Hindawi
Mobile Information Systems
Volume 2021, Article ID 2502083, 8 pages
https://doi.org/10.1155/2021/2502083

Research Article

A Dynamic Declarative Composition Scheme for Stream Data Services

Zhongmei Zhang,[1] **Zhongguo Yang**,[2] **Sikandar Ali**,[3] **Muhammad Asshad**,[3] **and Shaher Suleman Slehat**[4]

[1]*School of Management Engineering, Shandong Jianzhu University, Jinan 250101, China*
[2]*School of Information Science and Technology, Beijing Key Laboratory on Integration and Analysis of Large-Scale Stream Data, North China University of Technology, Beijing 100144, China*
[3]*Department of Information Technology, The University of Haripur, Khyber Pakhtunkhwa, Pakistan*
[4]*Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, Australia*

Correspondence should be addressed to Sikandar Ali; sikandar@cup.edu.cn

Received 27 August 2021; Revised 5 September 2021; Accepted 30 September 2021; Published 12 October 2021

Academic Editor: Fazlullah Khan

With the fast development of Sensor Network, Internet of Things, mobile devices, and pervasive computing, enormous amounts of sensor devices are deployed in physical world. Data streams produced by these sensor devices, deployed broadly, can be used to create various value-added applications. Facing continuous, real-time, high-frequency, low-valued data streams, how to flexibly and efficiently cooperate them for creating valuable application is very crucial. In this study, we propose a service-oriented manner to realize flexible streams integration. It considers data stream produced by one sensor data as a stream data service and utilizes composing multiple services to realize the cooperation among sensor devices. Firstly, we propose a stream data service model based on Event-Condition-Action rules, which can encapsulate steam data as services and continuously and timely process stream data into value-added events. Then, we propose a declarative method which can dynamically compose stream data services. Based on two kinds of declarative rules, that is, sink-rules and connect-rules, multiple data streams can be dynamically integrated through flexible service composition. To ensure the performance of service composition, we also employ a sensor partition strategy and process multiple service compositions in parallel. Through comprehensive evaluations by experiments, our service composition method shows both good efficiency and effectiveness.

1. Introduction

Recently, the Internet of Things (IoT) technology has been developed rapidly; enormous amounts of sensor devices are deployed in physical world [1]. Sensor data produced by sensor devices is a typical kind of data stream in which data records arrive continuously and sequentially. Such data present great opportunities to create unprecedented value-added services [2, 3], for example, violations detection based on analysis of the traffic data and carpooling recommendation based on trajectory pattern. Value-added services usually need the cooperation among multiple sensor devices instead of analysing data stream produced by single sensor. Data stream has some characteristics such as continuous, real-time, high-frequency, and low-valued, and the requirements of data stream are multitudinous. Taking Automatic Number Plate Recognition (ANPR) data as an example, which is detected information for vehicle captured by camera sites deployed at major road and crosses in metropolis. Huge amounts of ANPR data can be produced in a short period of time from lots of camera sites simultaneously, especially during traffic peak times. It is a big challenge to integrate and process such amounts of data steams for extracting valuable information facing lots of possible requirements.

Recently, a simple manner to integrate multiple sensors is to transmute all the data streams into a central server where data analysis and integration is running [4]. It is easy to realize while the server can quickly become overloaded with the increase of data streams or the concurrent requests. Another manner to create value-added service is to directly cooperate among sensors with computation inner sensor and communication intersensors [5, 6]. However, this manner needs sensors to have additional capability. Nowadays, there are lots of sensor devices deployed in physical world which are only capable of collecting data, for example, camera sites deployed at roads. It is not possible or desirable to improve or replace these sensor devices, which makes the direct manners not able to be applied universally.

In this paper, to flexibly cooperate with relative sensors, we propose an alternative manner which considers data stream produced by one-sensor data as a stream data service and utilize composing multiple services to realize the cooperation among sensor devices. With our service-oriented manner, stream data service enables sensor with easily extended abilities as computing and communication in software defined manner, and the relationship between sensors can be represented by relationship between corresponding services. Since the number of sensors and user's requirements can be huge, a standalone environment may not be able to process large numbers of services and requirements. To ensure the performance of service composition, we examine a distributed framework in a cloud computing environment to cooperate among services in parallel. Specifically, the contributions of this paper are as follows:

(1) We design and realize a declarative stream data service model based on ECA (Event-Condition-Action) rules. Based on the service model and ECA rules definitions, data streams can be created as stream data services. The stream data service is similar to psychical sensor with communication and processing abilities in software-defined manner.

(2) We propose a declarative service composition method. Based on two kinds of declarative rules, we can integrate multiple data streams through flexible stream data service composition. Specifically, different services can fuse their outputs together with *sink-rules*; and certain service can trigger the starting and stopping of relevant services with *connect-rules* under specific conditions.

