Smart Structures and Systems, *Vol. 28, No. 2 (2021) 275-287*
https://doi.org/10.12989/sss.2021.28.2.275

Real-time structural health monitoring system based on streaming data

Qilin Zhang [1a], Siyuan Sun [1b], Bin Yang*[1], Roland Wüchner [2], Licheng Pan [1] and Haitao Zhu [1]

[1] *College of Civil Engineering, Tongji University, Shanghai, China*
[2] *Technical University of Munich, Arcisstr. 21, D-80333 München, Germany*

(*Received November 3, 2020, Revised February 7, 2021, Accepted March 20, 2021*)

Abstract. In this paper, a novel real-time structural health monitoring (SHM) system based on streaming data is proposed. In contrast to a traditional SHM system, the proposed system implements a series of optimizations for data transmission and processing to reduce the latency and better satisfy the real-time requirement. The concept of the watermark in the streaming system is adopted to address the problem of when to trigger the time window calculation under the real-time requirement. Moreover, a well-designed parallel mechanism is used to satisfy the multistage computation requirement in the parallel data stream. A case study in which the proposed system is applied to the Shanghai Tower is presented. The peak picking method is used as an example in the test environment to track the latency of each main operation under different parallelism schemes. The results show that computing in parallel effectively reduces the latency and provides a reference for integrating the random decrement technique (RDT), stochastic subspace identification (SSI), or other more complex but effective algorithms in parallel into the system in the future. The total latency under the test environment from data generation to data transmission to the web server is approximately only 200-400 ms, which indicates the excellent real-time performance of the system.

Keywords: data stream; latency; computing; real-time; structural health monitoring; parallel

1. Introduction

Structures are often designed to last for decades or even longer. During their operation, structures are inevitably affected by various dynamic and static loads, natural environmental erosion, and material fatigue and aging, which cause accumulated damage and resistance attenuation (Yi *et al*. 2013). Continuous bearing capacity and durability degradation mean that the ability of the structure to resist natural disasters and even normal operating loads will decline over time (Li *et al*. 2006, Yi *et al*. 2017). Consequently, it is important to be able to evaluate structural integrity, durability, and reliability throughout a structure's life cycle and to issue early warnings of damage or deterioration to minimize repair costs and avoid catastrophic collapses (Ko and Ni 2005). Thus, the main objectives of a real-time structural health monitoring (SHM) system are to be able to analyze the key structural and environmental parameters under operating conditions in real time, detect structural defects as early as possible, and send warning information in a timely manner to avoid further damage or collapse (Zhou *et al*. 2016, Kaya and Safak 2015).

A real-time system has been described as one that controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment immediately (Martin 1965). The distinction between "near-real-time" and "real-time" is somewhat nebulous and must be defined for each individual situation (Federal Standard 1037C 1997). In some cases, processing described as "real-time" would be more accurately described as "near-real-time". The distinction between "near-real-time" and "real-time" varies. Real-time responses are often understood to be on the order of milliseconds or microseconds (Sivasankar and Sakthivel 2017), whereas the latency in near-real-time situations typically ranges from several seconds to several minutes (Arnold *et al*. 2016, Orcutt and Vernon 2014). In a real-time SHM system, the latency caused by different algorithms may belong to either one of these two concepts. For simplicity, in this paper, the term "real-time" denotes both real-time and near-real-time.

The key requirement in a real-time SHM system is that the recording, processing, and analysis of the data should all be performed in real time (Kaya and Safak 2015). In most SHM applications, data recording is continuous and occurs in real time. Some SHM systems can process data in real time, such as comparing a measured value with a predefined threshold and raising an early warning if necessary. However, due to the lack of real-time data analysis capabilities, some real-time monitoring algorithms are restricted. Over the last several decades, a few real-time SHM systems that can achieve real-time data processing and analysis have emerged, such as Masri *et al*. (2004), who implemented a real-time SHM system based on an efficient multithreaded software design in which the system acquires data from a large number of channels simultaneously. They implemented real-time data preprocessing and analysis, such as the fast Fourier

*Corresponding author, Assistant Professor,
E-mail: yangbin@tongji.edu.cn
[a] Professor, E-mail: zhangqilin@tongji.edu.cn
[b] Ph.D. Student, E-mail: 2010052@tongji.edu.cn

ISSN: 1738-1584 (Print), 1738-1991 (Online)

transform and root mean square extraction, for each measurement point. Achieving any real-time rapid analysis tasks in 2004 was a remarkable achievement. The system was applied to an important large-span flexible bridge (Vincent Thomas Bridge) in San Pedro, California, USA. Kijewski-Correa *et al*. (2013) designed a real-time SHM system called SmartSync for high-rise buildings that was based on wireless sensor networks. To solve the time synchronization problem of distributed sensors, the system assumes that the data from multiple sensors are transmitted to the central server in a small amount of time; thus, its impact can be ignored. Consequently, SmartSync does not rely on clocks and timestamps from individual devices but instead ties everything to the server clock. The system can display the power spectral density of each measured point in real time, obtain the vibration frequencies of the structure based on picked peak values, and provides an interface for data analysis and data visualization. This system was applied to the world's tallest building, Burj Khalifa. Kaya and Safak (2015) developed a MATLAB-based real-time SHM system called REC_MIDS, which achieved more complex real-time data processing and analysis. Nine different spectrum estimation techniques were implemented in REC_MIDS; the user has the option to use any one or a combination of them simultaneously. Additionally, the acceleration data can be filtered and quadratically integrated in real time to obtain the corresponding displacement. Then, the interstory drift can be calculated to be compared with predefined threshold values to detect damage. The system is installed and operational in a large number of different structures (tall buildings, suspension bridges, mosques, museums) in Turkey and the UAE.

Overall, while substantial achievements have been made for real-time SHM systems under the conditions available when they were designed, some shortcomings still exist in current real-time SHM systems. These problems mainly include the following: (1) Most real-time SHM systems conduct data processing and analysis based on the central server clock time when data arrive. This is a reasonable scheme under normal circumstances. However, after the system has been running for a long time, it will inevitably encounter network congestion, system freezing, or other situations that may extend the data transmission time from the sensor to the central server, making it no longer negligible, which will cause inconsistent and inaccurate processing results. Instead, it is preferable to perform data processing and analysis based on the time when the data are generated. Nevertheless, for the above reasons, data generated later by one sensor may arrive at the system earlier than data generated earlier by another sensor. This phenomenon is even more common when the system uses wireless sensors or real-time algorithms that require sensor data of different types. Such disordered data then causes another problem: when to trigger the time window calculation under the real-time requirement. Therefore, a time analysis capability is needed to help the system determine when the data were collected and to trigger the time window calculation in a timely manner. (2) In a real-time SHM system, to better satisfy the real-time requirement, the system often processes the data from multiple sensors in parallel at the same time and then accumulates and calculates the results to obtain the global result. For example, when the peak picking method (Clough and Penzien 2003) is used to identify structural modal frequencies, it is first necessary to calculate the power spectrum information of each measuring point. After the results of all the same-time windows have been judged to be collected, they are averaged and regularized, and then the peak is selected as the structural vibration frequency. The problem here is to determine when the data from the previous stage have been collected to trigger the calculation in the subsequent stage. Therefore, a reasonable parallel mechanism for a real-time SHM system is needed to satisfy the requirement of multistage computation in the parallel data streams.

