Data Stream of Wireless Sensor Networks Based on Deep Learning

https://doi.org/10.3991/ijoe.v12i11.6232

Li Yue-jie
Ordos Institue of Technology, Inner Mongolia, China

***Abstract*—The sensor data in wireless sensor networks are continuously arriving in multiple, rapid, time varying, possibly unpredictable, unbounded streams, and no record of historical information is kept. These limitations make conventional Database Management Systems and their evolution unsuitable for streams. Thereby there is a need to build a complete Data Streaming Management System (DSMS), which could process streams and perform dynamic continuous query processing. In this paper, a framework for Adaptive Distributed Data Streaming Management System (ADDSMS) is presented, which operates as streams control interface between arrays of distributed data stream sources and end-user clients who access and analyze these streams. Simulation results show that the proposed method can thus improve overall system performance substantially.**

***Index Terms*—control system, data stream, deep learning, wireless sensor networks**

I. INTRODUCTION

Sensor networks are sort of wireless networks that used for environment monitoring, target tracking, structural health monitoring, precision agriculture, active volcano monitoring, transportation, human activity monitoring, and other monitoring applications[1-2]. These sensor data behave very differently from traditional database sources: they are continuous arrival in multiple, rapid, time varying, possibly unpredictable, unbounded streams, and keeping no record of historical information[3].

The underlying framework provides stream management and query processing mechanisms to support the online acquisition, management, processing, storage, and integration of data streams for distributed sensor networks. The rise of large-scale monitoring infrastructures such as wireless sensor networks poses distributed query processing challenges; the queries must be processed inside the system in a distributed fashion so that the performance of the typically resource-constrained processing nodes is maximized[4].

Furthermore, to enable high throughput and low latencies in the presence of high-rate data streams, the query processing operators must be placed adaptively across the network to minimize the data movement cost[5]. Three optimization levels are proposed to provide the maximum reduction in resources required to process data streams. This becomes possible due to the simultaneous use of different optimization levels and methods.

The growth of electronic commerce and the widespread use of sensor networks have created the demand for online processing and monitoring applications Traditional query execution techniques, which assume finite persistent datasets and aim for producing a one-time query result, become largely inapplicable in this new stream paradigm due to the following reasons:

The data streams are potentially infinite. Thus the existence of blocking operators in the query plan, such as group-by, may block query execution indefinitely because they need to see all input data before producing a result. Moreover, stately operators such as join may require infinite storage resources to maintain all historical data for producing exact results.

Data streams are continuously generated at query execution time. Meta knowledge about streaming data, such as data arrival patterns or data statistics, is largely unavailable at the initial query optimization phase. Therefore the initial processing decisions taken before query execution commences, including the query plan structure, operator execution algorithm and operator scheduling strategy, may not be optimal [6].

Stream environments are usually highly dynamic. For example, the data arrival rates may fluctuate dramatically. Moreover, as other queries are registered into or removed from the system, the computing resources available for processing may vary greatly. Hence an optimal query plan may become sub-optimal as it proceeds, requiring runtime query plan restructuring and in some cases even across-machine plan redistribution.

Resource management is a key challenge in a data stream management system, and it is a specific focus of our project [7-8]. There are a number of relevant resources in a DSMS: memory, computation, I/O if disk is used, and network bandwidth in a distributed DSMS. An important challenge in DBMSs has always been how to optimally utilize resources in order to maximize performance, while at the same time balancing other factors such as recoverability and reliability. This remains true in DSMS, but often with a different emphasis. DSMS deals with push-based sources that often feed streams in through a continuous query registered with the system. Often, the usefulness of a result depends upon how quickly it was produced. This means that minimizing latency and maximizing throughput are typically very important, making it highly desirable to minimize CPU time and memory usage. Techniques to accomplish this range from shedding tuples in order to reduce load on the system, to scheduling operator queues in order to optimally reduce the amount of tuples needed by the system. Many of these techniques, such as load shedding, fundamentally affect the accuracy of the query by, in essence, changing it. Therefore, it is necessary to develop approximation techniques and measures of performance to balance performance versus accuracy and to give some guarantee of a certain level of accuracy.

