International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 17 (2017) pp. 6323-6329

Wireless Sensor Network Storage with Light Weight File System using Update Logic for Cloud Interface

R.Kumaravelu[1], Satheesh. A[2], Poornima. S[3] and V. Murugesh[4]

[1, 2]School of Computer Science and Engineering, VIT University, Vellore, India.
[4]Department of Computer Science, College of Computer Science, King Khalid University, Abha, Saudi Arabia.
[3]Vivekha Charitable Trust, Kanchipuram, India.

ORCIDs:[1]0000-0002-2442-812X, [2]0000-0003-3961-1581, [3]0000-0002-6403-6884, [4]0000-0002-5861-2857

Abstract

This paper focuses on a wireless sensor network used for an intelligent storage technique for the data acquisition process. The data acquired from the various sensors are stored locally with timestamp using the update logic in this work. The rapid development of IoT enables the embedded system to send/receive the information across the two remote entities. In the multi-sensor wireless sensor nodes, the data are usually acquired and stored in a structured block file I/O. This data collection is done periodically through multi-sensor wireless sensor nodes and file system is updated in block level. Then, the wireless sensor data are organized without duplication between successive data collection. But in Hadoop Distributed File System (HDFS), the block size is fixed with 64MB. Whereas in other implementation has File system concept with successive redundant and this would significantly increase the file size. The proposed update logic flat file system is a variable with light weight feature. Our proposed logic with Update logic minimizes the network contention by the lightweight file without successive redundant information.

Keywords: File System, SPI, UART, ARM controller

INTRODUCTION

Environmental sensors monitor various climate factors, including temperature, humidity, and ambient light, air quality or a combination of these. As the Internet of Things (IoT) grows, so does the demand for wirelessly connected environmental sensors. Sensors are the eyes and ears. Sensors are the troops of the "Internet of Things", the on-the-ground pieces of hardware doing the critical work of monitoring processes, taking measurements and collecting data. Sensors are now found in a wide variety of applications, such as smart mobile phones, automotive systems, industrial control, health care, oil exploration and climate monitoring. Sensors are used almost everywhere, and now sensor technology is beginning to closely mimic the ultimate sensing machine to the human being.

RELATED WORK

Using the wireless sensor network the way to measure the temperature in an intelligent way is described with wifi network [1]. Using the Distributed Memory File System the integration of Chunk and Master server with Write Scheduling and Write Cache for merging towards HDFS is illustrated [2]. The process execution policy like dispatch, replay and resource release implemented in the Falkon architecture with Push – Pull method [3]. The indexing based Light weight based file system JFFS2 and YAFFS2 with in-memory structure using B-tree [4].

The File System APIs are incorporated in the Light Weight File system with Process management service [5]. The Cloud Integrity Security and access control at local level cloud implementation is proposed. The scientific challenges of integration of copper physical systems have been explored [6].

The Graph model based RFID and Bluetooth based indoor positioning and tracking has been deployed [7]. The Master Fault tolerance, Storage Layer and Resource allocation through Edge algorithm has been implemented [8].

The importance of Scalable and elastic data management in Big data analytics is explored [9]. The Layered architecture for IndexFS Server and IndexFS Client with Log structured metadata storage format has been implemented [10].

IMPLEMENTATION AND METHODS

The Lightweight File System for the Wireless Sensor node is created using Flash Memory with FAT (File Allocation Table) system. The data acquisition happens in two ways with the following names 1) Continuous Data Acquisition in a period way using Update Logic 2) Query based Data Acquisition.

In the Continuous Data Acquisition mode, every wireless sensor node is capable of organizing its sensor information continuously at periodical interval. The periodical interval is configurable. The main logic behind Update logic is to eliminate successive environmental data acquisition values and this leads to significant reduction of the network contention during the uploading process. In Query based Data Acquisition, the sensor information is sent from the sensor node at the instant.

