International Journal of Applied Science and Engineering
2020. 17, 2: 191-206

Improve Data Storage Efficiency with Service-oriented Architecture underlying the Internet of Things

Shu-Ching Wang[a], Wei-Ling Lin[a], Mao-Lun Chiang[b*] and Shin-Jie Wang[a]

[a]*Department of Information Management, Chaoyang University of Technology, Taiwan, R.O.C.*
[b]*Department of Information and Communication Engineering, Chaoyang University of Technology, Taiwan, R.O.C.*

Abstract: Sensor hardware technology and network bandwidth have developed rapidly. Through the combination of the Internet and different types of sensors, the Internet of Things (IoT) can facilitate the development of applications, and it has become a hot issue in many countries. The IoT can facilitate the development of applications such as smart cities, smart healthcare and smart homes. In the IoT, people can exchange information to anything, such as people and objects, objects and machines, and even between machines. However, in order to develop related application services of the IoT, hundreds of millions of objects will exchange data on the Internet, and the amount of data in the IoT is very large. In order to provide the different real-time service requirements, the Service-oriented Architecture underlying IoT (SoAIoT) and the data storage operating mechanism of SoAIoT are constructed in the study. The operating mechanism of SoAIoT will reduce the load of network, improve the quality of data processing scheduling, and provide users with different services to meet service needs. Finally, in order to evaluate whether the proposed operating mechanism of SoAIoT can achieve the goal of our research, experimental examples are given through the application of the earth and rock flow monitoring.

Keywords: Internet of Things; Service-oriented Architecture; scheduling; data storage.

1. Introduction

Due to the increasing processing power of sensor, various applications combined with sensors have become more and more deeply in people's lives, such as smart monitoring, smart health care and smart home [1]. In order to improve the quality of life, the Internet of Things (IoT) is being explored and developed rapidly [2]. The IoT creates smart objects by embedding sensors into objects of daily life, and develops into an application platform in conjunction with the Internet. Therefore, IoT combines the objects in the real world into the virtual world, so that the world reaches the ubiquitous state.

Because the IoT combines a large amount of sensing data, a huge amount of data circulation and service requirements from a large number of users will increase the processing load on the Internet [3]. Therefore, without a specific mechanism to assist in the processing of data, it may cause serious delays in data circulation and affect efficiency. This makes users get poor service efficiency when enjoying the services provided by the IoT.

Service-oriented Architecture (SoA) is an example for organizing and leveraging distributed functions that may be under the control of different ownership domains [4-6]. Often entities (people and organizations) create capabilities to solve or support solutions to problems they encounter in their business processes. It is natural to think that the capabilities provided by others can meet one's needs; or in the world of distributed computing, a computer agent belonging to

*Corresponding author; e-mail: mlchiang@cyut.edu.tw
Received 16 March 2020
doi:10.6703/IJASE.202005_17(2).191

Accepted 30 March 2020

another owner can meet the requirements of a computer agent [7]. There is not necessarily a one-to-one relationship between requirements and capabilities. The perceived value of SoA is that it provides a powerful framework to match requirements and functions, and to meet these requirements by combining functions. This allows service providers and consumers to interact and provide decision points for any policies and contracts that may be in effect [8].

In the SoA, there are three roles, including: service provider, service agent, and service demander [5]. Service providers can publish related software services to service agents through the Internet, provide service interfaces to service agents, and through service agents, enable service providers to interact with service demanders and provide services. The service agent acts as an intermediary between the service provider and the service demander, and publishes the services registered by the service provider, and provides the relevant catalogue of the service to the service demander, so that the service demander can find the relevant service needed by the service agent. In addition, the service agent can collect relevant requirements from service demanders, find out the services required by service demanders, and respond. Service demanders can call software services from service providers according to their own needs, so they can send service requirements to service agents, or search for service needs they need through service agents.

Therefore, a Service-oriented Architecture underlying IoT (SoAIoT) to serve diverse applications is used in this research. This research uses a cloud computing environment and uses parallel computing features to perform a large amount of data processing, and then a Service Data Processing Mechanism (SDPM) is proposed in this study. SDPM allows sensing data to be efficiently processed by cloud nodes when storing data in a cloud environment, and provides differentiated data processing services.

The remainder of this paper is arranged as follows: section 2 illustrates the related work. Section 3 illustrates the proposed SoAIoT. The operations of SDPM in SoAIoT are given in section 4. The simulation experiments on SDPM under the SoAIoT architecture are explained in section 5. Finally, conclusions are presented in section 6.

2. Related Work

In this section, the CPU scheduling algorithm will be explained. The CPU scheduling algorithm is a process that uses a single processor for scheduling. When a single processor may need to execute multiple programs at the same time, these programs cannot be executed at the same time, so the remaining unexecuted programs will be sent Wait in the I/O temporary queue, until the processor is idle or the program is aborted, the next program will be selected from the temporary queue for execution [9].

The FIFO (First In First Out) scheduling algorithm is based on the program's arrival time as a sorting order and has a non-preemptive feature. The program that arrives earlier will be prioritized for scheduling. Therefore, a program with a shorter execution time may be affected by a program with a longer execution time, resulting in a longer waiting time for the program.

The RR (Round-Robin) scheduling algorithm is a fair scheduling algorithm that enables each program to obtain a fair fixed time service quota (Time Quantum) and execute each program in a round-robin manner until all programs are completed. However, under the mechanism of fair allocation of service quotas, the RR scheduling algorithm cannot provide differentiated service quotas, which affects the execution efficiency of the program. When the service quota is set too high, it may approach the FIFO scheduling algorithm; if it is set too low, it will cause the program exchange frequency to be too frequent and affect the scheduling operation.

The Priority scheduling algorithm is an unfair scheduling algorithm. Tasks are sorted according to priority, and programs with high priority are executed first. Programs with the same priority are based on FIFO scheduling algorithm. The advantage of the priority scheduling algorithm is that it can meet the needs of the program and flexibly change the scheduling of the program. However, when the high-priority program is executed preferentially, the low-priority program will be delayed, which may cause indefinite blocking or starvation of the low-priority program.