In a conventional DBMS, queries are run against datasets which are finite and which do not change while the queries are executing. The goal of the query optimizer in these systems is usually to minimize the average time required to execute each query. In a streaming data system, however, queries run indefinitely, and they must process datasets that are constantly growing. So DSMS must have a new query optimizer which should be more adaptive to generate optimum query plan to minimize the execution cost of the queries. If the system does not schedule the execution of query operators intelligently, the backlog of tuples at some of the operators may exhaust the available memory [9]. As mentioned above, conserving memory is important in a stream system because it reduces the demands on other system resources. Therefore, in a DSMS, intelligent operator scheduling is critical to effective resource management.

II. OVERVIEW

The first optimization level is sensor deployment. Sensor deployment is a critical issue, as it affects the cost, the amount of data that needs to be processed, and detection capabilities of a wireless sensor network. Although many previous efforts have addressed this issue, most of them assume that the sensing field is an open space. In this work, we consider the sensing field as conditional regions. The sensor location problem (SLP) is a nonlinear nonconvex programming problem which aims to locate sensors to monitor a constrained region. The objective is to determine the locations that will maximize the coverage. Three evolutionary algorithms, particle swarm optimization (PSO), genetic algorithm (GA) and adaptive hybrid optimization (AHO) were used to solve the SLP. AHO uses fuzzy logic controller (FLC) as an intelligent switching technique agent between different types of optimization techniques. Several variants (sensing patterns number of sensors and region constrains) were tested and compared in terms of percentage coverage and computation cost.

We can extract the following characteristics of data streams and processing requirements from its applications:

A data stream is a potentially unbounded sequence of data items generated by an active data source. A single data item is called stream element.

Stream elements arrive continuously at the system, pushed by the active data source.

The system neither has control over the order in which stream elements arrive nor over their arrival rates. Stream rates and ordering could be unpredictable and vary over time. A data sources transmits every stream element only once. As stream elements are accessed sequentially, a stream element that arrived in the past cannot be retrieved unless it is explicitly stored. The unbounded size of a stream precludes a full materialization.

Queries over data streams are expected to run continuously and return new results as new stream elements arrive. The ordering of stream elements may be implicit, i. e., defined by the arrival time at the system, or explicit if stream elements provide an application timestamp indicating their generation time. Complementary to the pure stream model, some applications need to combine data streams with stored data.

Sensor nodes are deployed in a sensor field. The deployment can be either done directly, by placing the sensor nodes in specific positions, or randomly, e.g., via aerial scattering in inaccessible terrains or disaster relief operations. Thus, the position of the sensor nodes in the sensor field may not be known in advance. After deployment, the sensor nodes perform some self-organization mechanisms to set up the network, e.g., by determining their neighbors and setting up routing tables. Self-organization is also required to adapt to changes of the network, e.g., caused by node failure due to energy exhaustion.