(3) We employ a distributed processing environment and introduce a sensor partition strategy, which divides sensors into partitions and can process service composition in parallel.

2. Motivation Scenario

The motivation example of this paper is a real-time vehicle tracking application for police department illustrated in Figure 1. The real-time vehicle tracking application can be created based on ANPR data because of its wide and continuous monitoring range. It can be utilized to locate suspicious vehicles and discover their trajectories. For example, a user can initiate the following query: "report the position of the vehicle when it is captured by any camera site."

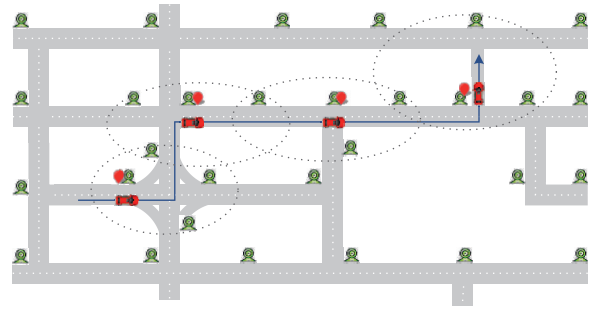

Figure 1: A real-time vehicle tracking application: Here, a vehicle is moving on the road. When it is captured by one camera site, the application will evaluate the information captured by neighboring camera sites. The information of the vehicle will be returned once it is captured by any camera site.

The application can be easily created in a centralized manner through processing data streams produced by all camera sites. Since, for one vehicle, it can only be captured by one camera site at one moment, the processing of ANPR data collected by all camera sites is a waste of resources; and the waste will be seriously increased when a number of users request the application concurrently.

Since sensors are usually correlated through their spatiotemporal relationship, the real-time vehicle tracking application can be created based on cooperation only among relevant camera sites. For camera sites deployed on road net, there are road constraints, and therefore prior knowledge of possible vehicle trajectories can be exploited. For the sake of simplicity, we focus on the sensor collaboration phase and ignore the sensor selection phase. When a user requests the trajectory *tra* for vehicle v, the steps of sensor's cooperation are as follows:

(1) Locate camera site c_1 which captured v by monitoring all camera sites, and return v's detection information.

(2) Find camera site c_{i+1} which captured v next by only monitoring the neighboring camera sites of c_i, and return the detection information.

(3) Repeat step (2) until user stops the request.

However, it is impossible to collaborate with camera sites directly. A service-oriented manner can be utilized to realize the sensor collaboration. The data stream collected by each camera site is regarded as a stream data service and the relationship of camera sites can be mapped to corresponding services. We can generate trajectory by dynamically composing relevant services.

3. Declarative Method for Service Composition

We firstly introduce some basic concepts and then introduce the stream data service model and declarative method for cooperating among multiple stream data services.

3.1. Basic Definition. Arasu et al. [7] supported that a stream is a sequence of time stamped tuples, which has an associated schema consisting of a set of named attributes, and all tuples of the stream conform to the schema. This paper regards a data stream as a sequence of data records generated by sensor devices or applications. As in the standard relational model, each relation has a fixed schema consisting of a set of named attributes. For each data record in one data stream arrivals, we assume it also has a fixed schema. To enable user better understand the data stream, the stream's representation must be enriched with metadata such as names and schemas.

Definition 1 (data stream). A data stream s can be presented as a tuple $s = <uri, A, R>$, in which *uri* is the unique identification of the stream data, $A = \{a_1, a_2, \ldots, a_m\}$ is the schema of the stream data with a set of attributes of the data record in the stream data, and $R = \{r_1, r_2, \ldots, r_i, \ldots\}$ is the sequence of data records, in which $r_i = \langle v_{i1}, v_{i2}, \ldots, v_{im} \rangle$, and v_{ik} is the value for the corresponding attribute a_k in record r_i.

Take ANPR data as an example; ANPR data can be denoted as $<ANPR, \{cid, vid, time\}, \{r_1, r_2, \ldots, r_i, \ldots\}>$, in which *ANPR* is the identification of ANPR data, {*cid*, *vid*, *time*} is the schema of ANPR data, and $\{r_1, r_2, \ldots, r_i, \ldots\}$ is the infinite data record sequence in ANPR data. Specifically, *cid* is the identification of camera site, *vid* is the plate number of the captured vehicle, and *time* is the detected time. One data record in the record sequence of ANPR data can be represented as follows: $r_1 = \{CAM0311812, jingK36452, 2012\text{-}11\text{-}1\ 08:30:01\}$, and each data record has the same attributes.