The SJF (Shortest Job First or Shortest Task First) scheduling algorithm is an unfair scheduling algorithm. When the program is executed by the CPU, the shortest completion time is used as the sorting order. The smaller the execution completion time of the program, the higher the priority is to be executed. Programs with the same completion time will be sorted according to the FIFO algorithm. The advantage of the SJF scheduling algorithm is that the waiting time between programs will not be too long, and a smaller average waiting time can be obtained. However, the SJF scheduling algorithm is based on the short-program priority execution, which is as unfair as the priority scheduling algorithm, which is likely to cause hunger in the program. In addition, when larger programs need to be executed preferentially, short-program priority scheduling algorithms cannot flexibly schedule programs.

In order to solve the problem that the long program execution time caused by the SJF scheduling algorithm is delayed too long, the E-SJF (Enhanced SJF) scheduling algorithm is an improvement of the SJF scheduling algorithm and was proposed by Wang et al. [10]. E-SJF regards the execution time complexity of the longest program as the threshold as the basis for preemptive scheduling. When the total time of the programs that have been executed exceeds the threshold, the preemption mechanism is used to execute the program with the longest execution time to reduce the problem of long waiting times. However, the E-SJF scheduling algorithm cannot flexibly adjust the execution order of the program according to the needs of the program. Therefore, although E-SJF can reduce the delay time of long programs, it cannot give priority to programs that need to be executed immediately.

3. SoAIoT

In this study, a four-layer framework of IoT (SoAIoT) is proposed, including the perception layer, the middleware layer, the cloud computing layer and the application layer. The structure of SoAIoT is shown in Figure 1.

The lowest layer of the SoAIoT architecture is the perception layer. The perception layer is composed of a variety of different types of sensors. These sensors can be embedded in a variety of different objects and can be distributed in different areas according to their applications. Because the capabilities of each sensor are usually limited; hence, the complex task calculations, large amounts of data storage, and high-speed transmission cannot be accomplished by sensors.

The second layer in the SoAIoT architecture is the middleware layer, which is composed of sinks located in different regions. The sink has a certain degree of computing power, storage space and transmission capacity, and can continuously receive the sensing data from the perception layer, and collect and process these sensing data. These sinks are located in different areas. Depending on the changes in the environment and the type of sensor, the types of data transmitted and processing operations may be different. In this study, sensors are grouped according to their area and the type of sensing data processed by the clustering mechanism, and subsequent data processing is performed by a specific sink to reduce the complexity of data processing. In addition, the data is transferred to the cloud computing environment through the sink for the subsequent related application services.

The third layer in the SoAIoT architecture is the cloud computing layer, which consists of a cloud computing environment. The cloud computing layer receives the sensing data from the middleware layer, and the huge data from different regions are collected and stored in the cloud computing layer. At the same time, the cloud computing layer will send various sensing data from the perception layer to the corresponding application server according to the service requirements of different applications, so that specific application services can be performed normally.

The fourth layer in the SoAIoT architecture is the application layer, which consists of diversified applications. In order to meet the requirements of application services, according to different requests made by each service demander, the application layer requests data from the cloud computing layer, and finally provides it through these application services. The service corresponding to the service demander. Therefore, service demanders do not need to know how to obtain data and the conversion process of data circulation in each domain. They only need to connect to the Internet to enjoy the services brought by the IoT.

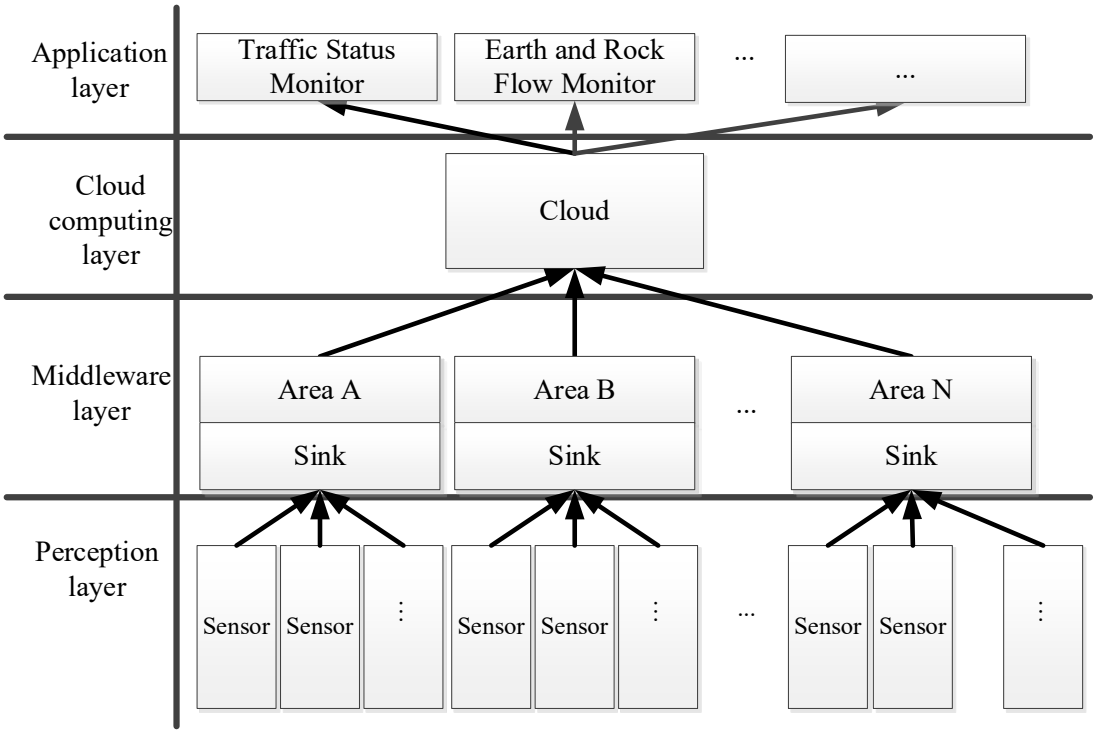

Figure 1. The structure of SoAIoT.

