TaskRouter: A Newly Designed Online Data Processing Framework

Minhao Gu, Kejun Zhu, Fei Li, Wei Shen

***Abstract*—TaskRouter is a newly designed framework for distributed computing. It can be used to develop online processing system for High Energy Physics experiments. The framework takes the responsibility of data transmission. Users can determine how data being processed and routed on each node by implementing a single callback interface. One or more backup slaves can be configured for critical nodes in TaskRouter system. The processing procedure can recover from any single-point failure without data loss. TaskRouter is flexible, easy to use and with high availablity. This paper presents the core design and implementation of TaskRouter. For funtion and performance test, a dummy online prcessing software based on TaskRouter is developed.**

***Index Terms*—TaskRouter, online processing, distributed computing, error recovery, data acquisition.**

I. Introduction

ONLINE data processing is part of general data acquisition (DAQ) software of high energy experiments. It is between electronics readout and event storage components, doing processing tasks, such as data collection, software trigger, data quality monitoring, etc. Since servers and network hardware are becoming more and more powerful, more computational tasks can be done online. In a typical high energy physics experiment, For example JUNO, the online processing tasks include waveform compression, software trigger, and merging events from different type of detectors[1].

Challenges of developing an online processing framework include

- Complicated processing procedure
- High throughput of event data
- Making full use of the CPU power of clusters
- Fault-tolerant issues

TaskRouter is a flow based, distributed processing framework. This paper introduces the objective, design, implementation of an online processing prototype based on TaskRouter, and gives some preliminary performance test results.

II. Objective of TaskRouter

Many good online processing frameworks already exist, such as Atlas TDAQ[2] framework. And we has successfully developed BESIII[3] and DayaBay[4] online processing software based on TDAQ. TDAQ is powerful but too complicate to use and hard to maintain, especially in small-scaled experiments. So we plan to develop a light weighted, easy to use framework, called TaskRouter.

Minhao Gu, Kejun Zhu, Fei Li, Wei Shen are with IHEP, China

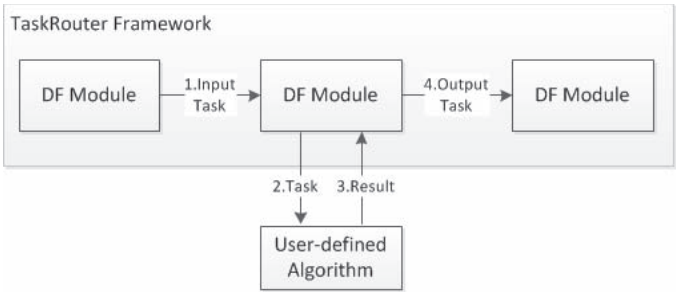

Fig. 1. Basic function of TaskRouter

When designing a flow based online processing system, the following issues need to be considered:

1 Data transmission between DF modules
2 Deployment and monitoring of DF modules
3 Load balance when using distributed architecture
4 Failures detection and recovery
5 Design of processing procedure
6 Implementation of processing algorithm...

TaskRouter intend to solve the issue 1-4 above. Users can focus on design of data flow and implementation of processing algorithms.

There is no critical node in TaskRouter based system. All single point failures, hardware or software, could recover automatically with no loss of event data.

III. Design and Implementation

TaskRouter design is organized into two layers: the transport layer and logical layer (Figure 2).

- The transport layer manages TCP connections, providing methods for data output and a synchronous callback interface for data input. The upper layer can register a callback to process a specific input data type. Usual transport exception handling, such as auto reconnecting, is also implemented in this layer.
- The logical layer contain two main components: the Task Process(TP) and Error Recover(ER) components. The TP component is responsible for data buffering, processing and releasing. The ER component monitors the status of neighboring DF modules using the heartbeat method, executing auto error recovery logic when some DF module dies.

More detailed descriptions will be given in the following chapters. The basic deployment of TaskRouter is simple. Only one background process called Processing Unit(PU) is required running on each compute node.