In the early big data computing field, large-scale data processing, as we recognize it today, made its debut with MapReduce (Dean and Ghemawat 2004) and its open-source version Hadoop (ASF 2003). The basic idea with MapReduce is to provide a simple data processing application programming interface (API) centered around two well-understood operations from the functional programming realm: map and reduce. As a result, engineers only need to focus on the business logic of their data processing needs. However, some features of MapReduce, such as batch processing mode and disk-level computing, restrict it to mainly offline computing scenarios. To solve the problems raised in the previous paragraph, Marz (2011) proposed the famous Lambda architecture in 2011. The basic idea is to use a real-time system to provide a low latency but relatively inaccurate result; then, an offline batch system performs the same calculation later to output the correct result. This system was quite successful because it solved the problem of accuracy while ensuring low latency (Hasani *et al*. 2014), even though the truly accurate results may arrive later. However, maintaining a Lambda system is difficult: such systems require building, provisioning, and maintaining two independent versions of the pipeline and a method to finally merge the results from the two pipelines (Kleppmann 2015, Akidau *et al* 2017). Kreps (2014a) put forward a new architecture named Kappa to improve the Lambda architecture. It only retains the streaming system in the Lambda architecture. When there is a requirement to recalculate from a certain historical moment to obtain the correct result, using a replayable system such as Kafka, a new stream processing task can be started to reprocess the historical data. When the new task catches up with the old task, the old task is replaced. However, storing historical data in a replayable system undoubtedly exerts great pressure. Moreover, replaying a large amount of historical data will also consume substantial real-time computing resources.

Due to the development of big data real-time computing, some excellent real-time big data processing frameworks have emerged, including Google Cloud Dataflow (Google 2013), Apache Flink (ASF 2014), Apache Beam (ASF 2016), etc. These frameworks support solutions to the above problems with reasonable configurations (Mandal 2018, Tantalaki *et al*. 2020). In this paper, a low-latency real-time SHM system is proposed based on streaming data utilizing

Apache Kafka (ASF 2011), Apache Flink, and WebSocket (Fette and Melnikov 2011) technology. In the next section, the design details of the real-time SHM system are presented, including the sensor subsystem, the data processing and analysis subsystem, and the data storage and display subsystem. The real-time mechanisms to ensure low latency and solutions to the above problems are addressed in the corresponding subsystems. The third section reports the results of a case study in which the system was applied to the Shanghai Tower. The latency caused by each main operation and the total latency from data generation to output are visualized and discussed. Finally, the last section concludes the paper.

2. Real-time structural health monitoring system

Most real-time SHM systems adopt a client/server (C/S) structure. The advantage of this architecture is that real-time data processing and analysis can be performed locally, which reduces pressure on the server. However, a C/S structure may require different software versions for different client operating systems, and the computing capabilities of the included devices may differ. It is thus difficult to satisfy the needs of users who want to be able to use any device to observe the current structural health monitoring status at any time. Therefore, this paper builds a real-time SHM system based on a B/S structure, that is, a browser/server structure. The system architecture is shown in Fig. 1.

Under the system architecture shown in Fig. 1, the data stream generated by the sensor first enters the message middleware and then flows into the data storage subsystem, which stores the raw data. Simultaneously, the data stream flows from the message middleware into the real-time data processing subsystem to ensure timely data processing and analysis. The output of data processing flows back into the message middleware and then into the data storage subsystem, which stores the results of the data processing and analysis. The output stream also flows from the message middleware into the web server, so that it can be displayed to users promptly.

The proposed system uses Apache Kafka as the message

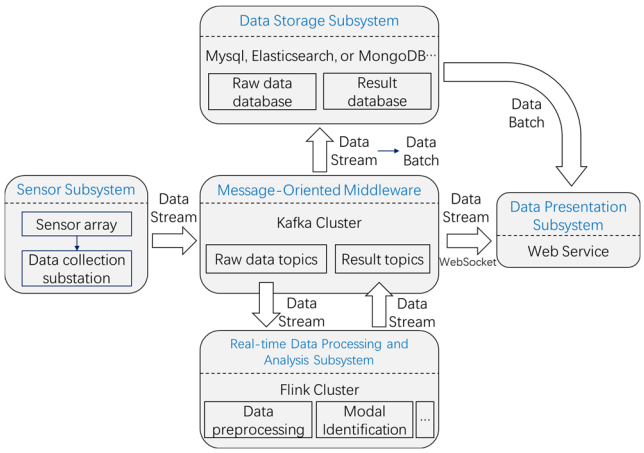


Fig. 1 System architecture

middleware. It is a distributed publish-subscribe messaging system. Kafka constructs a real-time streaming data pipeline, which can reliably obtain data between systems or applications. It categorizes streaming data by topic. When new data on one topic arrives, the data can be immediately pushed to all systems or applications that subscribe to the topic. Kafka is well qualified for this real-time structural health monitoring system, and it supports Apache Flink in data processing and analysis subsystem very well. More details about Kafka and the foundations upon which it is built can be referenced in Kreps (2014b).

Note that under the publish/subscribe mechanism of the message middleware, whenever new data are generated—whether raw data or output results—they are immediately accessible by the data storage subsystem and other subsystems simultaneously in the form of a data stream. Compared with a traditional SHM system based on a B/S structure, where the strategy involves first storing data to the data storage subsystem and then reading batched data from the subsystem, the proposed system reduces unnecessary latency. Furthermore, new data flowing into Kafka are first written to the page cache. Data read in real time are likely to be stored in the page cache; thus, they can be transmitted directly from the page cache to the socket buffer and then to other subsystems. This arrangement avoids disk I/O and further reduces the latency.

2.1 Sensor subsystem

This section does not focus on sensor selection or layout optimization; instead, it focuses on responding to time issues in real time. Because most sensors used for SHM generate electrical signals that do not carry a timestamp, the electrical signals of nearby sensors can be collected through a data collection substation, in which the electrical signals from sensors are converted into a digital signal, and a timestamp is generated. After the data have been timestamped, all subsequent data processing and analysis are based on the timestamp, which ensures the temporal credibility of the data.

Since SHMs typically include a large number of sensors with relatively low sampling frequencies, assigning a thread to each sensor data stream is not conducive to scheduling system hardware resources. Therefore, the data of multiple data collection substations are summarized into one data pipeline (topic in Kafka) based on the sensor type where the data originated. For example, the pipeline named “acceleration” carries data from all the acceleration sensors (Fig. 2).

2.2 Real-time data processing and analysis subsystem

In the real-time data processing and analysis subsystem, the real-time data processing subsystem consumes the timestamped original signal data stream received by the message middleware in real time. Additionally, the publish/subscribe mechanism of the message middleware is particularly amenable to data analysis methods that require multiple types of sensor data simultaneously. After the data processing and analysis are completed, the results are

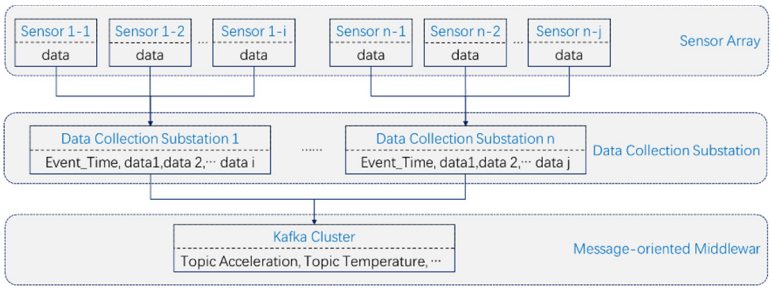


Fig. 2 Sensor subsystem

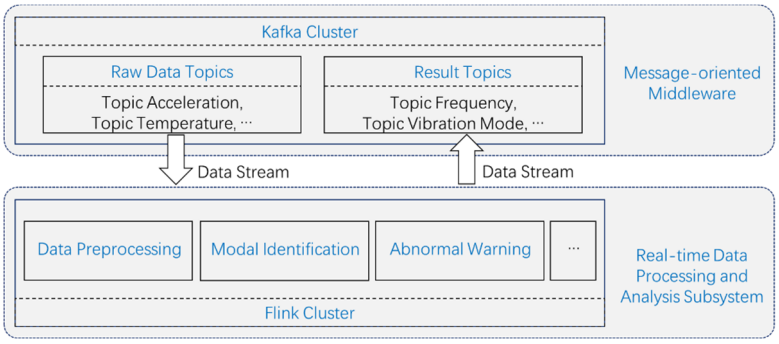


Fig. 3 Real-time data processing and analysis subsystem

immediately output to the corresponding result data pipeline in the message middleware for storage and display (Fig. 3).

