International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 12 Issue 1 ǁ January 2024 ǁ PP. 331-338

Design and Implementation of EDNA-based Time Series Caching Framework

Zhenlin Zhang[1], Mingxi Zhang[1], Zhou Liu[2], Fei Zhou[2], Gaobin Hu[2], Haiyang Guo[3]

[*1] *College of Communication and Art Design, University of Shanghai for Science and Technology, Shanghai, China*
[2] *Jiangsu Guoxin Jingjiang Power Generation Co., Ltd., Taizhou, China*
[3] *Henan Cryptography Administration, Henan, China*
Corresponding Author: Mingxi Zhang

Abstract
Sensors in the power production process generate a large amount of time-series data, and common power data systems, such as Electricity Data Network Architecture (EDNA), suffer from slow processing speeds and inflexible processing methods. To address the above problems, we design a time-series caching framework to transfer time-series data from low-speed devices to memory and use a sliding window model to organize power data into a time-series cache structure in a reasonable and efficient way, thus improving the processing speed of the data. We conducted extensive experiments on the production environment of Jiangsu Guoxin Jingjiang Power Generation Co. Ltd. and the results show that the proposed framework significantly improves the performance in data reading and processing.

Keywords: *Caching system, Smart power plants, Stream data processing, Sliding window model*

Date of Submission: 19-01-2024 Date of acceptance: 02-02-2024

I. INTRODUCTION

In recent years, as coal-fired power plants have gradually increased their requirements in terms of energy saving and emission reduction [1] and deep data peaking [2], the power industry has been increasingly relying on the Internet of Things [3] and computer [4] technologies to ensure production and transportation, and time-series power data processing [5] has become the basis and key to the whole process. Many power plants currently use relational databases [6] to access and process power data, such as the EDNA database. With the development of modern power networks, power plants have higher requirements for data reading, processing speed and processing flexibility [7], and traditional databases are difficult to meet this demand [8].

Without changing the main database of the power plant, in order to improve the speed and flexibility of sensor data processing, this paper develops a time series caching framework based on EDNA for managing time series power real-time and historical data. After the practical application of the framework in Jingjiang Power Plant, it obviously improves the data processing speed and meets the requirements of power plant data management [9].

II. FRAMEWORK IMPLEMENTATION

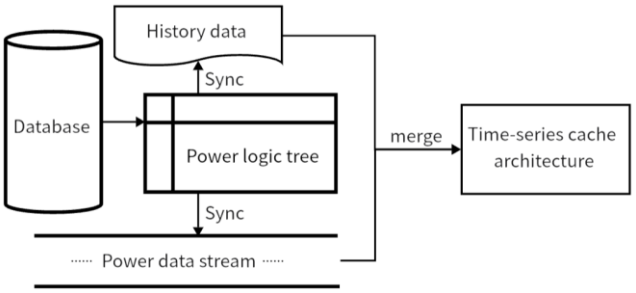

Figure1: EDNA based time series cache architecture diagram

In this paper, we develop a time series caching framework based on EDNA, Figure 1 shows the overall architecture of the caching framework. The framework mainly consists of four parts: constructing the power logic tree, acquiring data, constructing the time-series cache structure and data visualization, constructing web

services with the Flask, in which the time-series cache structure maintains all the power data as the core of the framework, at the same time, the framework creates multiple threads to continuously update the data in the time-series cache structure.

Compared to the traditional data analysis system [10], the main idea of the framework proposed in this paper is to load the data into the memory from the IO device with slow access speed, the foundation and difficulty lies in how to read and maintain the data. In order to cope with the problem of large-scale real-time streaming data and historical data reading, on the one hand, the idea in MapReduce programming model [11] can be borrowed: the historical data and real-time streaming data are quickly merged in the Map phase, the whole of the merged data is maintained in memory as a hash table. On the other hand, the processing of real-time streaming data relies on power plant business information, which is often stored in traditional relational databases such as MySQL. When the framework is initialized, the logical tree of the power plant is constructed through relational object mapping, and all subsequent streaming data reading operations rely on the logical tree instead of the database. This reduces the number of database accesses and overall shortens the processing time of real-time streaming data.