978-1-5090-2014-0/16/$31.00 ©2016 IEEE

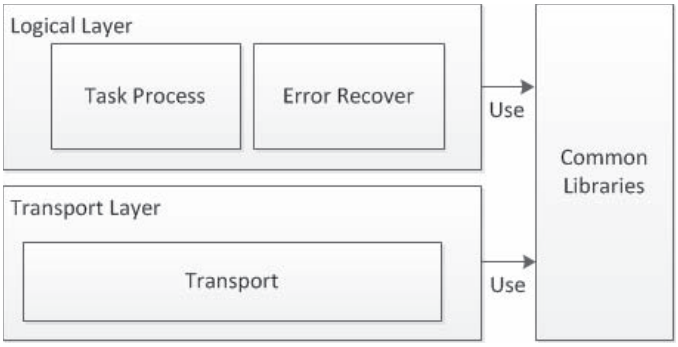

Fig. 2. TaskRouter layers and components

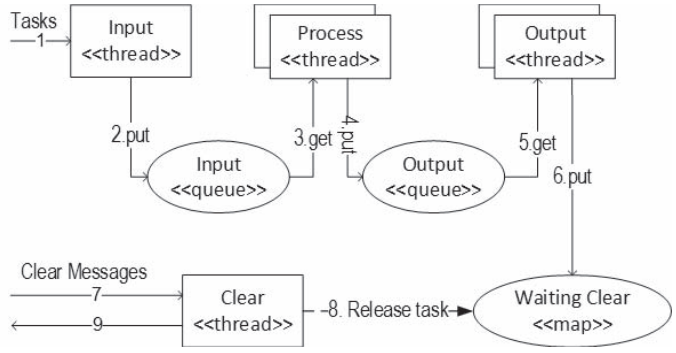

Fig. 3. Working threads in the Task Process component

A. Transport Component

Data fragments are transfer over TCP/IP protocol. The transport components are initialized when PU starts, waiting for connections from other PUs. Deployment information (Host addresses and ports of other PUs) is also collected at initial time. Before data sending, the current PU checks a connection pool for an active connection to the target PU. If no active connection is available, it will start a new connection. Both current and target PUs will use this connection for data transmission.

Connection and data sending timeouts are also checked by the transport component. Exceptions are thrown to the logical layer, when timeout events occur.

B. Task Process Component

Data Source is a program, generating source tasks to TaskRouter system. Each task is tagged by Type ID (TID) and Indexed ID (IID). Tasks with the same TID has the same data structure. IID is auto increasing, generated in Data Source. Tasks tagged by the same IID are computation results of the same source task.

This component is designed using producer/consumer model. Several kinds of threads together achieve the processing procedure, including input, processing, new task creation and output (Figure 3). The Input Thread receive task data and put them into the Input Queue. The Process Threads get data from Input Queue, call processing algorithms, create new tasks for next-level processing, and finally put them into the Output Queue; The Output Threads finally route tasks out to other PUs.

The number of Process Threads determines how many tasks can be processed on a node in parallel. It could be set according to node's CPU core count and memory size. The number of Output Threads can be adjusted in order to obtain best transmission performance, typically 2 to 4.

Task data are not released until the whole processing procedure has been done. Any PU can generate clear messages. For example in DAQ system, processing is always end with event storage. When an event has been successfully recorded to a data file, the storage PN will release the task data(event) and send a clear message to the previous PN. Clear messages are transferred to previous level of PUs following the processing path. Tasks with the same IID are released by all related PUs.

The framework provides an interface for development of processing algorithms. The algorithms are built into a dynamic library, loaded by the framework at runtime. Framework users can determine: 1. The processing logic; 2. Whether to release or route out the result data; 3. Target PNs of new tasks

C. PU Classification and Error Recovery Component

TaskRouter has two kinds of PUs: Dispatch and Compute. A Dispatch PU is used when a type of tasks cannot be processed on different nodes. Compute PUs can be deployed on several nodes processing a type of tasks in parallel. For example in JUNO experiments, event merge and sort from 3 different detectors[1] could be done on a single node, so a Dispatch PU could be used; As for the event process modules responsible for software trigger and data quality monitoring[1], events can be analyzed in parallel. So, compute PUs could be used.