The proposed system uses Apache Flink as the engine of the real-time data processing and analysis subsystem. It is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink implements many techniques from the dataflow model (Akidau *et al.* 2015), which is represented by a directed acyclic graph (DAG). In a DAG, each node represents calculation, and each edge represents data dependency. Fig. 4 shows the data stream diagram of the formula

$$z = ax + by + c \tag{1}$$

For the multiplication calculation of node A, once the variable a and the variable x arrive, the multiplication task will be executed. It can be expressed as

$$N_A.res = f_{N_A}(a, x), \quad once\ variable\ a, x\ arrive \tag{2}$$

where N_* represents Node *; $N_*.res$ represents the calculation result of Node *; and $f_{N_*}(\cdot)$ means the operator of Node *. Node A and node B have different computing dependencies, and all their computing tasks can be scheduled and executed asynchronously. When the calculation task of node A is completed, its calculation result will be sent to the next-level computing node (Node C in Fig. 4) that depends on it. The calculation of node C can be expressed as

$$z = N_C.res = f_{N_C}(N_A.res, N_B.res, c), \quad once\ N_A.res, N_B.res, c\ arrive \tag{3}$$

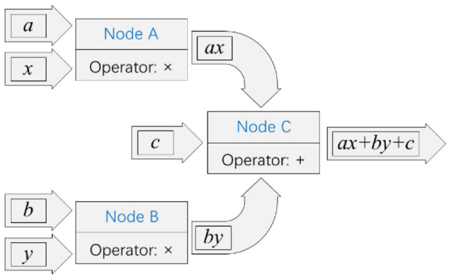


Fig. 4 An example of the dataflow model

In a structural health monitoring scenario, when the data generation time is used instead of the data arrival time at the server, in actual system operation, there will always be a certain lag between the time when data enter the real-time data processing subsystem and the time when the data collection substation generates the time stamp. This lag time is random due to factors such as network congestion, system freezing, or other situations, and it may extend the transmission from the sensor to the central server to non-negligible durations over long-term system operation. Such situations can cause data generated by one sensor at a later timepoint to arrive at the system earlier than data generated at an earlier timepoint by another sensor. This phenomenon is more common when the system uses wireless sensors or real-time algorithms that require sensor data of different types. In other words, the data timestamps arrive out of sequence.

To perform data processing and analysis, it is usually necessary to add a time window to the data in the time dimension (common choices include tumbling windows, sliding windows, and so on) and then perform modal analysis. Therefore, the problem of temporal disorder can

be solved simply by sorting the data by timestamp in the time window and then performing the analysis. However, because the data enter the time window out of sequence, another problem occurs: when to trigger the time window calculation under the real-time requirement, that is, to determine when the once bar, as in Eqs. (2)-(3), is satisfied. Assume that the time window covers the period from 01:00-02:00. When the system observes data with a timestamp of 02:01, it is unreasonable to assume that all the data for that window have been collected and thus trigger the calculation ecause data are continuously generated. The operation will continue to receive data, and because the data are out of order, it is impossible to know whether the timestamp of the next arriving data belonged to the 01:00-02:00 window (such as 01:58) or lie outside the window (such as 02:02). If the calculation of the window is triggered immediately, as soon as the system observes any data with a timestamp of 02:01, the calculation results may be erroneous. Therefore, a time management scheme is needed to help the system determine when all the appropriate data have been collected and trigger the time window calculation in a timely manner.

The traditional SHM system usually timestamps the data when they are generated. In the offline analysis, it only needs to sort the data according to the timestamp to solve the above problem. However, in those systems the results cannot be obtained in real time. For real-time SHM systems, to avoid disorder, the strategy of using the time when data enter the server is usually adopted because the server system time increases monotonically. However, the time sequence of data arrival cannot be guaranteed. To solve the problem of triggering the window calculation in time under the real-time requirement while avoiding the side effects of the above two systems, the real-time SHM system proposed in this paper is based on the timestamp of the data analogous to a traditional SHM system and uses the concept of a watermark in the streaming system (Akidau *et al* 2013, 2015, 2017).

2.2.1 Watermark

Watermarks are the way the system measures progress and completeness relative to the timestamp carried by the data being processed in the data stream. The watermarks flow with the data and carry a timestamp. In this case, a watermark is an assertion that no data with timestamps earlier than the watermark timestamp exist. As shown in Fig. 5, a watermark of "W29" means that there are no data with a timestamp <= 29 that will arrive after the watermark. Thus, the problem of when to trigger the window calculation under the real-time requirement can be solved as

$$TW.res = f_{TW}(\boldsymbol{TW.data}), \quad once\ TW.watermark \geq TW.end \tag{4}$$

where TW represents the time window; $\boldsymbol{TW.data}$ means all the data in the time window; $TW.watermark$ means the latest watermark in the time window; and $TW.end$ represents the maximum time allowed for this time window. Assuming that the time window is [0,30), when the W29 watermark enters the window, the window calculation can be immediately triggered.

The timestamp value carried by the watermark can be considered comprehensively based on both data integrity and the timeliness of the output. In an SHM scenario, data disorder is statistically relatively stable when the system is running normally. Thus, when data with a timestamp of T enters the system, one can assume that 99% of the data of *T-2 s* has entered the system, and that 97% of the data of *T-100 ms* has entered the system. When considering timeliness, it is a better choice to set the watermark to *T-100 ms*. It should be noted that when an unexpected situation occurs in a data collection substation, such as network congestion, its channel data will be accumulated and sent to the system all at once after recovery. This part of the data will be discarded due to the existence of the watermark. While losing data is not desirable, compared to mixing the data in the current time window with the data before the congestion, it at least does not produce incorrect results. Additionally, the window can be allowed to exist for some period after the calculation is triggered, so that if additional data belonging to the window arrive, the calculation of the window can be retriggered to help guarantee data integrity to a certain extent while also ensuring timeliness.

The watermark tells the window when the data have been collected and triggers the time window calculation in a timely manner. This approach allows a trade-off between data integrity and the timeliness of the output by setting different values. However, in the real-time SHM system, to better satisfy the real-time requirement, the system often calculates the data of multiple sensors in parallel at the same time and then collects and calculates the results to obtain the global results. For example, when the peak picking method is used to identify structural modal frequencies, it is necessary to initially calculate the power spectrum information of each measuring point. After the results of all the same-time windows have been judged to be collected, they are averaged and regularized, and then the peak is selected as the structural vibration frequency. The problem here is to determine when the data from the previous stage have been completely collected to trigger the calculation of the next stage. Therefore, a reasonable parallel mechanism for a real-time SHM system is needed to satisfy the requirement of multistage computation in the parallel data stream.

