International Journal of Engineering & Technology, 7 (3.33) (2018) 243-247

SPC

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper



Real-time Big Data Processing System to Improve Semiconductor Production Efficiency in Smart Factory

Hyeopgeon Lee[1]*, Young-Woon Kim[2], Ki-Young Kim[3]

[1,2]Department of Data Analysis, Seoul Gangseo Campus of Korea Polytechnic
[3]Department of Computer Software, Seoil University
**Corresponding author E-mail: hglee67@kopo.ac.kr*

Abstract

Semiconductor production efficiency is closely related to the defect rate in the production process. The temperature and humidity control in the production line are very important because these affect the defect rate. So many smart factory of semiconductor production uses sensor. It is installed in the semiconductor process, which send huge amounts of data per second to a central server to carry out temperature and humidity control in each production line. However, big data processing systems that analyze and process large-scale data are subject to frequent delays in processing, and transmitted data are lost owing to bottlenecks and insufficient memory caused by traffic concentrated in the central server. In this paper, we propose a real-time big data processing system to improve semiconductor production efficiency. The proposed system consists of a production line collection system, task processing system and data storage system, and improves the productivity of the semiconductor manufacturing process by reducing data processing delays as well as data loss and discarded data.

Keywords: *Real-time Big data, Big data Processing System, Smart Factory, Spark, Memory DB*

1. Introduction

New services based on artificial intelligence platforms, which are the core of the fourth industrial revolution currently being implemented industry-wide, are distributing a large amount of data and promoting the consumption of media content. As a result, the demand for various memories and other semiconductors in the global semiconductor industry has surged by at least 25% over the past three years. Furthermore, global semiconductor companies are investing in research and development as well as facility expansion to improve semiconductor production [1].

Semiconductor production efficiency[1, 2] is closely related to the defect rate in the production process. Hence, temperature and humidity control in the production line are very important because these affect the defect rate. Hundreds of sensors are installed in the semiconductor process, which send huge amounts of data per second to a central server to carry out temperature and humidity control in each production line. However, big data processing systems that analyze and process large-scale data are subject to frequent delays in processing, and transmitted data are lost owing to bottlenecks and insufficient memory caused by traffic concentrated in the central server. In this respect, big data processing systems are not suitable for real-time data processing and analysis. Big data processing techniques can solve these problems and reduce delays in data processing, unlike conventional methods. However, because of the frequent occurrence of missing data due to transmission losses between clusters, such techniques are difficult to apply in semiconductor manufacturing, where the reliability of data transmission as well as real-time data processing are important.

This study proposes a real-time big data processing system to improve semiconductor production efficiency. The proposed system consists of a production line collection system (MCS), task processing system (TPS), and data storage system (DSS), and improves the productivity of the semiconductor manufacturing process by reducing data processing delays as well as data loss and discarded data.

The structure of the paper is organized as follows: Section 2 describes Flume-based big data processing system, Spark-based big data processing system and Clustering-based big data processing system; Section 3 proposes the big data processing system; Sections 4 analyzes the data throughput and data loss rate; and Section 5 concludes this paper.

2. Previous studies

2.1. Flume-based big data processing system

Flume[3] is a big data collection technology that collects the logs which are loaded onto servers that provide various services, and onto a log collection server. Flume is based on stream-oriented data flow, collecting logs from all designated servers and loading them into the central storage, such as the Hadoop Distributed File System. Flume is suitable for building a Hadoop-based big data processing system. Fig. 1 illustrates a Flume-based big data processing system.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

