Sensors and Materials, Vol. 35, No. 1 (2023) 285–305
MYU Tokyo

S & M 3168

Approach to Urban Geospatial Monitoring Combining Sensor Web and High-performance Computing Infrastructure

Xi Zhai, Wanzeng Liu,* Ying Yang,** Xiuli Zhu,
Xinli Di, Yunlu Peng, and Tingting Zhao

National Geomatics Center of China, 28 Lianhuachi West Road, Haidian District, Beijing 100830, China

(Received October 31, 2022; accepted December 19, 2022; online published January 5, 2023)

Keywords: urban geospatial monitoring, sensor web, Apache Spark, 3D scene construction, stream computing

Urban geospatial monitoring is a dynamic early warning in the process of urban development. It reflects urban spatial changes from a geospatial perspective. Modern urban digital governance must use spatial data infrastructures, integrate multisource data, and implement dynamic real-time monitoring in 3D space. To meet the real-time monitoring requirements of urban geographic space in 3D environments, the rapid construction of 3D scenes, the rapid processing of monitoring information, and dynamic process simulation have become key challenges. This paper presents an approach to urban geospatial monitoring that combines a sensor web and a high-performance computing infrastructure. The approach leverages Apache Spark and stream computing to transform the traditional processing algorithm into a data retrieval and analysis process supported by high-performance computing, which drives the rapid construction of 3D scenes and the dynamic calculation of sensor web observations. The effectiveness of this method is demonstrated through a case of urban waterlogging monitoring.

1. Introduction

Urban geospatial space is the basis of urban development and the background resource of urban planning. Monitoring and analyzing urban geospatial space are of great significance for optimizing the spatial layout, preventing disasters, and promoting urban digital governance.[1,2] Urban geographic space is a complex 3D space.[3] To meet the needs of urban geospatial 3D scene monitoring, the rapid construction of 3D scenes, the rapid extraction of information on changes, and dynamic process simulation have become key challenges for researchers.

Geographic information service (GIS) technology is a 3D, dynamic, and intelligent development trend. GISs are moving from 2D to 3D, aiming to provide people with more intuitive and realistic scene description capabilities.[4] The research on traditional GISs in data processing/computing, scene description, and other aspects is mainly focused on 2D services. After years of development, a service system with perfect functions and mature applications has

*Corresponding author: e-mail: lwz@ngcc.cn
**Corresponding author: e-mail: yangying@ngcc.cn
https://doi.org/10.18494/SAM4216

ISSN 0914-4935

been formed. The 2D scene description service represented by the Web Map Service (WMS) and Web Map Tile Service (WMTS) has been widely used in key projects at home and abroad, such as Tianditu, Google Earth, and OpenStreetMap.[5–7] However, these 2D service products abstract 3D scenes to a 2D plane display, lose 3D space information, and lack the intuitive description of scenes, making them inconducive to a timely and effective understanding of the real information of depicted objects. Therefore, it is necessary to carry out urban monitoring through 3D scenes.

There are two main deficiencies in carrying out urban monitoring in 3D space. Firstly, the research on the rapid construction of specific 3D monitoring scenes is insufficient, and the main problem is the lack of technical means to quickly retrieve 3D data. In a distributed web environment, 3D model data are stored in different network nodes. Therefore, an efficient query method must be designed to quickly obtain the 3D data in a specified area when describing a 3D scene. The second deficiency is the inadequate real-time calculation methods of 3D dynamic variables in the environment of a sensor web. The fundamental requirement of urban geospatial monitoring is to discover changes in information in a timely manner. The development of a sensor web generates a massive dynamic data flow, which brings opportunities and challenges in the timely discovery of urban change information. Traditional data processing technologies, such as offline data processing and data batch processing, must store data before processing, which cannot meet the requirements for the real-time processing of dynamic data. In this paper, we attempt to combine a sensor web with high-performance computing to achieve dynamic and real-time 3D urban geospatial monitoring.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 introduces the method of urban geospatial 3D scene construction. Section 4 introduces sensor web observation management and stream computing in detail. In Sect. 5, a case of urban waterlogging monitoring is provided. Section 6 concludes this work and gives our future work.

2. Related Work

The key to implementing urban geospatial monitoring in 3D space is to quickly find the required data according to the construction requirements of specific 3D scenes and calculate variable information in real time based on the dynamic data flow of a sensor web. Such monitoring requires sensor web technology and high-performance computing technology. In recent years, there have been some promising research achievements toward achieving 3D urban geospatial monitoring.

The International Open Geographic Information Union (OGC) has successively launched the Web 3D Service (W3DS) and 3D Portrayal Service (3DPS) to realize the description and visualization of 3D scenes.[8] In addition, regarding sensor webs, since 2006, OGC has developed a series of sensor web service specifications and information models, forming a systematic theoretical method for defining sensors and sensor observation codes and service interfaces.[9] After nearly ten years of development, sensor webs have developed into a global shared and interoperable web service system, which can obtain real-time observations. The introduction of sensor webs has enabled GISs to develop from “static services” to “dynamic services”.[10]

Through the real-time perception of the real world and data acquisition, dynamic data streams can be provided, which can provide a massive amount of spatio-temporal dynamic information for environmental monitoring, resource management, ecological analysis, disaster emergencies, and other applications.[11]

With the development of a number of high-performance technologies represented by Hadoop, Spark, and Spark Streaming, real-time or near-real-time spatial querying and analysis have become possible.[12–14] These technologies greatly improve the computing efficiency. Benefiting from the high-performance computing framework, numerous spatial data retrieval strategies have been proposed and many data management platforms with excellent performance have emerged.[15,16] A breakthrough has also been made in the instant processing of multisource heterogeneous spatial data.[17] However, most of the research on high-performance computing for large-scale data has been on batch processing schemes for offline historical data, and the research objects have basically been static data, and there has been insufficient research on real-time computing for dynamic streaming data.

3. Construction of Urban Geospatial 3D Scene

To build a dynamic 3D scene, three problems must be solved: 3D data query, real-time extraction of variables in the scene, and dynamic scene construction. Focusing on the query requirements of 3D model data, in this paper we select a 3D data query method based on the Spark framework by comparing the query performance of the Hadoop and Spark frameworks. There are two steps in constructing the urban 3D scene: building the basic scene and adding dynamic variables. The response result of the GetScene request of the OGC 3DPS standard is returned as an extensible 3D (X3D) file. We follow the OGC 3DPS specification, which is conducive to the service release of scenario description results. In this study, the process of building 3D scenes is that of building X3D files.

3.1 Data preprocessing

High-performance queries on 3D model data must store the data to be processed in a distributed file system (HDFS), which is used as the data source for queries. There are various types of data for 3D terrain models, including City Geography Markup Language (CityGML), 3D Studio Max (3DX), OBJ, Keyhole Markup Language (KML), and X3D. The data size of the single 3D model established according to the above data format generally does not exceed 10 MB. Because of its operation mechanism, the HDFS is not suitable for storing a large number of small data files. Therefore, when a large amount of 3D terrain model data is transferred to the HDFS module for storage, data preprocessing is first required, and a large number of small 3D data files are integrated into a large dataset file. The preprocessing of 3D model data in this study includes four steps: parsing data files, calculating the bottom range of a single mode, coordinate conversion, and generating preprocessing data files.

3.1.1 Parsing data files

The 3D figure model data include Extensible Markup Language (XML)-based data (such as CityGML, KML, and X3D), from which the figure identification (ID), point coordinates of the figure, and other information can be obtained by parsing the XML file. The 3DS file is a binary file. The ID, figure center point, and other information in the 3DS information block can be obtained through program reading.