2.2.2 Parallel multistage calculation

In the system proposed in this paper, to calculate

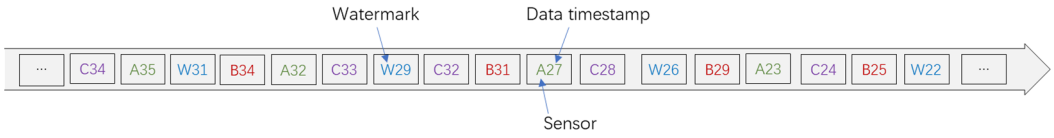


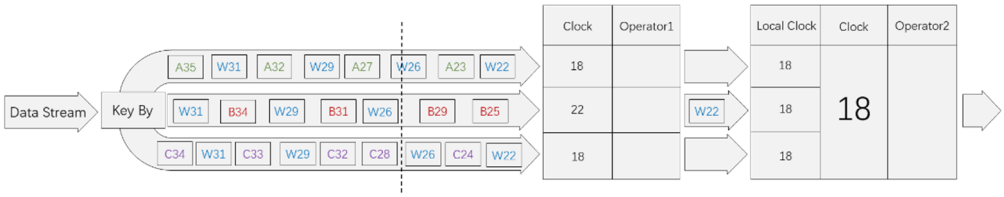
Fig. 5 Watermark

multiple sensor data at the same time, the data are divided into assigned multiple parallelisms based on sensor type through the KeyBy operation. A hash algorithm is used in KeyBy to ensure that data from the same sensor are sent to the same parallelism. Each parallelism holds data for one or more sensors, and watermarks are broadcast to all parallelisms. For instance, if the data from 10 sensors are divided into 5 parallelisms, each parallelism holds the data from 2 sensors and all watermarks. Within one parallelism, the data of two sensors are calculated serially. Unlike the data stored in the window to be calculated when it passes through the window, when the watermark passes through the window, it is immediately sent downstream. If the current window is triggered to calculate, the watermark is sent downstream along with the calculated result. The downstream window sets a local clock for each upstream parallelism. When a watermark is encountered, the local clock is updated. The downstream window takes the minimum value of all local clocks as its global clock. In this way, when the global clock reaches the maximum value allowed by the window, the window calculation can be triggered immediately. It can be expressed as

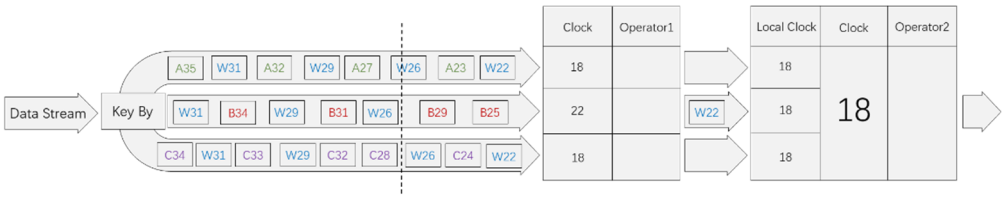
$$TW_{ds}.res = f_{TW_{ds}}(\boldsymbol{TW_{us}.res}), \quad once\ TW_{ds}.clock \geq TW_{ds}.end \tag{5}$$

$$TW_{ds}.clock = \min(\boldsymbol{TW_{ds}.local_{clock}}), \quad k = 1,2 \ldots n \tag{6}$$

where TW_{ds} represents the downstream time window; $\boldsymbol{TW_{us}.res}$ means all the calculation results of upstream time windows on which the downstream time window depends; $TW_{ds}.clock$ represents the clock of the downstream time window and $\boldsymbol{TW_{ds}.local\_clock}$ represents all the local clocks in the downstream time window. As indicated by Eqs. (1)-(5), there is no special limitation for the operator used in the time window. Conventional methods suitable for real-time analysis can be

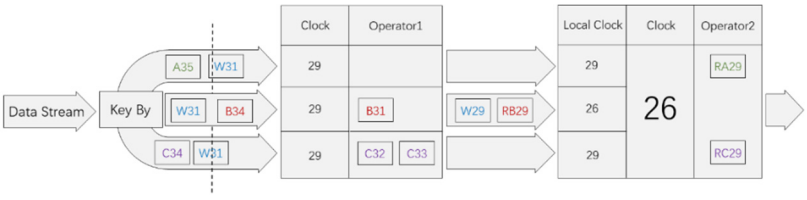


(a)

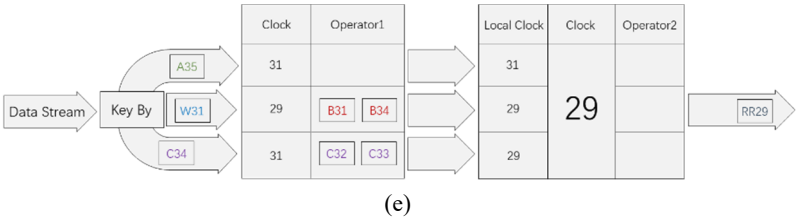


(b)

(c)



(d)



(e)

Fig. 6 The propagation mechanism of the watermark in parallel

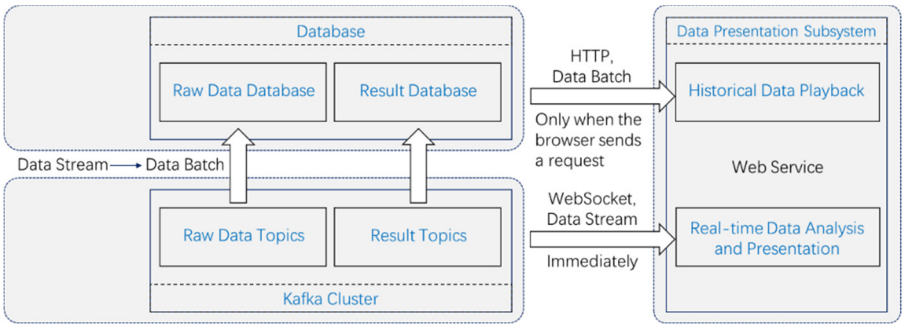


Fig. 7 Data storage and data presentation subsystem

included, so it has a great flexibility. Fig. 6 illustrates this process using the data stream described in the previous subsubsection. A tumbling window is used, involving time windows of [0,30) and [30,60). The data stream is first divided into three parallelisms (hereinafter referred to as parallelism A, B, or C) based on the sensor type through the KeyBy operation, and each parallelism holds the data from one sensor. The dotted line in the figure indicates the moment of the next picture. In Fig. 6(a), the W22 watermark of parallelism B has passed Operator 1, updated the clock of Operator 1 at parallelism B and is propagating to downstream Operator 2. In Fig. 6(b), sensor data are accumulated by Operator 1, and the watermark continues to propagate downstream. When the W22 watermarks from all parallelisms have been sent to Operator 2, Operator 2 updates its global clock from 18 to 22 and sends the W22 watermark downstream. The W26 watermark updates the clock of Operator 1 in parallelisms A and C. In Fig. 6(c), in parallelisms A and C, the W29 watermark is sent to Operator 1, which triggers the calculation of the [0,30) window. Then, the W29 watermark is sent downstream to Operator 2 together with the calculation result. The result carries the maximum timestamp (29) allowed in the window. In parallelism A, the result is stored in Operator 2, and the watermark updates the local clock. In parallelism C, the result and the watermark are going to involve the same operations. In Fig. 6(d), the W29 watermark of parallelism B also triggers the window calculation for the Operator 1 window, and the calculation result and the watermark are sent downstream. In Fig. 6(e), when Operator 2 receives the W29 watermark of parallelism B, its local clock is updated to 29; therefore, its global clock is also updated to 29, which immediately triggers the calculation of the [0,30) window of Operator 2.

Under this parallel mechanism, when the data from the previous stage have been collected, the system can immediately trigger the calculation of the next stage, and no unnecessary latency is produced. In addition, this mechanism also applies to real-time algorithms that require multiple data sources. Moreover, we only need to set the watermark when the data enter the real-time data processing subsystem; then, the system automatically determines when to trigger the time window calculation in a parallel environment. This approach also simplifies system development and later maintenance.

