Modeling Stream Preprocessing with WebGME

Basic Representation of Stream Task Scheduling and Resource Allocation

Xingyu Zhou
Electrical and Engineering Department
Vanderbilt University
zhouxingyu319@gmail.com

***Abstract*—This short paper aims to present a basic model for task scheduling and resource allocation in the stream computing framework of Amazon Kinesis. WebGME (Generic Modeling Environment) is the tool for this project. A model has been constructed and an interpreter for this model has been made to realize some functions.**

Keywords—stream computing; Kinesis; task scheduling; WebGME; modeling;

I. Introduction

This short paper aims to present a basic model for task scheduling and resource allocation in the stream computing framework of Amazon Kinesis. WebGME (Generic Modeling Environment) is the tool for this little project. A model has been constructed and an interpreter for this model has been made to realize some functions.

The Generic Modeling Environment is a configurable toolkit for creating domain-specific modeling and program synthesis environments. The configuration is accomplished through metamodels specifying the modeling paradigm (modeling language) of the application domain. The modeling paradigm contains all the syntactic, semantic, and presentation information regarding the domain; which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment.[1]

WebGME is an extension to the traditional PC-based GME environment. It is a novel, web- and cloud-based, collaborative, scalable (meta)modeling tool that supports the design of Domain Specific Modeling Languages (DSML) and the creation of corresponding domain models. The unique prototypical inheritance, originally introduced by GME, is extended in WebGME to fuse metamodeling with modeling. The tool also introduces novel ways to model crosscutting concerns. These concepts are especially useful for multi-paradigm modeling. The main design drivers for WebGME have been scalability, extensibility and version control. The web-based architecture and the constraints the browser-based environment introduces provided significant challenges that WebGME has overcome with balanced trade-offs.[2]

The project topic is “Stream Task Scheduling and Resource Allocation”. This is a significant topic in the application and analysis of distributed systems.

One thing worth point out is that the topic of this paper emphasizes on the task scheduling during the preprocessing of the stream computing process. For real applications, it should not be regarded as an independent entity but should be a component for a whole stream computing system.

II. Key Concepts About Streaming

Although much has been said about the topic, there are some key concepts that may not be very familiar to all people. This part illustrates some key concepts involved in this modeling process.

A. Stream Computing

Stream computing, also known as data stream processing, has emerged as a new processing paradigm that processes incoming data streams from tremendous numbers of sensors in a real-time fashion. Data stream applications must have low latency even when the incoming data rate fluctuates wildly. This is almost impossible with a local stream computing environment because its computational resources are finite.[3]

To address this kind of problem, various kinds of tools have been developed. Such as S4[4] and Amazon Kinesis. The latter is the computation mode this paper would discuss.

B. Amazon Kinesis

Amazon Kinesis is a managed service that scales elastically for real-time processing of streaming big data.[5]

Amazon Kinesis takes in large streams of data for processing in real time. The most common Amazon Kinesis use case scenario is rapid and continuous data intake and aggregation. The type of data used in an Amazon Kinesis use case includes IT infrastructure log data, application logs, social media, market data feeds, web clickstream data, and more. Because the response time for the data intake and processing is in real time, the processing is typically lightweight.

Amazon Kinesis enables sophisticated streaming data processing, because one Amazon Kinesis application may emit Amazon Kinesis stream data into another Amazon Kinesis stream. Near-real-time aggregation of data enables processing logic that can extract complex key performance indicators and metrics from that data. For example, complex data-processing graphs can be generated by emitting data from multiple Amazon Kinesis applications to another Amazon Kinesis stream for downstream processing by a different Amazon Kinesis application.

Data records are the units of data that are stored in an Amazon Kinesis stream. Data records are composed of a sequence number, a partition key, and a data blob, which is an un-interpreted, immutable sequence of bytes. The Amazon Kinesis service does not inspect, interpret, or change the data in the blob in any way. The maximum size of a data blob (the data payload after Base64-decoding) is 50 kilobytes (KB).