III. CONSTRUCTION OF POWER LOGIC

Data acquisition and analysis can not be separated from specific business scenarios, it is no practical significance to simply obtain the sensor value of power equipment and detach it from the specific power plant business. Power production and transmission process involves a wide variety of equipment, regular circumstances can be roughly divided into four levels - power plant, unit, system and physical equipment structure [12], between the upper and lower levels are respectively a one-to-many relationship, through the uppermost level of the power plant can be searched for all the equipment of the power system. The algorithm for construction of power logic tree is as follows:

Input: equipment vector V_d 、 system vector V_s , unit vector V_g , unit-system relationship vector V_s^g 、system-equipment relationship vector V_d^s

Output: power logic tree $F=\{G,S,D\}$

1: Perform combined calculations on V_s^g and V_d^s , generate a logical matrix $A(m*n*k)$

2. if $\text{size}(V_d)>k\&\&k\neq 0$

3. Iterate over V_d , set the system property d for the intermediate result F_S , get the intermediate result $F_S=\{F_{D1},F_{D2},...,F_{Dk}\}$

4. if $\text{size}(V_s)>n\&\&n\neq 0$

5. Iterate over V_s , set the system property s for the intermediate result F_G , get the intermediate result $F_G=\{F_{S1},F_{S2},...,F_{Sn}\}$

6: if $\text{size}(V_g)>m\&\&m\neq 0$

7. Iterate over V_g , Set the unit attribute g for the power plant object F, get to the end result $F=\{F_{G1},F_{G2},...,F_{Gm}\}$

8.return power logic tree F

The power logic tree requires to be constructed before fetching the data after this framework is started, as it is needed for subsequent operations such as fetching the data and initializing the timing cache structure. Since the power logic tree is used throughout the operation of the framework to simplify operations, the power logic tree must be memory-resident.

IV. DATA ACQUISITION

4.1 ACQUISITION OF REAL-TIME DATA

The acquisition of real-time sensor data is the basis for real-time monitoring and real-time calculations in the power production process. Time series data is typically derived from underlying production equipment sensors which are received and processed through the EDNA database system. The DNA data system provides real-time services that can effectively receive real-time sensor data and provide external interfaces for other programs to facilitate real-time and accurate access to timing data. Once the data is received, the real-time service sends it to the history service, compresses it and saves it in a data file, discards the current data in order to receive the latest data.

Although the EDNA database provides an interface to access real-time data, in order to obtain real-time data associated with the plant's operations through EDNA, it is necessary to obtain the plant's business logic information as a parameter first. This business logic information is usually kept in the database of the application system, which creates a challenge for real-time data. To solve this problem, power business logic information is stored on the power logic tree in memory as part of a time series based caching framework. With the help of power logic tree to assist in real-time data search, can reduce the number of visits to the relational database, indirectly improve the speed of access to real-time data for power production and transportation to provide a more reliable data base. The process of obtaining power real-time data using EDNA and power logic tree is shown in Figure 2:

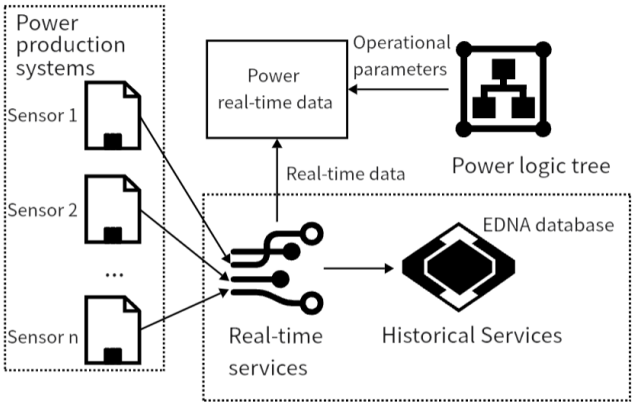

Figure2: Obtain power real-time data

4.2 ACQUISITION OF HISTORICAL DATA

To ensure that all historical data from sensors is preserved as well as to provide complete and accurate historical data to other applications, power database systems often have built-in historical data service modules for permanent data storage. The EDNA database provides an independent history service that is responsible for performing operations related to historical data. This service includes modules such as configuration center, compression algorithms, and service image files. Parameters such as storage paths, compression algorithms, sensor business information, and historical data ranges need to be provided when reading historical sensor data. Our framework utilizes the power logic tree to accelerate the process of acquiring sensor business logic and to quickly complete the process of reading historical data. The process of reading historical data using EDNA and Power Logic Tree is shown in Figure 3:

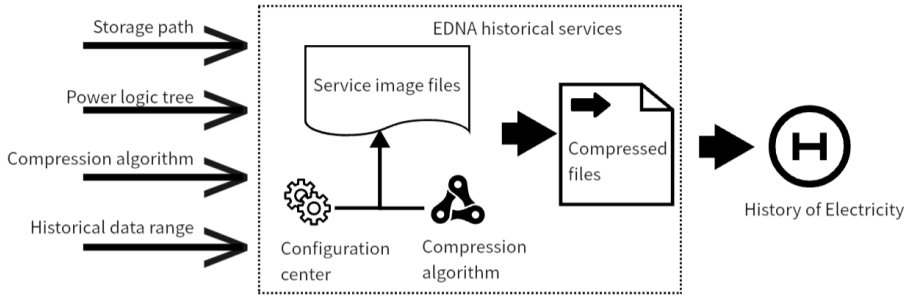

Figure3: Obtain historical power data

Power plants of different sizes have a large number of underlying sensor devices, and the number of sensors in even a small-scale power plant may reach thousands. It means that the historical data accumulated over the years need to occupy a lot of hard disk space, which brings huge data storage costs. To meet this challenge, power plants need to minimize the storage space of the data while ensuring that they do not lose any of the original data, which requires the use of lossless compression to store the data. In lossless compression algorithms, there are various techniques available such as Deflate algorithm, Hoffman coding, arithmetic coding etc. EDNA uses a Hoffman compression algorithm which uses a write-optimized SSTable structure [13] file read and write strategy. It means that some optimization operations are performed when writing data, but some additional overhead is incurred when reading data. For example, BigTable uses append writes when writing in-memory cached data to disk, but requires a merge operation when reading. For historical data stored locally by EDNA, the framework wants to perform read optimization to speed up the process of power data extraction. The read overhead estimation formulas are given below for merged reads and random reads respectively:

If the seek time for the combined read method is a constant T_{s1}, the seek time for the random read method is a constant T_{s2}, the average data read and write overhead functions are T_r and T_w, the data merge overhead function is T_m.

Merge Read Data Existing data d and new data Δ include a total of two seeks, two reads, and data merge overhead:

$$T_m = 2T_{S1} + T_r(d) + T_r(\Delta) + T_m(d,\Delta) \tag{1}$$

Random Read Data Includes seek time and read data d overhead:

$$T_r = T_{s2} + T_r(d) \tag{2}$$

Based on the above analysis of the data reading process, when reading electric power historical data compressed using the Hoffman compression algorithm, in order to select a faster speed to read the historical file in different environments. Comparing $2T_{S1} + T_m(d,\Delta)$ and $2T_{S1} + T_m(d,\Delta)$, If $2T_{S1} + T_m(d,\Delta) \le 2T_{S1} + T_m(d,\Delta)$, random read is selected, otherwise merged read is selected.

V. SEQUENTIAL CACHE STRUCTURE

The historical data maintained in our framework typically involves sensor data over a specific time period, which is relatively stable and generally used for data characterization over a specific time period. In contrast, real-time power data reflects the state of the sensor at the most recent moment. However, due to the limited memory capacity of the server, it is not possible to satisfy all the data storage requirements. Therefore for real-time data, the framework uses the sliding window model [14] to maintain a fixed-length list as shown in Fig. 4: at any point in time n, the data available for query in memory is $\{a_{\max(0,n-w+1)},...,a_n\}$, where w represents the size of the sliding window and a_x represents the at time power data. When the list capacity exceeds the limit value, a first-in-first-out replacement strategy is used to discard the earliest data into the cache and empty the storage space to save the latest power data , in order to maintain and update the data efficiently. This optimization measure can effectively balance the contradiction between the continuous growth of data and the limited storage capacity to ensure the feasibility and stability of the system.