A. MASTER NODE WITH SLAVE WIRELESS SENSOR NODE

Each Wireless Sensor Node with Fat File System interfaces with multiple sensors. Each Wireless Sensor Node is having unique ID and the sensors of the node are numbered. The Sensor scan scheduler which is running on the Wireless Sensor Node is intended to access and acquire the sensor information and it stores the information on the FAT File system using Block Level File I/O in Embedded C Language.

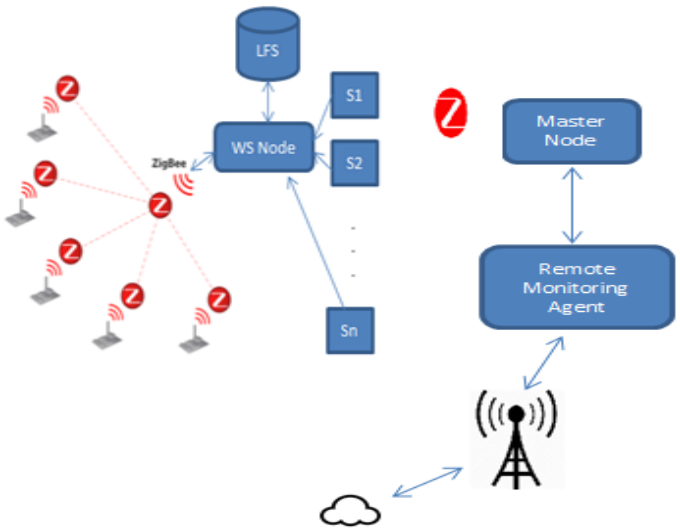

Figure 1: System Architecture

The above figure 1 depicts the Data Acquisition Architecture from Slave Nodes to Master Node. The Master node is capable to synchronize Slave nodes with itself in Data Acquisition. The Slave nodes and Master node is connected through Wireless. The Slave nodes are expecting a command from Master node. According the command, Slave nodes are operating and sense the data which is sent to Master node. Based on the Master node current mode the data is manipulated. The request command to a Slave node is in the following format.

CType(1)	Mode(1)	Slave_Node_Id(2)	Sensor_Id(2)

Figure 2: Command Format goes to the Slave node

The figure 2 – shows the format of a Command word. The data frame starts with a value CType and it should be 'C' which denotes that it is a Command to a Slave node. . The possible values of Mode is either 1 (Continuous Data Acquisition using Update Logic) or 2 (Query based Data Acquisition). The Master node is capable of controlling maximum of 99 Slave nodes could participate and thus Slave_Node_Id is ranging from 01 to 99. Each Slave node is capable of interfacing 99 sensors which are connected to it through wired/unwired. This enables a Master node to specify the value between 01 and 99 for Sensor_ID while sending the command frame.

The response generated by the Slave node to the Master node is in the following format.

CType (1)	Slave_Node_Id (2)	Sensor_Id(2)	Sensor_V alue(1)

Figure 3: Response Format of the Slave node

The figure 3 – shows the Response format. Here, the possible value of CType is R and Slave_Node_Id and Sensor_ID values are ranging between 01 and 99. The Sen_Value denotes the Sensory information and this field size is 10 bytes.

The Master node always makes a command in two ways. The Remote monitoring agent determines how to interact with a slave node from the Master node. During the Update logic the interval period also set by the Remote monitoring agent.

The Master node is having its external flash memory system using the FAT file system for organizing the sensory information. The FAT file system is light weight data storage logic with block level file I/O using Embedded 'C'.

```
struct LW_Sens_Info
{
Char Master_Node_Id [2]; /* Value between '00' and '99' */
  Char Slave_Node_Id [2];     /* Value between '00' and '99' */
  Char Sensor_Id [2];         /* Value between '00' and '99' */
  Char Sen_Value [10];        /* 10 Bytes of information */
  time_t Sen_Time;            /* Date and Time Stamp */
};
```

The Master node stores the sensor information from the Slave nodes when mode is set to 1 between them otherwise it is transferred to the Remote monitoring agent. The Lightweight local storage of Master node avoids the network contention by avoiding the every response transfer to the Remote monitoring agent. At a later point of time, when the Master node's light

weight file system size reaches the file size limit the light weight file system content is transferred to the Remote monitoring agent. After successful transfer of the light weight file system to the Remote monitoring agent it is flushed out. Hence, the Master node local storage will have an adequate amount of free storage area.

