Load balance oriented data processing mechanism for bounded and unbounded data in smart cities

Qinglong Dai¹, Jin Qian², Jianwu Li³, Jun Zhao⁴ and Xiaoxiao Liu¹

Abstract
The co-existence of bounded data and unbounded data gives a great challenge for the traditional single and inflexible data processing in smart cities. The wide promotion of the internet of things (IoT) makes the data amount rapidly increase. This leads to the further raise of the requirement for data processing in smart cities, especially the demand for low latency and abundant data in real-time video services. To solve this problem, a load balance oriented data processing mechanism for bounded and unbounded data in smart cities is proposed. A smart city framework is introduced to explain the role of data processing in smart cities. A load-balanced data processing mechanism is proposed. Based on the mathematical model for data processing in smart cities, the load-balanced data processing is abstracted into an optimization problem. Aiming to obtain the minimum load balance ratio (LBR), an LBR algorithm is presented. Through simulation and experiment, the superiority and feasibility of our work are validated via numerical simulation and prototype implementation, respectively.

Keywords
Smart cities, stream data, batch data, model, Flink

Date received: 30 October 2020; accepted: 13 April 2022

Introduction
Smart cities play a significant role in human’s modern life.¹ It has penetrated into all aspects of society, such as traffic, communications, city maintenance, and so on. With the development of technologies, especially information communications technology (ICT) and the internet of things (IoT), the city is becoming smarter. Smart devices like smart meters, smart vehicles, and other sensors, mostly replace their counterparts. Technologies greatly extended the management vision to cities’ every corner.²

A smart city is an intelligent electrical complex that includes a variety of operations, energy, measures, etc. Smart cities are constructed on an integrated high-speed network, to ensure information transport efficiency. A smart city is economic, safe, and environment-friendly. The key issue in smart cities is the application of data processing and acceleration of information transport.³

A smart city can be seen as an application of IoT in a modern city’s coverage.⁴ In the past, the collection of traffic situations, city monitoring, power consumption, resident account balance, device log, etc., depends on workers’ manual operations. Via IoT, the interaction of humans is largely reduced, which makes smart cities work smoothly and effectively. As a result of this, IoT is enjoying high favor. In 2016, the value of the IoT market was 157 billion USD. It is expected to a market cap, 771 USD, by 2026.⁵ With the deployment of fourth generation communication technologies, the addition of smart sensors, smart meters, smart vehicles, and other devices greatly increases the complexity and uncertainty of smart cities. It can be foreseen that the data in smart cities would sharply increase with the full commercial application of the fifth-generation (5G) communications technologies for IoT sensors. According to the prediction report of Cisco, 500 billion objects like mobiles, sensor, etc., will access to the Internet.⁶

¹Beijing Union University, Smart City College, Beijing, China
²Dareway Software Co. Ltd, Institute of Artificial Intelligence, Jinan, Shandong, China
³H3C Technologies Co. Ltd, Beijing, China
⁴China Telecom Beijing Research Institute, Research Institute of big data and artificial intelligence, Beijing, China

Corresponding author:
Qinglong Dai, Beijing Union University, No. 97, Northsihuan East Road, Haidian District, Beijing 100101, China.
Email: xxtqinglong@bnu.edu.cn

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).
Big data in smart cities is an attractive and meaningful topic. On one hand, high volume and velocity data from smart cities’ different components should be collected, cleaned, integrated, and analyzed. Through this, a view of the whole city can be grasped. Note that, the message queuing telemetry transport (MQTT) protocol, HyperText Transfer Protocol (HTTP), or javascript object notation (JSON) based wireless data transport between sensors and smart gateways gives an easy, ubiquitous, and continuous data collection.7 On the other hand, the results of data analysis are the basis of smart city management strategy making. Therefore, data processing in smart cities is crucial.

