktest1, and modify the task status by the interface. For example, when the central resource manager receives the tasktest1 status change instruction, we record the time as t1. When the local resource manager gets the modified status, and new data package triggers new computing, we record the time as t2. So we can get Δt = t2-t1, which is customer service response time. The Service response time experiment procedure is as follows.

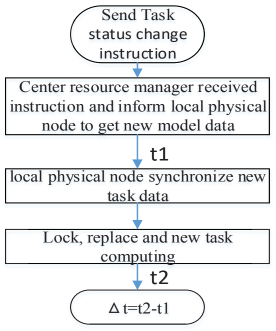
Send Task status change instruction
Center resource manager received instruction and inform local physical node to get new model data
t1
local physical node synchronize new task data
Lock, replace and new task computing
t2
Δt=t2-t1

Figure 7. The Service response time experiment procedure

The Evaluation results is as follow.

TABLE II. EVALUATION OF CUSTOMER SERVICE RESPONSE TIME

number of times	1	2	3	4	5	6	7	8	9	10
Δt=t2-t1 (ms)	287	70	188	64	248	218	87	191	475	236
Average Δt (ms)	206.4									

As can be seen in table II, the average response time of on-demand service is 206ms. Let's analyze the entire process of the task after receiving the instructions by figure8. After the task status is modified, the real-time processing engine needs to resynchronize the modified task model data to the storm distributed node, so most of the time is spent on synchronization of tasks. At present, customer service response time and synchronization method is not ideal, which is needed to be improved.

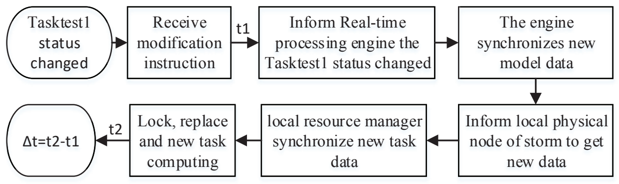
Tasktest1 status changed
Receive modification instruction
t1
Inform Real-time processing engine the Tasktest1 status changed
The engine synchronizes new model data
Inform local physical node of storm to get new data
local resource manager synchronize new task data
Lock, replace and new task computing
t2
Δt=t2-t1

Figure 8. Execution process after the task status is modified

c) Algorithm dynamic loading

Another feature of MDDRSPF is to provide algorithm dynamic loading, and the original algorithm can be updated or the new algorithm can be deployed dynamically without restarting the running topology.

Algorithm update and new algorithm deployment are the same as the task status modifying. First, set the original task to invalid state, second modify the algorithm, and then set the task to running status. Therefore, the performance of algorithm update is the same as customer service response time which depends on the efficiency of synchronization. But the performance demand of algorithm update and new algorithm deployment should be lower than on-demand service.

d) performance evaluation of MDDRSPF

Experiments were carried out to evaluated the average delay of stream data passing through the framework and the throughput of the MDDRSPF under different hardware configuration.

- Delay evaluation

The delay refers to the time difference between accessing stream data into the topology and outputting results from topology. The test method is shown in Figure 9.The real-time processing algorithm is the data packet decoding algorithm which divide the package into different working conditions value according to the ID of the sensor. Local timestamps are obtained as Figure 8.

- ✧ Time1: The time when it flows out from kafka1;
- ✧ Time2: The time when it flows into kafka2;
- ✧ System delay: time2-time1.