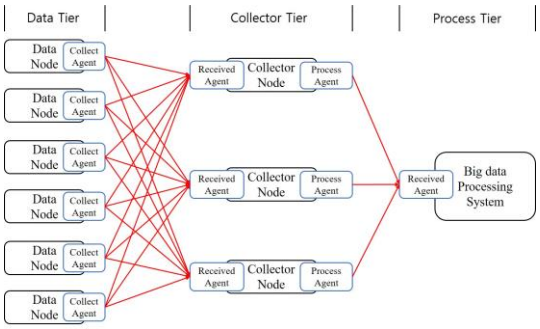

Fig. 1: Flume-based big data processing system

A Flume-based big data processing system has a hierarchical structure that reduces data processing delays by decreasing bottlenecks compared to conventional big data processing techniques. However, it is difficult to apply the Flume-based big data processing system in a large-scale environment such as the semiconductor manufacturing process. Because a Flume-based big data processing system centrally manages the data received from nodes and has more data collected than processed, the discard data phenomenon occurs frequently.

2.2. Spark-based big data processing system

Although Spark [4, 5] is a technology developed for real-time big data analysis, it is frequently used because Spark processes data using the memory of the nodes. Fig. 2 illustrates a Spark-based big data processing system.

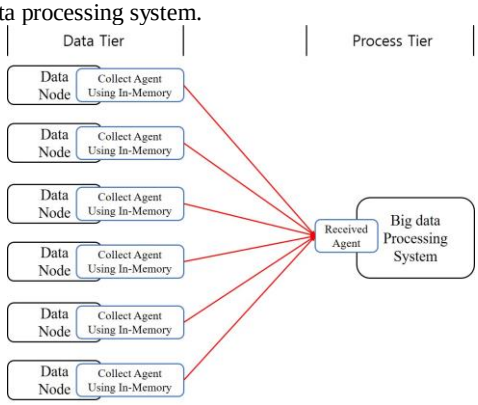

Fig. 2: Spark-based big data processing system

Because Spark-based big data processing technology uses memory to process large-scale data, the data processing speed is faster than the Flume-based big data processing system. However, it requires a large amount of memory, leading to high infrastructure management costs. Furthermore, because a Spark-based big data processing system installs a Spark daemon in each node that collects data, the system requires large memory size and high-performance micro control unit (MCU) for the data collecting nodes. However, since the nodes used in semiconductor manufacturing simply transmit the sensing results, the Spark-based big data processing system is difficult to apply.

2.3. Clustering-based big data processing system

A clustering-based big data processing system[6-8] is a big data processing system that applies the clustering technique to conventional big data processing systems, such as the Flume-based and Spark-based big data processing systems. The clustering technique is used to group multiple big data processing techniques, improve the speed of big data processing, and increase scalability by connecting them in parallel. The clustering technique can be easily applied to various big data processing systems and is suitable for big data processing in large-scale environments. Fig. 3 illustrates clustering-based big data processing system.

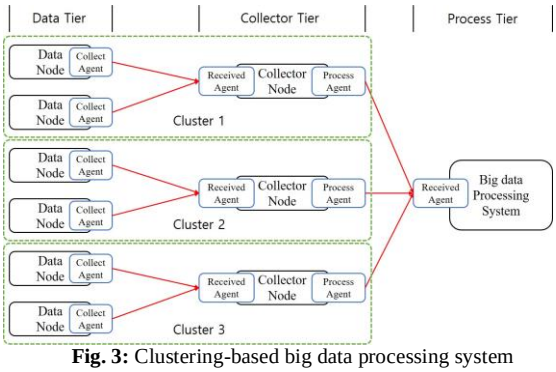

Fig. 3: Clustering-based big data processing system

A clustering-based big data processing system reduces data processing delays due to bottlenecks and insufficient memory by applying the clustering method. However, because the missing data phenomenon due to data transmission losses between clusters frequently occurs, it is difficult to apply the system in semiconductor manufacturing, where the reliability of data transmission as well as real-time data processing are important.

3. Proposed system

Since the productivity of the semiconductor manufacturing process is sensitive to temperature and humidity, the temperature and humidity control in the production line are important. Thus, sensors that control temperature and humidity are installed and managed in order to improve productivity. Sensors for temperature and humidity control in a single production line transmit thousands of data per second. Thus, this study proposes a real-time big data processing system to improve semiconductor production efficiency by real-time data processing. Fig. 4 illustrates proposed big data processing system.