2.3 Data storage and data presentation subsystem

After the real-time data processing and analysis subsystem sends the results to the message middleware, the data storage subsystem and the data presentation subsystem can simultaneously obtain the results. In addition to making requests to the database to display historical data, the data presentation subsystem uses WebSocket technology to establish a long link between the web server and the browser. It is different from the traditional HTTP protocol, which is a request-response model. If the user does not send a request to the server of the SHM system through the browser, the server cannot actively push monitoring data to the user's browser. To achieve the requirements for real-time display of monitoring results based on the HTTP protocol, the browser's JavaScript for regular polling can be relied on, which is inefficient and not real time. With WebSocket technology, when the message middleware receives new data, it can be acquired by the web server immediately and pushed to the browser by the established long link in real time (Fig. 7). Compared with a traditional SHM system, in which the results are first stored in the data storage subsystem, when the browser sends a request, the web server reads the latest data from the data storage subsystem in the form of the batch, further reducing the latency.

In the proposed real-time SHM system, timestamps are generated for the data by the collector in the data collection substation. The message middleware enables multiple types of sensor data to be used by multiple subsystems and data analysis methods simultaneously. In addition, it works with WebSocket technology to ensure that data are transmitted in streaming data form throughout the entire whole system, which reduces the latency compared with batched data. Moreover, subsystems that require real time are conducted in memory, which avoids the large latencies caused by disk I/O. In the real-time data processing subsystem, the watermark and a well-designed parallel mechanism are used to satisfy multistage computation requirements in the parallel data stream and the trigger window calculations at appropriate times, which also minimizes the latency. The proposed system was developed using Apache Kafka (the message middleware), Apache Flink (the real-time data processing subsystem), MySQL (the data storage subsystem), and Spring Boot (the data presentation subsystem). In the next section, a case study in which the proposed system was applied to the Shanghai Tower is

introduced. Moreover, the latency caused by each main operation and the total latency from data generation to output are reported and discussed to provide an intuitive understanding of the latency caused by each operation and explore the contribution of each operation to the overall latency, providing a reference for subsequent possible optimization work.

3. Case study: The Shanghai Tower application

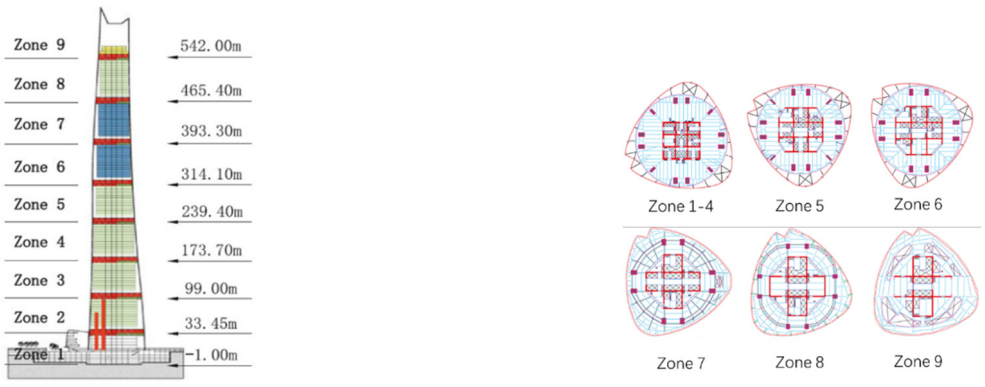
The Shanghai Tower is a supertall building with a structural height of 580 m and an architectural height of 632 m; it is the tallest structure in China (Fig. 8). The building is used for office spaces, entertainment facilities, hotel and sky-gardens as well as various retail and cultural venues. A triangular outer facade encloses the entire structure that gradually shrinks and twists clockwise by approximately 120 degrees over the height of the entire building (Su *et al* 2013). The building is vertically divided into nine zones separated by eight independent strengthening floors (Fig. 9(a)). At the interface of the adjacent zones, a two-story, full-floor area is created to house mechanical, electrical and plumbing equipment and serve as that zone's life safety refuge area. This full-floor platform creates a base for the atrium spaces directly above. The structural layouts of typical floors in each zone are shown in Fig. 9(b). The Shanghai Tower adopts a mega-frame core-wall structural system, which comprises a core-wall inner tube, an outer mega frame, and a total of six levels of outriggers between the tube and the frame. The outriggers are s*et al*ong the height of the building at zones 2, 4, 5, 6, 7, and 8. The core acts in concert with the outrigger and supercolumn system that is supplemented by the mega frame. There are four paired supercolumns - two at each end of each orthonormal axis. In addition, four diagonal supercolumns along each 45-degree axis are required by the long distances at the base between the main orthonormal supercolumns. These distances are approximately 50 meters and reduced to 25 meters at the diagonal columns (Xia *et al*. 2010). The steel-concrete composite superstructure resists lateral loads with the central reinforced concrete shear wall core interconnected with the composite mega frame via six two-story-high outrigger trusses and supercolumns system. Gravity loads are resisted by the steel-concrete composite floor system (Zhang *et al*. 2015).

Fig. 8 A view of Shanghai Tower

To monitor the response of Shanghai Tower, Tongji University, Hong Kong Polytechnic University and Tongji Architectural Design Institute constructed a long-term structural health monitoring system that collects and analyzes data related to wind velocity, displacement, internal forces, temperature, corrosion, cracks, and earthquakes (Li *et al*. 2015). More details of Shanghai Tower and its SHM system can be referenced in Su *et al*. (2013), Xia *et al*. (2010) and Li *et al*. (2015).

3.1 Application

In this paper, we updated the existing SHM system of Shanghai Tower to the proposed real-time SHM system. In addition to providing a real-time display of the raw sensor data, the system also achieves real-time data processing and analysis functions, such as structural modal recognition. The system not only provides a plug-and-play interface for new algorithms that may be added in the future, but the hardware resources required to accommodate new sensor data and algorithms can also be expanded by increasing the number of nodes in the cluster, taking advantage of the characteristics of Kafka and Flink's distributed architecture. Additionally, with Savepoint in Flink, it is possible to easily upgrade, migrate, suspend, and resume algorithms.



(a) (b)

Fig. 9 Strengthening floors (Su *et al*. 2013 and Zhang *et al*. 2015)

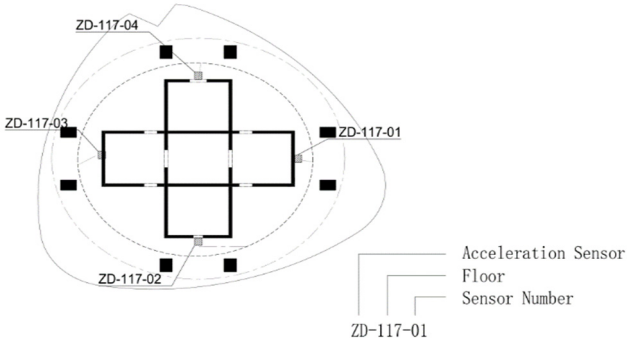


Fig. 10 The layout of the acceleration sensors on the 117th floor

Acceleration sensors are arranged throughout 10 zones of Shanghai Tower. Fig. 10 shows the layout of the acceleration sensors on the 117th floor. As shown in Figs. 9(a) and (b), the cylinder of Shanghai Tower is regular in the vertical direction. The 117th floor corresponds to zone 8 in Fig. 9(b). The acceleration sensor layouts on other floors are similar. Fig. 11 displays the raw data from 40 acceleration measurement points in Shanghai Tower in real time. By displaying the data of multiple measurement points on one graph, comparisons can be made more intuitively. Fig. 12 applies the peak picking method to these 40 acceleration measurement points. The results show that the system works well.