SoAIoT is mainly building the infrastructure of the IoT, and focuses on the circulation of IoT data and service provision. Therefore, the main operation mechanism under the SoAIoT is to enable the data of the IoT to be processed efficiently and to provide IoT services to users. In SoAIoT, the "Service Data Processing Mechanism (SDPM)" is implemented in the cloud computing layer. In the SDPM, the main operation is to process a large amount of sensing data and store the data to a cloud data storage center. The sensing data stored in the cloud data storage center can provide application services to meet the needs of IoT users. And, the SDPM is considered in this study.

4. The Operations of SDPM

The rise of the IoT has brought about an automated environment and intelligent life. In order to provide diversified applications for the IoT, the massive data obtained through the sensors in the IoT, leading to an increase in load. As the behavior pattern of tens of thousands of users in the IoT has been transformed from the original real object operation to the virtual operation in the Internet. The development of various applications has resulted in a large number of IoT data demand services, which may cause the system to be unable to effectively handle huge demands [5].

When a large number of service requirements are issued from the application side of the IoT at the same time, if there is not an effective control mechanism, service processing may be severely delayed during the process. Therefore, the scalability and parallel computing characteristics of cloud computing resources are used in this study to enable massive data to be used in a large amount in the cloud computing environment to provide diversified application services.

In order to provide diversified and high-quality services for the IoT, under the SoAIoT proposed in this study, the data collected by sensors is processed in the cloud computing layer with "Service Data Processing Mechanism (SDPM)". Through SDPM, a large amount of sensing data can be efficiently stored in a cloud data storage center. In order to achieve this goal, a scheduling algorithm is used in SDPM to perform task processing. It is considered that all tasks can be serviced by the node and different applications in the shortest time. Therefore, the minimum makespan is considered, and different task priorities are set to allow the data to provide differentiated services. In other words, the method proposed by this research can serve the massive data and a large number of service requirements in the IoT efficiently, can reduce the congestion of data processing, and can provide differentiated data processing services.

In SDPM, the processed data transmitted by the sinks in different regions of the middleware layer is received continuously. A large number of heterogeneous data received from the perception layer are processed by the middleware layer and transmitted to the cloud computing layer continuously. According to different properties, the data will be divided into "urgent data" and "general data". In order to provide differentiated services from application users, tasks will be divided into "urgent tasks" and "general tasks" in the research. The urgent data usually needs to be processed in time, while the general data can be processed sequentially. Therefore, when the data is transmitted from the sink to the cloud computing layer, the priority of task execution will be determined by agent.

In order to process a large number of heterogeneous data, when the data enters into the cloud computing layer, tasks will be assigned to the "urgent data queue" or the "general data queue". First, the data determined as urgent is assigned to the urgent data queue. And, the general data will be assigned to the corresponding application general data queue for scheduling. There are two parts in SDPM, one is the task selection from the data queue, and the other is the agent selection from the cloud nodes, which are explained separately below.

4.1 The Task Selection from the Data Queue

The data in the general data queue has less influence on the application service. In the context of the IoT, there is a large amount of general data from each sink to the cloud computing layer. Therefore, enabling a large amount of data to be efficiently processed by the nodes in cloud computing layer is a goal that general data queues must achieve.

To achieve this goal, a large number of tasks can be performed and tasks can be stored in the cloud data center at a faster rate will be considered, so tasks in the general data queue will be scheduled. Therefore, considering the minimum overall waiting time of the task, the task can be executed with the minimum delay time, so that users on the application side can quickly obtain related services.

Because the FIFO scheduling algorithm does not consider the execution time of multiple tasks; hence, tasks with short execution times are susceptible to delays caused by tasks with long execution times, so that short waiting times cannot be obtained. Using the RR scheduling algorithm, each task can be performed at a fair schedule, but it cannot meet a large number of output requirements and obtain a lower waiting time. Using the SJF scheduling algorithm, tasks with longer execution times can be executed preferentially, shorter waiting times can be obtained, and tasks can be executed in large numbers. However, the SJF scheduling algorithm is based on the shortest execution time of the task. In an unfair scheduling, when the task with the longer execution time enters and waits, it will be affected by the task with the shorter execution time. As a result, starvation can occur in tasks with long execution times. E-SJF uses the longest task execution time as a threshold as the basis for preemptive execution of tasks. Therefore, tasks with a long execution time can be preferentially processed when the threshold is exceeded using a preemption mechanism, so the problem of long waiting time for tasks with a long execution time can be reduced. However, E-SJF does not consider the different requirements of data processing, so it cannot be applied in the IoT environment.

Therefore, in order to enable the IoT to process a large amount of data, this study uses multiple application data processing queues in the general data queue to enable a large number of tasks to be processed in parallel and scheduled. This research proposes WE-SJF (Weight Enhanced SJF) and improved E-SJF scheduling algorithm. In addition to solving the problem of hunger caused by short tasks affected by short tasks, it can also provide different Weight values for different application services, so that general tasks can provide differentiated services.

In order to respond to the calculation of the WE-SJF scheduling algorithm, the Weight value corresponding to each task will be assigned, and the change in weight will be set according to whether an urgent task has occurred. If an urgent task has occurred within the specified conditions, the Weight value will increase. If the task has no urgent task within the limited conditions, the initial Weight value is set to 1. The WE-SJF scheduling algorithm is divided into 7 steps:

[Step 1]: The tasks in the queue are sorted according to their respective Weight values, so that the scheduling of tasks can be shifted according to the Weight value from high to low.

[Step 2]: The waiting task execution time in each level Weight value is sorted from small to large according to the different rotations of the Weight value.