The Local Storage file system stores the LW_Sens_Info in a CSV format and a sample view is given below:

Table 1: Sample Sensor Information in CSV format

Master_Node_ID	Slave_Node_ID	Sensor_ID	Sensor_Value	Date	Time
1	1	1	223	27-07-2017	10:30:45
1	1	2	400	27-07-2017	10:30:50
1	1	3	100	27-07-2017	10:31:45
1	1	4	180	27-07-2017	10:32:50

B. CONTINUOUS DATA ACQUISITION WITHOUT UPDATE LOGIC

The sensor data arrival rate is estimated using the equation:

DAR = (DS / IV) * (NS) * (LW_Sens_Info)

Where

DAR => Data Arrival Rate

DS => Seconds Count Per Day

IV => Data Collection Threshold Period in Seconds

NS => No. of Sensors connected to a Node

LW_Sens_Info => Sensor Information Data Size in Bytes

Table 2: Data Arrival Rate in KB for IV is 60 seconds

DS	IV	NS	LW_Sens_Info	DAR in KB
86400	60	1	30	42.19
86400	60	2	30	84.38
86400	60	3	30	126.56
86400	60	4	30	168.75
86400	60	5	30	210.94
86400	60	6	30	253.13
86400	60	7	30	295.31
86400	60	8	30	337.50
86400	60	9	30	379.69
86400	60	10	30	421.88

The Local FAT File system storage writes the information in a sequential organization. In this accessing mode repeated values from each sensor at successive time stamps are organized in a flash memory file system.

Table 3: Data Arrival Rate in KB for IV is 180 seconds

DS	IV	NS	LW_Sens_Info	DAR in KB
86400	180	1	30	14.06
86400	180	2	30	28.13
86400	180	3	30	42.19
86400	180	4	30	56.25
86400	180	5	30	70.31
86400	180	6	30	84.38
86400	180	7	30	98.44
86400	180	8	30	112.50
86400	180	9	30	126.56
86400	180	10	30	140.63

C. UPDATE LOGIC IMPLEMENTATION FOR CONTINUIUS DATA ACQUISITION

The redundant data storage in a successive data acquisition is eliminated using Update logic significantly, which is based on the sensor acquisition. The sensor data arrival rate for the update logic implementation of continuous data acquisition is estimated for every sensor using the equation below:

DAR = (DS / (Rep_Val * IV)) * (LW_Sens_Info)

Where

DAR => Data Arrival Rate

DS => Seconds Count Per Day

Rep_Val => Successive Repeated Sensor Value

IV => Data Collection Threshold Period in Seconds

NS => No. of Sensors connected to a Node

LW_Sens_Info => Sensor Information Data Size in Bytes

From the successive repeated values from the sensor, only any one of the value is considered for storage purpose other values will be discarded, this significantly reduces the file system size.

Table 4: Data Arrival Rate in KB with Update Logic

					Update Logic DAR in KB		
DS	IV	NS	LW_Sens _Info	W/o UL DAR	Rep_Val 3	Rep_Val 5	Rep_Val 7
86400	60	1	30	42.19	14.06	8.44	6.03
86400	60	2	30	84.38	28.13	16.88	12.05
86400	60	3	30	126.56	42.19	25.31	18.08
86400	60	4	30	168.75	56.25	33.75	24.11
86400	60	5	30	210.94	70.31	42.19	30.13
86400	60	6	30	253.13	84.38	50.63	36.16
86400	60	7	30	295.31	98.44	59.06	42.19
86400	60	8	30	337.50	112.50	67.50	48.21
86400	60	9	30	379.69	126.56	75.94	54.24
86400	60	10	30	421.88	140.63	84.38	60.27

The sensor information is retrieved and compared before the local file storage. The Rep_Val determines the repetition factor of the sensor value in the update logic implementation and this significantly reduces the local file system storage during the sensor's similar data across the different time stamp.