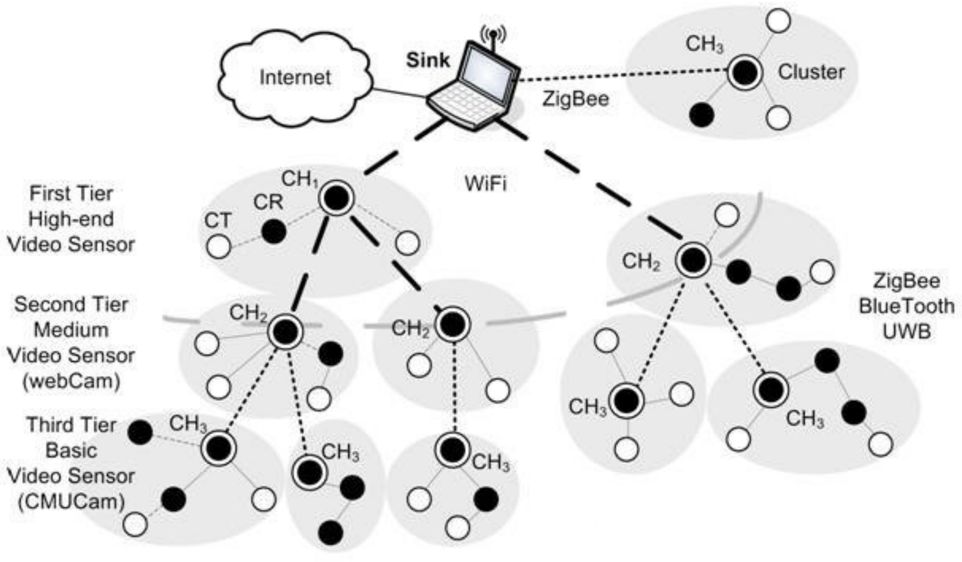

Figure 1. The basic framework of WSN

During operation of the WSN, sensor nodes perform measurements of some physical phenomena, e.g., the temperature at a certain location. This data is sent to the base station (BSI, called sink, for further processing. Since the transmission range of a sensor is limited, the sink may IOT be directly reached. Thus, messages are forwarded in a multichip communication to other sensor nodes which act as routers. Also sensor nodes may perform some operations on the data, e.g., data aggregation to decrease the amount of transmitted data. Since the transmission of data is much more cost-intensive than data processing, this is a commonly used approach to decrease the overall energy consumption. However, this may not be possible in all scenarios [10-11].

Figure 1 shows an example WSN. Multiple sensor nodes are deployed in a sensor field. The WSN autonomously performs measurements, e.g., following a certain time schedule, and send these measurements in regular intervals to the BS which send the collected data to the DSMS. The user generates a continuous query, e.g., "What is the highest temperature at the sensor field'?" using the DSMS. Then the user will get the query result continuously based on the sensors' data.

III. METHOD AND ALGORITHM

Data Sources: DBMSs operate on passive, persistent data sources, namely, finite relations stored on disk. In contrast, DSMSs operate on active data sources that continuously push data into the system as possibly unbounded, rapid, and transient data streams. Due to the stringent response time requirements of many streaming applications, data management primarily takes place in main memory, whereas DBMSs excessively make use of external memory. Moreover, it is unfeasible for a DSMS to store an entire stream due to the unknown and potentially unbounded stream size. Even if Larger stream fragments were written to disk, operating on this vast amount of data would drop system performance drastically and, thus, would conflict with fast response times. At most, fragments of query results may be stored in streaming applications whenever these need to support ad-hoc queries referencing the past. Access to databases is also necessary if applications need to combine relations with streams.

Query Types: While DBMSs execute one-time queries over persistent data, DSMSs execute continuous queries over transient data. Whenever the user issues a query, the DBMS computes and outputs the results for the current snapshot of the relations. After that, the processing for this query is completed. In a DSMS, however, queries are long-running; they remain active in the system for a long period of time. Once registered at the DSMS, a query generates results continuously on arrival of new stream elements until it is deregistered.

Query Answers: DBMSs always produce exact query answers for a given query, whereas continuous queries usually provide approximate query answers. The reasons are: Many continuous queries are not computable with a finite amount of memory, e. g., the Cartesian product over two infinite streams.

Some relational operators such as group-by are blocking because they must have seen the entire input before they are able to produce a result. The data can accumulate faster than the system can process the data. In general, high quality approximate answers are acceptable for users or applications. Moreover, recently arrived data is considered more accurate and useful.

Processing Methodology: A DBMS employs a demand-driven computation model, where processing is initiated when a query is issued. Tuples are typically read from relations in a pull-based manner using scan-based or index-based access methods.

Conversely, query processing in a DSMS is data-driven because the query answer is computed incrementally on arrival of new stream elements. Hence, the underlying active data sources trigger processing in a push-based fashion. Without explicit buffering, DSMSs have to access stream elements sequentially in arrival order, whereas DBMSs have random access to tuples.

Query Optimization: DBMSs optimize queries prior to execution. First, the optimizer generates a set of semantically equivalent query plans but with different performance characteristics. Based on a cost model incorporating metadata about the underlying data and system conditions, the optimizer then selects the plan with the lowest estimated costs. While this static optimization is adequate for one-time queries, the long-running nature of continuous queries entails DSMSs requiring further query optimizations at runtime to adapt to changing stream characteristics and system conditions. Data distributions and arrival rates of streams and also query workload may vary over time. Without runtime adaptations, plan and system performance may degrade significantly during the lifetime of a continuous query.