[Step 3]: Pick the longest task execution requirement in the queue and set it as the maximum tolerance execution threshold.

[Step 4]: According to the order of [Step 2], select the task execution schedule with the smallest task execution requirement.

[Step 5]: Accumulate the output tasks to the output threshold according to the output task execution requirements.

[Step 6]: (1) When the accumulated output threshold value exceeds the maximum tolerance execution threshold value, the task with the longest execution demand in the queue area is preemptively outputted.
(2) Reselect the longest task execution requirement to set the maximum tolerance execution threshold.

[Step 7]: Repeat [Step 1] to [Step 7] until the assignment of tasks in queue is completed.

The urgent data queue stores urgent data, which may have a huge impact on practical applications. Disaster hazards such as earth and rock flow monitoring systems and fire monitoring systems may pose a threat to people's lives. In addition, the incidence of urgent tasks is low and does not occur frequently. Therefore, in this study, in the processing of urgent tasks, the FIFO scheduling algorithm was used.

In other words, because the execution order of urgent tasks cannot be determined for different application tasks, such as the priority execution order of fire and flood, this study uses the FIFO scheduling algorithm for the processing of urgent tasks. When an urgent task arrives, the execution priority of the task is determined based on the arrival time of the task.

4.2 The Agent Selection from the Cloud Nodes

In this study, through the characteristics of parallel computing in the cloud, when a large amount of heterogeneous sensing data is collected by the middleware layer, it is transmitted to the cloud computing layer. In the study, agents are selected through cloud nodes to assign task to the appropriate cloud nodes for data processing, so that the cloud nodes can serve a large number of data processing tasks. However, when a cloud node is performing task processing, when a large amount of tasks needs to be performed, if there is not an effective mechanism, it may cause a delay in the task processing and then affect the service quality. And urgent task needs to be performed at a faster speed. Therefore, this study provides the agent of cloud nodes selection, and uses the characteristics of parallel computing in the cloud to enable tasks to be dispatched to the nodes for data processing in an orderly manner. In addition, when the task is assigned to the appropriate cluster nodes, each agent of cloud nodes changes at any time based on the information provided by the cloud resources. Therefore, in this study, an Urgent First Min-Min scheduling algorithm is proposed as a node selection.

Because the traditional Min-Min scheduling algorithm does not consider the priority of task execution. Therefore, the Min-Min scheduling algorithm is improved in this study. The proposed Urgent First Min-Min scheduling algorithm will perform the task assignment according to different demand nature to provide users with different services.

The Urgent First Min-Min scheduling algorithm establishes a minimum task completion time for each currently unscheduled task, and assigns the task to a cloud node with a minimum task completion time for processing, resulting in a smaller waiting time. When the Urgent First Min-Min scheduling algorithm is executed, the tasks are divided into general task and urgent task according to its urgency, and urgent task will be executed first.

There are three steps of Urgent First Min-Min scheduling algorithm:

[Step 1]: The time required to complete the task waiting to be executed at each cloud node is calculated by each agent.

[Step 2]: (1) If there is more than one task waiting to be executed, the urgent task will be executed first.
(2) If the general work is being executed by the cloud node, execute (3), if not, execute (4).
(3) Because urgent task must be performed preemptively, tasks in the urgent queue are assigned to cloud nodes for preemptive services. At this time, the cloud node that can complete the urgent task in the shortest time will be selected, the task will be allocated to the node in order, and the ongoing general task will be temporarily interrupted.
(4) The node with the shortest task completion time required for general task execution will be selected, the task will be dispatched to this node for service, and wait for the ongoing task to finish.

[Step 3]: Repeat [Step 1] to [Step 3] until all tasks in the queue have been performed.

When a large amount of data has been scheduled in the urgent data queue or the general data queue, it will be allocated to the appropriate cloud nodes according to the different data weights and stored in the cloud database. Because urgent task must be performed in a faster order, and in order to reduce the waiting time of the task, this study uses the Urgent First Min-Min scheduling algorithm to enable the task to be efficiently allocated to the processing nodes based on different characteristics and services. In other words, when all cloud nodes are performing task, two situations will be considered.

(1) If all cloud nodes are performing general task, a cloud node that can complete urgent task in the shortest time will be selected, the task currently being performed will be interrupted, and the urgent task is set as the first serviced task.
(2) If all cloud nodes are performing urgent task, the cloud node that can complete the current task in the shortest time will be selected by the agent. After the current urgent task is completed, this cloud node is selected to execute the new urgent task.

5. Experimental Simulation and Results

In this section, simulation experiments on SDPM under the SoAIoT will be performed. In the experiment, a Network Simulation 2 (NS2) [11] designed by the American Wireless Communications Association (The VINT Project) was used to carry out the network topology and environment construction of this study.

5.1 Experimental Assumptions

The method proposed in this study will be simulated in the following experimental and environmental assumptions. The assumptions of this experiment are as follows:

(1) Take the application of earth and rock flow monitoring as an example.
(2) The transmission of sensing data will not be delayed by environmental factors.
(3) No damage or loss will occur during the transmission of sensing data.
(4) The execution time of each task can be predicted.
(5) Each task will have different execution time in different cluster nodes.
(6) Nodes can support different requirements based on different applications.
(7) All tasks can be processed by the node, that is, all tasks are within the capacity of the node, so when the task enters, it can be executed and completed.

Table 1 shows the experimental environment parameters used in this experiment. The network topology architecture is used to simulate the network topology of the IoT, and is the SoAIoT architecture proposed by this research. The experimental simulation executes the SDPM mechanism under SoAIoT, and the running application is the earth and rock flow monitoring system. Table 2 shows the total number of tasks sent by users and the environmental parameters of the nodes. When users from different regions send 1,000 service requests to the earth and rock flow monitoring system, there will be 50 service processing nodes to perform task processing to meet the needs issued by users. In order to verify that the method proposed by this research can achieve the research purpose, the evaluation factor is compared in the experiment to verify the feasibility and correctness of the method proposed by this research.