3.1.2 Calculating bottom range of single mode

Generally, the spatial range is used as the filter condition to query the model data required to build a 3D scene. Therefore, the bottom range of the single model must be extracted in the data preprocessing process. Different formats of model data have different methods for calculating the bottom range. This paper only considers 3DS data and CityGML data.

3D Max produces 3DS model data according to the existing base map vector data to ensure the accuracy of the spatial position of model data. By reading the 3DS data position vector structure, the coordinates of the center point of each model can be obtained. Then, taking the center point coordinates as the origin, a certain distance is extended along the X and Y axes to form a rectangular box as the bottom range of the single model.

GityGML uses the boundary representation model to provide a description of 3D model objects. To extract the building bottom range in the GityGML data, it is necessary to parse the GityGML document and decompose the GityGML data into several independent building models. Then, the polygon 3D coordinate point sequence is extracted under the GroundSurface node.

3.1.3 Coordinate conversion

To realize the spatial query of 3D model data, it is necessary to ensure that the coordinate point sequence constituting the bottom range of the figure and the spatial range coordinates as the query filter conditions are in the same coordinate system. In this study, in the process of 3D data file preprocessing, the coordinates of the extracted bottom edge range of the ground objects are converted into the coordinate form under the World Geodetic System 1984 (WGS84) geodetic coordinate system.

3.1.4 Generating preprocessing data files

After preprocessing, the 3D model data are not the model data itself in the generated big data file. Therefore, the mapping relationship between the model ID and the data storage address should be established. When writing 3D model information to a new data file, the storage address of data is queried by its ID according to the mapping relationship table. As shown in Fig. 1, we manage the basic information of the model using XML documents. The unified use of Building node management is conducive to simplifying the processing of subsequent

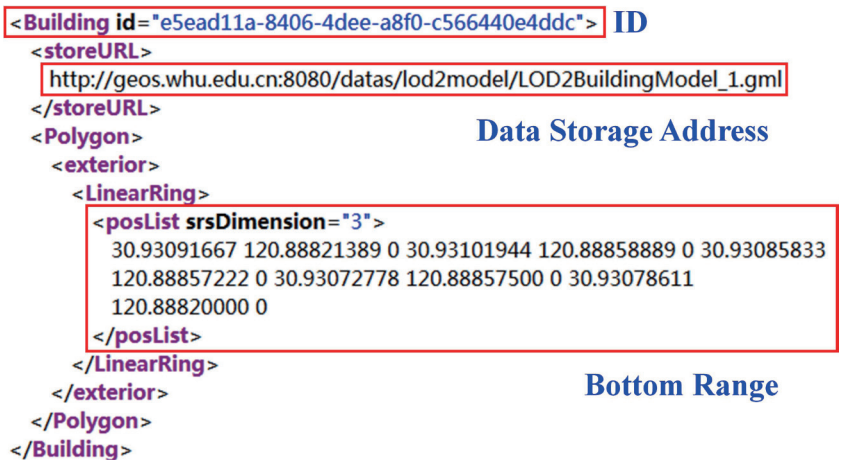


Fig. 1. (Color online) XML fragments for recording individual model.

preprocessing-generated files. The storeURL node stores the data address, and the Polygon node stores the bottom range of the model.

3.2 High-performance query method for 3D data

For the processing flow of 3D model data, the Hadoop and Spark frameworks provide two different schemes. Hadoop and Spark do not support the processing of a large number of small data files. In view of this, we design a preprocessing scheme for 3D model data of 3DS and CityGML, and we use XML structure documents to manage the ID, storage address, and bottom range information of ground objects, which facilitates subsequent query processing operations. We make a comparative evaluation of the query performances of Hadoop and Spark from three aspects: cluster size, query data size, and query algorithm iteration complexity. The results show that Spark has better performance than Hadoop in the field of geographic services. On the basis of the performance evaluation results, a 3D data query method based on the Spark framework is proposed to provide model data for the subsequent construction of 3D geospatial scenes.

3.2.1 Comparison of query performance between Hadoop and Spark

Hadoop is a distributed large-scale data storage and processing framework, which can provide data storage management and batch processing functions. It is suitable for data mining, statistics, analysis, and other application scenarios of large static data. Hadoop is a disk-based batch processing mode with a large number of network transmission and disk input/output (I/O) operations. Spark is a cluster computing framework based on memory computing that is designed and implemented on the basis of Hadoop. It performs operations on the distributed elastic dataset (RDD) of the cluster. It aims to improve the speed of large-scale data analysis and calculation, and it is suitable for batch processing, streaming data processing, interactive analysis, and other application scenarios. To evaluate the query performance of the Hadoop and

Spark frameworks in a distributed environment, we conduct experimental tests from three aspects: cluster size, query data volume, and query algorithm iteration complexity, and we compare the operation performance of Hadoop and Spark clusters.

3.2.1.1 Evaluation experiment design

The experimental data are generated by data preprocessing, and the data format is an XML document (Table 1). This document contains <Building>…</Building> fragments of hundreds of thousands to millions of records. The model ID, the bottom range of the model, and the data storage address corresponding to each 3D model are encapsulated in a Building node. To make a full performance comparison, five groups of data are generated using script batch writing, and the data sizes are 1.47, 2.94, 4.41, 5.88, and 7.35 GB.

All performance tests are conducted in a local area network (LAN) environment built by 40 virtual machines. Ten Dell desktops are used to build the LAN. Each desktop is equipped with a 3.60 GHz Intel i7 4 core processor, 8 GB memory, 1 TB disk storage space, and a Microsoft Windows 8.1 64-bit operating system. Each desktop uses VMware 10.0.1 software to deploy four virtual nodes, and each virtual node uniformly has the Ubuntu 12.04 Linux 64-bit operating system installed. The Hadoop cluster deploys a master node and 39 slave nodes. The master node is equipped with 2 GB memory and 40 GB disk storage space, and the slave node is equipped with 1 GB memory and 20 GB disk storage space. The Spark cluster deploys a master node and 39 worker nodes. The configuration scheme of the master and worker nodes is the same as that of the Hadoop cluster. The Hadoop cluster adopts Hadoop version 0.23.11, the HDFS data block size adopts the default value of 64 MB, and the cluster bandwidth in the LAN is 100 Mbps. The Spark cluster adopts Spark 2.0, and it relies on the HDFS distributed file system of the Hadoop cluster to manage data. The data blocking strategy is the same as that of the Hadoop cluster. A standalone independent cluster manager is used.

Table 1
Basic information of three-dimension model preprocessing test data.

File format	XML file	
Single building model fragment	<Building id="e5ead11a-8406-4dee-a8f0-c566440e4ddc"> <storeURL>http://geos.whu.edu.cn:8080/datas/lod2model/LOD2BuildingModel_1.gml</storeURL> <Polygon> <Exterior> <LinearRing> <posList srsDimension="3">30.93091667 120.88821389 0 30.93101944 120.88858889 0 30.93085833 120.88857222 0 30.93072778 120.88857500 0 30.93078611 120.88820000 0</posList> </LinearRing> </Exterior> </Polygon> </Building>	
Test data	Data size: 1.47 GB	385351 building model records
	Data size: 2.94 GB	770703 building model records
	Data size: 4.41 GB	1156055 building model records
	Data size: 5.88 GB	1541406 building model records
	Data size: 7.35 GB	1926758 building model records

Three types of experiments (Types A–C) are designed to test the performance of the Hadoop and Spark clusters. In each experiment, the bottom range of the building model is extracted and the spatial relationship with the query conditions is calculated. Type A: Set the query algorithm to perform only one iteration, and conduct four sets of comparative tests on the 1.47 GB test data to compare the query efficiency of Hadoop and Spark clusters composed of 6, 10, 20, and 40 nodes. Type B experiment: Set the query algorithm to perform only one iterative operation. On the Hadoop and Spark clusters consisting of 40 nodes, perform five sets of performance tests on the five different test data to compare the performance of Hadoop and Spark query data with different sizes. Type C: On the Hadoop and Spark clusters consisting of 40 nodes, the query data volume is set to 1.47 GB, and the 3D data query algorithm is configured to perform 1, 4, 8, and 12 iterations. Four groups of tests are conducted to compare the performance of the Hadoop and Spark frameworks in the face of iterative operations with different levels of complexity. To avoid the influence of accidental factors, all the test times are the average of 10 test times.