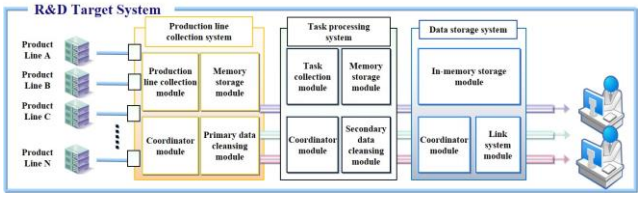

Fig. 4: An illustration of proposed big data processing system

As shown in Fig. 4, the proposed system consists of the MCS, TPS, and DSS. The main role of the MCS is to collect data from the production line in the manufacturing process, cleanse the data, and transfer the data to the TPS in accordance with the work process. The main role of the TPS is to validate whether the data received from the production line collection system are transferred according to the work process, cleanse the data according to the process, and transmit information to the data storage system. The main role of the DSS is to store the transmitted data in a memory-based database (DB), create links with other systems, and transmit the information.

3.1. Data structure

Fig. 5 illustrates data structure commonly used in proposed big data processing system.

```
1  //Memory Database Object
2  struct memory Object
3    int nodeSeq;
4    int lineSeq;
5    int taskType;
6    struct validHumidity;
7    struct validTemperature;
8  }
9
10 //Humidity
11 struct validHumidity{
12   float min;
13   float max;
14 }
15
16 //Temperature
17 struct validTemperature{
18   float min;
19   float max;
20 }
21
22 //ValidDataDictionary
23 struct dataDictionary{
24   int taskType;
25   int lineSeq;
26   struct validHumidity;
27   struct validTemperature;
28 }
```

Fig. 5: Data structure commonly used in proposed big data processing system

The data structure shown in Figure. 5 is commonly used in the proposed big data processing system. The *memoryObj* was used as a data structure to store the structure of the data received from the production line, as well as the data cleansed through the primary and secondary data cleansing algorithms. The *nodeSeq* was used to identify a node with a unique, nonredundant value of the data collecting node. The *lineSeq* is a nonredundant value for easy access to the production line of the data collecting node, which enables faster search than the values in node identification. The *maskType* was used as a delimiter for the production line process. MemoryObj uses *validHumidity* and *validTemperature* as substructures to manage the valid humidity and temperature data.

3.2. Production line collection system

The MCS for each production line collects data and performs the primary data cleansing. The MCS consists of a production line collection module, memory storage module, coordinator module, and primary data cleansing module. The production line collection module is responsible for receiving data from the production line. The memory storage module stores the data collected from the production line collection module in a memory. The coordinator module manages the status of the MCS. The primary data cleansing module determines the validity (normal temperature and humidity) of the data transmitted through the primary data cleansing algorithm to cleanse the data. Among the data collected from the production line, the primary data cleansing algorithm discards the data included in the validity range of the predefined data, and transmits only invalid data to the TPS suitable for the work process. Fig. 6 illustrates the primary data cleansing algorithm using the data structure described above.

```
1  getReceivedNodeData(pNodeObj){
2  while(true){
3
4    //Humidity
5    if(getDataVaildAlgorithm(pNodeObj.validHumidity.min) &&
6       getDataVaildAlgorithm(pNodeObj.validHumidity.max))
7       return sendNodeInfo(pNodeObj);
8    else
9       return false;
10   }
11
12   // Temperature
13   if(getDataVaildAlgorithm(pNodeObj.validTemperature.min) &&
14        getDataVaildAlgorithm(pNodeObj. validTemperature.max))
15      return sendNodeInfo(pNodeObj);
16   else
17      return false;
18   }
19 }
```

Fig. 6: The primary data cleansing algorithm

The primary data cleansing algorithm compares the temperature and humidity data transmitted from the production line collection module via *getReceivedNodeData*, and the predefined validity range through *getDataVaildAlgorithm*, and stores only the valid data through the memory storage module. All invalid data are discarded to minimize the data stored in the memory storage module. Furthermore, the primary data cleansing algorithm must minimize the memory and CPU usage used in the computation to reduce data processing delays as well as data losses due to bottlenecks in collecting and cleansing large-scale data transmitted from the production line. Therefore, the data validation by the primary data cleansing algorithm cleanses the temperature and humidity data by predefining the temperature and humidity valid for various production lines.