The data streams generated by sensor devices may need to be transformed and/or fused to a new data stream which is more meaningful for users. We propose a declarative method that can create stream data service and federate different services into a new powerful service based on declarative rules. The declarative method uses events to transmit between decoupled data streams, services, and users. The semantics of declarative rule are derived from ECA (Event-Condition-Action) rules which can be described straightforward as follows: when an event occurs, its specified condition will be evaluated, and if the condition is satisfied, the corresponding actions can be executed. In this paper, we regard each data record of a data stream as an event, and the declarative rule can be used to create stream data service for data stream source and connect stream data service with its relative services. For better understanding, we redefine some basic definitions referring to the definitions of ECA.

Definition 2 (event). An event can be formalized as $e = \{uri, r, t\}$, in which *uri* is the *uri* of event, r is the data record carried by e, and t is the timestamp when e happened.

We regard events produced form the same source as a group, which have the same *uri* and carry data records having the same schema. For example, events produced by one data stream can have same *uri* as s and can be represented as $E(s)$.

Definition 3 (condition). A condition consists of k ($k \geq 0$) predicates and logical operators of "$\cap$" or "$\cup$." A predicate is presented by a relation expression with relation operators, for example, <, >, =, and so forth. Formally, a condition can be represented as follows: $c = p_1\ lo\ p_2\ lo\ p_3, \ldots, lo\ p_k$, in which p_i is a predicate and *lo* is a logical operator. The possible brackets are ignored for simplicity.

For example, "$vid = jingK36452$" could be both a predicate and a condition, and "$vid = jingK36452 \cap time > 8:00$" is a condition consisting of two predicates.

Definition 4 (action). An action is the actual operation of a declarative rule.

There are two kinds of actions in this paper: one is to emit the events that satisfied specific condition to a specific destination, and the other is to trigger the processing of specific stream data services.

Definition 5 (declarative rule). A declarative rule can be represented as a triple $r = <E, c, a>$, where E is an event set produced from one source, and c is a condition which determines when action a will be triggered.

3.2. Stream Data Service Modeling and Service Composition. To possess and enhance sensor's ability, stream data service should be able to continuously deliver data and communicate with users or other services under certain condition. While traditional Web service or data service is generally a better fit for a request/response exchange, it is not suitable for stream data service. The declarative rule provides long-running asynchronous process capabilities by using events to communicate among processes. Meanwhile, the development of Web push techniques and tools has enabled the creation of event-driven applications directly over the web [8]. By referring to our previous work [9] about data service modeling, we regard the data stream as resource which can be opened through stream data service.

Definition 6 (stream data service). The stream data service can be formalized as $sds = \{uri, r, input, s\_output\}$, in which *uri* is the unique identification of the stream data service; r is the declarative rule which indicates the data stream source, the condition constraint to evaluate events, and the specific action; *input* is given by users to specify the condition; and *s_output* is the event stream which can cache the latest events and deliver them to user actively.

Stream data service can constrain the content of data records that can be provided through defining the condition in declarative rule. Take the following stream data services as examples:

$sds = \{getANPRfromCAMa, <E(CAMa), , emit\ e\ to\ ANPRa>, , ANPRa\}$;

$sds' = \{getANPRfromCAMaByVid, <E(CAMa), vid = \{v\}, emit\ e\ to\ ANPRaofV>, v, ANPRaofV\}$.

They both regard data stream *CAMa* produced by camera site a as data source by defining the events in declarative rules; and, for *sds*, each event will be emitted to

ANPRa since there is no condition defined. Meanwhile, for *sds'*, the only event satisfying the condition in which *vid* in event is equal to a given *v* will be sent to users.

As single sensor is not enough to created applications, and single stream data service cannot satisfy user's requirements. Services composition is a convenient manner to integrate a group of single existing services and create one service which is more complex and useful.

Definition 7 (service composition). A service composition can be represented as a tuple $SC = \{SDS, R\}$ in which *SDS* is a set of stream data services which are initially invoked, and *R* is a set of declarative rules that can connect services in *SDS* with other services.

When user invokes a composited service, the initial services will be invoked and keep producing events respectively, and the rules will be checked for triggering relevant services' invoking or stopping. This process will be executed iteratively until the stopping of the service composition. According to Definitions 5 and 6, there are two kinds of declarative rules that can be used to combine services, namely, *sink-rule* and *connect-rule*, in this paper.

The *sink-rule* can be used by stream data services whose outputs have the same schema and can merge their outputs into one stream through defining the same action, that is, emit respective events into the same data stream. Figure 2 shows part of topological structure of camera sites. Based on Figure 2, the relationship of corresponding service can be shown. For example, the services {*getANPRfromCAMa*, <*E*(*CAMa*), , *emit e to ANPRa*>, , *ANPRa*}, {*getANPRfromCAMb*, <*E*(*CAMb*), , *emit e to ANPRb*>, , *ANPRb*}, {*getANPRfromCAMd*, <*E*(*CAMd*), , *emit e to ANPRd*>, , *ANPRd*} together can provide the ANPR data for one certain road. To obtain ANPR data of this road, three sink-rules can be defined for these services as <*ANPRa*, , *emit e to Road*>, <*ANPRb*, , *emit e to Road*>, and <*ANPRd*, , *emit e to Road*>, in which *ANPRa*, *ANPRb*, and *ANPRd* are the outputs of above services and they can be merged into one event stream *Road* according to the same action "*emit e to Road*."