3.2.1.2 Experimental results and analysis

As shown in Fig. 2, the Spark cluster is more efficient than the Hadoop cluster for processing test data of the same size with different cluster sizes. With an increasing number of cluster nodes, the rate of performance improvement of the Hadoop and Spark clusters decreases, and the slowdown of the Hadoop cluster is greater than that of the Spark cluster. As the number of nodes in the Spark and Hadoop clusters increases, their communication costs increase. However, Spark is based on the memory computing mode and it has fewer network transmission and disk I/O operations than Hadoop. As the number of cluster nodes increases, the communication cost of the Spark cluster is less than that of the Hadoop cluster.

According to the test times illustrated in Fig. 3, when the 1.47 GB data query is executed, Spark is 29.8 s faster than Hadoop. With increasing data size, the time consumption difference

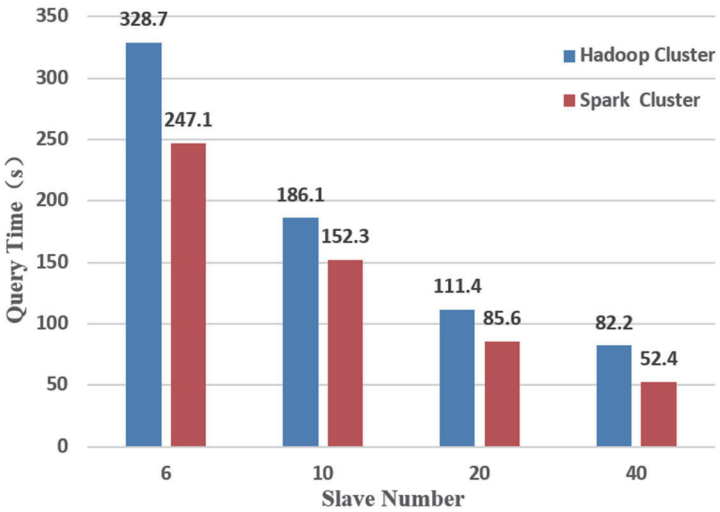


Fig. 2. (Color online) Comparison of query times between Hadoop and Spark for different cluster sizes.

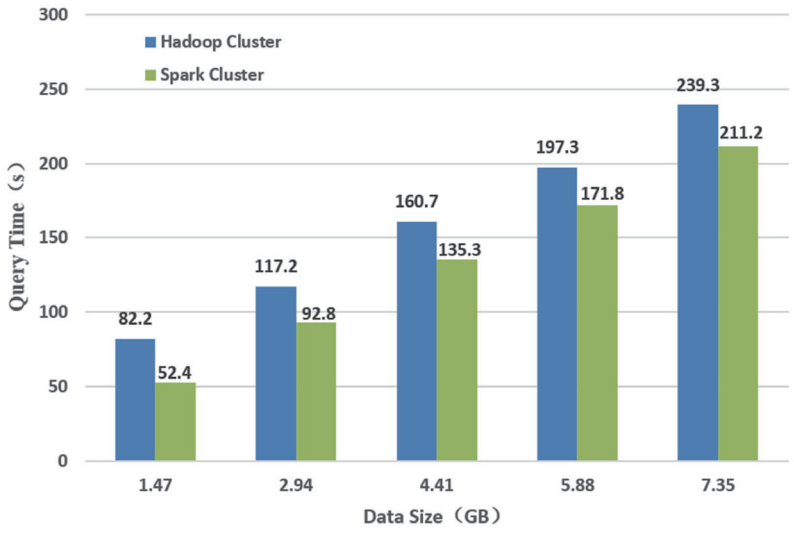


Fig. 3. (Color online) Comparison of query times between Hadoop and Spark for different data sizes.

between Spark and Hadoop fluctuates very little. Therefore, it can be concluded that when the cluster size and the iteration complexity of the query algorithm are the same, the Spark cluster is always better than the Hadoop cluster in terms of data query for test data of different sizes.

The performance test is conducted on 40-node Hadoop and Spark clusters with the 1.47 GB test data in the C-type experiment. The experimental results are shown in Fig. 4. When executing a query with one algorithm iteration, Spark is 32.8 s faster than Hadoop. As the number of algorithm iterations increases, Spark's advantageousness gradually increases. When executing queries with 12 algorithm iterations, Spark is 594 s faster than Hadoop. Therefore, it can be concluded that with the same cluster size and test data size, the performance advantage of the Spark framework over the Hadoop framework increases with the number of data query algorithm iterations. This is because Spark caches the data in the memory for the calculation when running the driver program for iterative calculation. It, therefore, does not need to read the data for each iteration, reducing the amount of data transmission and disk operations. In the process of executing iterative operations, Spark turns the RDD directed acyclic graph (DAG) into a physical execution plan. In this process, Spark optimizes the logical plan, combines the steps that can be executed in a pipeline, and simplifies the operation process. Therefore, Spark performs well in the implementation of algorithms that converge repeatedly.

We have found that the Spark framework outperforms the Hadoop framework in the process of 3D spatial data query. Therefore, in the process of 3D geospatial construction, the Spark framework is adopted as the high-performance computing infrastructure.

3.2.2 Method for 3D data query based on Spark framework

In this paper we propose a method of 3D data query based on the Spark framework, which can reduce disk I/O access and communication costs through memory computing, thus improving the efficiency of 3D data query. The Spark cluster adopts the master-worker

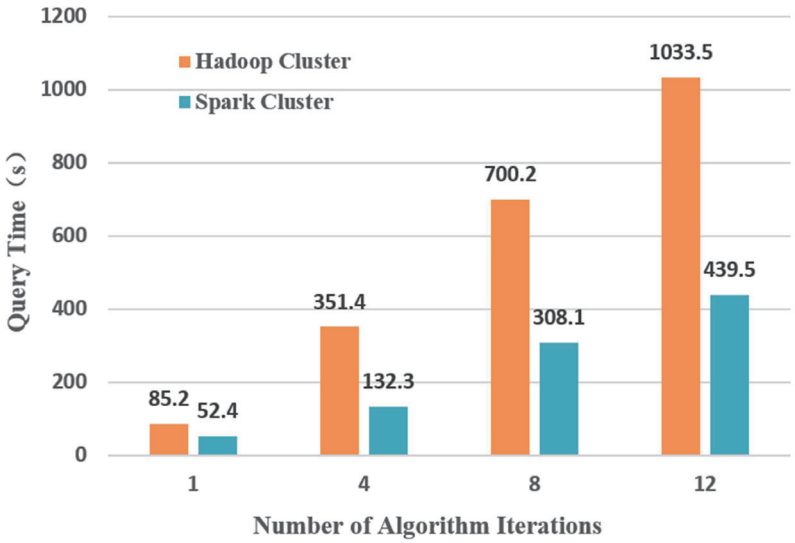


Fig. 4. (Color online) Comparison of query times between Hadoop and Spark for different algorithm iterations.

architecture and usually has one master node and multiple worker nodes. The master node is responsible for managing the cluster by coordinating and scheduling cluster resources. The worker node is responsible for managing the resources of the node and reporting the resource utilization to the master node in the mode of "heartbeat communication" during the running of Spark. The Spark cluster has two main types of application processes at runtime: a driver main process and several executor subprocesses. The driver main process runs on the master node and is responsible for transforming the application into an executable task, managing the executor subprocess, and scheduling task execution. The executor subprocess runs on the worker node, is responsible for executing a specific business logic code, and returns the query results to the driver main process.