Ping message is sent periodically between neighboring PUs. Failure of a PU is discovered when the response ping message does not arrived within a timeout cycle. After detection of PU failure, previous-level PUs will check tasks in the Waiting Clear buffer. Tasks assigned to the dead PU will be put back into the Output Queue, waiting for another assignment. Error recover schemes are different for the 2 PU types.

- For Compute PUs, several nodes are doing the same job. Failure of any PU does not affect the whole data flow. After error detection, the dead PU will be disabled and all failed tasks will be re-assigned.
- For Dispatch PUs, one or more backup PUs are deployed. They ping each other periodically. When the working PU dies, a backup PU with the smallest transport ID takes its place.

According to the clear logic, tasks are buffered in each level of PUs until completely processed. So no data are lost after error recovery.

D. Common Library

Several reusable functions are implemented in Common Library.

- Configuration. Configuration data are store in XML format containing deployment information. CodeSynthesis XSD[5] is used for mapping XML file to C++ classes. Framework users can define new configuration parameters in an XSD schema file, setting the parameter values

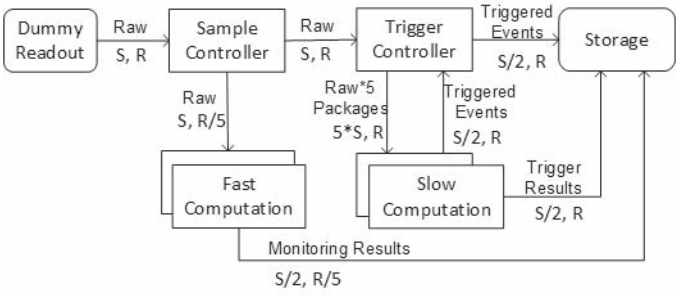

Fig. 4. Data flow of an online processing prototype

TABLE I
THROUGHPUT AND DISPATCH DELAY

		Output Data Rate (GBytes/s)	DT[2] Avg. (ms)	DT Max (ms)
Case 1	TC[1]→ SC	1.2	2.9	14.1
	SC[2]→ TC	0.12	1.6	14.8
Case 2	TC[1]→ SC	1.4	46.7	97.0
	SC[2]→ TC	0.14	5.4	22.6

[1] TC: Trigger Controller module
[2] SC: Slow Computation module
[3] DT: Task dispatch time

in a data file. Then, values of new parameters can be accessed in C++ code.
- Pre-allocated buffer management. Buffer pages of typical sizes are allocated at initial time, for example, 1 KBytes, 10 KBytes, 100 KBytes, 1000 KBytes, etc. If a 50-KByte buffer is requested, a 100-KByte buffer in the free-buffer list will be returned. If buffers of required size is not available in the free-buffer list or larger than the max pre-allocated size, A new memory page of required size will be allocated and returned.
- Shared Queue. It can be accessed by all working threads and is thread safe.

IV. PROTOTYPE ONLINE PROCESSING SOFTWARE BASED ON TASKROUTER

A prototype processing software base on the TaskRouter is developed for function and performance tests. Shown in Figure 4, each DF module is also a TaskRouter PU. Functions of these DF modules are:

- Dummy Readout, generating dummy raw packages (source tasks)
- Sample Controller, sampling some of raw packages for fast computation
- Fast Computation, processing sampling raw packages for monitoring
- Trigger Controller and Slow Computation, executing complex algorithm (software trigger, etc.) on farm. The Trigger Controller packs five successive packages for software trigger.
- Storage, collecting and storing triggered events and all computational results

Processing algorithms are implemented for PUs, generating dummy result data to feed the input of next level. Assuming the source package size is S(Bytes), and the rate is R(Hz). Data rates between DF modules are labeled in Figure 4.