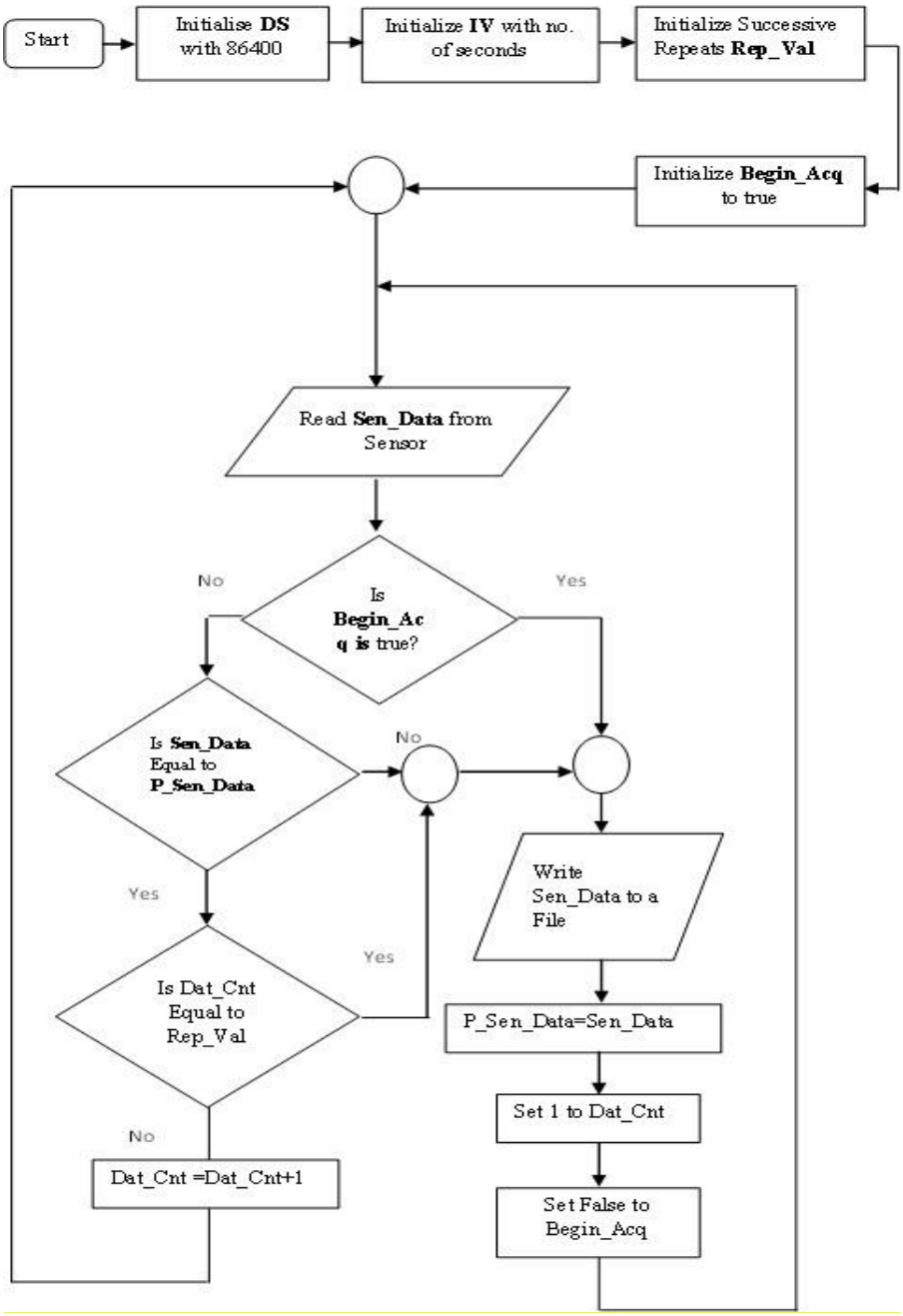

Figure 4: Update logic – Flow chart

The figure 4 shows the Update logic implementation in Continuous Data Acquisition mode. After resetting the Wireless Sensor Node, a new Time Stamp is recorded. Based on the reset Time Stamp a parameter DS is considered in the evaluation. After getting the configuration parameters from the Master Node, the slave node initializes a new Stamp and associate with new parameters like IV, Rep_Val.