For 3D model data query, the ModelInputFormat class is introduced by inheriting the InputFormat class, which is used to preprocess file segmentation and reading and to generate<key_{map}, $value_{map}$>. The ModelInputFormat class mainly implements two abstract methods, getSplits and createRecordReader. getSplits solves the problem of data file segmentation. It records the starting position, data length, node list, and other information of each partition split according to the size of the HDFS data block. The results are saved in the List <InputSplit> set and transferred to createRecordReader. createRecordReader reads data from List <InputSplit> to solve the problem of reading the data in the partition. The key value pairs generated by createRecordReader() are <key, value>. The key is the starting position of each building in the preprocessed build file, and the value is the XML fragment embedded in the <Building></Building> node. These <key, value>={start position | Building Node XML Fragment} key value pairs are used as the input data of the map function.

The 3D data query method based on Spark is designed and implemented around the creation of 3D data RDD, RDD filter mapping, and RDD actual calculation. Figure 5 shows the design of the 3D data query process based on the Spark framework.

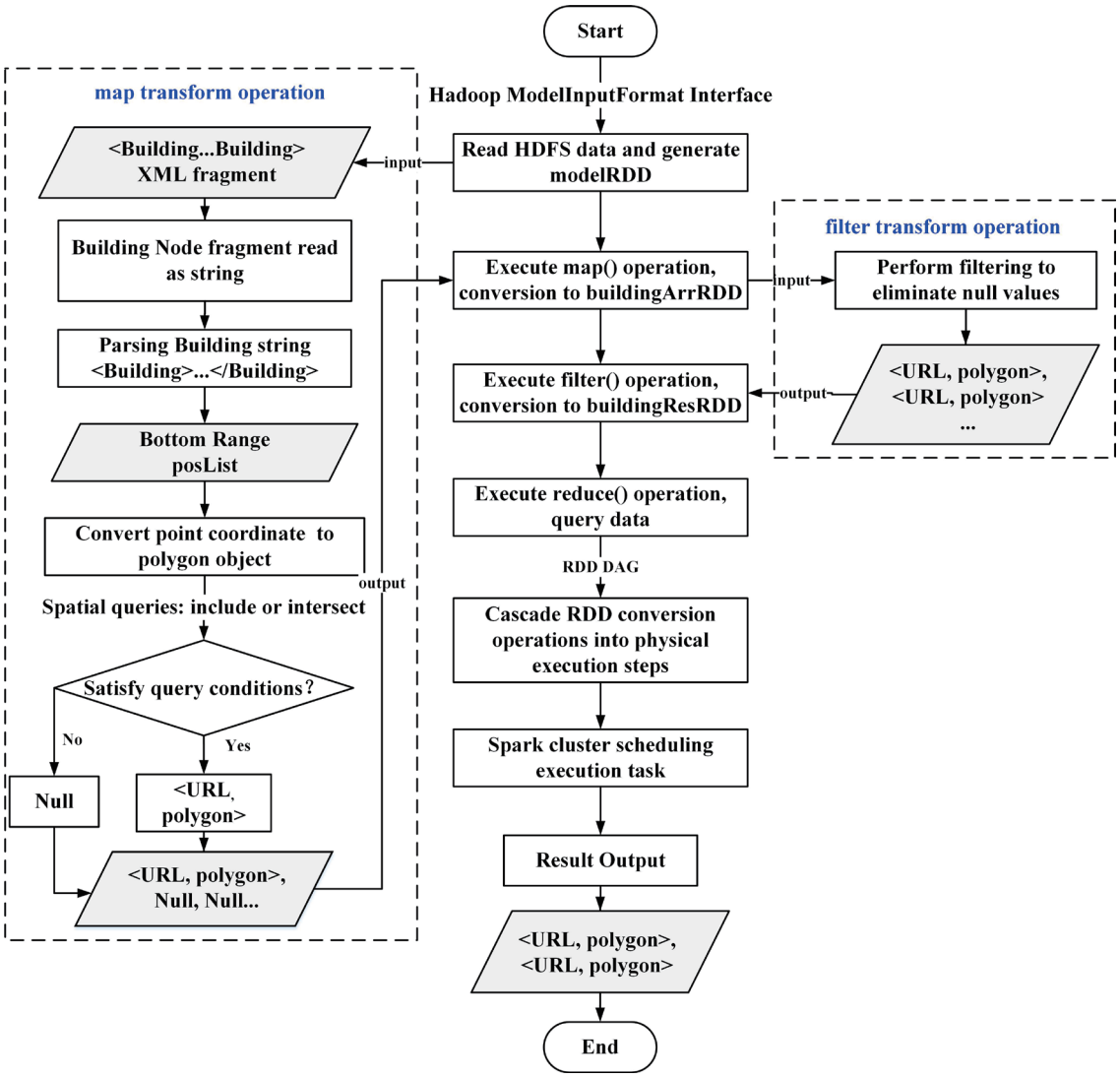


Fig. 5. (Color online) Design of 3D data query process based on Spark framework.

3.2.2.1 Creation of 3D data RDD

The Spark framework is developed on the basis of Hadoop. It is designed to support the InputFormat and OutputFormat interfaces of the MapReduce model to read file format data and data stored in the HDFS, Hadoop database (HBase), and other storage systems. By implementing the ModelInputFormat interface for reading 3D model data, the Spark distributed cluster can directly read data blocks from the HDFS to create an RDD of 3D model data. modelRDD=sc. newAPIHadoopFile [LongWritable, Text, ModelFormat] (fileIn) is invoked to create a modelRDD. fileIn is the path of data in the HDFS. The partition in modelRDD corresponds to the data block of each worker node in the HDFS. When the data are read into Spark, the data are split into data blocks and read to generate <key, value> key value pairs.

3.2.2.2 RDD filter mapping

Execute map transform operation: this process performs a spatial query on all 3D model objects in modelRDD and maps them to a new buildingArrRDD. When Spark creates a modelRDD, it generates a series of <Building Start Position, Building Node XML Fragments> key value pairs. The value "Building Node XML Fragment" is read as a string by the program in turn, and the <Building>...</Building> string is parsed to extract the building ID, storage address uniform resource locator (URL), and bottom range posList of the 3D model. The 3D point coordinate sequence in the bottom range posList is converted to a polygon object. The polygon object calculates the spatial relationship with the given spatial query conditions to determine whether there is an intersection or inclusion relationship. If the query conditions are met, a <storage address URL, bottom range polygon> key value pair is output. If the query conditions are not met, a null value is output. This point is different from the implementation process of the map task under the Hadoop framework. The map function of Hadoop MapReduce directly filters and outputs the data that meet the query criteria. However, the map transform operation of the Spark framework is used to map the elements in the modelRDD one by one to a new RDD. Therefore, there are both qualified output data <URL, polygon> and null values in the new RDD. The data in the new buildingArrRDD dataset are {<URL, polygon>, null, null, <URL, polygon>, null...}, and the buildingArrRDD partition still corresponds to the data block in the HDFS.