Any kind of data is produced as a stream of events. In a smart city, the data can be classified into bounded data and unbounded data. Bounded data, also called batch data, has a defined start and end. Meanwhile, unbound data, also called stream data, has a start but no end. Usually, users’ account information, personal information, and other relatively stable information are bounded data. Users’ consumption information, sensor measurements, machine logs, and so on are unbounded data. Bounded data is infrequently changing and can be processed by ingesting all data before performing any computations. Ordered ingestion is not necessary for bounded data processing because a bounded data set can always be sorted. Unbounded data is dynamic, frequently changing, and scrappy. And must be continuously processed, that is, unbounded data must be promptly handled after it has been ingested. It is impossible to wait for all input data to arrive because the input has no end and will not be completed at any time. In unbounded data processing, the ingested data in a specific order is required, such as the order in which data generated, to be able to reason about result completeness.

As an important part of data science, several data processing frameworks are designed and applied for investigating the pattern of data. Apache Hadoop is only used for bounded data processing. Apache Storm and Samza are only used for unbounded data processing. Apache Spark and Flink are not only used for bounded data processing but also unbounded data processing. The core of Spark is using multiple micro-batches to simulate a stream. In spark, unbounded data or a stream is treated as special bounded data. Unbounded data is seen as the composition of a lot of bounded data. Unbounded data or a stream is split into an ordered series of micro-batches or bounded data. On the contrary, in Flink, bounded data is treated as a special situation of unbounded data. Bounded data is seen as a fixed-sized data stream. Via precise control of time, Flink can handle any data, no matter bounded data or unbounded data.

In smart cities, any operation upon data or information would consume resources, including CPU, bandwidth, frequency, storage space, and so on. Considering the cost and the construction period, the resource in smart cities is limited and relative deficiency. In the running of a smart city, the realization of a specific service is step-wise. An operation upon data is normally executed after its previous one or more operations’ completion. Service systems are normally distributed and do not have a centralized controller. This means that there are multiple paths for the realization of a specific service, from this service’s source to sink. The data distribution upon these multiple paths is random, without any intervention. This leads to the situation that some paths hold more data than their capability, while other paths are idle. It is also called unbalanced load distribution. Because each operation upon data is accomplished by specific nodes and nodes have their limited computing resource like buffer, the nodes would discard exceeded data or require re-transmission. Both actions would make data transmission latency grow. Therefore, unbalanced load distribution will make an operation spend more time to waiting for all its previous operations’ completion, compared to balanced load distribution. And then the latency of the corresponding service increases. Besides, balanced load distribution will reduce the resource occupancy ratio of entire smart cities.

With intervention, the load among multiple paths can be evenly distributed. Load balance is a multi-commodity flow issue. Through distributing the load evenly upon multiple paths, one or some effects are achieved. These effects include high performance, high scalability, high stability, and low energy consumption. Depending on different situations, such as pursuing lowest latency, one or several indicators would be selected and composed as the objective function in multi-commodity flow issues.

Currently, the big data in smart cities has been used on the base of Hadoop and Spark. Hadoop is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Sensors, like smart meters, report the user’s power data every 15 min. With the advance of technologies and the requirement of the city’s refined management, the interval between two reports may be decreased to second-level. Correspondingly, the data volume would explosive increase. For example, for 10,000 smart meters, the data volume will increase from 32.61 Gb to 114.6 TB. In a big city, there are millions of residents. Considering the co-existence, separated processing, and rapidly increased data amount, for bounded data, and unbounded data, this is a great challenge for load-balanced data processing in smart cities.

To cope with unbalanced load in big data with explosive volume, Flink is chosen to realize real-time data processing for smart grids, as a basic framework. The contribution of our work are listed as follows:

(1) A smart city framework is introduced, to explicit the role of data processing in smart cities. The layers, data processing workflow, and data operation in a smart city are explained in detail.
Especially, under a general workflow, unbounded data is treated as a data stream, and bounded is treated as a data set, to bridge the gap between the data processing for bounded data and that for unbounded data.

(2) A load-balanced data processing mechanism is proposed. The basic concepts of load-balanced data processing are defined. Load-balanced data processing in smart cities is formulated into an optimization problem. The load balance ratio (LBR) is used to demonstrate the load distribution situation upon the entire data processing network. To obtain the minimum LBR, an LBR algorithm is presented. And the time complexity of the LBR algorithm is analyzed.

(3) Via simulation and experiment, the superiority and feasibility of our work are validated respectively, in the aspect of average resource occupancy rate (ROR) for nodes, average ROR for edges, the number of accepted flow request, and the testbed system.