D. FLASH MEMORY CARD SYSTEM INTERFACE

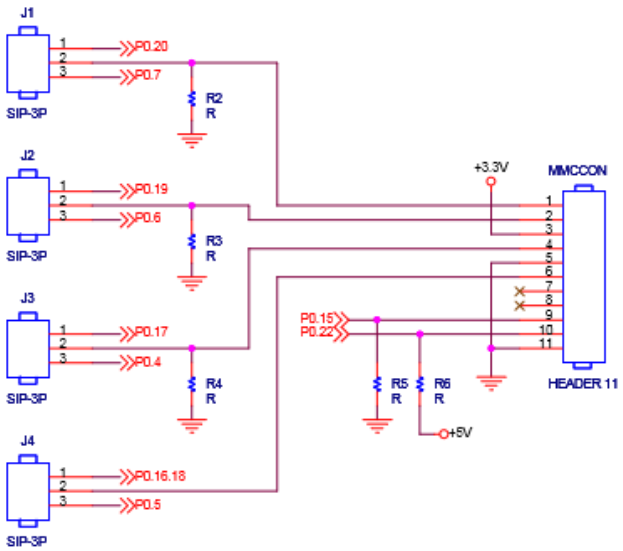

Figure 5: SPI Interface for Flash Memory Card Interface with LPC2148

The GPIO pins P0.20/P0.7, P0.19/P0.6, P0.17/P0.4 and P0.15,P0.16/P0.5 uses SPI0 and SPI1 for the local file system implementation with external detachable flash memory.

Table – SPI Interface Descriptions

Pin Description	Pin ID
SSEL0/SSEL1	P0.7/P0.20
MISO0/MISO1	P0.5/P0.18
MOSI0/MOSI1	P0.6/P0.19
SCK0/SCK1	P0.4/P0.17

Using the SPI Interface of LPC2148 the interface is configured using the switches J1, J2, J3 and J4. The SPI mode is compliant with the Serial Peripheral Interface (SPI) specification. Its bus architecture includes signals based on the above table. With both SD/MMC and SPI modes use the single master/multiple slave bus architecture to communicate with the end devices and/or media cards. The Wireless Node with Local File system creates a new file after the successful transmission of Light Weight Update logic file to a Master Node else it works in append mode in order to preserve the existing content.

E. LIGHT WEIGHT FILE CREATION LOGIC

```
//----- PROCESS FAT FILING SYSTEM-----
ffs_process ();
If (ffs_card_ok)
{
  our_file = ffs_fopen("lwfile_01.txt\0" , (const char*)"w");
}
```

The above Embedded C code creates a new lightweight file for update logic during the reset time.

*int ffs_fwrite (const void *buffer, int size, int count, FFS_FILE *file_pointer)*

The above API is intended for appending the sensor information to the light weight file system.