Execute filter transform operation: this process filters all the data in the buildingArrRDD according to the set conditions to obtain a new buildingResRDD. The application traverses the data in the buildingArrRDD dataset, removes null values, and generates new data in the buildingResRDD dataset of {<URL, polygon>,<URL, polygon>,...}. At this time, the data volume of buildingResRDD decreases. It is necessary to perform a repartition operation, set a reasonable partition, and return the RDD of the specified new partition. Reducing the number of partitions will cause errors in the node reduction program running the calculation. It is necessary to set shuffle=true to increase the shuffle operation. Repartition is the default shuffle=true repartition operation.

3.2.2.3 RDD actual calculation

The map operation and filter operation of the 3D data modelRDD implicitly generate the DAG, and the scheduler combines the map and filter into pipeline execution steps. When buildingResRDD encounters the reduced operation, the executor process on the worker node reads the data block in the HDFS into the memory and runs the query and filtering of 3D model data in parallel in the memory. All queries and filters are completed, and the calculated results <URL, polygon>={3D data storage address | bottom range} are returned to the driver main process and written to the console for output.

3.3 Construction of 3D scene

For the subsequent service release of 3D scenes, the 3D scene construction method proposed in this paper is based on the OGC 3DPS specification. The visualization of a scene involves rendering and displaying the X3D file returned by the GetScene operation; thus, the process of building a 3D scene is that of building an X3D file. The construction of an urban geospatial 3D scene is divided into two steps: building a basic scene and adding dynamic variables.

3.3.1 Build basic 3D scene

The nodes in the X3D file have strict hierarchical relationships. Each X3D file contains a scene root node, and the remaining geometry nodes, rendering effect nodes, texture painting nodes, group nodes, and so forth, are nested under the scene root node. When describing geometric figures in 3D scenes, it is necessary to use the shape model node. One or more figure models are allowed under each shape model node. Shape uses the appearance node as a sub-node to define the object's appearance attributes such as its material, texture, and texture coordinate transformation. Geometric nodes are used to express the geometric shape and position of objects. The key step in the process of constructing a 3D scene is the addition and position transformation of geometric nodes. The most commonly used solid geometric node in a 3D scene is the IndexedFaceSet geometric surface node, which is the basic unit of a 3D scene. The IndexedFaceSet contains four sub-nodes: coordinate, normal, TextureCoordinate, and color. Among them, coordinate, normal, and color belong to the rendering effect node, and TextureCoordinate belongs to the texture painting node. The coordinate node stores the 3D coordinates of 3D geometric objects. The coordinates under each coordinate node form the face represented in the IndexedFaceSet. The normal node records the normal vector for lighting effect rendering. The color node can specify the spatial color of solid geometric faces, which is applicable when there is no texture. The TextureCoordinate node has embedded texture coordinates that specify the location of the texture map. A 3D scene often contains multiple shape model nodes. The organization of these nodes depends on the transform node. The X3D file uses the Cartesian coordinate system. Each transform node creates a sub-coordinate system to determine the center point position of the figure geometry translation, rotation, and scale. In fact, when building a 3D scene, the "translation" field value information under the transform node is used to lay out the locations in the figure model.

The so-called basic 3D scene is a scene within the specified geographical space range without overlapping spatial variables. The process of building a basic 3D scene involves combining and laying out the model data in the specified space obtained by high-performance queries. We design six basic scene objects: buildings, vegetation, pools, bottom planes, street lamps, and railings (Fig. 6). Each object is organized under a separate transform node according to its location. When converting the original data into X3D data in batches, the center point coordinates of each figure model are set to the origin of the Cartesian coordinate system "translation = '0 0 0'", and the center point of the basic scene is also set to the origin of the Cartesian coordinate system. When the figure object is laid out according to the spatial

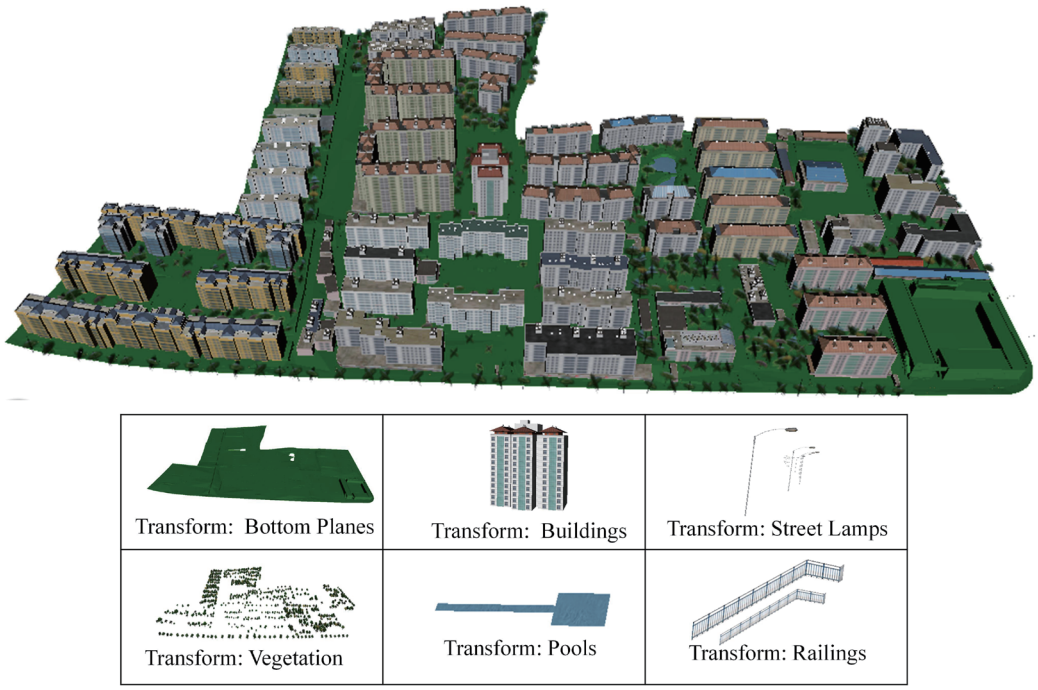


Fig. 6. (Color online) Basic 3D scene.

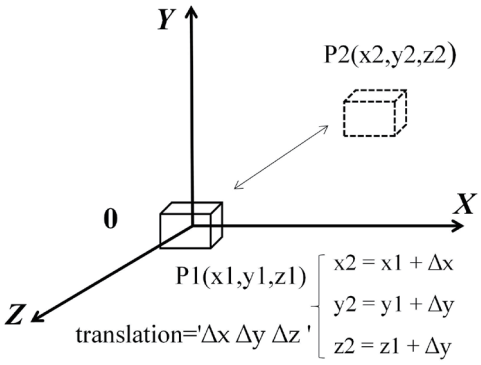


Fig. 7. Coordinate translation of model object in Cartesian coordinate system.

relationship between the actual center point position of the figure and the actual center point position of the basic scene, the coordinates of the center point of the figure object are determined in the scene "translation = 'Δ x Δ y Δ z'", and then the coordinates of all the 3D points that make up the model are translated according to the translation parameters in the translation (Fig. 7).

3.3.2 Dynamically add spatial variable objects

The spatial variable objects refer to the data that change over time and directly or indirectly affect the visualization effect of 3D scenes in an urban space. These data are time sensitive and change frequently in the process of urban geospatial monitoring. After the basic 3D scene is

generated, spatial variable objects must be added dynamically, taking the construction of an urban waterlogging scene as an example. When visualizing the waterlogging scene, the most intuitive 3D variable is the change in the waterlogging range. The waterlogging range obtained from waterlogging treatment is added to the basic scene as a geometric object. The ponding range is actually the point sequence coordinates of the vector polygon. The coordinate conversion function in the geospatial data abstraction library (GDAL) is used to convert the point sequence coordinates of the vector polygon to the coordinates in the Cartesian coordinate system. Because the ponding range object is for the entire 3D scene, it is not necessary to place the ponding range in the transform node. The ponding range is usually composed of one or more ponding polygons, each of which is encapsulated in a shape node. When adding a waterlogging variable object, the waterlogging polygon shape node is directly placed under the scene root node (Fig. 8).