The rest of this paper is organized as follows: Section II reviews the related works. The smart cities framework is introduced in Section III. In Section IV, the load-balanced data processing mechanism is modeled. Numerical evaluations and experiments are given in Section V and Section VI, respectively. Section VII concludes this paper.

Relate work

Flink is an open-source framework that is supported by the Apache software foundation and designed as a distributed stream data processing engine. Havers et al. built a Flink-based prototype to evaluate their proposed DRIVEN framework. The DRIVEN framework was used to cope with a common problem in vehicular networks’ application, that is, the conflict between the limited communication bandwidth and data transmission’s costs. The engineering implementations of distributed stream processing frameworks for data processing in smart cities were examined and these frameworks’ adoption and maturity among IoT applications were analyzed by Nasiri et al. Apache Storm, Apache Flink, Apache Spark, Apache Heron, Samza, and Akka were selected as distributed stream processing frameworks. An open-source benchmark for the emerging frameworks, structured streaming, Kafka streams, Spark streaming, Flink and its extensive analysis was proposed by van Dongen and Van den Poel. The relationships among between latency, throughput, and resource consumption were discussed. The performance impact of adding different common operations to the pipeline was also measured. Antaris and Rafailidis proposed an approximate indexing mechanism to index and store massive image collections with varying incoming image rates. Flink was used to appraising the proposed mechanism, comparing it to a baseline with a disk-based processing strategy. A stream processing was seen as a Directed Acyclic Graph (DAG), but a mathematical elaboration was missing. To improve computational ability, Chen et al. introduced a GFlink architecture, extending the original Flink from Central Processing Unit (CPU) clusters to heterogeneous CPU-Graphic Processing Unit (GPU) clusters. They further raised a novel parallel hierarchical extreme learning machine (H-ELM) algorithm based on Flink and GPUs, to accelerate Flink for big data. CPUs and GPUs cooperated to fulfill the works assigned to them, thus achieving a better acceleration than previous work. Espinosa et al. suggested a property-based testing tool for Apache Flink. It used a bounded temporal logic to guide how random streams were generated and to define the properties. Xu et al. established fault-tolerant mechanisms for graph and machine learning analytics that ran on a Flink based distributed dataflow system. Isah et al. presented a comprehensive study of distributed data stream processing, and analytics frameworks, and gave a critical. Kaitoua et al. developed a GenoMetric Query Language (GMQL) to operate on heterogeneous genomic datasets. Flink was used for data computation and management in gene territory. Katragadda et al. suggested a neighborhood-centric graph processing approach to handle graphs. This approach exploits the locality, parallelism, and incremental computation of existing distributed frameworks to calculate graph features with exact results. Graph stream processing in link prediction was executed on the base of Flink. Zachelias et al. provided a novel approach that enables the execution of top-k join queries over sliding windows. They reduced the amount of data that need to be analyzed by the stream processing operators. Flink combined with Kafka was applied in experiment evaluation to prove the proposed approach’s superiority. Li et al. put forward a Flow-network based auto rescale strategy for Flink in, to solve the problem that a load of big data stream computing platform was increasing with fluctuation while the cluster was not able to rescale efficiently. However, the change of data transfer rate brought by Flink’s operators was ignored.

Smart cities framework

Architecture

As a complex system, smart cities are hierarchical. From bottom to top, as shown in Figure 1, there are infrastructure layer, control layer, and application layer.

(1) Infrastructure layer: The infrastructure layer is the aggregation of all the physical devices, including buildings, networks, and sensors. Buildings provide the physical space for smart cities’ everything. Networks provide data transport for smart cities’
information, via wired and wireless networks. Sensors provide real data collection from smart cities’ every corner. The composition of buildings, networks, and sensors makes the infrastructure of a smart city. The infrastructure layer is the fundament and skeleton of smart cities.