F. UART INTERFACEFOR ZIGBEE COMMUNICATION

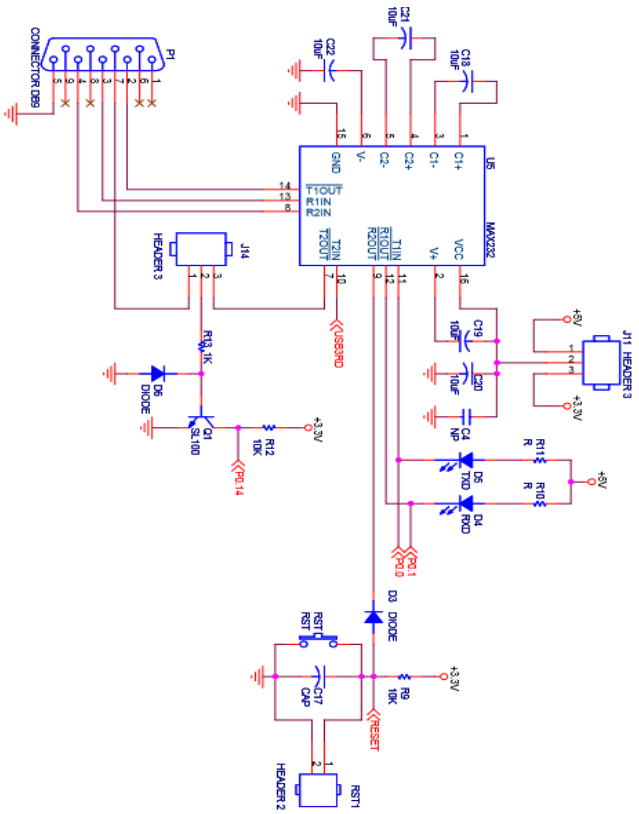

Figure 6: UART Interface for Zigbee

The Zigbee unit is interfaced with UART0 of the Microcontroller at the baudrate of 9600bps. The following Embedded C code configure the RS232 based Zigbee interface.

```
Void UART0_INIT(unsigned int baud_rate)
{
PINSEL0 |=0x5;
U0LCR =0x83;
baud_value=(pclk/16)/baud_rate;
U0DLM =baud_value/256;
U0DLL=baud_value % 256;
U0LCR =0x3;
U0FCR =0x07;
}
```

RESULTS AND DISCUSSIONS

During Data Arrival the light weight file is getting appended on the Flash memory card. Significantly write access is minimized by the update logic implementation. Without Update logic the repeated value storage has no purpose. But that could lead to network contention during the file transferring to a Master Node.

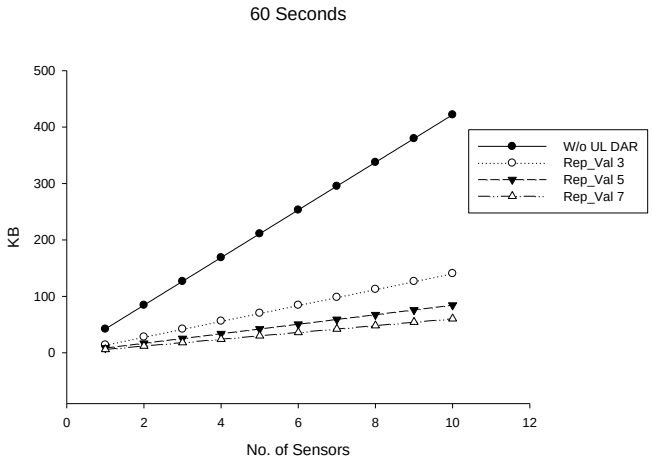

Figure 7: Data Arrival Rate comparison of Update Logic and without Update Logic for the Interval 60 seconds

In general, the repeated value factor 3 requires 33.33% write access, the repeated value factor 5 requires 20% write access and repeated factor 7 requires 14.29% write access when compared to Without Update logic method. This would save disk area and file transfer time significantly illustrated in the Figure5 , Figure 6 and Figure 7.

The Update logic based data collection enables the wireless sensor node data logging in a smarter way and network traffic contention in Big Data environment.

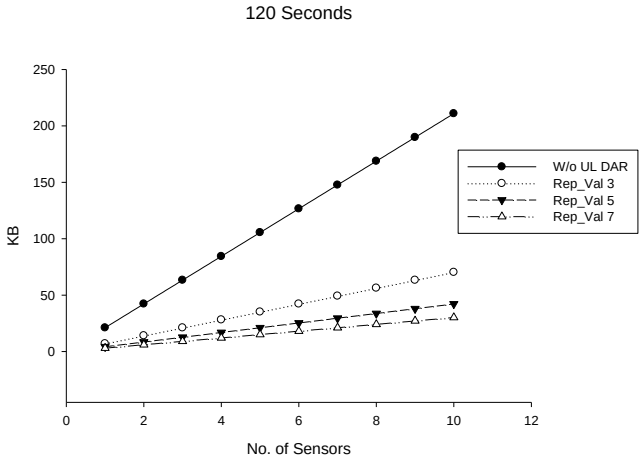

Figure 8: Data Arrival Rate comparison of Update Logic and without Update Logic for the Interval 120 seconds