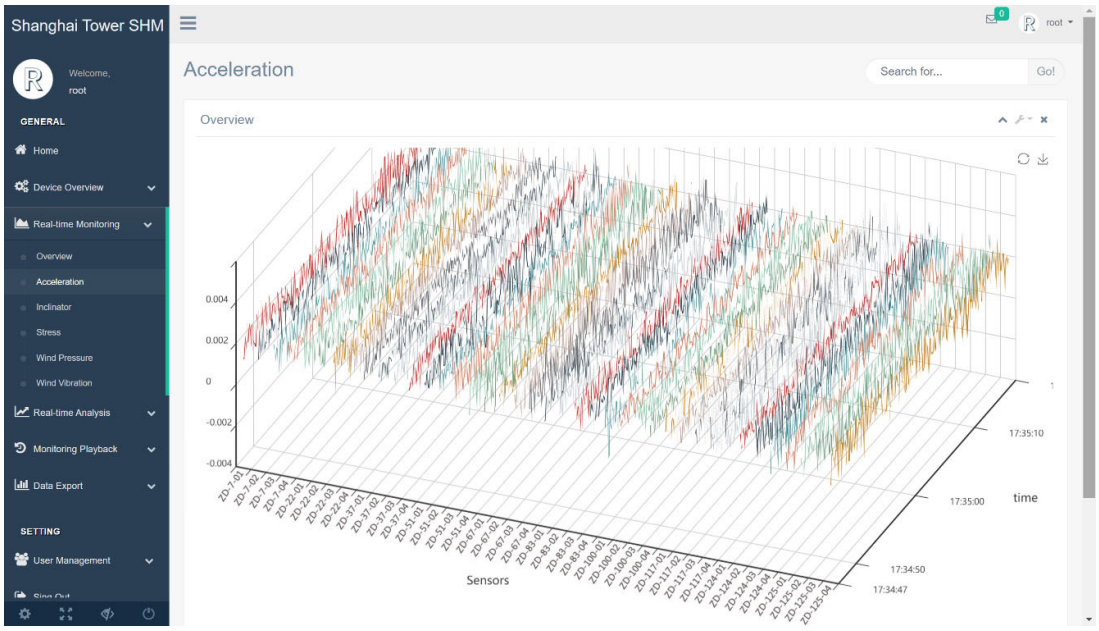


Fig. 11 General view of real-time acceleration monitoring of all acceleration sensors

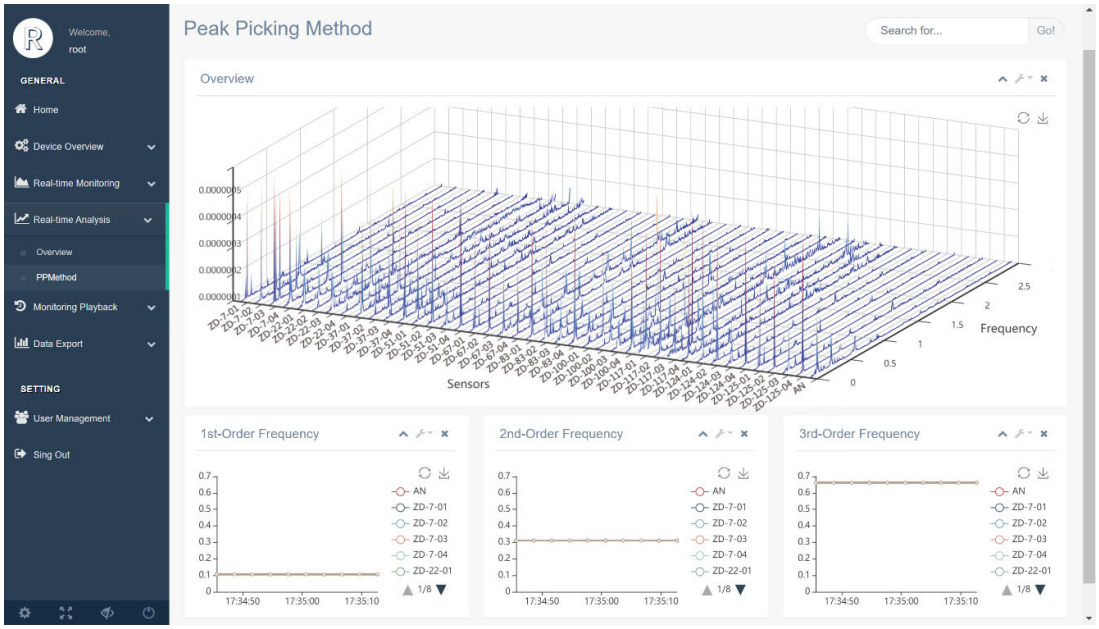


Fig. 12 General view of real-time peak picking method for the whole building

3.2 Latency analysis

For simplicity, but without loss of generality, the peak picking method is taken as an example to simulate the real scene using a standalone method with historical data to test the data latency from generation to transmission to the web server. The parallel strategy of the peak picking method is shown in Fig. 13. It first reads the data from the Kafka topic "Acceleration" and assigns a watermark based on the data timestamp. Then, KeyBy is used to divide the data from 40 acceleration measurement points into the 4 parallelisms of Operator 1 by sensor (the degree of parallelism can be set to other values). After computing Operator 1, the results are collected and sink into the corresponding Kafka topic by the data collector. Simultaneously, Operator 2 continues to read the power spectral density data from the data collector, performs the next calculation, and stores the results in the data collector again, which again sink into the corresponding Kafka topic.

This study not only provides the end-to-end latency (from Kafka to Flink to Spring Boot) but also tracks the latency of each main operation in Flink to gain an intuitive understanding of the latency caused by each operation and explore the contribution of the operation to the overall latency, providing a reference for subsequent possible optimization work. Each main operation is shown in Fig. 14. The symbols in the upper left and upper right corners of each operation indicate the time at which data enter and leave the operation; the difference between these values indicates the latency in or between operations. For instance, $T\_Pi - T\_Po$ represents the time consumed by Operator 1, that is, the latency generated by Operator 1. The specific implementation is shown in Fig. 15. Thus, T_Po-T_Ai represents the transmission latency resulting from the transmission of the result of Operator 1 to Operator 2 and latency resulting from the watermark. Note that—except for Operator 1 and Operator 2—the remaining operations do not involve time windows; they apply to each data value. The method used here is to add the same window to these operations in the latency analysis and show the time when the latest processed data in the window enters and leaves the operation for the symbols in the upper left and upper right corners of the operation. Therefore, $T\_Di - T\_Wi$ represents the latency between when Flink receives the last piece of data belonging to a time window to the time when the window results are sent to the web server. In addition, to

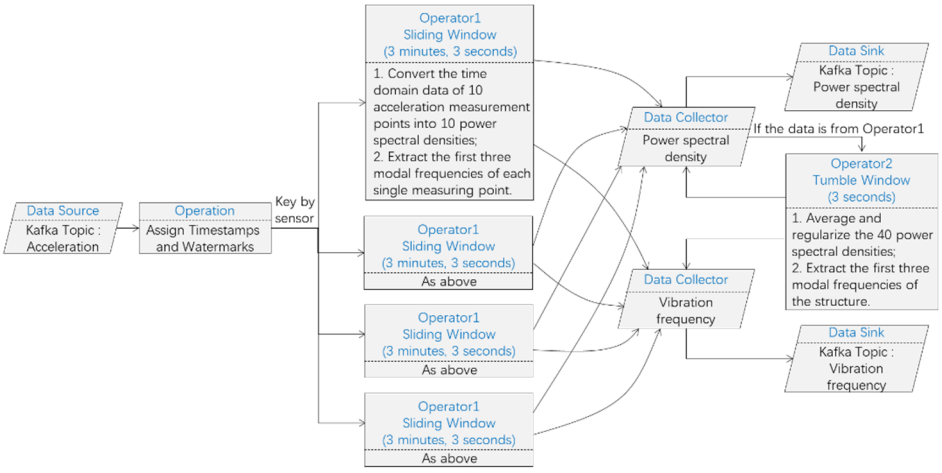


Fig. 13 Peak picking parallel strategy

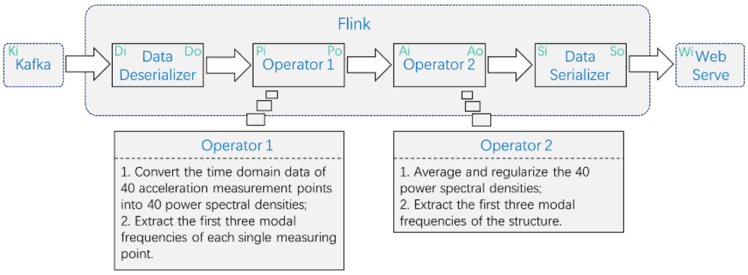


Fig. 14 Main operations of the peak picking method in the real-time SHM system

```
Time Window:
    Time_Pi=current_time;
    exect Operator 1;
    Time_Po=current_time;
    T_Pi - T_Po = Time_Po - Time_Pi
end
```