(2) Control layer: The control layer is the core of smart cities, including multiple components like device access, resource management, load balance, operation monitoring, data processing, and so on. Devices access provide the information update when devices access to or quit from smart cities. Resource management provides the scheduling of smart cities’ resources. Load balance provides even flow distribution in the entire smart city, to avoid load fluctuation. Operation monitoring provides real-time visualization of a smart city. Data processing provides a calculated regular pattern based on the data from the infrastructure layer or other components. The control layer is logical, nonintuitive, and is realized by software in servers. Note that our work in this paper is closely related to data processing. The control layer is the brain of smart cities.

(3) Application layer: The application layer is the top layer of smart cities and directly interacts with users, including city management, healthcare, power grid, traffic, industry, etc. The components in the application layer are the application areas of smart cities. With ICT technologies, each aspect of a city is completely reformed. For example, most operations in the traditional industry are accomplished by workers, relying on experience. With the help of ICT, the monitoring of work progress and operations can be partially or completely replaced by sensors or machines. This brings the reduction of labor costs and the improvement of work efficiency. The application layer consists of several specific functions of smart cities.

In smart cities, each component is relatively independent and pervasively connected with other components. As the middle layer, the control layer is a bridge between the infrastructure and the application layer. On one hand, the control layer gives an operation-based infrastructure layer’s devices, information, and data. On the other hand, the output of the control layer provides the basis of different areas in the application layer. Besides, the effect of multiple components in application layers conversely is the base of feedback adjustment upon the infrastructure layer.

Workflow

With the help of other components, data processing deals with a large amount of collected data from the infrastructure layer and outcomes processed results for different kinds of smart city applications. The workflow of data processing is shown in Figure 2.

In data processing, the data in databases or files are first entered into Kafka. For databases, the data can be ingested into Kafka in a real-time way, by the monitor of databases’ log. For files, the data is ingested into Kafka at one time, by file read. Kafka is a distributed event streaming component for high-performance data pipelines and data integration. Through Kafka, data
automatically gets into the execution environment. For bounded data, such as the data from files, it is treated as a data set. And for unbounded data, such as the data from databases, it is treated as a data stream. In the transformation module, several operations are carried out on the combination of different operators like `Map`, `FlatMap`, `Filter`, `KeyBy`, `Reduce`, and so on, to obtain the pattern hidden in a large amount of data. At last, the calculated results are restored in the sink module, as the basis of the UI dashboard.

The data from databases or files to sink is the process of separating the wheat from the chaff, according to a service’s specific requirement. For example, in a company, a manager would like to check employees’ work efficiency, to decide the department adjustment program. The data in internal office system logs can be ingested into this workflow. Via search, splitting, filtering, and statistics, the manager can see each employee’s volume of work, work efficiency, and the statistical data by department. According to these results of the workflow, the manager can make his mind to promote an employee or not, to expand a department or not. Note that, the data set realizes the processing of inventory data, meanwhile, the data stream realizes the processing of incremental data. Essentially, the workflow of the data process is implemented by Java. The data stream is the inheritance and extension of the stream in Java, without an end. The data set is the inheritance and extension of the stream in Java, with an end.

### Data operation

In data processing, the operation of data is mostly completed in the transformation module, in the form of transformation operators in the data processing script program. Common transformation operators are `Map`, `FlatMap`, `Filter`, `KeyBy`, `Reduce`, etc. Note that for a transformation node, the data transfer rate for input flow and that for output flow are different, as shown in Figure 3. In Figure 3, \( N \) is the ratio of data transfer rate for output flow to that for input flow.

From Flink’s official website, we can see that operator `Map` takes one element and produces one element. This means that for operator `Map`, \( N \) is equal to 1. Operator `FlatMap` takes one element and produces zero, one, or more elements. For operator `FlatMap`, \( N \) is greater than 1. Operator `Reduce` takes a “rolling” reduce on a keyed data flow. For operator `Reduce`, \( N \) is smaller than 1.