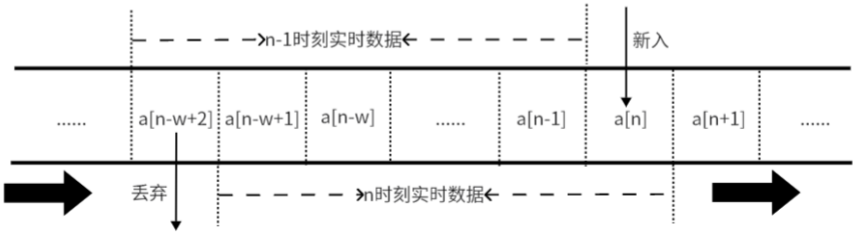

Figure4: Sensor real-time data sliding window model

In order to ensure efficient access to power plant data and time series cache retrieval, this framework utilizes a composite data structure [15] to maintain data from all sensors. Specifically, we use a composite data structure of array + linked list + tree with the sensor's unique identifier ID as the key. In this structure, the real-time and historical data of each sensor are stored in the corresponding values. The structure of the time series based power data cache is shown in Figure 5: arrange all the sensor ID's in an array according to the hash value, and connect the sensors with the same hash value to form a chain table. If the length of a linked table exceeds the set limit, the system will automatically reorganize the linked table to form a binary sorted tree structure to ensure efficient data storage and retrieval. The design of this composite data structure enables efficient management and retrieval of large-scale sensor data, while ensuring efficient and accurate data manipulation. With this data structure, we are able to better meet the needs of power plant data processing, thereby improving the efficiency and accuracy of data access.

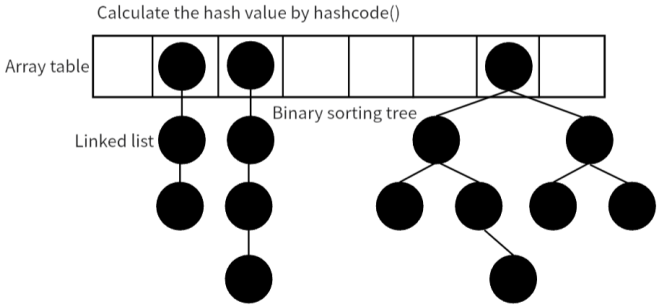

Figure5: Sequential cache structure

VI. DATA VISULAIZATION

Electricity data maintained in computer memory is obscure and difficult to understand for power plant staff, while data visualization techniques [16] can make the data move and show it in a more human-like way such as charts or diagrams. Data visualization in the power production process is important for staff to quickly understand the operation of equipment and the health of the power generation system.

In Section 4 we have built the time series structure which efficiently holds power data in memory. In order to further visualize the in-memory data and make it easier for the power plant staff to view it, we used Flask to build a server-side application and Vue to build a client-side application, with the client and server side transmitting data through the power plant's internal network. In order to ensure the real-time and efficient client-side display of data, the server-side cache automatically obtains real-time data at regular intervals, while the client utilizes Ajax technology to launch network requests to the server side at regular intervals. The flow of data interaction between the client and the server is shown in Figure 6:

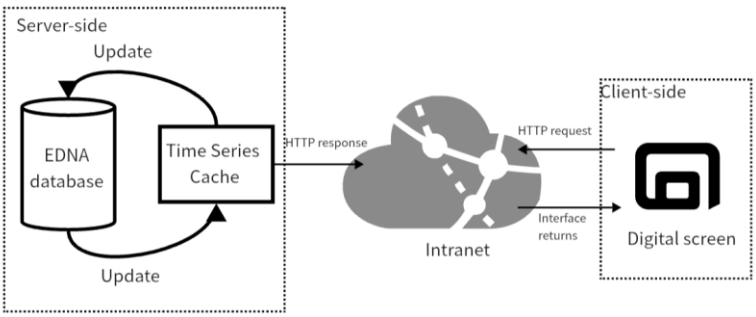

Figure6: Flow chart of client-server data interaction

VII.EXPERIMENTS

7.1 EXPERIMENTS ENVIRONMENT

The acquisition of real-time sensor data is the basis for real-time monitoring and real-time calculations in the power production process. Time series data is typically derived from underlying production equipment sensors which are received and processed through the EDNA database system. The DNA data system provides real-time services that can effectively receive real-time sensor data and provide external interfaces for other programs to facilitate real-time and accurate access to timing data. Once the data is received, the real-time service sends it to the history service, compresses it and saves it in a data file, discards the current data in order to receive the latest data.