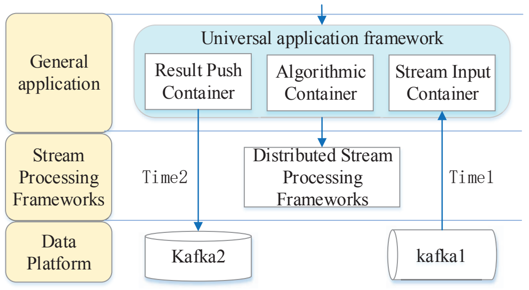

Figure 9. Delay Evaluation Method

Delayed experiment results is as follow:

TABLE III. TIME DELAY FROM TABLE

No	Input Packet Rate	Max Delay (ms)	Min Delay (ms)	Average Delay (ms)
1.	10 packs / sec	302	12	23.28
2.	100 packs / sec	771	11	25.89
3.	1000 packs / sec	555	9	33.68

We can observe that the time delay from the exit of kafka1 to the entry of kafka2 is around 30ms. Since the number of sensor ID in each packet and the complexity of decoding the package are different, the difference between the maximum and the minimum delay is large, But this is the normal delay phenomenon for the storm in processing the decoding algorithm. It shows that the model driven framework does not lead to poor latency performance.

- Throughput evaluation

We use the average decoding rate to represent throughput, The MDDRSPF pulls the original data package from kafka1, and then push the parsed data into kafka2. We get the aggregate inflow rate of the data packet at the kafka2, which is the actual throughput of the MDDRSPF. The test method is also as Figure8.Throughput experiment results is as follow.

TABLE IV. THE RELATIONSHIP BETWEEN THE NUMBER OF SERVERS AND PARSING PERFORMANCE

ID	Server Number	Average Decoding Rate (packs / sec)
1.	1	10693.15
2.	2	20345.2
3.	4	41911.7
4.	8	76937.5
5.	10	85309.8
6.	12	106450.3

We can observe that the relationship between decoding performance and the number of servers is nearly linear, and twelve servers make up a cluster with a parsing speed of 106,450 packages per second. which can satisfy the requirement of logistics vehicle data service provider for the real-time stream processing system. It also shows that the MDDRSPF can dynamically adjust the computing resources according to the sale demand of users' businesses and meet the elastic computing requirement of enterprises at different seasons.

The performance evaluation results show that the MDDRSPF does not cause performance loss compared with running the algorithm directly on SPFS, but it brings the convenience of development and powerful practicality.

V. Conclusion

In this paper, a model driven distributed real-time stream processing framework is proposed. By establishing project and task models, the functional model is separated from the implementation platform, The Framework is able to provide on-demand services and algorithm dynamic loading. Service oriented architecture is deployed to lower the threshold of algorithm development and deployment. Furthermore, the functionality and performance of the framework is evaluated with a practical case to verify if the system can meet the needs of practical application.

In the future, we will improve the synchronization method to decrease service response time, which is important to on-demand service and algorithm dynamic loading.

Acknowledgment

This work was supported by the National Key R & D Program of China (grant No: 2017YFC0804307).

References

[1] Yuanzi Xu. Research on Valuable Event Recognition in Web Data Integration. Shandong: School of Computer Science and Technology, Shandong University, 2014.

[2] Poormohammady E , Reelfs J H , Stoffers M , et al. Dynamic algorithm selection for the logic of tasks in IoT stream processing systems[C]// International Conference on Network & Service Management. IEEE Computer Society, 2017.

[3] He M, He D. A Deep Learning Based Approach for Bearing Fault Diagnosis[J]. IEEE Transactions on Industry Applications, 2017:1-1.

[4] Rafael Orozco, Shuangwen Sheng , Caleb Phillips. Diagnostic Models for Wind Turbine Gearbox Components Using SCADA Time Series Data[C]//IEEE International Conference on Prognostics and Health Management (ICPHM). 2018:1-9.

[5] Li T , Xu Z , Tang J , et al. Model-Free Control for Distributed Stream Data Processing using Deep Reinforcement Learning[J]. Proceedings of the VLDB Endowment. 2018 Vol 11: 705-718.

[6] Y. Wen, L. Zhang, H. Zhu, K. Sun and Y. Tao, "Fault Pattern Mining Based on Fault Information Event Model," 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 2019, pp. 1361-1366.

[7] Mutschler C, Philippsen M. Reliable speculative processing of out-of-order event streams in generic publish/subscribe middlewares[C] //Proceedings of the 7th ACM international conference on Distributed event-based systems. ACM, 2013: 147-158.