A stream is an ordered sequence of data records. Each record in the stream has a sequence number that is assigned by the service. The data records in the stream are distributed into shards.

A shard is a uniquely identified group of data records in an Amazon Kinesis stream. A stream is composed of multiple shards, each of which provides a fixed unit of capacity. Each open shard can support up to 5 read transactions per second, up to a maximum total of 2 MB of data read per second. Each shard can support up to 1000 write transactions per second, up to a maximum total of 1 MB data written per second. The data capacity of your stream is a function of the number of shards that you specify for the stream. The total capacity of the stream is the sum of the capacities of its shards.

If your data rate increases, then you just add more shards to increase the size of your stream. Similarly, you can remove shards if the data rate decreases.[6]

III. Model Construction

A. General Procedure

First, the procedure of the task scheduling and resource should be determined.

Even though there could exist infinite number of data flow combinations in real networks, the basic components are the same.

One fundamental procedure for this follows the data flow from data producer to sampling buffer meanwhile getting the partition key from the partition key generator and then go to shard producer, then ordering buffer, then stream producer and finally the application.

Details involved in this procedure might be quite different. But the process itself is the same all the time.

The model made here is just a rather abstract one that will only display the process with these. The details will be hidden in the black boxes ready for a more flexible framework to realize.

B. Model of Computation

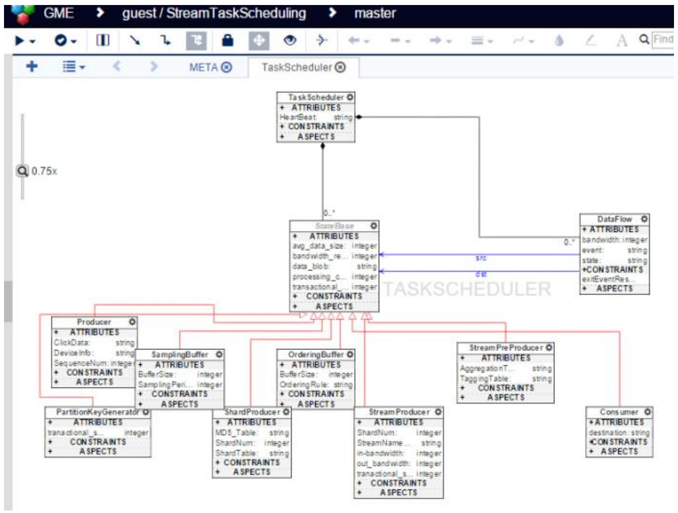
Considering the problem of modeling, the theoretical model of computation must first be pointed out to make things clear. Obviously, this stream task scheduling and resource allocation procedure involves not only the data flow but also the control flow. From the definition, it should be considered as a Petri Net, essentially an extended form of state machine.

However, to make the modeling process easier to conduct and understand, the modeling here directly uses the simpler form of state machine and regard all kinds of data flows as equivalent.

As mentioned above, this state machine has states like producer, sampling buffer, shard producer, ordering buffer and so on. Each state can be triggered by its previous state and all processes will finally lead to the applications state.

C. Metamodel

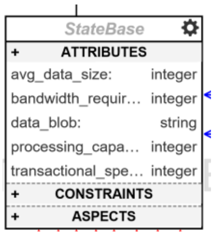
The overall metamodel is shown below.



The figure may not be clear enough to show the details. But the main idea is not complex.

All components are in a very basic metatype defined as the TaskScheduler. Every new instance of this kind of model is a kind of TaskScheduler.

All states inherit from an abstract type defined as StataBase. StateBase itself has already defined attributes of "avg_data_size","bandwidth_requirement","data_blob","processing_capacity" and "transactional_speed".