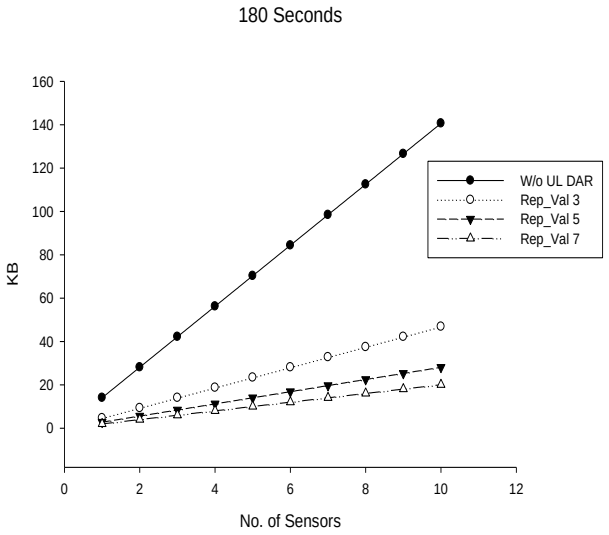

Figure 9: Data Arrival Rate comparison of Update Logic and without Update Logic for the Interval 180 seconds

CONCLUSIONS

In this paper, Data acquisition using wireless sensor node has been implemented Wihout Update logic and Update Logic method. The Without Update logic methods gets the sensor information and stores it in a flash memory system continuously as per the IV period. This method organizes the information in a successive redundant. Hence, according to the time period IV (Interval Value) sensor data accumulates in flash memory system. This leads to creating network contention during the file transfer. Whereas the Update Logic method nullifies the successive redundant values of the sensors. Hence, this method significantly reduces the file size and network contention.

REFERENCES

[1] U. Sarojini Naidu, R. Malikarjun, "ARM Based Wireless Sensor Networks for Temperature Measurement" International Journal of Mathematics and Computer Research, vol. 1 issue 1, pp. 1-4, Feb 2013.

[2] Xingjun Hao, Peiquan Jin, "Efficient Storage of Multi-Sensor Object -Tracking Data", IEEE Transactions on Parallel and Distributed Systems, vol. 27 issue 10. pp 2881-2894, October 2016

[3] loan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster and Mike Wilde, "Falkon: a Fast and Light-weight tasK executiON framework",

[4] Jaegeuk Kim, Hyotaek Shim, Seon_Yeong Park and Seungryoul Maeng, "Flashlight : A Lightweight Flash File System for Embedded Systems",

ACM Transactions on Embedded Computing Systems, vol. 11S issue 1, pp. 18.1-18.23, June 2012

[5] Zhan Shi, Dan Feng, Heng Zhao and Lingfang Zeng, "USP: A Lightweight File System Management Framework", 2010 Fifth IEEE International Conference on Networking, Architecture, and Storage, pp. 250-256, 2010

[6] Yenumula B Reddy,"Cloud-based Cyber Physical Systems: Design Challenges and Security Needs", 2014 10th International Conference on Mobile Ad-hoc and Sensor Networks", pp. 315-322, 2014

[7] C. S. Jensen, H. Lu, and B. Yang, "Graph model based indoor tracking," in Proc. 10th Int. Conf. Mobile Data Manage.: Syst., Services Middleware, 2009, pp. 122–131.

[8] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in Proc. ACM Symp. Cloud Comput., 2014, pp. 1–15.

[9] B. Cui, H. Mei, and B. C. Ooi, "Big data: The driver for innovation in databases," Nat. Sci. Rev., vol. 1, no. 1, pp. 27–30, 2014.

[10] K. Ren, Q. Zheng, S. Patil, and G. A. Gibson, "IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion," in Proc. Int. Conf. High Performance Comput., Netw., Storage Anal., 2014, pp. 237–248.