**Figure 3.** The input and output of a node.

### Load balanced data processing mechanism

#### Basic concepts

**Definition 1.** Data flow graph: The topology of a cluster is defined as a directed acyclic graph (DAG), \( G = (V, E) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set of all nodes in \( G \), \( S \) is the set of source nodes in \( G \), \( D \) is the set of destination nodes in \( G \), \( S \subseteq V \), \( D \subseteq V \), \( E = \{(v_i, v_j)|i, j \in [1, n], n = |V|\} \) is the set of directed edges in \( G \), \((v_i, v_j)\) is a link from \( v_i \) to \( v_j \), \( \forall v_i \in V \), there is \( c(v_i, v_j) \) which is the maximum data transfer rate of edge \( v_i, v_j \) and is called the capacity of node \( v_i \). \((v_i, v_j)\) is the actual data transfer rate of edge \( v_i, v_j \) at the current juncture and is called the current data flow of edge \( v_i, v_j \). \( f(v_i, v_j) \) is the actual data transfer rate of node \( v_i \) at the current juncture and is called the current data flow of node \( v_i \).

The sum of \( v_i \)’s input data transfer rate from its directed connected nodes to \( v_i \) is called \( v_i \)’s input flow and is denoted as

\[
|f_{v_i,\text{in}}| = \sum_{v_{\text{directed}, j} \in V} f(v_{\text{directed}, j}, v_i). \quad (1)
\]

The sum of \( v_i \)’s output data transfer rate from \( v_i \) to its directed connected nodes is called \( v_i \)’s output flow and is denoted as

\[
|f_{v_i,\text{out}}| = \sum_{v_{\text{directed}, j} \in V} f(v_i, v_{\text{directed}, j}) \quad (2)
\]

where the node that directly connected to \( v_i \) is denoted as \( v_{\text{directed}, j} \). \( v_{\text{directed}, j} \in V \). In \( G \), the unit of capacity and flow are both tuple/s and \( 0 \leq f(v_i, v_j) \leq c(v_i, v_j) \). \( G \) is denoted as a non-symmetric matrix.

\[
G = (V, E) = \begin{bmatrix}
    c(v_1) & c(v_1, v_2) & \cdots & c(v_1, v_n) \\
    c(v_2, v_1) & c(v_2) & \cdots & c(v_2, v_n) \\
    \vdots & \vdots & \ddots & \vdots \\
    c(v_n, v_1) & c(v_n, v_2) & \cdots & c(v_n)
\end{bmatrix} \quad (3)
\]

In Equation (3), because that \( G \) is a DAG, there is one non-zero value between \( c(v_i, v_j) \) and \( c(v_j, v_i) \), and the other one is zero. When \( i = j \), the corresponding value in the matrix is \( c(v_i) \).

**Definition 2.** Augment network: Based on the \( G \) with existing flow \( f \), \( G'(V', E') \) is the augment network. \( \forall v_i' \in V' \), corresponding to the node \( v_i \) in \( G \), its augment
capacity function $c'(v_i)$ is related to the situation of the next flow, i.e. adding flow and deleting flow.

$$
 c'(v_i) = \begin{cases} 
 c(v_i) - f(v_i) & \text{adding flow} \\
 f(v_i) & \text{deleting flow} \\
 0 & \text{otherwise}
 \end{cases} \quad (4)
$$

where $c(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$. $v_i, v_j$ is the occupied resource by existing flow $f$ in edge $(v_i, v_j)$.

The augment network $G'$ is the variable space of $G$ at the current juncture. Therefore, in Definition 2, if the next flow is added from $S$ to $D$, the augment capacity of node $v_i$ is the subtraction of node $v_i$'s capacity and existing flows’ data transfer rate, the augment capacity of edge $(v_i, v_j)$ is the subtraction of edge $(v_i, v_j)$’s capacity and existing flows’ data transfer rate. If the next flow is deleted from $S$ to $D$, the augment capacity of node $v_i$ is the existing flows’ data transfer rate of node $v_i$, the augment capacity of edge $(v_i, v_j)$ is the existing flows’ data transfer rate of edge $(v_i, v_j)$. Otherwise, the augmented capacity of node $v_i$ and edge $(v_i, v_j)$ are both 0. Thus, finding an acyclic path from $S$ to $D$ in the augment network is equivalent to calculate the variable space of the data transfer rate in $G$.