4. Sensor Web Observation Data Access and Calculation

The real-time access to sensor web data provides powerful data support for urban geospatial monitoring. The observation data have the characteristics of dynamic and continuity. Through the sensor web, a large amount of dynamic observation data can be obtained. For example, satellite sensor monitoring, meteorological monitoring, traffic supervision, and other sensor network systems generate large-scale observation data, and a large amount of updated data is acquired in a short time. A problem caused by this is that the traditional processing mode of storing data before computing can no longer meet the demand for the real-time computing of dynamic data in urban geospatial monitoring.

4.1 Sensor web observation data access and management

The OGC Sensor Observation Service (SOS) can register sensors according to their description, insert corresponding sensor observation data, and publish a real-time observation

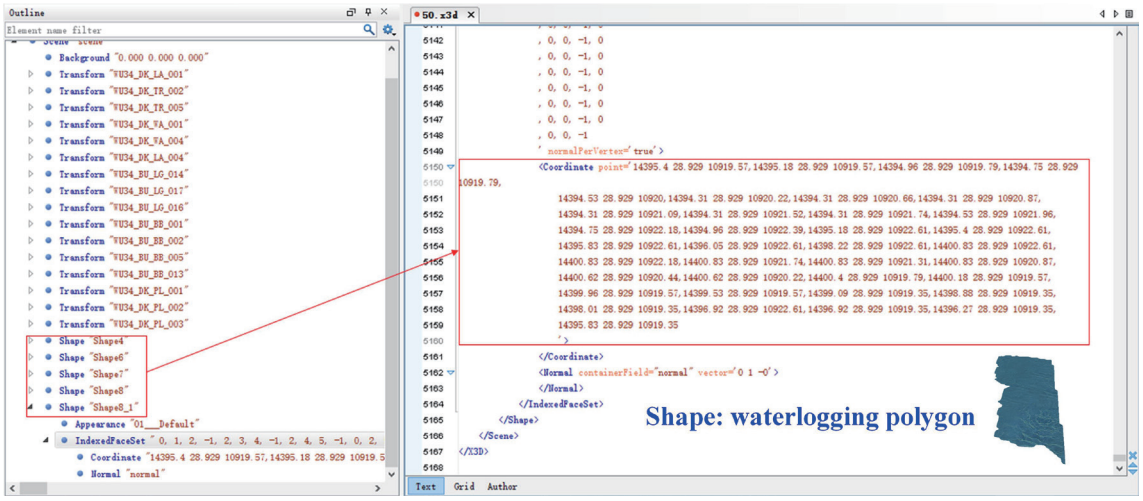


Fig. 8. (Color online) Shape node of waterlogging range variable.

data stream. Therefore, the SOS is the core component of the sensor web for realizing data access and management. We use the open-source framework 52-North SOS to build the SOS for accessing and managing sensor network data. The operations of the SOS used to implement insertion include InsertSensor, InsertObservation, InsertResult, and InsertResultTemplate. The operations used by the SOS to realize data retrieval include GetCapabilities, GetObservation, GetResult, GetResultTemplate, DescribeSensor, and GetFeatureOfInterest. The core operations include InsertSensor, InsertObservation, and GetObservation. The InsertSensor operation is used to insert sensor metadata and establish an association relationship with the corresponding observation record. The InsertObservation operation is used to insert the observation data of a sensor. The GetObservation operation is used to request observation and return results in XML format.

The access process of the observations collected by the sensor is as follows. Firstly, the InsertSensor operation is executed to register the sensor in the SOS. In this study, we provide a SensorML description file for each sensor. When registering a sensor, the metadata information of the sensor in SensorML is actually registered to the SOS. Secondly, the observed values are inserted into the configured sensor storage through the InsertObservation operation. Finally, a GetObservation request is sent to access the SOS to obtain observation data.

4.2 Calculation of observation data supported by stream computing

The 3D dynamic variables refer to elements that change over time and directly or indirectly affect the urban environment. The variables mentioned in this paper are not necessarily 3D data but generally refer to all data that affect urban scenes. These data are time sensitive and change frequently. To meet the demand for real-time computation of 3D variables in urban geospatial monitoring, in this paper we propose a real-time method of computing 3D dynamic variables supported by stream computing that combines sensor networks and stream computing technologies. Figure 9 shows the design scheme of the proposed method. First, through a series of processes such as sensor network construction, access to existing sensor network systems, sensor registration, observation data insertion, and observation value release, the real-time access method of the sensor network is used to provide dynamic observation data support. Then, the observation data of the sensor network is continuously requested, the published topic is subscribed to, the observation values are valued according to the needs of the topic, and they are converted into <key, value> key value pair data. Dynamic streaming data are published for <key, value>, and the published dynamic data stream is used as the input data of the flow calculation module. For specific application scenarios, before stream computing, users must find such computing models/algorithms and analyze whether they meet the conditions. After the user selects or improves a reasonable algorithm, the algorithm must be implemented on the basis of Spark Streaming coding for the stream computing module to call. The stream computing module continuously acquires dynamic data streams <key, value> through the receiver process, splits the data to create RDDs, then performs RDD conversion operations around the called algorithm processing flow to generate physical execution steps, and then performs the real-time calculation of 3D dynamic variables.