Table 1. The experimental environment parameters.

Item	Parameter
Network topology	SoAIoT
Application	Earth and rock flow monitoring
Number of rainfall sensors by region	10
Total number of regions	100
Areas with high frequency of earth and rock flow	10
Number of users	500

Table 2. The total number of tasks sent by users and the environmental parameters of the nodes.

Task	Parameter
Service type	Rainfall query
Total number of service processing tasks	1,000
Total number of service processing nodes	50

In the experiments to evaluate the factor, two different scenarios will be designed to simulate SDPM. In the experiments, the weight of each task, the number of tasks in low-frequency earth and rock flow areas or in high-frequency earth and rock flow areas, the requirement for task execution in low-frequency earth and rock flow areas or in high-frequency earth and rock flow areas are given in two scenarios. Table 3 shows the task data parameter range for [Scenario 1], and Table 4 shows the task data parameter range for [Scenario 2].

Table 3. Task and task profile parameter ranges used in evaluating factors (Scenario 1).

Item	Parameter
Weight=1 (Number of general tasks)	5
Weight=0 (Number of general tasks)	55
Number of tasks in low-frequency earth and rock flow areas	50
Number of tasks in high-frequency earth and rock flow areas	10
Requirement for task execution in low-frequency earth and rock flow areas	31.8~43.9(KB)
Requirement for task execution in high-frequency earth and rock flow areas	71.4~82.9(KB)
Range of weight	0~1

Table 4. Task and task profile parameter ranges used in evaluating factors (Scenario 2).

Item	Parameter
Weight=1 (Number of general tasks)	10
Weight=0 (Number of general tasks)	90
Number of tasks in low-frequency earth and rock flow areas	90
Number of tasks in high-frequency earth and rock flow areas	10
Requirement for task execution in low-frequency earth and rock flow areas	31.8~43.9(KB)
Requirement for task execution in high-frequency earth and rock flow areas	71.4~82.9(KB)
Range of weight	0~1

The requirement for task execution in the evaluation factor will vary according to the size of the file. This experiment is divided into two types based on the probability of earth and rock flow occurrence. The minimum value of the file in the high frequency earth and rock flow area is 71,493 Bytes to the maximum 82,893 Bytes; the minimum value of files in areas with low frequency of earth and rock flow is 31,893 Bytes to the maximum value of 43,893 Bytes. Therefore, the task execution requirement value will be designed with Kilobyte (KB) as different task execution requirement values, and will generate 90 task requirement values of 31.8 KB to 43.9 KB with random numbers, and 10 task requirement values of 71.4 KB to 82.9 KB. In the setting of the Weight value, it can be set according to different application requirements. This experiment is based on the lifting of the earth rock flow alert issued by the Soil and Water Conservation Bureau in Taiwan, and distinguishes between the different Weight values. Since it takes a certain amount of time for soil moisture to dissipate, in order to avoid the occurrence of major disasters in the area caused by the occurrence of earth and rock flows, if there is subsequent scattered rainfall at this time, it may also cause earth and rock flows, so the earth and rock flow alert may not be lifted in synchronization with the typhoon alert [12]. In this experiment, the setting values of the Weight values are “0” and “1”, respectively. “0” indicates that no earth rock flows have occurred in the area, and “1” indicates that earth rock flows alerts have been sent in the area within 12 hours.

Finally, in the evaluation factor, when the task has been calculated by the shift, the schedule time of the node will be calculated, and the task will be assigned to the node for processing. Therefore, the total number of tasks will be equal to the total number of tasks. The range of task setting parameters for [Scenario 1] in this experiment is shown in Table 5, and the range of task setting parameters for [Scenario 2] is shown in Table 6. The nature of task can be divided into urgent task and general task. Due to the small amount of data for urgent task, considering real-life situations, only when a continuous rainfall phenomenon or typhoon disaster occurs in a particular season can the warning threshold be exceeded.

Table 5. Parameter range of task data for evaluation factors (Scenario 1).

Item	Value
Number of urgent tasks	10
Number of general tasks	60
The simulation time of urgent tasks	3~10 (microseconds)
The simulation time of general tasks (low-frequency earth and rock flow area)	10,000~30,000 (microseconds)
The simulation time of general tasks (high-frequency earth and rock flow area)	55,000~90,000 (microseconds)

Table 6. Parameter range of task data for evaluation factors (Scenario 2).

Item	Value
Number of urgent tasks	10
Number of general tasks	100
The simulation time of urgent tasks	3~10 (microseconds)
The simulation time of general tasks (low-frequency earth and rock flow area)	10,000~30,000 (microseconds)
The simulation time of general tasks (high-frequency earth and rock flow area)	55,000~90,000 (microseconds)

Moreover, urgent task only sends warning data that exceeds the threshold value, so the task execution time of the urgent task at the node is shorter than the general task. The general task will continue to collect rainfall data on the occurrence of rainfall, so the number of tasks is more than the urgent task. In the evaluation factor, the simulation execution time of the node is changed. The execution time of the general task simulation in the low-frequency earth and rock flow area is 10,000 microseconds and the maximum time is 30,000 microseconds. The simulation execution time of urgent tasking on the node has a minimum value of 3 microseconds and a maximum value of 10 microseconds.

5.2 Analysis and Comparison of Simulation Results

In this experiment, the execution changes of SDPM will be simulated when different tasks are received in different time periods. Experiments include: the impact of SDPM on the order of task execution in the earth and rock flow monitoring and the impact of SDPM on task execution time in the earth and rock flow monitoring. There are two different scenarios designed to simulate SDPM receiving changes in the execution of different tasks in different time periods. Because this experiment uses the time zone method for task processing, the number of tasks received may be different in different time zones. Therefore, this experiment assumes that there are two different situations for the tasks received:

[Scenario 1]: The total number of tasks executed is 60, in which the number of general tasks with Weight equal to 1 is 5 and the number of general tasks with Weight equal to 0 is 55.