The linear equation can be expressed into the following simplified forms:

$$L(\nabla,\omega)f(x,\omega)=0$$

$$L(\nabla,\omega)=T(\nabla)+\omega^2\rho \mathrm{J} \quad (1)$$

In which,

$$T(\nabla)=\begin{Vmatrix} T_{ik}(\nabla) & t_i(\nabla) \\ t_k^T(\nabla) & -\tau(\nabla) \end{Vmatrix},\ \mathrm{J}=\begin{Vmatrix} \delta_{ik} & 0 \\ 0 & 0 \end{Vmatrix},$$

$$f(x,\omega)=\begin{Vmatrix} u_k(x,\omega) \\ \varphi(x,\omega) \end{Vmatrix} \quad (2)$$

Consider delay, the L can be expressed as:

$$L^0=\begin{Vmatrix} C^0_{ijkl} & e^0_{kij} \\ e^{0T}_{ikl} & -\eta^0_{ik} \end{Vmatrix} \quad (3)$$

These functions can be expressed in the following form:

$$C(\mathrm{x})=C^0+C^1(\mathrm{x}),\quad e(\mathrm{x})=e^0+e^1(\mathrm{x}),$$

$$\eta(\mathrm{x})=\eta^0+\eta^1(\mathrm{x}),\quad \rho(\mathrm{x})=\rho_0+\rho_1(\mathrm{x}) \quad (4)$$

The value with superscript of 1 represents the difference below:

$$C^1=C-C^0, e^1=e-e^0,$$

$$\eta^1=\eta-\eta^0, \rho_1=\rho-\rho_0 \quad (5)$$

And local fractional integral of $f(x)$ defined by Eq.6.

$$ {}_aI_b^{(\alpha)}f(t)=\frac{1}{\Gamma(1+\alpha)}\int_a^b f(t)(dt)^\alpha $$
$$ =\frac{1}{\Gamma(1+\alpha)}\lim_{\Delta t\to 0}\sum_{j=0}^{j=N-1} f(t_j)(\Delta t_j)^\alpha \quad (6)$$

Its local fractional Hilbert transform, denoted by $f_x^{H,\alpha}(x)$ is defined by

$$H_\alpha\left\{f(t)\right\} = \hat{f}_H^\alpha(x) = \frac{1}{\Gamma(1+\alpha)} \fint_R \frac{f(t)}{(t-x)^\alpha}(dt)^\alpha \tag{7}$$

Where x is real and the integral is treated as a Canchy principal value, that is,

$$\frac{1}{\Gamma(1+\alpha)} \fint_R \frac{f(t)}{(t-x)^\alpha}(dt)^\alpha = \lim_{\varepsilon\to 0}[\frac{1}{\Gamma(1+\alpha)} \int_{-\infty}^{x-\varepsilon} \frac{f(t)}{(t-x)^\alpha}(dt)^\alpha + \frac{1}{\Gamma(1+\alpha)} \int_{x+\varepsilon}^{\infty} \frac{f(t)}{(t-x)^\alpha}(dt)^\alpha] \tag{8}$$

Due to advances in wireless communications and electronics over the last few years, the development of networks of low-cost, low-power, multifunctional sensors has received increasing attention. These sensors are small in size and able to sense, process data, and communicate with each other, typically over an RF (radio frequency) channel.

The well-known IEEE 802.11 family of standards was introduced in 1997 and is the most common wireless networking technology for mobile systems. It uses different frequency bands, for example, the 2.4-GHz band is used by IEEE 802.11b and IEEE 802.11 g, while the IEEE 802.11 a protocol uses the 5-GHz frequency band. IEEE 802.11 was frequently used in early wireless sensor networks and can still be found in current networks when bandwidth demands are high (e.g., for multimedia sensors). However, the high-energy overheads of IEEE 802.11-based networks make this standard unsuitable for low-power sensor networks.