3.3. Task processing system

The TPS collects the data transmitted from the MCS and performs secondary processing. The TPS consists of a work process collection module, memory storage module, coordinator module, and secondary data cleansing module. The work process collection module is responsible for collecting the primarily processed data transmitted from the MCS. The memory storage module stores the data collected from the work process collection module into a memory. The coordinator module manages the status of the TPS. The secondary data cleansing module determines the validity (normal temperature, humidity) of the data transmitted through the secondary data cleansing algorithm to cleanse the data. The secondary data cleansing algorithm checks whether the cleansed data transmitted from the primary data cleansing module are included in the predefined validity range of the data for each work process in order to transmit only the invalid data to the DSS, and to discard the valid data. Fig. 7 illustrates the secondary data cleansing algorithm.

```
1  TaskProcessSystem(){
2    while(true){
3     memoryObj = getNotValidNodeData(pNodeObj);
4     if (getDataValidValue('h', memoryObj) &&
5          getDataValidValue('t', memoryObj)){
6       MemoryStoreModule(memoryObj)
7     }
8     CordinatorModule();
9    }
10  }
11
12  //Store Collecting Data
13  MemoryStoreModule(pData){
14    memoryObj = pData;
15  }
16
17  //Data Valid Range / pType values(h:humidity / t: temperature)
18  getDataValidValue(pType, pNodeObj){
19    //if humidity;
20    if (pType='h'){
21      validHumidity=getValidDictionary(pNodeObj.taskType);
22      if (validHumidity.min > pNodeObj.min &&
23          validHumidity.max < pNodeObj.max)
24        return true;
25      else
26        return false;
27
28    //if temperature;
29    else if (pType='t'){
30      validTemperature=getValidDictionary(pNodeObj.taskType);
31      if (validTemperature.min > pNodeObj.min &&
32          validTemperature.max < pNodeObj.max)
33        return true;
34      else
35        return false;
36    }
37  }
```

Fig. 7: the secondary data cleansing algorithm

The secondary data cleansing algorithm receives the abnormal temperature and humidity data transmitted from the MCS through *getNotValidNodeData*, and further validates the data by accessing the valid value range in the data validity dictionary through *getDataVaildAlgorithm*. The valid value range in the data validity dictionary is retrieved from the *DataDictionary* structure described above through *getDataVaildDictionary*. The secondary data cleansing algorithm transmits only invalid data to the DSS and discards all valid data. As a result, the DSS stores only the minimized abnormal data that have undergone two data cleansing processes.

3.4. Data storage system

The DSS stores abnormal temperature and humidity data transmitted from the TPS into a memory DB, linking it to other systems and transmitting it to users. The DSS consists of an in-memory storage module, link system module, and coordinator module. The in-memory storage module stores abnormal temperature and humidity data into the memory DB. The link system module is responsible for transmitting information to the system and users connected to the memory DB. The coordinator module manages the status of the DSS.

4. Performance evaluation

The performance of the proposed system (Proposed BPS) was analyzed using JMeter to measure the data processing and data loss rates of the proposed system, Flume-based big data processing system (FBPS), and clustering-based big data processing system (CBPS), respectively. The assumptions for performance evaluation are described as follows.

Assumption 1: The number of nodes communicating with one evaluation target system is defined as 1,000.

Assumption 2: Each node communicates with the evaluation target system for 100 s, once per second.