[Scenario 2]: The total number of tasks executed is 100, in which the number of general tasks with a Weight value equal to 1 is 10, and the number of general tasks with a Weight value equal to 0 is 90.

5.2.1 The Impact of SDPM on the Order of Task Execution in the Earth and Rock Flow Monitoring

At this stage, two different methods will be used to compare each other. The WE-SJF scheduling method proposed in this study will be compared with the E-SJF scheduling method. When [Scenario 1] was received during the time phase: There are 60 tasks, of which tasks with weight equal to 1 include tasks 1, 5, 6, 17, and 34.

In the SDPM mechanism, when tasks are scheduled via the WE-SJF and E-SJF scheduling algorithms, the results can be obtained as shown in Figure 2. The simulation result of the WE-SJF scheduling algorithm is the maximum execution order task in the fifth task, and the execution order is the fifth order. The simulation result of the E-SJF scheduling algorithm is the maximum execution order task at the 34th task, and the execution order is the 27th place. Because the WE-SJF scheduling algorithm preferentially screens tasks with a Weight value of 1 for scheduling calculations, tasks with a Weight value of 1 will execute faster than the results by the E-SJF scheduling algorithm. Based on the environmental results of the experimental simulation time [Scenario 1], the WE-SJF scheduling algorithm proposed by this research enables the data processing tasks with a Weight value of 1 issued by the earth and rock disaster area to obtain higher execution priority. Therefore, the tasks in the disaster area can be sent to the cloud node to assign agents for task scheduling.

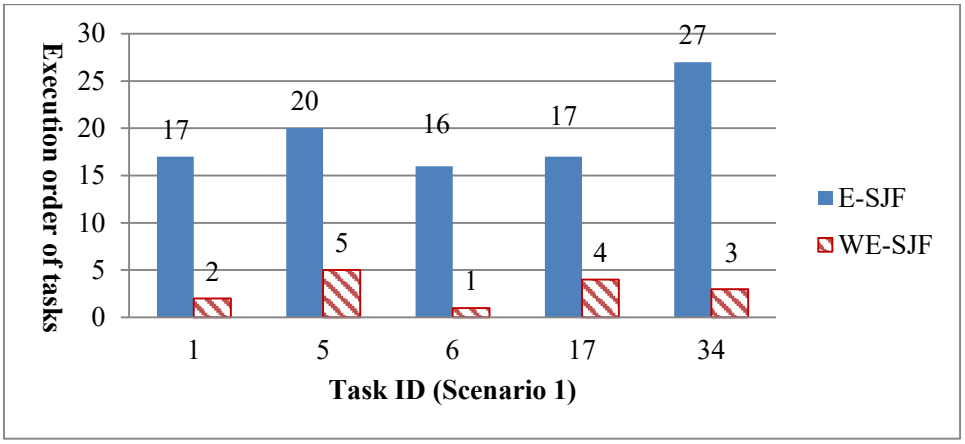

Figure 2. Scheduling results of WE-SJF and E-SJF (Scenario 1).

When the time period received [Scenario 2]: There are 100 tasks, of which tasks with weight equal to 1 include tasks 1, 5, 6, 17, 24, 27, 31, 32, 34, and 49. After the tasks are scheduled by the WE-SJF and E-SJF scheduling algorithm, the results can be obtained as shown in Figure 3. The simulation result of the WE-SJF scheduling algorithm is the maximum execution order task in the fifth task, and the execution order is the tenth order. The simulation result of the E-SJF scheduling algorithm is the maximum execution order task at the 34th task, and the execution order is the 90th order. Because the WE-SJF scheduling algorithm preferentially screens tasks with a Weight value of 1 for scheduling, tasks with a Weight value of 1 will execute faster than the results by the E-SJF scheduling algorithm. According to the results of the experimental simulation [Scenario 2], the WE-SJF scheduling algorithm proposed by this research enables the data processing tasks with a weight of 1 sent by the earth and rock disaster area to get higher execution priority. Therefore, the tasks in the disaster area can be sent to the cloud node to assign agents for task scheduling.

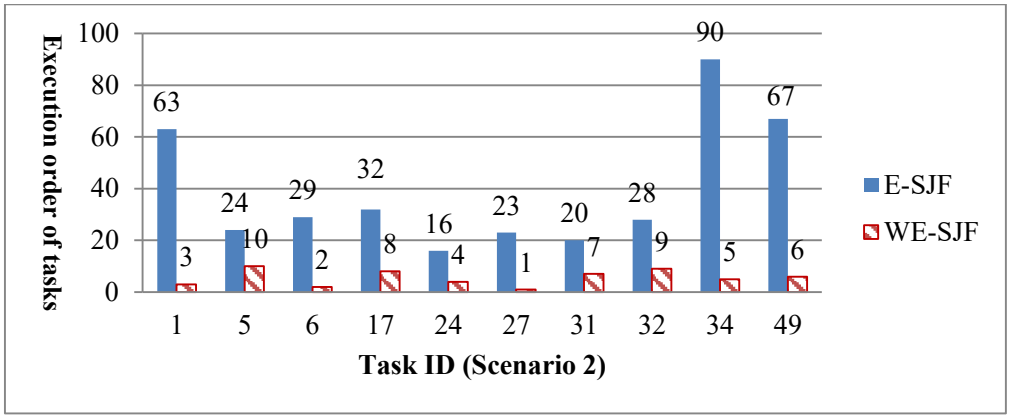

Figure 3. Scheduling results of WE-SJF and E-SJF (Scenario 2).