Typical data rate requirements in sensor networks are comparable to the bandwidths provided by dial-up modems, therefore the data rates provided by IEEE 802.11 are typically much higher than needed. This has led to the development of a variety of protocols that better satisfy the networks' need for low power consumption and low data rates. For example, the IEEE 802.15.4 protocol has been designed specifically for short-range communications in low-power sensor networks and is supported by most academic and commercial sensor nodes. When the transmission ranges of the radios of all sensor nodes are large enough and the sensors can transmit their data directly to the base station, they can form a star topology. In this topology, each sensor node communicates directly with the base station using a single hop. However, sensor networks often cover large geographic areas and radio transmission power should be kept at a minimum in order to conserve energy; consequently, multi-hop communication is the more common case for sensor networks.

IV. EXPERIMENT RESULT

The data stream is limitless, and the arrival of the data is in individual (normal) or in batch (bursts). Data arrival mode is often described with arrival interval. The random arrival mode applied in the system appears very complex, and different probability distributions have to be adopted for different systems. Data stream model almost obeys the exponential distribution function; so our simulation will generate the data according this function using random numbers generation for the time intervals and generate the data in normal and burst mode to investigate the different impact of the two modes.

Random numbers in simulation are never random. Rather, they are produced using deterministic algorithms. Algorithms take a seed value and perform some deterministic calculations on them to produce a random number and the next seed. Such algorithms and their implementations are called random number generators or RNGs, or sometimes pseudo random number generators or PRNGs to highlight their deterministic nature.

Starting from the same seed, RNGs always produce the same sequence of random numbers. This is a useful property and of great importance, because it makes simulation runs repeatable. RNGs produce uniformly distributed integers in some range, usually between 0 or 1 and 232 or so. Mathematical transformations are used to produce random variants from them that correspond to specific distributions, in our case it will be the exponential distribution and manually configure seed values to use the same seeds for several simulation runs for ensuring a fare comparison between the different scheduling algorithms and the proposed algorithm.

Queue is the memory schema which is used to store the data packet received from the data stream source, and to store the intermediate data between the operators. These queues use the FIFO as the plan to output the data packet with its arrival order to be the inputs for the next operator. In the proposed approach each queue has an embedded ready condition. This condition is tested after each data insertion in the queue. If the condition is met, queue will send a message to the scheduler to inform it that the bonded operator has enough data and ready to be run.

After receiving request message form the bonded operator the queue send the stored data to the operator according to the service discipline; individual means to send only one packet each request and bulk means to send all stored data packets. These modes differ from scheduling algorithm to another, as example FIFO uses individual mode to ensure the arrival order and the Chain uses the bulk mode to maximize the memory reducing.

The clustering process will be done by the query analyzer as shown in the ADDSMS system design. So the execution engine in the query processor will have a number of threads that equal to the number of clusters resulted from the clustering phase. The adaptive manner of that scheduling schema resulting from reiterating calculates the operators' clusters during the query running either periodically or when operators' parameters have significant changes. These changes result from the input stream rate changes which affect the cost model of the operators or the varying of data itself may resulting the change of the selectivity of the operators. The query analyzer will recalculate the clusters when it receives a new query plan from the distribution manger that may be different from the plan.

The S-mean clustering method now is ready to apply. First the features of operators should be extracted in order to be used in the similarity measurements. Query execution can be captured by a data flow diagram, where every tuple passes through a unique operator path. Thus queries can be represented as rooted trees. Over adequately large memory units, we can assume that if an operator with selectivity s operates on inputs that require one unit of memory, its output will require s units of memory.

In all experiments only select, project and join operators are used. Join had been modeled as a sliding-window joins. The query plan consists of multiple queries with the same structure with different operator selectivity. Each query uses two data stream sources and applies two successive selections for each data stream. After that is applying a join operator with specific sliding window. The output from join operator will pass thought anther 2 filtering operators (select operators). Finally using a projection operator the output stream will be generated and sent to the data stream sink. Each query consists of 8 operators: 6 selections, one projection and one join operator. During the experiment, each operator uses a fixed computing cost per tuple processing, the same cost is used for calculate the total cost model and then the clustering phase. Figure 2 shows average memory requirements and latency for 40 operators query plan in normal data mode while figure 3 shows average memory requirements and latency for 40 operators query plan in burst data mode. Average memory requirements and latency for 320 operators query plan in normal data mode is shown in figure 4 and average memory requirements and latency for 320 operators query plan in burst data mode is shown in figure 5.