Fig. 15 Specific implementation of $T\_Pi - T\_Po$

Table 1 Standalone configuration

Hardware	Configuration
CPU	Intel Xeon Gold 6242R @ 2.8 GHz with 16 cores ×1
Memory	64 GB
Disk	512 GB M.2

trigger the time window at an appropriate time, this test sets a watermark based on the timestamp of each data value. Note that this choice does have a certain impact on the performance. In practical applications, sending a watermark periodically is a better choice.

The standalone configuration is shown in Table 1. Fig. 16 shows the latency of each operation, where *Ki-Di* represents the time between when Kafka receives data to the time it is sent to Flink, that is, $T_{\_Di} - T_{\_Ki}$. The other labels are similar. Because Operator 1 needs to calculate 40 acceleration measurement points, parallel calculations are used to reduce the latency. Figs. 16(a), (b), (c) and (d) show cases in which Operator 1 is calculated with 1, 2, 5, and 10 parallelism(s).

Fig. 16 shows that the main contributions to the latency are *Do-Pi*, *Pi-Po*, and *Po-Ai*. These operations have latencies between tens of milliseconds to more than 100 milliseconds, while the latencies of other operations range from approximately 0-2 milliseconds. Both *Do-Pi* and *Po-Ai* represent the latency from the previous operation to the trigger window calculation. Because *Do-Pi* sets a watermark of data timestamp-100 ms, a latency of at least 100 ms will inevitably occur. Therefore, it and *Po-Ai* both result in latencies of approximately 50 ms when processing watermarks and trigger window calculations. *Pi-Po* represents the latency of converting the time-domain data from 40 acceleration measurement points into a power spectral density and extracting the first three order modal frequencies of each measurement point. As the degree of parallelism in Operator 1 gradually changes from 1 to 10, the average latency of *Pi-Po* is effectively reduced from 122.64 ms to 36.69 ms. Note that in the selected peak picking method example, the system can not only perform modal analysis of a single measuring point but can also summarize and analyze the results of multiple measuring points. This provides a reference for integrating the stochastic subspace identification (SSI) (Candy 2019) method, random decrement technique (RDT) (Ibrahim 2012) or other more complex but effective algorithms in parallel into the system in the future. Additionally, the *Do-Pi* in the figure shows a slow rise and then a sudden drop. This result occurs because the test generates historical data at the actual sampling frequency of 100 Hz, that is, one data value is generated every 10 ms. However, in addition to the 10 ms rest time, the system also consumes a small amount of time to extract and send data; thus, the actual data generation interval is slightly larger than 10 ms. Finally, even though it is related to the algorithm used, the total latency from data generation to data transmission to the web server averages only 344.4 ms even with a parallelism of 1 in Operator 1, which demonstrates the excellent real-time performance of the proposed system.

4. Conclusions

When damage occurs, the faster a system responds, the more time it can buy for decision makers to assess the damage. In this paper, a real-time SHM system with low latency is proposed based on streaming data. The

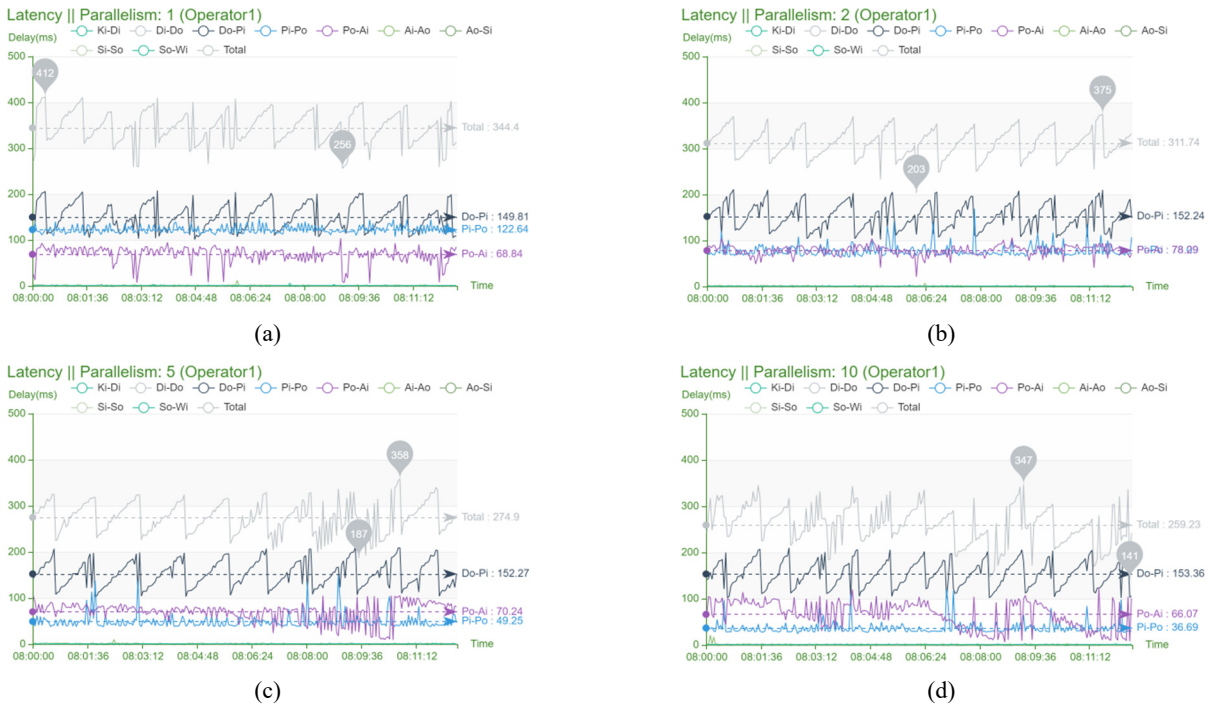


Fig. 16 Latency of each main operation and of different parallelisms in using the peak picking (PP) method for the whole building

characteristics of this system are summarized as follows.

- In the sensor subsystem, a data timestamp is generated using the collector in the data collection substation. After the data have been assigned a timestamp, subsequent data processing and analysis are based on the time stamp, ensuring the temporal credibility of the data.
- In the data processing and analysis subsystem, the watermark concept in the streaming system is used to solve the problem of when to trigger the time window calculation under the real-time requirement. A well-designed parallel mechanism is used to satisfy the requirement of multistage computation in the parallel data stream.
- In the data display subsystem, WebSocket technology is used to establish a long link between the web server and the browser. When new data to be displayed are generated, they can be immediately obtained by the web server and pushed to the browser in real time.
- Message middleware is adopted in this system. Under its publish/subscribe mechanism, when new data are generated, whether raw data or result output, they can be immediately accessed by the data storage and other subsystems simultaneously. Compared with a traditional SHM system, whose data are first stored in the data storage subsystem and then read from it, our approach further reduces the latency. In addition, this mechanism is particularly suitable for data analysis methods that need to obtain multiple types of sensor data simultaneously.
- Message middleware works with WebSocket technology to ensure that data are transmitted in streaming data form throughout the entire system, which reduces the latency compared with batched data. Subsystems that involve real time are conducted in memory to avoid the large latencies potentially caused by disk I/O.