4.1. Data Throughput

The data throughput of the proposed BPS was obtained by measuring the time between the data transmission of the nodes installed in the production line and the storage of abnormal data by the secondary data cleansing algorithm of the TPS through MCS. The data throughput of the FBPS was obtained by measuring the time between the data transmission of nodes installed in the production line and the storage of abnormal data in the storage layer through the agent layer and collector layer. The CBPS was clustered by dividing the FBPS by production line. The data throughput of the CBPS was obtained by measuring the time between the data transmission of nodes and the storage of abnormal data in the clustered storage layer. Fig. 8 illustrates the r Results from measurement of data throughput for proposed BPS, FBPS, and CBPS.

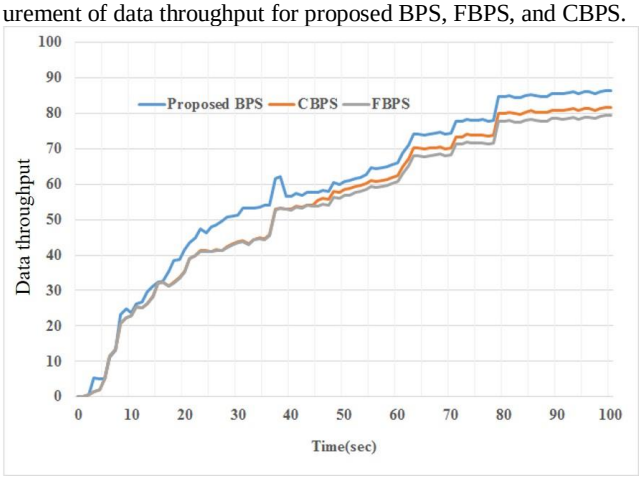

Fig. 8: Results from measurement of data throughput for proposed BPS, FBPS, and CBPS

According to the results shown in Figure 8, the data processing rates of the proposed FBPS and CBPS were not significantly different from those of the proposed big data processing system at the 0–16 s interval. However, after 17 s, the rate of the proposed BPS started to show a sharp difference from that of the compared subjects, and also showed approximately 15.2% difference from the FBPS at 20 s. The difference in data processing rates of CBPS and the proposed BPS gradually decreased from approximately 50 s. The results can be attributed to the characteristics of the clustering technique that the data processing in each cluster starts at the initial stage, before the data are transferred and stored in the cluster responsible for storing the data.

4.2. Data loss rate

The data loss rate of the Proposed BPS was measured by summing up the data loss rate at which the MCS failed to receive the data transmitted from the node installed in the production line, and the data loss rate at which the TPS failed to receive the data transmitted from the MCS. The data loss rate of the FBPS was measured by summing up the data loss rate at which the collector layer failed to receive the data transmitted from the agent layer, and the data loss rate at which the storage layer failed to receive the data transmitted from the collector layer. The data loss rate for the clustering-based big data processing system was measured by the data loss rate at which the storage layer failed to receive the data transmitted from each cluster. Fig. 9 illustrates the results from measurement of data loss rate for proposed BPS, FBPS, and CBPS.

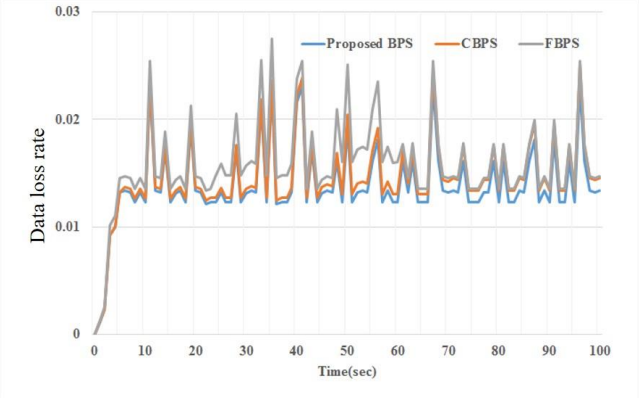

Fig. 9: Results from measurement of data loss rate for proposed BPS, FBPS, and CBPS