The *connect-rule* can be used to invoke or stop relevant services when a specific event happens. As shown in Figure 2, for each camera site, there exist a set of neighboring camera sites which directly connect with it. Correspondingly, for each stream data service, there also exist a set of neighboring stream data services, and the relationship of the stream data service can be modeled according to the location relationship of the camera sites. For example, service *getANPRfromCAMa* provided by camera site *a* has a set of neighboring stream data services *getANPRfromCAMb* and *getANPRfromCAMc*. According to the relationship between stream data services, a set of connect-rules can be used by *getANPRfromCAMa* as <*ANPRa*, *vid* = {*v*}, *invoke service getANPRfromCAMb*>, <*ANPRa*, *vid* = {*v*}, *invoke service getANPRfromCAMc*>, and <*ANPRa*, *vid* = {*v*}, *stop service getANPRfromCAMa*>. With these rules, when the *vid* of event in the output of service *getANPRfromCAMa* is equal to given *v*, services *getANPRfromCAMb* and *getANPRfromCAMc* will be invoked, and *getANPRfromCAMa* will be stopped automatically.

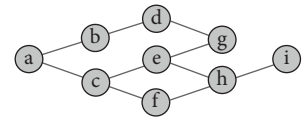

Figure 2: Part of topological structure of camera sites.

4. Service Composition Implementation

Figure 3 illustrates the architecture of our method which can create stream data service based on data streams produced by sensors and combine different stream data services into more powerful services dynamically for flexibly satisfying users' requirements.

To shield the details about the data stream source and provide different representation to users, the Representational State Transfer (REST) architectural style can be followed. Every data stream can be a URI-addressable resource whose representation can be manipulated using a limited and fixed set of verbs, for example, for HTTP, GET, POST, DELECT, and so forth. Data streams can directly exchange information transparently with each other or with other Web resource thanks to the loose coupling of REST, so we implement the stream data service using RESTful design pattern.

Since ANPR data is essentially a stream, traditional Web services cannot help developers to effectively retrieve live data streams due to their "request-and-response" pattern. We utilize a Web push technology, for example, Server-Sent Events (SSE), and declarative rules to realize stream data service. With SSE, the stream data service can actively send events carrying data records to users upon the Web, and, with the declarative rules, users can flexibly obtain events satisfying the declarative rules, that is, users' requirements.

The sensors are not independent from each other, and so are the stream data services; we map the relationship of sensors to relationship of services and aim to realize the cooperation among sensors through dynamical service composition. Declarative rules can be utilized to cooperate among stream data services. A repository can be established to store predefined or user-defined declarative rules. The predefined declarative rules can be provided by service providers or domain experts who can model the relationship of different stream data services through specifying the service event and actions when building a declarative rule.

4.1. Sensor Partition Strategy. We adopt a direct method that divides the sensors into several partitions and processes the corresponding services in parallel. Figure 4 illustrates the parallelism of our method. The partition of stream data services is in charge of the master node. The worker node is the one that actually combines and connects multiple services and outputs the result of composite services. Since service composition needs cooperation between different services, the communication between different workers is inevitable. The master node not only indicates which stream data services are processed by workers but also manages the communication between workers.

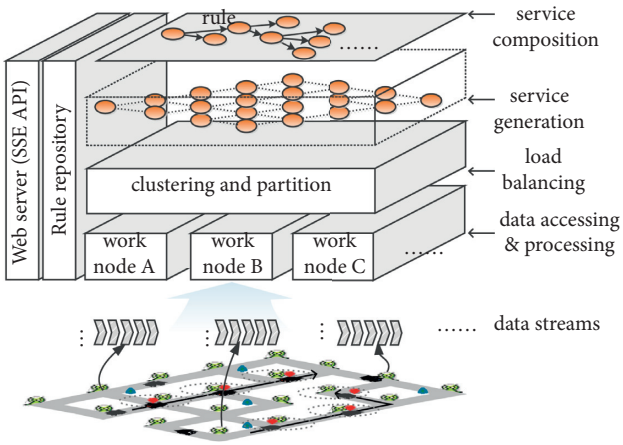

Figure 3: The architecture of our method.