In the reported case study, the proposed system was applied to the Shanghai Tower. The peak picking method was adopted as an example in the test environment to track the latency of each main operation under different parallelism settings. The results can be summarized as follows.

- In the test environment, the watermarking and trigger window calculations, the algorithm of converting the time-domain data of acceleration measurement points into power spectral densities and the process of extracting the first three order modal frequencies of each measurement point made the largest contributions to the overall latency.
- The peak picking method example shows that the system can not only perform modal analysis of a single measuring point but also summarize and analyze the results of multiple measuring points, providing a reference for integrating SSI, RDT, or other more complex but effective parallel algorithms into the system in the future.
- Even though the actual latency is related to the algorithm used, the total latency from data generation to data transmission to the web server is approximately only 200-400 ms, which indicates the excellent real-time performance of the system.

Acknowledgments

Special thanks to Hong Kong Polytechnic University and Tongji Architectural Design (Group) Co., Ltd., for their contribution in the construction of the structural health monitoring system of the Shanghai Tower.

References

Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P. and Whittle, S. (2013), "MillWheel: fault-tolerant stream processing at internet scale", *Proceedings of the VLDB Endowment*, **6**, 1033-1044. https://doi.org/10.14778/2536222.2536229

Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R.J., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E. and Whittle, S. (2015), "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing", *Proceedings of the VLDB Endowment*, **8**, 1792-1803.

Akidau, T., Chernyak, S. and Lax, R. (2017), *Streaming Systems*.

Arnold, D., Lutz, S., Dach, R., Jäggi, A. and Steinborn, J. (2016), "Near real-time coordinate estimation from double-difference GNSS data", In: *IAG 150 Years*, pp. 691-697. https://doi.org/10.1007/1345_2015_173

ASF (2003), Apache Hadoop; http://hadoop.apache.org/

ASF (2011), Apache Kafka; https://kafka.apache.org/

ASF (2014), Apache Flink: Stateful Computations over Data Streams; https://flink.apache.org/

ASF (2016), Apache Beam; https://beam.apache.org/

Candy, J. (2019), "Stochastic Subspace Identification", In: *Model-Based Processing*, pp. 309-389. https://doi.org/10.1002/9781119457695.ch7

Clough, R. and Penzien, J. (2003), *Dynamics of structures*.

Dean, J. and Ghemawat, S. (2004), "MapReduce: Simplified data processing on large clusters".

Federal Standard 1037C (1997), *Federal Standard 1037C*: *Glossary of Telecommunications Terms*, National Telecommunication Information Administration, USA.

Fette, I. and Melnikov, A. (2011), "The websocket protocol".

Google (2013), Datafolw | Google Cloud; https://cloud.google.com/dataflow.

Hasani, Z., Kon-Popovska, M. and Velinov, G. (2014), "Lambda architecture for real time big data analytic", *ICT Innovations*, 133-143.

Ibrahim, S.R. (2012), "Random decrement technique for modal identification of structures", *J. Spacecr. Rockets*, **14**, 696. https://doi.org/10.2514/3.57251

Kaya, Y. and Safak, E. (2015), "Real-time analysis and interpretation of continuous data from structural health monitoring (SHM) systems", *Bull. Earthq. Eng.*, **13**, 917-934. https://doi.org/10.1007/s10518-014-9642-9

Kijewski-Correa, T., Kwon, D.K., Kareem, A., Bentz, A., Guo, Y., Bobby, S. and Abdelrazaq, A. (2013), "SmartSync: An integrated real-time structural health monitoring and structural identification system for tall buildings", *J. Struct. Eng.*, **139**, 1675-1687. https://doi.org/10.1061/(ASCE)ST.1943-541X.0000560

Kleppmann, M. (2015), "A Critique of the CAP Theorem".

Ko, J.M. and Ni, Y.Q. (2005), "Technology developments in structural health monitoring of large-scale bridges", *Eng. Struct.*, **27**, 1715-1725. https://doi.org/10.1016/j.engstruct.2005.02.021

Kreps, J. (2014a), Questioning the Lambda Architecture; https://www.oreilly.com/radar/questioning-the-lambda-architecture.

Kreps, J. (2014b), *I heart logs*: *Event data*, *stream processing*, *and data integration*.

Li, H., Zhou, W.S., Ou, J.P. and Yang, Y.S. (2006), "A study on system integration technique of intelligent monitoring systems for soundness of long-span bridges", *China Civil Eng. J.*, **39**, 46-52.

Li, H., Zhang, Q.L., Yang, B., Lu, J. and Hu, J. (2015), "Development and application of construction monitoring system for Shanghai Tower", *Smart Struct. Syst.*, *Int. J.*, **15**(4), 1019-1039. https://doi.org/10.12989/sss.2015.15.4.1019

Mandal, K. (2018), "Evolution of Streaming ETL Technologies".

Martin (1965), *Programming real-time computer systems*.

Marz, N. (2011), How to beat the CAP theorem; http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html.

Masri, S.F., Sheng, L.H., Caffrey, J.P., Nigbor, R.L., Wahbeh, M. and Abdel-Ghaffar, A.M. (2004), "Application of a web-enabled real-time structural health monitoring system for civil infrastructure systems", *Smart Mater. Struct.*, **13**, 1269-1283. https://doi.org/10.1088/0964-1726/13/6/001

Sivasankar, D. and Sakthivel, P. (2017), "Enhancing Quality of Service in Real Time Communication services for Network on Chip", *Int. J. Electron. Eng. Res.*, **9**, 233-240.

Orcutt, J. and Vernon, F. (2014), "A Modern Operating System for Near-real-time Environmental Observatories", In: *EGU General Assembly Conference Abstracts*, p. 14949.

Su, J.Z., Xia, Y., Chen, L., Zhao, X., Zhang, Q.L., Xu, Y.L., Ding, J.M., Xiong, H.B., Ma, R.J., Lv, X.L. and Chen, A.R. (2013), "Long-term structural performance monitoring system for the Shanghai Tower", *J. Civil Struct. Health Monitor.*, **3**, 49-61. https://doi.org/10.1007/s13349-012-0034-z

Tantalaki, N., Souravlas, S. and Roumeliotis, M. (2020), "A review on big data real-time stream processing and its scheduling techniques", *Int. J. Parallel Emerg. Distrib. Syst.*, **35**, 571-601. https://doi.org/10.1080/17445760.2019.1585848

Xia, J., Poon, D. and Mass, D. (2010), "Case Study: Shanghai Tower", *CTBUH J*, **2010**, 12-18.

Yi, T.H., Li, H.N. and Gu, M. (2013), "Recent research and applications of GPS-based monitoring technology for high-rise structures", *Struct. Control Health Monitor.*, **20**, 649-670. https://doi.org/10.1002/stc.1501

Yi, T.H., Huang, H.B. and Li, H.N. (2017), "Development of sensor validation methodologies for structural health monitoring: A comprehensive review", *Measurement*, **109**, 200-214. https://doi.org/10.1016/j.measurement.2017.05.064

Zhang, Q., Yang, B., Liu, T., Li, H. and Lv, J. (2015), "Structural health monitoring of Shanghai tower considering time-dependent effects", *Smart Struct. Syst.*, *Int. J.*, **4**(1), 39-44. https://doi.org/10.21022/IJHRB.2015.4.1.039

Zhou, G.D., Yi, T.H. and Chen, B. (2016), "Innovative design of a health monitoring system and its implementation in a complicated long-span arch bridge", *J. Aerosp. Eng.*, **30**(3), B4016006. https://doi.org/10.1061/(ASCE)AS.1943-5525.0000603