The data used in this experiment comes from Jiangsu Guoxin Jingjiang Power Generation Co. Ltd, where the historical dataset covers the power production data of the past ten years (January 1, 2013 to January 1, 2023), the real-time data comes directly from the EDNA database. Table 1 shows a brief summary of the historical dataset, where the historical data contains only the previous year's data. The server side was built based on the Flask framework using PyCharm 2020, the client side was built based on the Vue framework using Visual Studio Code 2020, and the basic read/write modules were built in combination with Python and C++. The framework runs on a hardware environment of Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz 2.29 GHz, a RAM Lenovo server with 64GB of memory, and an operating system of 64-bit Windows 10.

Table 1 Jingjiang power plant data set description.

Year	Sensors (units)	Historical data (billions)	Data flow rate (MB / S)
2013	3854	-	1
2014	4738	153.16	1
2015	5182	203.27	1
2016	5563	221.69	1
2017	5764	231.74	1
2018	6319	246.38	1
2019	6882	267.52	1
2020	7502	289.81	1
2021	7502	432.93	2
2022	7545	436.47	2
2023	7693	456.78	2

7.2 HIT RATE ANALYSIS

Firstly, the relationship between sliding window size, real-time data flow rate and real-time data memory hit rate is explored. By continuously increasing the sliding window size in order to explore the relationship between the access performance and memory footprint of the framework under different window sizes and different data flow rates, the aim is to find the optimal window capacity value under different data flow rates. The experimental results are shown in Fig. 7:

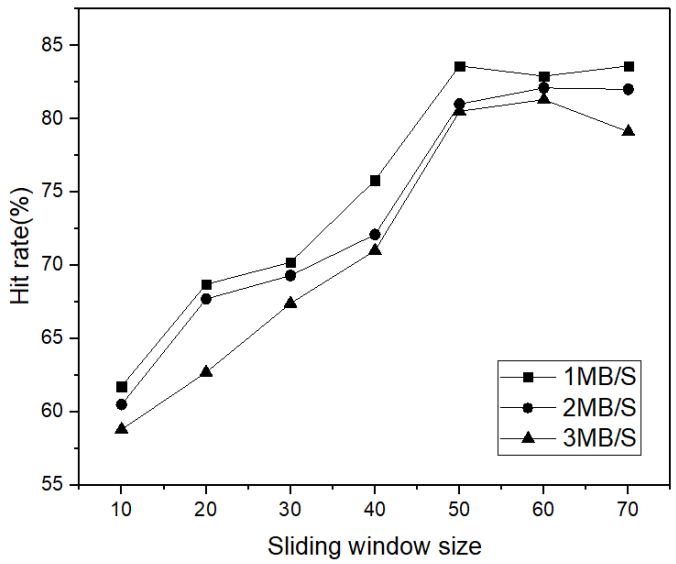

Figure7: Real-time data hit rate analysis

It can be found that as the sliding window size increases, the time-series power data queue keeps increasing, the real-time data stored in memory keeps increasing, at the same time the hit rate of the real-time data in memory also increases gradually. However, the hit rate is essentially maximized when the window size reaches 50. The experimental results show that while continuing to increase the window size can further improve the hit rate of real-time data in memory, the effect is not significant. In addition, any server has a limited amount of memory, a too-large sliding window may increase the retrieval overhead and additional internal and external replacement operations, even affecting the memory usage of other modules, thus decreasing the access performance of the whole framework.

7.3 ACCESS PERFORMANCE ANALYSIS

In order to further validate the overall access performance of power data in the caching framework, the experiment fixes the historical data size to 10 GB, adopts the Hoffman compression algorithm, the real-time streaming data flow rate is fixed to 100 KB/S, and each piece of data is about 16 B. 6250 pieces of data are processed per second. Each test is conducted 10 times, each lasting 20 minutes, and the average value is calculated as the experimental result.

As a result of the optimization of the caching framework, Table 2 demonstrates the effect of the improvement in real-time and historical read performance for power data. A sliding window model is used to maintain a list of real-time data, keeping the most recent and up-to-date data in memory, thus avoiding the read process that involves IO operations and improving the read speed. When the amount of real-time data exceeds the window maximum and is discarded, it can be accessed using EDNA's history module, which has resulted in a 54.61% improvement in real-time data read/write performance through the space-for-time method. In addition, this paper constructs a power logic tree to simplify the operation of obtaining sensor business data during historical data reading, which improves the performance of historical data access up to 12.38%. Combining the above two optimization methods, the combined access performance (number of reads and writes per unit time) of this framework is improved by nearly 17% compared to the EDNA system.