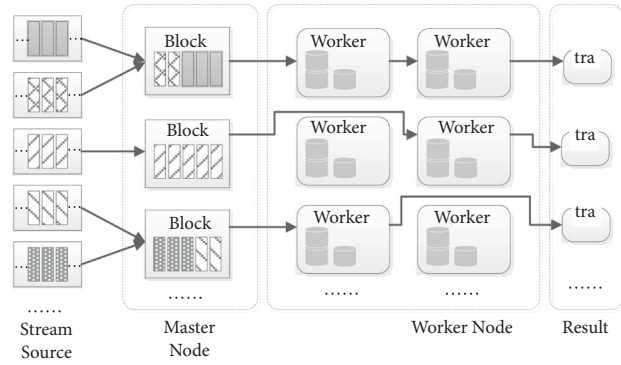

Figure 4: The parallel framework for service composition.

There are some issues to consider when dividing stream data services into partitions. Firstly, to avoid data skew in the worker nodes, the data volume in every partition should be as equal as possible. Secondly, to reduce the communication between workers, the neighboring camera sites should be parted in the same partition. To solve the above issues, we propose a sensor partition strategy and realize it by improving the Number Partition Problem (NPP) [10].

Definition 8 (sensor partition strategy). Given a set of sensors, S is the data stream set produced by them; the sensor partition strategy sp is a partition function $sp(S) = \{S_1, S_2, \ldots, Sk\}$, s.t.

(1) $S = \cup_{i=1,\ldots,k} S_i$.

(2) $S_i \cap S_j = \varnothing, i, j = 1, \ldots, k \cap i \neq j$.

(3) For arbitrary data stream $s \in S_i$, $\exists s' \in S_i$ and s' is data stream produced by its neighboring sensors which directly connect with it.

(4) min $(\sum_{i,j=1,\ldots,k}^{k} (|S_i| - |S_j|))$, where $|S_i|$ means the total frequency of the data stream in S_i.

To be mentioned, condition (3) and condition (4) may not be satisfied at the same time. In this paper, we first satisfy condition (3), which ensures there is not isolated sensor in one partition, and then satisfy condition (4) as possible. Reference [11] proposed a partition algorithm without rearrangement, which can do the partition without reordering any of elements. The relationship of services can be represented as a graph as shown in Figure 3. We firstly traverse the graph breadth firstly and then divide the sensors into n areas. All the sensor numbers of each area can form one array, and each sensor number is one element in the array. For example, we set the degree of parallelism as 2, and we get an array as {12, 4, 7}. We can divide the array as {12} and {4, 7}.

In this paper, we adopt the algorithm in [11] to realize our partition strategy, whose time complexity is $O(kn^2)$. It does not disturb the order of the original array elements and can achieve a suboptimal solution. This algorithm calculates the minimum possible cost $M[i][j]$ and the divider position $D[i][j]$, in which $M[i][j]$ represents the minimum cost of partitioning first i element of input list into j partitions and $D[i][j]$ is used to record the divider position. More details about this algorithm can be found in [11].

4.2. Case Study. In this subsection, we will use the scenario mentioned in Section 2 to illustrate how to compose different stream data services and realize the real-time vehicle tracking application.

For each camera site, stream data services can be created to provide real-time ANPR data. We suppose that there is one stream data service that can return ANPR data when new detected information is captured for every camera site. Take camera site a as an example; stream data service *getANPRfromCAMa* providing ANPR data *ANPRa* captured by a can be created by defining the declarative rule $r = <E(CAMa)$, *emit e to ANPRa*$>$. Users can request URL https://streamdataservice/getANPRfromCAMa by sending an HTTP GET request to obtain the data stream *ANPRa*. The server responds immediately with event carrying the latest data record, followed by updated events as new data records are captured.

To generate the real-time vehicle tracking application, we firstly need to locate given vehicle in whole road net; and there are three needed functions for the cooperation of services: (1) invoke the relative stream data service when the given vehicle is found in one service; (2) stop the services invoked before since the data are not meaningful anymore; and (3) collect all the detected information as a stream data and open it as a service. We can design the service composition as $TRA = \{SDS, R\}$, in which SDS is all stream data services that provided ANPR data for every camera site, and rules in R are as follows:

$r_1 = <sds.s\_output, vid = \{v\},$ invoke $sds.neighboringService>$

$r_2 = <sds.s\_output, vid = \{v\},$ stop $SDS'>$

$r_3 = <sds.s\_output, vid = \{v\},$ emit e to $tra>$

Among them, r_1 and r_2 are connect-rules and r_3 makes sure all specific events sink in one stream *tra*. In detail, r_1 means that, for the event produced by each stream data service $sds \in SDS$, if there exists an event e satisfying condition $vid = \{v\}$, e will trigger the invoking of the neighboring

services of *sds*, r_2 means to trigger the stopping of the invoked services *SDS*′ before, and r_3 means to trigger the updating of the trajectory *tra*, which is a data stream that user can obtain through its URL.

Figure 5 shows a part of the processing of service composition for creating real-time vehicle tracking application. It illustrates the updating of the specific data stream and invoking of the relative services, while the stopping of the invoked service is not illustrated.