According to the results shown in Fig. 9, the data loss rate of the proposed BPS had little change, and its data loss rate decreased by approximately 11.35% compared to that of FBPS. In comparison, the data loss rate of CBPS increased over time and its data loss was 21.18% greater than that of the proposed BPS. Furthermore, at the 0–5 s interval, the data loss rate of the proposed BPS did not show much difference compared to FBPS and CBPS. However, after 23 s, the data loss ratio of the proposed BPS and CBPS began to diverge rapidly, especially at approximately 85.8 s. The results were attributed to the characteristics of the clustering technique, where data communication in each cluster occurs frequently at the initial stage, and then the transmission of the collected data concentrates on the cluster responsible for storing data.

5. Conclusion

In this paper, we propose a real-time big data processing system to improve semiconductor production efficiency. The proposed system consists of a production line collection system, task processing system, and data storage system.
In order to evaluating the performance of proposed system, we have compared the data throughput and data loss rate. Through the performance evaluation, we showed that our proposed system has more improved the data throughput and data loss rate than the FBPS and CBPS.

References

[1] Hyeop-Geon Lee, Young-Woon Kim, Ki-Young Kim and Jong-Seok Choi, ”Design of GlusterFs Based Big Data Distributed Processing System in Smart Factory”, *Journal of Korea Institute of information, Electronics, and Communication Technology*, Vol.11, No.1, (2018), pp:70-75

[2] Hyeopgeon Lee, Young-Woon Kim and Ki-Young Kim, ”Implementation of an Efficient Big Data Collection Platform for Smart Manufacturing”, *Journal of Korea Institute of information, Electronics, and Communication Technology*, Vol.12, No.22, (2017), pp:6304-6307
http://dx.doi.org/10.3923/jeasci.2017.6304.6307

[3] In-Hak Joo, ”Spatial Big Data Query Processing System Supporting SQL-based Query Language in Hadoop”, *Journal of Engineering and Applied Sciences*, Vol.10, No.1, (2017), pp:1-8

[4] Young-Woon Kim and Hyeopgeon Lee, ” Implementation of Big Data Analysis System to Prevent Illegal Sales in the Cable TV Industry”, *Journal of Korea Institute of information, Electronics, and Communication Technology*, Vol.12, No.23, (2017), pp:6542-6545
http://dx.doi.org/10.3923/jeasci.2017.6542.6545

[5] Jianguo Chen, Kenli Li, Zhuo Tang, Kashif Bilal, Shui Yu, Chuliang Weng and Keqin Li, ”A Parallel Random Forest Algorithm for Big Data in a Spark Cloud Computing Environment”, *IEEE Transactions on Parallel and Distributed Systems EEE Transactions on Smart Grid*, Vol.28, No.4, (2016), pp:919-933
http://dx.doi.org/10.1109/TPDS.2016.2603511

[6] Neha Bharill, Aruna Tiwari and Aayushi Malviya, ” Fuzzy Based Scalable Clustering Algorithms for Handling Big Data Using Apache Spark”, *IEEE Transactions on Big Data*, Vol.2, No.4, (2016), pp:339-352
http://dx.doi.org/10.1109/TBDATA.2016.2622288

[7] Xing He, Lei Chu, Robert Caiming Qiu, Qian Ai and Zenan Ling, ”A Novel Data-Driven Situation Awareness Approach for Future Grids—Using Large Random Matrices for Big Data Modeling”, *IEEE Access*, Vol.6, No.1, (2018), pp:13855-13865
http://dx.doi.org/10.1109/ACCESS.2018.2805815

[8] Ling Hu, Qiang Ni and Feng Yuan, ”Big data oriented novel background subtraction algorithm for urban surveillance systems”, *Big Data Mining and Analytics*, Vol.1, No.2, (2018), pp:137-145
http://dx.doi.org/10.26599/BDMA.2018.9020013