Finally, this study observes the comparison of task execution requirements with a Weight value of 1 in time [Scenario 1] and time [Scenario 2]. The observation results are shown in Figure 4. In [Scenario 1], the order different of the task with the last Weight value of 1 in the E-SJF and the task with the Weight value of 1 in the last order of the WE-SJF is 22, which means that WE-SJF has performed 22 fewer execution tasks, so WE-SJF has 933.2 fewer execution requirements than E-SJF execution requirements. In [Scenario 2], since the order different of the task with the last Weight value of 1 in the E-SJF and the task with the Weight value of 1 in the last order of WE-SJF is 80. That is, WE-SJF has less execution with 80 tasks; hence, WE-SJF has 3,192 fewer execution requirements than E-SJF. By observing the simulation results of [Scenario 1] and [Scenario 2], data processing tasks in disaster areas can be quickly processed by WE-SJF.

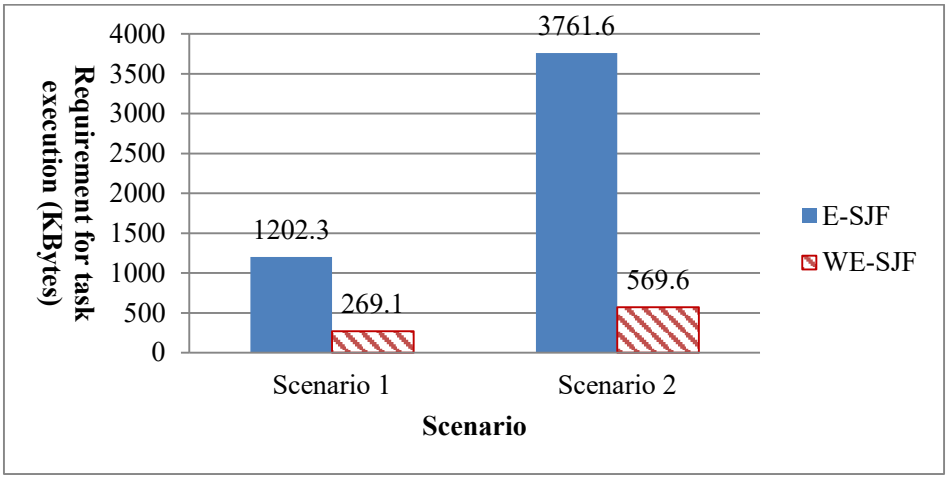

Figure 4. Comparison of task execution requirements.

5.2.2 The Impact of SDPM on Task Execution Time in the Earth and Rock Flow Monitoring

When tasks are sorted by a scheduling algorithm and sent to a cloud node to select an agent for task scheduling, the total work will be received as [Scenario 1] with 70 tasks; [Scenario 2] with 110 tasks to be performed task scheduling. In the experiment, it will be assumed that the general task and urgent task of the above two different scenarios are scheduled, and the number of urgent task is 10 in [Scenario 1] and [Scenario 2]. First observe the situation [Scenario 1], the schedule comparison results of 70 tasks, as shown in Figure 5. Figure 5 shows the comparison of the schedules of [WE-SJF+Urgent first Min-Min] and [E-SJF + Min-Min] for the tasks in the disaster area. The number of general tasks set in the scheduling experiment at this stage is to receive the results of 60 general task scheduling. Five general tasks with a Weight value of 1 are 1, 5, 6, 17, and 34, and the rest are 55. The number of urgent tasks at this stage is scheduled to generate 10 urgent tasks, which are tasks 101, 102, 103, 104, 105, 106, 107, 108, 109, and 110. The sum of general task and urgent task is 70.

From the experimental observations in Figure 5, we can know that the [WE-SJF + Urgent first Min-Min] proposed by this research is more efficient than the [E-SJF + Min-Min] in the disaster area when processing the disaster area. And, the total number of reduced time is 150,525. Because [Urgent first Min-Min] prioritizes urgent task scheduling and [WE-SJF] prioritizes task batching, the method proposed by this research can effectively reduce the data processing time in disaster areas.

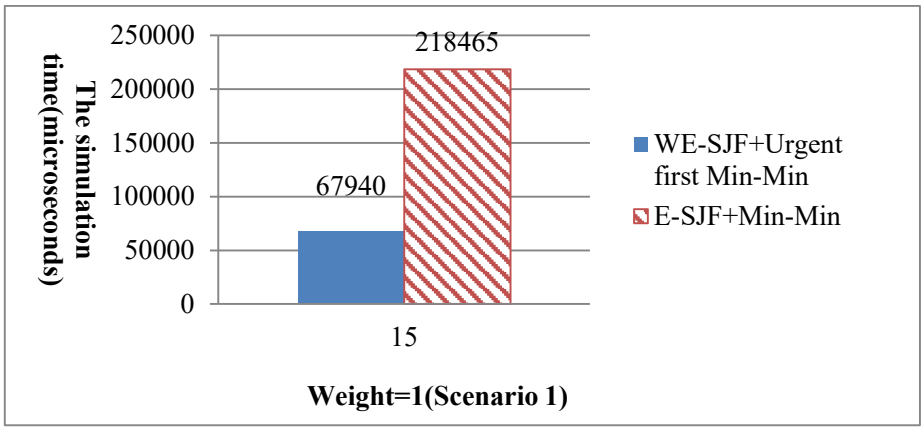

Figure 5. Comparison of WE-SJF + Urgent first Min-Min and E-SJF + Min-Min task schedules in disaster areas (Scenario 1).

The next observation is [Scenario 2]; Figure 6 shows the comparison result of the task schedule when there are 110 tasks. Figure 6 shows the comparison of [WE-SJF + Urgent first Min-Min] and [E-SJF + Min-Min] task schedules in disaster areas. The number of general tasks set in the scheduling experiment at this stage is to receive the results of 100 general tasks. The 10 general tasks with a Weight value of 1 are 1, 5, 6, 17, 24, 27, 31, 32, 34 and 49, and the rest are 90 general task with a Weight value of 0. The number of urgent tasks at this stage is scheduled to generate 10 urgent tasks, which are tasks 101, 102, 103, 104, 105, 106, 107, 108, 109, and 110. The total of general task and urgent task is 110.