The "avg_data_size" is in the unit of KB. The "bandwidth_requirement" is in the unit of M and "data_blob" is the part that carries true information and can be denoted in the form of string. The "processing_capacity" and "transactional_speed" are both referring to the hardware capacity of the server that carry on the job.

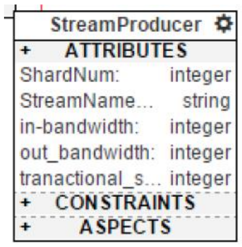
These five attributes are rather important ones in all states because they somehow indicate whether the node could be really realized. And they actually share common standards for parameter optimization.

For one specific state, it would have its own private properties different from other kinds of states.

The producer produces data records. For example, when we are trying to analyze the visiting records of a website, it will also include the information of the device the click data.

For both forms of buffers, the sampling buffer and the ordering buffer, they would of course have a buffer size limitation. The ordering buffer also involves the rule of ordering the buffer. The ordering rule attribute can be denoted as an enumeration, which can be methods of FIFO, Heap or Random.

The StreamProducer is the most complex part of this model. It must take the status of the current running shard and input shard into consideration at the same time. The "ShardNum" attribute in StreamProducer is automatically given through calculation based on the values of "in_bandwidth" and "out_bandwidth". The figure of the metamodel has been shown below.



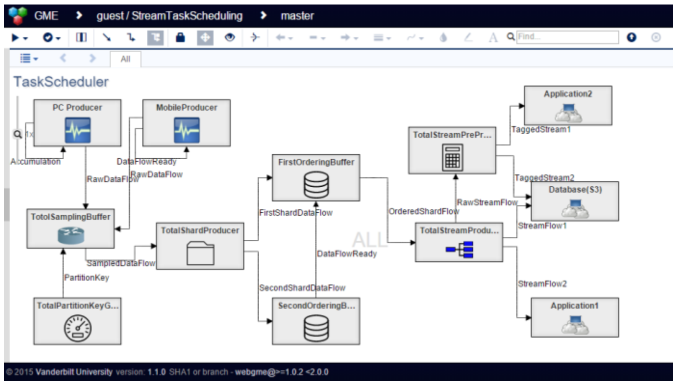
Although the scheduling idea is quite clear here, the realization of the function above is not actually the goal for generic modeling here. As a result, only an abstract model is presented here.

The final destination of the model is the application. Which executes some user defined functions based on the data flow preprocessed by the process discussed above.

The last thing I want to mention in the construction of the metamodel is the decorator. WebGME has provided a rather convenient way to change the display icons of a component.

IV. NETWORK INSTANCE

The metamodel is described as above. A simple instance model will be shown here.



The figure above shows a simple stream task scheduling and resource allocation.

The data flow simple runs from left to right.

In the model instance, there are two producers: the PC Producer and the Mobile Producer. This is catering to the website visiting record analysis which requires visitors' device information to be recorded.

The modeling here aims to simulate the network configurations of the network. For the simple reason that stream computing has issued a high demand for hardware, distributed systems must be utilized. Moreover, the ordering algorithm itself is a rather resource-consuming application. In this way, there could be more than one ordering buffer in this model.

The applications can be various kinds, including databases. The probable database destination after stream computing for Amazon Kinesis is Amazon S3 Database.

Settings of parameters in this instance model is also worth mentioning.

V. NETWORK OPTIMIZATION INTERPRETER

The metamodel and an instance model is described as above. Interpreter is a more sophisticated feature to realize more complex functions.

The interpreter has appeared since the PC-based GME version. Right now on WebGME, the interpreter is called plugin and is written in Javascript form.

The plugin here in the model is call NetworkRunner. It is used to check the configuration of the stream task scheduling network and also make some optimizations whenever possible.

There are two main types of functions that have been realized in this project. They will be discussed below in a more detailed way.

A. *Capacity Checking*

The capacity checking aims to judge whether the hardware resources can meet the demand of transactions.

The table below shows the core idea of this function. The total transactional size can be calculated with the product of transactional speed and the average data size.