[8] Mutschler C, Philippsen M. Adaptive Speculative Processing of Out-of-Order Event Streams[J]. ACM Transactions on Internet Technology, 2014, 14(1):1-24.

[9] Wang J, Wang T, Cheng L, et al. An Efficient Complex Event Processing Algorithm based on INFA-HTS for Out-of-order RFID Event Streams[J]. KSII Transactions on Internet & Information Systems, 2016, 10(9).

[10] Tschumitschew K, Klawonn F. Effects of drift and noise on the optimal sliding window size for data stream regression models[J]. Communications in Statistics, 2016, 46(10):5109-5132.

[11] T. Sirisakdiwan and N. Nupairoj, "Spark Framework for Real-Time Analytic of Multiple Heterogeneous Data Streams," 2019 2nd International Conference on Communication Engineering and Technology (ICCET), Nagoya, Japan, 2019, pp. 1-5.

[12] Storm - distributed and fault-tolerant realtime computation. [Online]. Available: http://storm.incubator.apache.org/

[13] Carbone, Paris & Katsifodimos, Asterios & Kth, † & Sweden, Sics & Ewen, Stephan & Markl, Volker & Haridi, Seif & Tzoumas, Kostas. (2015). Apache Flink™: Stream and Batch Processing in a Single Engine. IEEE Data Engineering Bulletin. 38.

[14] Zaharia M, Xin R S, Wendell P, et al. Apache Spark: a unified engine for big data processing[J]. Communications of the Acm, 2016, 59(11):56-65.

[15] Emmanuel Boachie, Chunlin Li. Big Data Processing with Apache Spark in Tertiary Institutions: Spark Streaming[J]. Journal of Information Engineering and Applications, 2017:2224-5782.

[16] Somasundaram N. Apache Samza - A Stream Processing Framework [J]. 2014.

[17] Riccomini C. How LinkedIn Uses Apache Samza[J]. Discusses.

[18] Introduction to Real-Time Processing in Apache Apex. [Online]. Available: www.ijrat.org

[19] Krishnan S P T, Gonzalez J L U. Google Cloud Dataflow[J]. 2015.

[20] J. Dhaouadi and M. Aktas, "On the Data Stream Processing Frameworks: A Case Study," 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sarajevo, 2018, pp. 104-109.

[21] W. A. Tanujaya, M. Z. C. Candra and S. Akbar, "Rapid data stream application development framework," 2017 International Conference on Data and Software Engineering (ICoDSE), Palembang, 2017, pp. 1-6

[22] H. Chen, Y. Wang, Y. Wang and X. Ma, "GDSW: A General Framework for Distributed Sliding Window over Data Streams," 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, 2016, pp. 729-736.

[23] X. Chen, H. Chen, N. Zhang and J. Huang, "Elastic Streaming Semantic Engine for Web of Things," 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), Singapore, 2015, pp. 305-308.

[24] A. Ichinose, A. Takefusa, H. Nakada and M. Oguchi, "A study of a video analysis framework using Kafka and spark streaming," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 2396-2401.

[25] MDA - The ARCHITECTURE OF CHOICE FOR A CHANGING WORLD [Online]. Available: https://www.omg.org/mda/.

[26] Atanasovski B , Bogdanovic M , Velinov G , et al. On defining a model driven architecture for an enterprise e-health system[J]. Enterprise Information Systems, 2018, 12(8-9):915-941.

[27] Shukla A, Simmhan Y . Model-driven scheduling for distributed stream processing systems[J]. Journal of Parallel and Distributed Computing, 2018:S0743731518300686.

[28] Cang-Hong J , Ze-Min L , Ming-Hui W , et al. FastFlow: Efficient Scalable Model-Driven Framework for Processing Massive Mobile Stream Data[J]. Mobile Information Systems, 2015, 2015:1-18.