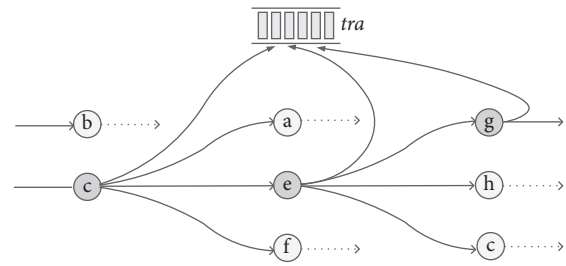

Figure 5: The process of service composition for real-time vehicle tracking application.

5. Evaluation

To evaluate the performance of our declarative method, we firstly compare our method with centralized manner, and then we evaluate our method with different degree of parallelism.

5.1. Experiment Setup

5.1.1. Datasets. The dataset used in our experiment is the real Automatic Number Plate Recognition (ANPR) data in Beijing. Specifically, the number of camera sites is 1794, and there are about 2.3 million vehicles and about 9.7 million records in the dataset. Each data record includes the vehicle information, the camera site information, the timestamp, and other additional information. We simulate the data streams completely according the timestamps in the datasets.

5.1.2. Environments. Our method is implemented in a cluster consisting of 5 nodes. All the nodes are running in virtual machines with CentOS system and Java 1.80. The detailed configuration of the cluster is shown in Table 1.

Table 1: Configuration of the cluster.

Role of node	CPU	Memory (G)
Master node	Intel Xeon E312xx	6
Work node 1	Intel Xeon E312xx	6
Work node 2	Intel Xeon E312xx	3
Work node 3	Intel Xeon E312xx	3
Work node 4	Intel Xeon E312xx	3

5.2. Performance Evaluation.

To evaluate the performance of our method, we firstly design the following criterion.

Definition 9 (processing latency). Processing latency for stream data service is the time difference for each data record between when it arrives and when its process ends. The latency for each data record can be denoted as $L_i = t_{\text{arr}} - t_{\text{end}}$.

To be mentioned, since parts of records are not required, not all the data records in a data stream will produce a transformed result in the output of service, and we only record ones that produce results.

Firstly, we simulated data streams with the real ANPR data. We regarded each camera site as a stream data source and also simulated one stream data source that provided all data. Then we realized two manners to generate the real-time trajectory. To realize the centralized manner, we access all the ANPR data into our system and process them to generate trajectory. We run both manners in standalone mode and compare the processing latency. The detailed experiment steps are as follows:

(1) We randomly selected 50 vehicles.

(2) We generated trajectories for 50 selected vehicles respectively through two manners and recorded the processing latency.

(3) We calculated the average processing latency for both methods and compared them with their changing curve.

As shown in Figure 6, at the beginning of all the invocation, the latencies are similar, since they all process all the ANPR data. Meanwhile, at the subsequent processing, the latency of our method is obviously less compared to the traditional methods. It is because ANPR data from only 2–4 camera sites are processed in our method, and the processing data is far less compared to other methods. However, since the amount of ANPR data is not very huge, the difference is not so outstanding. The advantage will be remarkable when the amount of data streams is huge.

Secondly, since our method also supports running in distributed environment, we then evaluate our method with different degrees of parallelism. More specifically, we randomly selected 1000 vehicles and set the degree of parallelism as 1, 3, and 5. The detailed experiment steps are as follows:

(1) We simulated data streams with the real ANPR data and regarded each camera site as a stream data source.

(2) We generated trajectories of the 1000 vehicles, respectively, with different degree of parallelism and recorded the execution latency during the trajectories' generating.

(3) After finishing all of generations, we calculated the average execution latency for each parallelism degree.

The final result is shown in Figure 7. With the increase of the degree of parallelism, the execution latency is reduced. Especially for the beginning of the real-time vehicle tracking

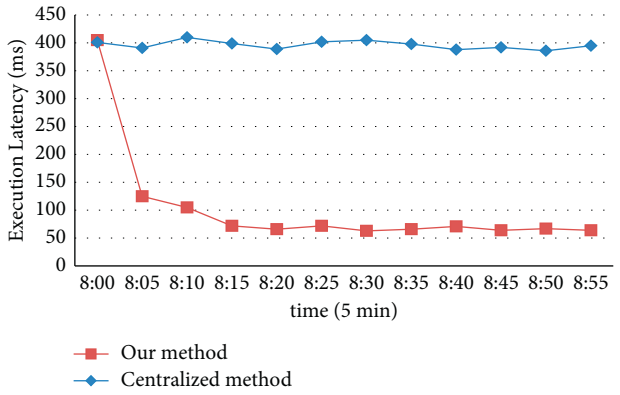

Figure 6: The execution latency for different methods.