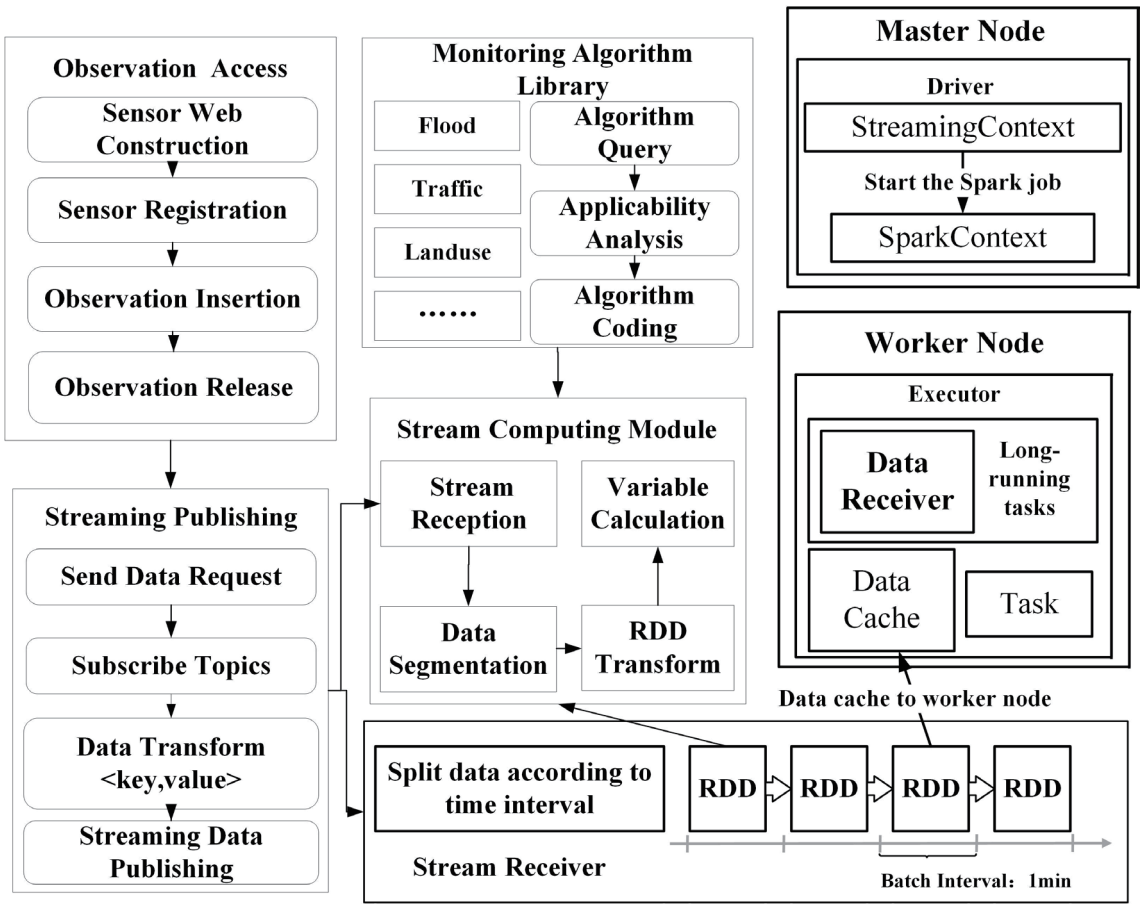


Fig. 9. Real-time computation of 3D dynamic variables supported by stream computing.

5. Case of Urban Waterlogging Monitoring

A case of urban waterlogging monitoring has been chosen to verify the validity of the proposed method. Urban waterlogging is a disaster in which high-intensity precipitation exceeds the upper limit of the urban drainage capacity. Urban waterlogging monitoring is a typical case of urban geospatial monitoring. Waterlogging monitoring requires precipitation, runoff, and other data obtained by meteorological sensors, and these data must be processed quickly. The experiment simulates the extreme situation of a local continuous rainstorm or a heavy rainstorm. The experimental area is a small district in Wuhan covering an area of about 1.5 km^2.

Firstly, according to the spatial scope of the area, the proposed method of 3D data query based on Spark is used to collect model data (CityGML) and build a basic 3D scene.

Secondly, stream computing is used to process the precipitation observation data in real time and calculate the waterlogging range. In the process of waterlogging, water accumulation is a direct factor affecting waterlogging. The ponding is the difference between the surface runoff and the drainage in the process of confluence. When waterlogging occurs in a city, the most intuitive variable is the change in the waterlogging range; this change is caused by a change in

the ponding depth. When digital elevation model (DEM) data of the local urban ponding area and ponding depth are available, the ponding range can be deduced by comparing the elevation of each grid in the DEM data with the ponding depth. Generally, there are five spatial variables in urban waterlogging: water accumulation, surface runoff, water discharge, accumulation depth, and waterlogging range.

We use the SOS to manage and publish precipitation observation data, and we use a distributed message subscription system (Apache Kafka) to stream the real-time precipitation observation obtained from the SOS. The Spark Streaming module is used to access the observation data stream obtained by dynamic query, and the 3D variable extraction calculation is decomposed into a series of waterlogging algorithm batch processing jobs. The precipitation data stream subscribed to from Kafka is split and converted into RDD data sets arranged by time series according to a time interval (1 min), and memory-based operations are directly carried out in the Spark distributed cluster to obtain urban waterlogging variables in near real time.

The waterlogging volume can be calculated by multiplying the difference between the runoff and drainage volumes by the catchment area. The actual runoff formula for the waterlogging scenario[18] can be expressed as

$$W = (Q - V) \times S, \tag{1}$$

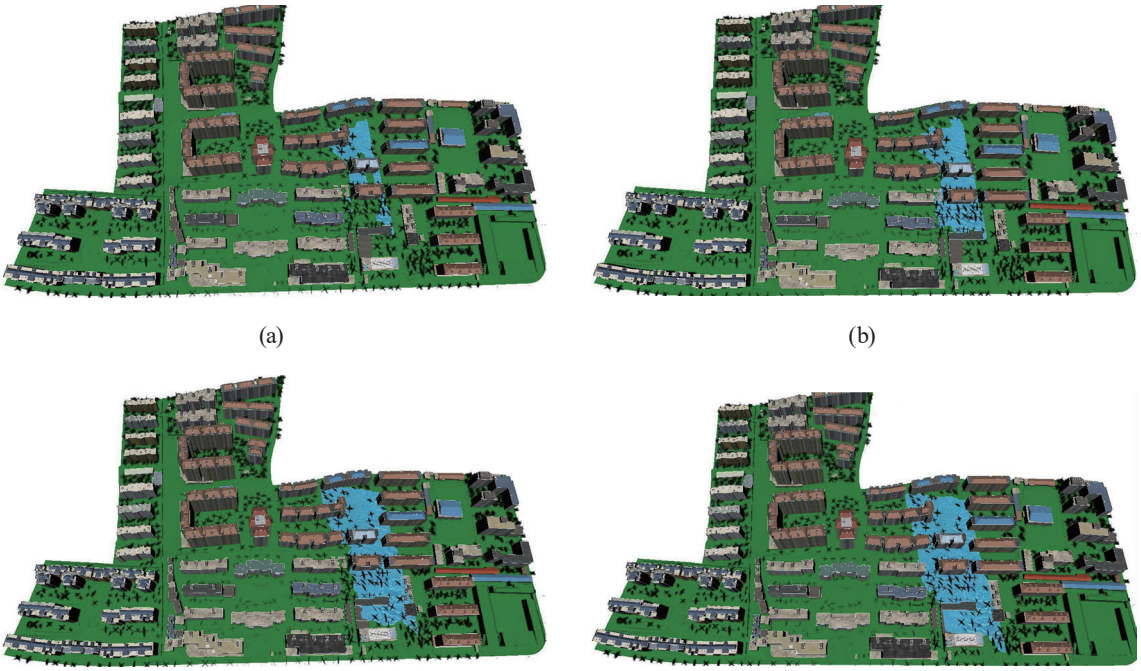
where W, Q, V, and S represent ponding, runoff, drainage, and catchment area, respectively.

Using the amount of waterlogged water in the catchment area, the equal volume method based on a GIS is used to calculate the waterlogged area and submergence depth.

$$W = \sum_{i=1}^{f(E_w, E_g)} \left[E_w - E_g(i) \right] \Delta\sigma = 0, i = 1, 2, \cdots, N \tag{2}$$

Here, W is the water accumulation in m^3, E_w is the elevation of the ponding surface in mm, $E_g(i)$ is the elevation value of the ith grid in the DEM data in mm, and $\Delta\sigma$ is the grid area in m^2. In Eq. (2), E_w is the only unknown quantity; thus, E_w can be solved through an iterative operation using dichotomy. After the ponding depth is solved, E_w is compared with the elevation values of each grid in the DEM data in turn, and all grids with elevation values lower than E_w are screened out. The several "flooded surfaces" formed by these grids represent the ponding range.

The waterlogging range is added to the waterlogging polygons, which are added to the basic 3D scene as shape nodes. The waterlogging variable extraction algorithm is executed once a minute, which can dynamically simulate the urban waterlogging changes in the experimental area over time. The four sub-images in Fig. 10 show the visualization effect of the 3D waterlogging scene corresponding to different time points, from which the dynamic changes in the waterlogging range can be viewed.



(a) (b)

(c) (d)

Fig. 10. (Color online) Waterlogging scenes when the cumulative precipitation time is (a) 50, (b) 60, (c) 70, and (d) 80 min.