From the experimental observations in Figure 6, we can know that the [WE-SJF + Urgent first Min-Min] proposed by this research is more effective than the [E-SJF + Min-Min] in the disaster area when processing the disaster area. And, the total number of reduced time is 285,829. Because [Urgent first Min-Min] prioritizes urgent task scheduling and [WE-SJF] prioritizes task batching, the method proposed by this research can effectively reduce the data processing time in disaster areas.

Based on the evaluation results of [Scenario 1] and [Scenario 2] in simulation, when [WE-SJF + Urgent first Min-Min] is used to process the task in the earth and rock flow monitoring, it is possible to reduce the task processing time more efficiently than [E-SJF + Min-Min], because the task in the earth and rock flow monitoring and the task that occurs in an urgent are preferentially processed than the faster data processing speed can be gotten.

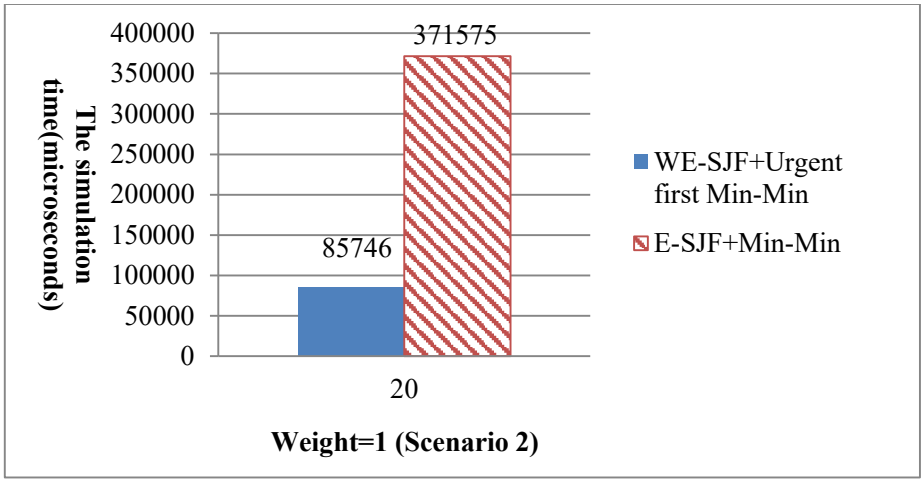

Figure 6. Comparison of WE-SJF + Urgent first Min-Min and E-SJF + Min-Min task schedules in disaster areas (Scenario 2).

6. Conclusion

Because the IoT uses smart objects embedded with sensors, and through the IoT communicate with each other, it can achieve the life of universal computing, automation, and remote operation of objects [1]. When the concept of the IoT is used, users can enjoy IoT services through various terminals connected to the Internet. In order to provide diversified services to users, these sensors from different applications will continue to provide mass information, so that the services of these applications can be maintained normally. However, when massive amounts of IoT data are continuously transmitted to the Internet and a large number of users use the Internet to send service requests to the IoT, it will cause an increase in the burden of Internet transmission.

In order to reduce the quality of services affected by the large amount of heterogeneous data and service requirements transmitted through the Internet, and considering that the current IoT architecture does not yet have a standard. In this research, a service-oriented architecture for the IoT (SoAIoT) is proposed, SoAIoT uses SDPM to reduce the burden of the IoT data transmission on the Internet, while also meeting the needs of users for different services. Overall, the architecture and operating mechanism proposed by this research can provide a large number of application developments in the IoT, and can provide users' services of different needs of the IoT.

Acknowledgment

This work was supported in part by the Ministry of Science and Technology MOST 107-2221 -E-324-005-MY3.

References

[1] Atzori, L., Iera, A., and Morabito, G. 2010. The internet of things: A survey. *Computer networks*, 54, 15: 2787-2805.
[2] Ben-Daya, M., Hassini, E., and Bahroun, Z. 2019. Internet of things and supply chain management: a literature review. *International Journal of Production Research*, 57, 15, 4719-4742.
[3] Ashton, K. 2009. That 'internet of things' thing. *RFID journal*, 22, 7: 97-114.
[4] Arsanjani, A., Allam, A. 2006. Service-oriented modeling and architecture for realization of an SOA. *In 2006 IEEE International Conference on Services Computing (SCC'06)*, 521-521
[5] Ranjan, A. and Sahoo, B. 2020. Web service selection mechanism in service-oriented architecture based on publish–subscribe pattern in fog environment. *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, 269-281.
[6] Niknejad, N. and Amiri, I. S. 2019. Introduction of Service-Oriented Architecture (SOA) Adoption. *The Impact of Service Oriented Architecture Adoption on Organizations*, 1-8.
[7] Pulparambil, S. and Baghdadi, Y. 2019. Service oriented architecture maturity models: A systematic literature review. *Computer Standards & Interfaces*, 61: 65-76.
[8] Chiang, M. C., Huang, C. Y., Wu, C. Y., and Tsai, C. Y. 2020. Analysis of a Fault-Tolerant Framework for Reliability Prediction of Service-Oriented Architecture Systems. *IEEE Transactions on Reliability*.
[9] Abraham, S., Greg, G., and Peter, B. G. 2005. *Operating System Concepts*, John Wiley & Sons, Inc., seventh edition.
[10] Wang, S.C., Yan, K.Q., Wang, S.S. and Chen, C.W. 2010. Improving system service efficiency with a three-stage scheduling mechanism in a cloud computing environment. *16th Information Management and Practice Symposium*, 2265-2281.
[11] Breslau, L., Estrin, D., Fall, K., Floyd, S., Heidemann, J., Helmy, A., Huang, P., McCanne, S., Varahan, K., Xu, Y. and Yu, H. 2000. *Advances in network simulation. Computer*, 33, 5: 59-67.
[12] Zhu, X., Mukhopadhyay, S. K., and Kurata, H. 2012. A review of RFID technology and its managerial applications in different industries. *Journal of Engineering and Technology Management*, 29, 1: 152-167.