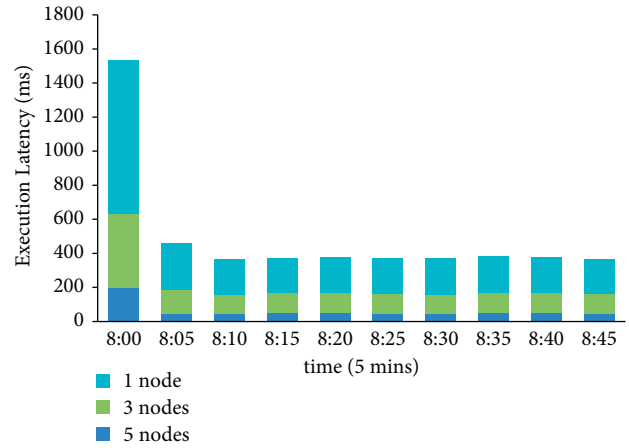

Figure 7: The execution latency with different degree of parallelism.

application, when all the ANPR data need to be processed, the difference is more obvious. It is indicated that the parallelism of service composition can improve its performance even with huge amount of data streams.

6. Related Work

In this section, we summarize the related work in two aspects, which are the work about sharing and accessing of the sensor data and the work about cooperating different services to create more powerful one.

6.1. Sensor as a Service. Recently, with the development of sensor device, Sensor Network, and Internet of Things, more and more devices are deployed in the physical world. In respect of effective access and sharing senor data, there are three kinds of work, which are Sensor as a Service (or Sensing as a Service) [12–14], Sensor Web [15–17], and Web of Things (WoT) [8, 18]. The term Sensor as a Service is generated following the recent trend towards Everything as a Service (XaaS); it regards sensors deployed in the realistic society as services that provide data resources to consumers. The term Sensor Web refers to a global network of Web-connected sensors. The term WoT is derived from Internet of Things. They all aim to provide users a uniform interface to accessing sensor data in real time and enable interactions with sensor devices based on the Web. Zeng et al. [8] presented a survey on WoT's research works, which proposed two main analysis criteria of WoT: integrating physical devices (directly or indirectly) on the Web and providing composable services to enable physical-virtual mashups.

These works make the sensor devices be accessible through URIs which can be manipulated using HTTP operations. Existing researches offer significant advantages such as on-demand resource sharing and feasibility of creating new value-added services based on existing services. In this paper, we kind of provide proxy-based stream data service and give sensor data the ability of computation and communication with stream data service.

In the aspect of realization for the stream data service, many works that do not have high real-time demand adopt the pooling method to access stream data [16, 17]. For example, although LiveWeb realized the alarm function with Subscribe\Alert mechanism, the realization was based on periodical execution of the users' queries. Meanwhile, there are researches that access stream data based on event-driven mode. The approach adopted in Homeport [17] is based on the Server-Sent Events (SSE) technology. In this paper, we also adopt SSE to realize the active sending of updated data records in the server side.

6.2. Service Composition. Service composition has been a widely researched issue in the past few years [19]. It allows users to create a more powerful service with the composition of multiple existing services. In our previous data mashup work [9], called Mashroom, we aimed to support users to easily process and combine data with visualized nested tables. Mashroom is a mashup tool with a novel programming model that allows the end-user to combine multiple data services to build mashup applications. Our previous work was focused on processing and combination of static data. In this paper, we aim at the composition of stream data service.

Web services composition is a highly active and widely studied research direction [20]. One way to build service composition is by modeling user interactions and services into events [21], as well as combining the typical Service-Oriented Architecture (SOA) with the more flexible Event-Driven Architecture (EDA). Chen et al. [22] proposed a situational-aware service coordination method based on the event-driven SOA paradigm, which can support asynchronous communication and on-demand distribution of data streams in distributed IoT environment.

Most existing methods needed to predefine the composition logic, and did not support adaptive service collaboration. In this paper, we proposed a flexible service composition method based on ECA rules, which can dynamically combine relative stream data services and build more efficient and powerful service.

7. Conclusion and Future Work

In this paper, we utilized dynamical service composition to realize cooperation among sensors. Firstly, we proposed a stream data service model that can access, process, and share data streams continuously and timely. Then, we proposed a declarative method which can dynamically compose stream data services with declarative rules to satisfy users' requirements flexibly. Finally, we proposed a stream data service partition strategy to implement service composition in parallel and ensure the performance. The evaluation of series comprehensive experiment results shows that our method has both good efficiency and effectiveness.

Many further improvements can be made over some details of our method. For example, we will verify and enhance the efficiency of our method when facing number of concurrent users' requirements and high speed and huge volume data streams.

Data Availability

The traffic flow data used to support the findings of this study have not been made available because the data were supplied by local management Transport Department under license with certain confidentiality level and so cannot be made freely available. Requests for access to these data should be made to the corresponding author for an application of joint research.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by Doctor Scientific Research Foundation of Shandong Jianzhu University (no. X21007Z).