6. Conclusions

The urban geospatial monitoring approach proposed in this paper combines a sensor web and a high-performance computing framework, adapts to the real-time processing of the dynamic data flow of the sensor web, and can quickly query 3D data and dynamically build 3D scenes. The case of urban waterlogging demonstrated the effectiveness of the method. In fact, this scheme provides a high-performance computing method for variable information extraction and scene reproduction in a sensor web environment, making it beneficial to explore applications of stream computing. The research results of this paper can be used to provide a more immersive and realistic user experience.

The proposed method still requires some improvements. Future works are as follows. First, we will investigate the coupling of the method with various processing models (e.g., hydrological model, land use model, meteorological model) to enhance its applicability. Secondly, by compliance with the OGC 3DPS service specification, the 3DPS service can be implemented on Servlet, which provides a dynamic description service for users by improving interfaces. Users only need to send description requests to the 3DPS service system to obtain the required urban geospatial 3D scene. Thirdly, this paper mainly focused on rapid data query and real-time computing, with little research on the visualization strategy. Investigation of the adaptive visualization strategy under a multithreading architecture and the realization of rapid visualization under high-concurrency conditions are also future work.

Acknowledgments

We appreciate the reviewers and editors for their constructive suggestions, which helped improve the quality of the paper. This work was supported by the Special Project of Science and Technology Basic Resources Survey (2019FY202503), the National Key R&D Program of China (2022YFB3904200), the Consulting Project of Chongqing Academy of Chinese Engineering S&T Strategy for Development (2022-DFZD-12), and the China Engineering Science and Technology Project "Geographic Information Professional Knowledge Service System" (CKCEST-2022-1-23).

References

1 Q. H. Weng and M. S. Wang: Photogramm. Eng. Remote Sens. **81** (2015) 692. https://doi.org/10.14358/PERS.81.9.692
2 B. G. Yang and X. B. Feng: Sci. Surv. Mapp. **32** (2007) 38. https://doi.org/10.3771/j.issn.1009-2307.2007.01.013
3 H. Chang, J. G. Gao, and P. Pan: 2009 WRI World Congr. Software Engineering (IEEE, 2009) 309–312. https://doi.org/10.1109/WCSE.2009.327
4 P. Yue, P. Baumann, K. Bugbee, and L. Jiang: Earth Sci. Inf. **8** (2015) 463. https://doi.org/10.1007/s12145-015-0229-z
5 N. C. Chen, C. J. Xiao, F. L. Pu, X. L. Wang, C. Wang, Z. L. Wang, and J. Y. Gong: Sensors **15** (2015) 2565. https://doi.org/10.3390/s150202565
6 S. Sheppard and P. Cizek: J. Environ. Manage. **90** (2008) 2102. https://doi.org/10.1016/j.jenvman.2007.09.012
7 M. Haklay and P. Weber: IEEE Pervasive Comput. **7** (2008) 12. https://doi.org/10.1109/MPRV.2008.80
8 V. Coors: ISPRS Int. J. Geo-Inf. **10** (2021) 707. https://doi.org/10.3390/ijgi10100707
9 J. Y. Gong, J. Geng, and Z. Q. Chen: Int. J. Health Geographics. **14** (2015) 1. https://doi.org/10.1186/1476-072X-14-2
10 X. Zhai, X. Y. Zhu, X. C. Lu, J. Yuan, M. Li, and P. Yue: Trans. GIS **16** (2012) 763. https://doi.org/10.1111/j.1467-9671.2012.01365.x
11 W. Wang, C. B. Hu, N. C. Chen, C. J. Xiao, and S. Jia: ISPRS Int. J. Geo-Inf. **5** (2016) 203. https://doi.org/10.3390/ijgi5110203
12 J. Dean and S. Ghemawat: Commun. ACM **53** (2010) 72. https://doi.org/10.1145/1629175.1629198
13 M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, and M. J. Franklin: Commun. ACM **59** (2016) 56. https://doi.org/10.1145/2934664
14 G. P. Gupta and J. Khedwal: Procedia Comput. Sci. **167** (2020) 2337. https://doi.org/10.1016/j.procs.2020.03.286
15 B. Chandramouli, J. Goldstein, M. Barnett, R. Deline, and J. Wernsing: Proc. VLDB Endowment **8** (2014) 401. https://doi.org/10.14778/2735496.2735503
16 F. Li, B. C. Ooi, M. T. Özsu, and S. Wu: ACM Comput. Surv. **46** (2014) 1. https://doi.org/10.1145/2503009
17 H. Rui and D. Sun: Int. J. Wireless Mobile Comput. **10** (2016) 317. https://doi.org/10.1504/IJWMC.2016.078204
18 B. Y. Shangguan, P. Yue, Z. R. Yan, and D. Tapete: Environ. Modell. Software. **119** (2019) 182. https://doi.org/10.1109/igarss.2017.8126973

About the Authors

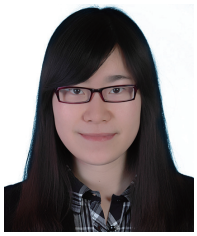
Xi Zhai received his B.S. degree in engineering of surveying and mapping from Shandong Agricultural University, China, in 2006 and his M.S. and Ph.D. degrees in cartography and geographical information engineering from Wuhan University, China, in 2012 and 2017, respectively. From 2017 to 2020, he was an engineer at the National Geomatics Center of China. Since 2020, he has been a senior engineer at the National Geomatics Center of China. His research interests are in emergency mapping, sensor webs, and spatial knowledge services. (zhaixi@ngcc.cn)

Wanzeng Liu received his B.S. degree in mine surveying, his M.S. degree in management science and engineering, and his Ph.D. degree in cartography and geographic information systems from China University of Mining and Technology in 1992, 2000, and 2005, respectively. From 1992 to 1997, he was an assistant engineer at Yongxia Mining Area Construction Management Commission, China. From 1997 to 2002, he was an engineer at Yongcheng Coal Power Group Co., China. Since 2006, he has been a senior engineer at National Geomatics Center of China. His research interests are in emergency mapping, spatial knowledge services, and geographic information systems. (lwz@ngcc.cn)

Ying Yang received his B.S. degree from Wuhan University, China, in 2007, his M.S. degree from the Chinese Academy of Sciences in 2010, and his Ph.D. degree from Wuhan University, China, in 2017. From 2010 to 2013, he was an assistant professor at the Chinese Academy of Surveying and Mapping. Since 2017, he has been an engineer at the National Geomatics Center of China. His research interests are in photogrammetry, UAV systems, and sensors. (yingy@whu.edu.cn)

Xiuli Zhu received her B.S. degree from Wuhan Technical University of Surveying and Mapping and her M.S. degree from Wuhan University, China, in 1998 and 2001, respectively. Since 2001, she has been a senior engineer at the National Geomatics Center of China. Her research interests are in geo-information services and emergency mapping. (zhuxiuli@ngcc.cn)

Xinli Di received her B.S. degree from Wuhan University, China, in 2007. From 2009 to 2016, she was an engineer at the National Geomatics Center of China. Since 2016, she has been a senior engineer at the National Geomatics Center of China. Her research interests are in geographic information services. (dixinli@ngcc.cn)



Yunlu Peng received her B.S. degree from the Beijing University of Civil Engineering and Architecture, Beijing, in 2012. From 2012 to 2017, she was an assistant engineer at the National Geomatics Center of China. Since 2018, she has been an engineer at the National Geomatics Center of China. Her research interests are in geographic information systems. (pengyunlu@ngcc.cn)

Tingting Zhao received her B.S. degree from Wuhan University, China, in 2000 and her M.S. degree from Chang'an University, Xi'an, China, in 2007. Since 2007, she has been a senior engineer at the National Geomatics Center of China. Her research interests are in thematic map compilation, surveying and mapping emergency support, and GIS development and applications. (zhaotingting@ngcc.cn)