Definition 3. Augment path: An augment path is the set of a series of nodes and edges. These nodes are ordered passing nodes from source node set $S$ to destination node set $D$. These edges are end to end from source node set $S$ to destination node set $D$. An augment path in $G$ is denoted as $P = \{ (v_i, v_j) | v_i, v_j \in S, v_k \in D \}$.

$$
 P = \bigcup_{(v_i, v_j) \in E \text{ end to end from } S \text{ to } D} (v_i, v_j) \bigcup_{v_i \in V} v_i \quad (6)
$$

The data transfer rate of path $P$ is

$$
 |f_P| = c_f(P) = \min \{ c_f(v_i), c_f(v_i, v_j) | v_i \in P, (v_i, v_j) \in P \} \quad (7)
$$

$\frac{c_f}{c_f}$ is the existing flows’ data transfer rate of node $v_i$, $c_f(v_i, v_j)$ is the existing flows’ data transfer rate of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

Definition 3. Augment path: An augment path is the set of a series of nodes and edges. These nodes are ordered passing nodes from source node set $S$ to destination node set $D$. These edges are end to end from source node set $S$ to destination node set $D$. An augment path in $G$ is denoted as $P = \{ (v_i, v_j) | v_i, v_j \in S, v_k \in D \}$.

$$
 P = \bigcup_{(v_i, v_j) \in E \text{ end to end from } S \text{ to } D} (v_i, v_j) \bigcup_{v_i \in V} v_i \quad (6)
$$

The data transfer rate of path $P$ is

$$
 |f_P| = c_f(P) = \min \{ c_f(v_i), c_f(v_i, v_j) | v_i \in P, (v_i, v_j) \in P \} \quad (7)
$$

$\frac{c_f}{c_f}$ is the existing flows’ data transfer rate of node $v_i$, $c_f(v_i, v_j)$ is the existing flows’ data transfer rate of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.

$$
 c_f(v_i, v_j) = c_f(v_i) + c_f(v_i, v_j)
$$

where $c_f(v_i)$ is the capacity of node $v_i$ in $G$, $f(v_i)$ is the occupied resource by existing flow $f$ in node $v_i$, $c_f(v_i, v_j)$ is the capacity of edge $(v_i, v_j)$.
The constraint conditions are
\[ \sum_{v_i \in S} f_{v_i,\text{in}} = 0 \]  
\[ \sum_{v_i \in D} f_{v_i,\text{out}} = 0 \]  
\[ f_{v_i} \leq c(v_i) \]  
\[ f_{(v_i, v_j)} \leq c(v_i, v_j) \]  
\[ f_{v_i,\text{in}} = N \times f_{v_i,\text{out}} \]

where equation (12) means that for each node in \( S \) there is no input data flow. Equation (13) means that for each node in \( D \) there is no output data flow. Equation (14) means that for each node in \( G \) the data transfer rate must be smaller than or equal to this node’s capacity. Equation (15) means that for each edge in \( G \) the data transfer rate must be smaller than or equal to this edge’s capacity. Equation (16) means that for each node in \( G \), its data transfer rate for output flow is \( N \) times of that for input flow.

In the running of smart cities, services’ flow loads are evenly distributed upon the entire \( G \), via Algorithm 1. We assume that the flow in \( G \) is from one source node to one destination node. Every source node in \( S \) can provide the same service data. Every destination node in \( D \) can accept handled service data. In the pseudocode of Algorithm 1, \( ROR_{v_i} \) is the resource occupancy rate of node \( v_i \). \( ROR_{(v_i, v_j)} \) is the resource occupancy rate of edge \((v_i, v_j)\). Note that in the calculation of Algorithm 1, when two or more maximum value exists, only one is chosen. Because our goal is to evenly distribute service data upon the entire network, a local choice between two or more equal candidates has little impact on global load distribution.

Algorithm analysis
Before the analysis of Algorithm 1, the concept of the hop is necessary. The hop for path \( P \) in this paper is the number of passing nodes from \( S \) to \( D \). From the definition, we can know that the source node and the destination node are not included in the hop. For Algorithm 1, the maximum loop number is the product of \( G \)’s maximum hop and the sum of \(|V|\) and \(|E|\). The time complexity of Algorithm 1, \( T(n) \) is
\[ T(n) = O(\max(\text{hop}(G))(|V| + |E|)) \]  

Algorithm 1 is a kind of distributed parallel algorithms. Its procedures are executed on different nodes. The communications between these nodes are coordinated by Zookeeper. The computing is normally simple and depends on the amount of input data and the construction of \( G \).