References

[1] D. A. Reed, D. B. Gannon, and J. R. Larus, "Imagining the future: thoughts on computing," *Computer*, vol. 45, no. 1, pp. 25–30, 2012.

[2] J. Zhang, B. Iannucci, and M. Hennessy, "Sensor data as a service—A federated platform for mobile data-centric service development and sharing," in *Proceedings of the IEEE International Conference on Services Computing*, pp. 446–453, Santa Clara, CA, USA, June 2013.

[3] Y. Han, G. Wang, J. Yu, and C. Liu, "A service-based approach to traffic sensor data integration and analysis to support community-wide green commute in China," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 9, pp. 1–10, 2015.

[4] C. C. Loy, T. Xiang, and S. Gong, "Multi-camera activity correlation analysis," in *Proceedings of the Computer Vision and Pattern Recognition*, Miami, FL, USA, June 2009.

[5] S. Steffen, B. Claus, A. Hamza et al., "Integrity and collaboration in dynamic sensor networks," *Sensors*, vol. 18, no. 7, Article ID 2400, 2018.

[6] J. Wan and Q. Liu, "Distributed data association in smart camera networks using belief propagation," *Distributed Smart Cameras (ICDSC)*, Association for Computing Machinery, New York, NY, USA, 2011.

[7] A. Arasu, B. Babcock, and S. Babu, "Stream: the stanford data stream management system," *Data Stream Management*, Springer Berlin Heidelberg, Berlin, Germany, 2016.

[8] D. Zeng, S. Guo, and Z. Cheng, "The web of things: a survey," *Journal of Clinical Microbiology*, vol. 6, no. 6, pp. 424–438, 2011.

[9] G. Wang, S. Yang, and Y. Han, "Mashroom: end-user mashup programming using nested tables," in *Proceedings of the 18th International Conference on World Wide Web*, Madrid, Spain, April 2009.

[10] S. Mertens, "The easiest hard Problem: number partitioning," *Computational Complexity & Statistical Physics*, vol. 125, pp. 125–139, 2003.

[11] S. S. Skiena, *The Algorithm Design Manual: Text*, Springer Science & Business Media, Berlin, Germany, 1998.

[12] A. Chakraborty, S. Misra, A. Mondal, and M. S. Obaidat, "SensOrch: QoS-Aware resource orchestration for provisioning Sensors-as-a-Service," in *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 1–6, Dublin, Ireland, July 2020.

[13] S. Chatterjee and S. Misra, "Adaptive data caching for provisioning sensors-as-a-service," in *Proceedings of the Black Sea Conference on Communications and Networking (BlackSeaCom)*, pp. 1–5, Varna, Bulgaria, June 2016.

[14] Z. H. Ali, H. A. Ali, and M. M. Badawy, "A new proposed the Internet of things (IoT) virtualization framework based on sensor-as-a-service concept," *Wireless Personal Communications*, vol. 97, no. 1, pp. 1419–1443, 2017.

[15] W. Du, N. Chen, S. Yuan, C. Wang, M. Huang, and H. Shen, "Sensor web-enabled flood event process detection and instant service," *Environmental Modelling & Software*, vol. 117, pp. 29–42, 2019.

[16] X. Yang, W. Song, and D. De, "Live web: a sensorweb portal for sensing the world in real-time," *Tsinghua Science and Technology*, vol. 16, no. 5, pp. 491–504, 2011.

[17] T. L. Guilly, P. Olsen, A. P. Ravn, J. B. Rosenkilde, and A. Skou, "HomePort: middleware for heterogeneous home automation networks," in *Proceedings of the Pervasive Computing and Communications Workshops (PERCOM Workshops)*, February 2013.

[18] F. Paganelli, S. Turchi, and D. Giuli, "A web of things framework for RESTful applications and its experimentation in a smart city," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1412–1423, 2017.

[19] M. J. Carey, N. Onose, and M. Petropoulos, "Data services," *Communications of the ACM*, vol. 55, no. 6, pp. 86–97, 2012.

[20] A. Urbieta, A. González-Beltrán, S. Ben Mokhtar, M. Anwar Hossain, and L. Capra, "Adaptive and context-aware service composition for IoT-based smart cities," *Future Generation Computer Systems*, vol. 76, pp. 262–274, 2017.

[21] Y. Zhang, L. Duan, and J. Chen, "Event-driven SOA for IoT services," in *Proceedings of the IEEE International Conference on Services Computing*, pp. 629–636, Anchorage, AK, USA, June 2014.

[22] B. Cheng, D. Zhu, and S. Zhao, "Situation-aware IoT service coordination using the event-driven SOA paradigm," *IEEE Transactions on Network & Service Management*, vol. 13, no. 2, pp. 349–361, 2017.