V. TEST RESULTS

11 blade servers are used for deployments of the prototype software: 4 servers for the Data Source, Sample Controller, Trigger Controller and Storage modules; 1 server for Fast Computation, 4 servers for 4 Slow Computation modules; 1 server running backup PUs. The servers' hardware and system configurations are: 3.40GHz CPU with 12 cores, 48GB memory, 56Gbps IB configured with SDP network, SLC6.5 OS.

PUs are configured with 5 Process Threads and 3 Output Threads, and send ping messages very 500ms. The node alive timeout is set to 2s, which means that if ping does not reply within 2s, error recovery logic will start.

Test results of two cases with different source event sizes and rates will be shown as follows. The source event generation parameters of the two cases are: 1MB event size, 240Hz rate (case 1); 24MB event size, 12Hz rate (case 2).

A. Throughput and Task Dispatch Time

The task dispatch time of some PU is defined as $(T_{input} - T_{succ\ output}) - \Delta t_{process}$. T_{input} represent the time when an input task arrives. $T_{succ\ output}$ is the time when PU has successfully sent the result (packed into a new task) to the next level. $\Delta t_{process}$ is the execution time of processing algorithm for this task. Histograms are used for statistics of dispatch time in each PU. Figure 5 shows dispatch time histograms of Trigger Controller.

The transfer link with highest load is between Trigger Controller and Slow Computation modules. Throughput and task dispatch time of this link are listed in table I.

B. Error Recovery Function Tests

The prototype software is also used for tests of the error recovery function. We deliberately kill processes, switch off some node's power supply or unplug network cables while the software is running. The system can automatically recover, no task is lost, and the max dispatch time is less than 4.5 seconds. Figure 6 shows task dispatch times before and after killing the Trigger Controller module in case 2.

VI. SUMMARY

The core design of TaskRouter has been implemented. TaskRouer can be used for development of flow based online processing software. Framework users are only required to implement the processing interfaces, focusing more on data processing algorithms. Computation results are packed as new tasks for the next level processing. Users can determine the route of each new task.

A prototype online processing software has been developed base on the framework. Test results show that the throughput of a single PU is over 1GBytes/s and dispatch time is