Simulation
In this section, numerical results are achieved to analyze the performance of the proposed model, through the simulation using MATLAB. Unless explicitly stated otherwise, the simulation parameters shown in Table 1 are used. In our simulation, there are three values for \( N \). The work of Li et al. in\textsuperscript{21} is selected as a comparison group which is the situation of \( N = 1 \).

The simulated data processing network is shown in Figure 4 and generated randomly. We can see that \( S = \{v_1, v_2, v_3, v_4, v_5\}, D = \{v_{24}, v_{25}\} \). For each flow, a path from \( S \) to \( D \) should be selected. If an appropriate path cannot be got, the flow transport request would be declined. To have an insight into the entire \( G \), average \( ROR \) for nodes and edges, the number of accepted flow requests are selected as evaluation indicators.

The results are shown in Table 2. With the increase of \( N \), the average \( ROR \) for nodes and edges both increase. No matter nodes or edges, the increase of...
average ROR is not in proportion to the increase of $N$. The reason for this phenomenon is that nodes and edges are both involved in the path calculation. Once the available resource is not enough for supporting a flow path, flow requests would be declined. When $N = 2$, the number of accepted flow requests is 4, not like that of $N = 0.5$ and $N = 1$. Because the data transfer rate in node $v_{12}$ exceeds its capacity.

### Table 2. Average ROR for nodes and edges.

| $N$  | Average ROR for nodes | Average ROR for edges |
|------|-----------------------|-----------------------|
| 0.5  | 25.0%                 | 8.3%                  |
| 1    | 41.7%                 | 13.3%                 |
| 2    | 45.0%                 | 23.3%                 |

### Table 3. Experiment parameters.

| Component    | Version      | Purpose                                      |
|--------------|--------------|----------------------------------------------|
| CentOS       | 7.7.1908(Core) | Server basic operation system                |
| Java         | 1.8.0_232    | Development environment for data processing script |
| Maven        | 3.0.5        | Project version management tool              |
| Flink        | 1.11.0       | Data processing platform                     |
| Docker       | 19.03.8      | Container environment                        |
| Docker-compose | 1.25.5   | Container orchestration command line         |

### Experiment

To verify the feasibility of our model, we experimented in the real world. The experiment corresponding components’ purposes are listed in Table 3. Note that this experiment was executed on a server with 2.60 GHz CPU, 16 G memory, and 1 TB hard disk.

In Figure 5, a Flink cluster was started up in the form of multiple containers. Each container completes a task in data processing, on its own. At the same time, the status of multiple containers is also shown in Figure 5, with the information of communication ports.

In Figure 6, the workflow of a data processing task is displayed. A window word count task was executed on the base of the Flink platform. The specific task ID, start time, duration, status, received, and sent Bytes can also be seen.

### Conclusion

To solve the problem of data scale explosive growth and services’ increasing requirement, we proposed a load balance oriented data processing mechanism for bounded and unbounded data in smart cities. The position, function, and workflow of data processing were introduced in a smart city framework. Base on defined basic concepts, a load-balanced data processing mechanism, including LBR algorithm, was proposed to get the path with the minimum resource occupancy ratio. Through the numerical simulation and the experiment, the superiority and feasibility of our work had been proved. In our future work, further research on the handle, migration, and optimization of data processing in smart cities will go on.

![docker-compose ps](image_url)

**Figure 5.** Successful startup of multiple containers.
Author’s note
Jianwu Li is now newly affiliated to Beijing Institute of Technology, Advanced Research Institute of Multidisciplinary Science, Beijing, China.

Declaration of conflicting interests
The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is supported by R&D Program of Beijing Municipal Education Commission (Research on Optical and Wireless converged Access Network Networking Technology in Smart Traffic, No. KM20211147010), China Computer Federation (CCF) Opening Project of Information System (Research on Massive Event Flow oriented Stream Computing Framework, No. CCFIS2019-01-01). And thanks to Ms. Li.