Table 2 Power data reading performance analysis.

Test metric	Test method	Number of reads and writes (TPS)	Enhancement effect
Real-time read performance	EDNA EDNA improved	6712.3 10381.0	1
Historical read/write performance	EDNA EDNA improved	23251.3 26132.1	1
Overall Read/Write Performance	EDNA EDNA improved	28731.6 33616.4	1

VIII. CONCLUSION

In this paper, we propose an EDNA-based time-series power caching framework aimed at quickly obtaining real-time streaming data and historical data for visualization to reflect the production of power plants in real time. The foundation and focus of the framework lies in the construction of the power logic tree, through which the power data acquisition process is simplified, the acquired data is efficiently integrated using the sliding window model and hash tables, and the core temporal caching structure is established to provide the underlying support for a variety of advanced data computation tasks. The effectiveness of the proposed framework is demonstrated through extensive experiments on the Jingjiang Power Plant dataset, where the combined access performance of the framework is improved by nearly 17% compared to EDNA. In the future, we will continue to optimize the caching framework and delve into data feature-driven adaptive caching structures.

REFERENCES

[1]. Liu H, Guo G, Chen Z. Study on WESP multi-pollutant emission reduction and energy efficiency test of ultra-low emission unit[J]. Power Generation Technology,2023,44(01):94-99.
[2]. Luo J. Innovative management mode leads power enterprises to improve quality and efficiency [J]. China Businessman,2023 (10):144-145.
[3]. Zhang S, Xiao Y, Li Y, et al. Collaborative operation of electricity-carbon-green market of new-type power system based on blockchain technology[J]. Electric Power Construction,2023,44(11):1-12.
[4]. Li Z. A perfect solution for intelligent power plant-a case study on xinguang power generation co., ltd.[J]. Industrial Technology Innovation,2017,4(3):4.
[5]. Zhong Z, Zhang G, Yin L, et al. Description and Analysis of Data Security Based on Differential Privacy in Enterprise Power Systems[J]. Mathematics, 2023, 11(23): 4829.
[6]. Liu S. New development of power plant informatization-research on intelligent power plant construction [J]. Science and Technology,2016,26(11):103.
[7]. Pan D, Liu H, Li Y. A wind speed forecasting optimization model for wind farms based on time series analysis and Kalman filter algorithm[J]. Power System Technology,2008,(07):82-86.
[8]. Yang D, Yu J, He Z, et al. Applying self-powered sensor and support vector machine in load energy consumption modeling and prediction of relational database[J]. Scientific Reports, 2023, 13(1): 19097.

[9]. Liu X. Analysis of data engineering and management control system in power plants[J]. Application of IC,2023,40(03):330-332.
[10]. Yamate S, Otomo J. Design of cost-effective and highly efficient systems for protonic ceramic fuel cells based on techno-economic analysis[J]. Energy Conversion and Management, 2024, 301: 118016.
[11]. Lämmel R. Google's MapReduce programming model—Revisited[J]. Science of computer programming, 2008, 70(1): 1-30.
[12]. Seven S, Yao G, Soran A, et al. Peer-to-peer energy trading in virtual power plant based on blockchain smart contracts[J]. Ieee Access, 2020, 8: 175713-175726.
[13]. Chandakanna V R. REHDFS: A random read/write enhanced HDFS[J]. Journal of Network and Computer Applications, 2018, 103: 85-100.
[14]. Liu Z, Wang J, Hui Z, et al. Research on positioning and mapping algorithm of sliding window optimization for substation monitoring robot[J]. Energy Reports, 2023, 9: 898-908.
[15]. Yan T, Qu Z Z, Hui D, et al. The Business Optimization Analysis of the Virtual Power Plant Based on the Large-Scale BESS System[J]. Advanced Materials Research, 2015, 1070: 1524-1533.
[16]. Ajibade S S, Adediran A. An overview of big data visualization techniques in data mining[J]. International Journal of Computer Science and Information Technology Research, 2016, 4(3): 105-113.