We assume that the processing capacity, which is decided by hardware, should be at least 50% higher than the transactional size. Because the transactional size here is only an average value and it cannot deal with the maximum situation.

```
    var transactional_size

    = stateData.transactional_speed*stateData.avg_data_size;


    if( 1.5*transactional_size>=stateData.processing_capacity )
{
    stateData.processing_capacity=transactional_size*1.5;
    stateData.improvements.push("processing_capacity");
}
```

B. *Shard Num Decision*

This step can be considered as an optimization job. In the Amazon Kinesis, as the developer guide mentioned, the shard num should be set by the user manually, however, the modeling of preprocessing network here makes it possible to get the value automatically.

Luckily, the computation formula has been provided by Amazon.

```
number_of_shards
=max(incoming_write_bandwidth_in_KB/1000,
outgoing_read_bandwidth_in_KB/2000)
```

This formula has been used in the interpreter and after running the interpreter, the output JSON file will include the calculated results.

C. *Output Analysis*

The interpreter will output a JSON file with components and their attributes.

With the optimization algorithms implemented as discussed above, the interpreter would change the values of some attributes to make the network configuration more appropriate.

If the value of one attribute were changed, the name of the changed attribute would be logged in the output JSON file.

An example output paragraph of the StreamProducer would be like the code in the table below shows.

```
"name": "TotalStreamProducer",
"transitions": [
  {
    "event": "StreamFlow2",
    "targetId": "/1829395557/1083137865",
    "targetName": "Application1"
  },
  {
    "event": "RawStreamFlow",
    "targetId": "/1829395557/1996059387",
    "targetName": "TotalStreamPreProducer"
  },
  {
    "event": "StreamFlow1",
    "targetId": "/1829395557/125845890",
    "targetName": "Database(S3)"
  }
],
"avg_data_size": "200",
"bandwidth_requirement": 6400,
"processing_capacity": 600000,
"transactional_speed": "2000",
"attributes": [
  [
    "transactional_speed",
    "avg_data_size",
    "bandwidth_requirement",
    "name",
    "out_bandwidth",
    "ShardNum",
    "StreamNameTable",
    "in_bandwidth",
    "data_blob",
    "processing_capacity"
  ]
],
"improvements": [
  "processing_capacity",
  "bandwidth_requirement",
  {
    "shard_num": 1000
  }
]
```

One big problem is that there still exists a gap between this network configuration and the real preprocessing. The reason for this is obvious, the plugin function is a static execution process. It can only deal with the state variables at a certain time point and is not designed to run simutaneausly.

VI. Discussion

After all the discussions above, a simple stream task scheduling and resource allocating process has been created and shown.

Although this paper emphasizes on the preprocessing procedure of Amazon Kinesis. It has actually provided a generalized stream preprocessing framework design ideology for all kinds of stream computing.

There are many things can be done in the future.

First of all, to simplify the modeling process, all routes between states are set equivalent. This may cause some paradox in real applications.

In addition, more attributes could be added. The model here has only one level. It does not involve multi-hierarchical modeling. In real systems, multiple levels must be taken into consideration.

Finally, interpreters can only be used for static state analysis. It still lacks a bridge to fill the gap between the network configuration and the real scheduling.

VII. Acknowledgment

I want to thank Professor Jules White for offering me this topic and giving me much tutor.

References

[1] http://www.isis.vanderbilt.edu/Projects/gme/.

[2] http://webgme.org/WebGMEWhitePaper.pdf.

[3] Ishii A, Suzumura T. Elastic stream computing with clouds[C]//Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011:195-202.

[4] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed stream computing platform[C]//Data Mining Workshops (ICDMW), 2010 IEEE International Conference on. IEEE, 2010: 170-177.

[5] http://docs.aws.amazon.com/kinesis/latest/dev/introduction.html.

[6] http://docs.aws.amazon.com/kinesis/latest/dev/key-concepts.html