ORCID ID
Qinglong Dai https://orcid.org/0000-0001-8812-3416

References
1. Ejaz W, Naeem M, Shahid A, et al. Efficient energy management for the internet of things in smart cities. IEEE Commun Mag 2017; 55(1): 84–91.
2. Nassar MA, Luxford L, Cole P, et al. The current and future role of smart street furniture in smart cities. IEEE Commun Mag 2019; 57(6): 68–73.
3. Zhao L, Wang J, Liu J, et al. Routing for crowd management in smart cities: A deep reinforcement learning perspective. IEEE Commun Mag 2019; 57(4): 88–93.
4. Santos J, Vanhove T, Sebrechts M, et al. City of things: Enabling resource provisioning in smart cities. IEEE Communications Magazine 2018 2018; 56(7): 177–183.
5. Growth Enabler. Discover Key Trends & Insights on Disruptive Technologies & IoT innovations. Report, UK, 2017.
6. Abie H. Cognitive cybersecurity for CPS-IoT enabled healthcare ecosystems. In: 2019 13th International Symposium on Medical Information and Communication Technology (ISMICT), Oslo, Norway, 8-10 May 2019, paper no. 8743670, pp. 1–6. New York: IEEE, 2019.
7. Stoyanova M, Nikoloudakis Y, Panagiotakis S, et al. A survey on the internet of things (IoT) forensics: Challenges, approaches and open issues. IEEE Commun Surv Tutor 2020; 22(2): 1191–1221.
8. Apache Software Foundation. What is apache Flink? https://flink.apache.org/flink-architecture.html (2019, accessed 20 January 2020).
9. Havers B, Duvignau R, Najdattai H, et al. Driven: a framework for efficient data retrieval and clustering in vehicular networks. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 4-11 April 2019, paper no. 201900201 pp.1850–1861. New York: IEEE, 2019.
10. Nasiri H, Nasehi S and Goudarzi M. Evaluation of distributed stream processing frameworks for IoT applications in smart cities. J Big Data 2019; 6(1): 52–52.
11. van Dongen G and Van den Poel D. Evaluation of stream processing frameworks. IEEE Trans Parallel Distrib Syst 2020; 31(8): 1845–1858.
12. Antaris S and Raphaelidis I. In-memory stream indexing of massive and fast incoming multimedia content. IEEE Trans Big Data 2018; 4(1): 40–54.
13. Chen C, Li K, Ouyang A, et al. Gflink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data. In: 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, USA, 16-18 August 2016, paper no. 101109, pp. 542–551, New York: IEEE.

14. Chen C, Li K, Ouyang A, et al. GPU-accelerated parallel hierarchical extreme learning machine on Flink for big data. IEEE Trans Syst Man Cybern Syst 2017; 47(10): 2740–2753.

15. Espinosa CV, Martin-Martin E, Riesco A, et al. Flink-check: Property-based testing for apache Flink. IEEE Access 2019; 7: 150369–150382.

16. Xu C, Holzemer M, Kaul M, et al. On fault tolerance for distributed iterative dataflow processing. IEEE Trans Knowl Data Eng 2017; 29(8): 1709–1722.

17. Isah H, Abughofa T, Mahfuz S, et al. A survey of distributed data stream processing frameworks. IEEE Access 2019; 7: 154300–154316.

18. Kaitoua A, Pinoli P, Bertoni M, et al. Framework for supporting genomic operations. IEEE Trans Comput 2017; 66(3): 443–457.

19. Katragadda S, Gottumukkala R, Pusala M, et al. 2018 IEEE International Conference on Big Data (Big Data). In: Distributed real time link prediction on graph streams. New York, USA: IEEE, 2018, pp.2912–2917. https://ieeexplore.ieee.org/document/8621934

20. Zacheilas N, Dedousis D and Kalogeraki V. Scalable distributed top-K join queries in topic-based pub/sub systems. In: 2018 IEEE International Conference on Big Data (Big Data), Seattle, USA, 10-13 December 2018, paper no. BigD563, pp. 378–383, New York: IEEE 2018.

21. Bian ZL, Zhang JY, Pu C, et al. Flow-network based auto resale strategy for Flink. Journal on Communications 2019; 40(8): 85–101.