V. DISCUSSION

In this experiment, the performance of the COS will be compared with two GTS algorithms, which are FIFO and Chain algorithms besides the OTS (multi-threaded). The experiment will inspect two metrics. The first is the data arrival modes, there are two modes normal mode and burst mode. The second metric is the number of the operator in the query plan we have run the simulation for different number of operators starting from 10 operators until 320 operators. Two examples will be shown only, one for small query plan (40 operators) and one for large query plan (320 operators). As a result we will have four set of testes (small query plan with normal data arrival, small query plan with burst data arrival, large query plan with normal data arrival and large query plan with burst data arrival). The four scheduling methods (GTS-FIFO, GTS-Chain, OTS and COS) will be applied for all experiments in order to compare them. Note that GTS-FIFO, GTS-Chain, OTS and COS algorithms will be mentioned in figures as FIFO, Chain, Multi-threads and clustered respectively.

In all experiments, the COS proves its superior performance. As shown in figure 2 that illustrate the result for small query plan with normal data arrival, the COS have an average memory requirement almost as Chain and less than FIFO and multi-threaded and having a great improvement in the latency issue near to FIFO which have the worst memory requirement. In the case of small query plan with burst data arrival shown in figure 3, the COS proves its advantage in burst mode by having the lowest memory requirement and nearly the best latency.

As shown in figure 4 that illustrate the result for large query plan with a normal data arrival, the COS had an average memory requirement less than Chain, FIFO and mufti-threaded and having a great improvement in the latency issue better than all of them. In the case of the large query plan with the burst data arrival shown in figure 5, the COS proves its advantage in burst mode by having the lowest memory requirement and nearly the best latency. For the small query plan, the power of the COS is not quite clear but in the large query plan the enormous performance has been verified for normal and burst data modes.

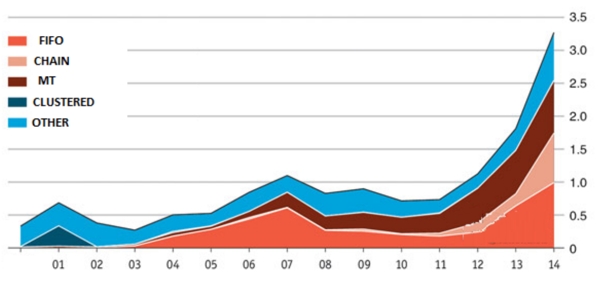

Figure 2. Average memory requirements and latency for 40 operators query plan in normal data mode

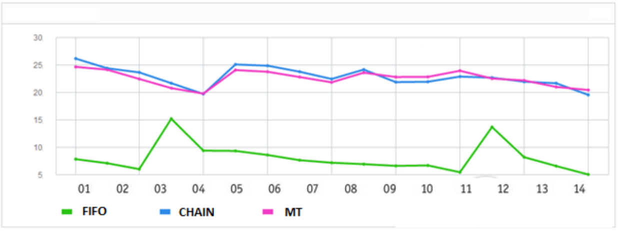

Figure 3. Average memory requirements and latency for 40 operators query plan in burst data mode

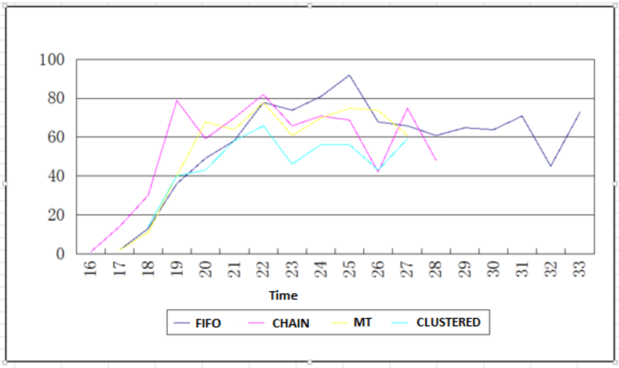

Figure 4. Average memory requirements and latency for 320 operators query plan in normal data mode