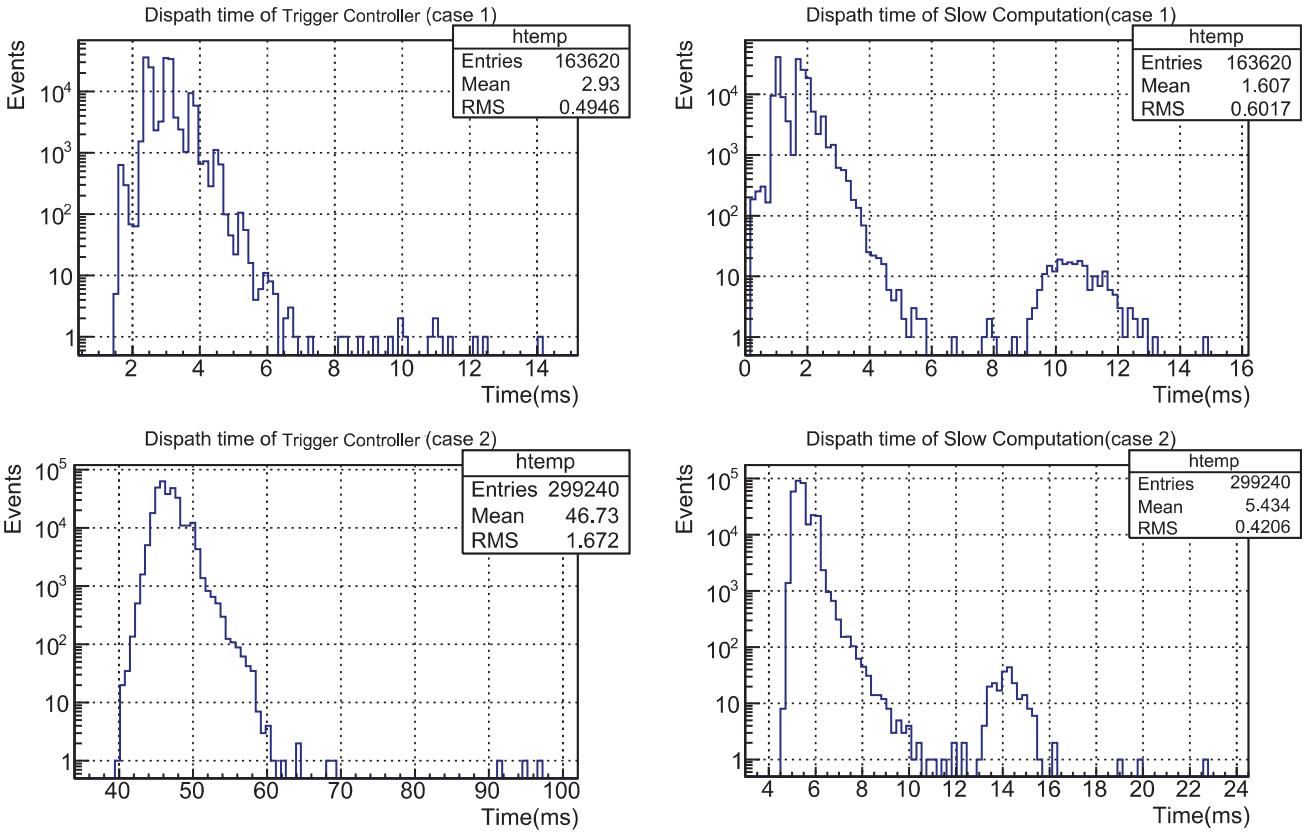

Fig. 5. Data flow of an online processing prototype

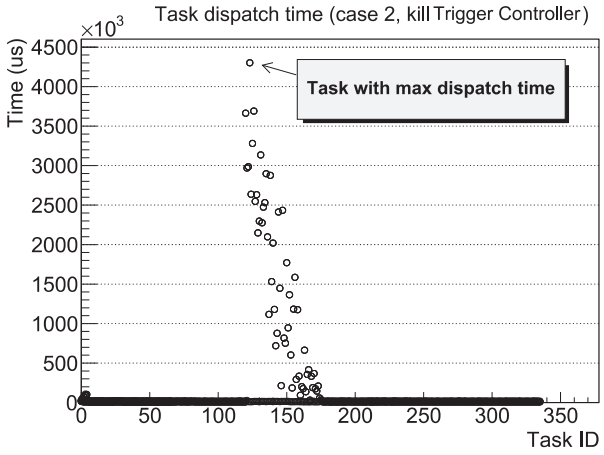

Fig. 6. Task dispatch time, before and after error recovery

controllable. There is no critical point in TaskRouter. It can automatically recovery from any single-point failures with no data lost.

TaskRouter is still under development. Many functions (such as run control, PU status monitoring, GUI, etc.) are not implemented yet. Lots of efforts are also required to improve its stability.

REFERENCES

[1] Djurcic Z. JUNO Conceptual Design Report,(2015)[J]. arXiv preprint arXiv:1508.07166.

[2] ATLAS C, kesson T, Eerola P, et al. ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report[J]. ATLAS Technical Design Reports, 2003.

[3] Li F, Zhu K, Chen L P, et al. Online data processing and analyzing in BESIII DAQ[C]//Real Time Conference, 2009. RT'09. 16th IEEE-NPSS. IEEE, 2009: 458-460. MLA

[4] Li F, Ji X, Li X, et al. DAQ architecture design of Daya Bay reactor neutrino experiment[J]. Nuclear Science, IEEE Transactions on, 2011, 58(4): 1723-1727.

[5] Code Synthesis, http://www.codesynthesis.com/projects/xsd