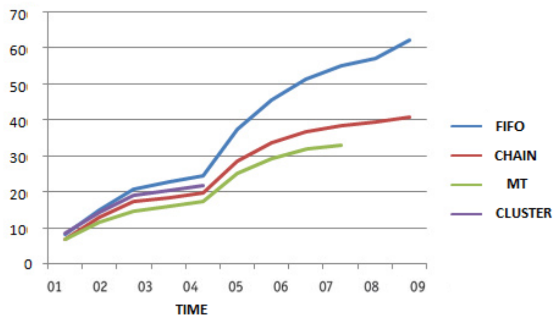

Figure 5. Average memory requirements and latency for 320 operators query plan in burst data mode

VI. CONCLUSION

The issue of continuous query processing was studied. The problem of operator scheduling in query processor has been focused on, with the goal of minimizing memory requirements and tuples latency. An adaptive operator scheduling COS is proposed. It clusters different operators into a number of groups based on its selectivity and computing cost. S-mean is used to operators' clustering that does not require specification of clusters count.

COS is Compared with the conventional scheduling methods FIFO, Chain and OTS.COS proved its high performance for all situations compared with other techniques. Furthermore, we showed that COS scheduling performs very well in terms of scalability and robustness. COS also able to use the memory and the computation resources in an efficiency manner that makes it continue works with limited resources, where other techniques lose their stability.

As a final result the novel proposed technique provides adaptive, flexible, reliable, scalable and robust design for continuous query processor that can be the core for adaptive DSMS.

REFERENCES

[1] H. Jing, “Node deployment algorithm based on perception model of wireless sensor network,” *International Journal of Automation Technology*,vol.9, no.3, pp. 210-215, April 2015. https://doi.org/10.20965/ijat.2015.p0210

[2] Z. Zhang, D. He, L. Jing, et al., “ Robot Arm Trajectory Planning Method,” *Sensors & Transducers,* pp. 1621-1626, 2014.

[3] H. Jing, “Routing optimization algorithm based on nodes density and energy consumption of wireless sensor network,” *Journal of Computational Information Systems*, vol. 11, no.14, pp. 5047-5054, July 2015.

[4] L.S. Li, et al.,"Fractal-Based Outlier Detection Algorithm over RFID Data Streams," *International Journal of Online Engineering*, vol. 12, no 01, pp. 35-41, January 2016. https://doi.org/10.3991/ijoe.v12i1.5171

[5] Z. Lv, T. Yin, H. Song, et al., “Virtual Reality Smart City Based on WebVRGIS,” *IEEE Internet of Things Journal* 2016.

[6] Z. Zhang, J. Tang, I. Huang, et al., “Research on Kinematics for Inhibition Fluttering of Robot Arm,” *Sensors & Transducers*, pp. 161-171, 2013.

[7] L. Wang, et al., “Mechanics and energetics in tool manufacture and use: a synthetic approach,” *Journal of the Royal Society Interface*, pp. 111-114, 2014. https://doi.org/10.1098/rsif.2014.0827

[8] Z. Li, et al., "A low latency, energy efficient MAC protocol for wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 10, no. 6, pp.1-9, 2015. https://doi.org/10.1155/2015/946587

[9] J. Niu, et al., "R3E: Reliable Reactive Routing Enhancement for Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics,* vol. 10, no. 1, pp.784-794, 2014. https://doi.org/10.1109/TII.2013.2261082

[10] C. Huang, Y. Tseng, and L. Lo, "The coverage problem in three-dimensional wireless sensor networks," *Journal of Interconnection Networks,* vol. 08, no. 03, pp. 3182-3186, 2015.

[11] H. Yang, et al., "Toward resilient security in wireless sensor networks," *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing ACM,* pp.34-45, 2015.

AUTHOR

Li Yue-jie is with the Ordos Institue of Technology, Inner Mongolia, China (liyuejieli@tom.com).

The paper is supported by Inner Mongolia university research project (No. NJZY16383) Submitted 09 September 2016. Published as resubmitted by the authors 23 October 2016.