symmetry

MDPI

Article

Resource Symmetric Dispatch Model for Internet of Things on Advanced Logistics

Guofeng Qin *, Lisheng Wang and Qiyan Li

Department of Computer Science and Technology, Tongji University, Shanghai 201804, China; lishwang@tongji.edu.cn (L.W.); qylcad@sina.com (Q.L.)

* Correspondence: gfqing@aliyun.com; Tel.: +86-21-6598-1423

Academic Editor: Yuhua Luo

Received: 16 December 2015; Accepted: 31 March 2016; Published: 5 April 2016

Abstract: Business applications in advanced logistics service are highly concurrent. In this paper, we propose a resource symmetric dispatch model for the concurrent and cooperative tasks of the Internet of Things. In the model, the terminals receive and deliver commands, data, and information with mobile networks, wireless networks, and sensor networks. The data and information are classified and processed by the clustering servers in the cloud service platform. The cluster service, resource dispatch, and load balance are cooperative for management and monitoring of every application case during the logistics service lifecycle. In order to support the high performance of cloud service, resource symmetric dispatch algorithm among clustering servers and load balancing method among multi-cores in one server, including NIO (Non-blocking Input/Output) and RMI (Remote Method Invocation) are utilized to dispatch the cooperation of computation and service resources.

Keywords: Internet of Things; network integration; symmetric resource dispatch; cooperative systems; logistics service lifecycle

1. Introduction

The Internet of Things is a hot-point in theory and engineering field because it involves the complexity of big data, massive amounts of information, the concurrency of operation and application, and real-time services. The Internet of Things is an auto-cooperative system and platform with integration of Internet, Intranet, Wide area network (WAN), Local area network (LAN), 3&4G, and other sensor network such as Radio-frequency Identification networks (RFID), Bluetooth, controller area network bus (CANBUS), video collection networks and so on. For high performance and reliable response, a hybrid cluster-based (HC) wireless sensor network (WSN) architecture was proposed by Huang *et al.*, [1], which could transmit emergency data packets in an efficient manner during an emergency case. An insect organization model was setup in sensor networks by Ma and Krings [2], which justified how and why insect sensory systems might promote computation and communication ability. A distributed detection method on a binary target was considered to send local decisions to a fusion center with wireless sensors by Kim *et al.*, [3], which proposed two new concepts called detection outage and detection diversity in the long-term system performance. Luis and Sebastia, [4] considered that the strong components of the system could provide a cost-effective, real-time, and high resolution means of wildfire prevention in a sensor network.

It is very important to study component structure for high performance of systems with low power. Saaty and Shih, [5] considered a network structure must satisfy two requirements. The first one is that the similar sensors should be identified and grouped together. The other is that the relationship among them should be kept accurately according to the flow of networks. A new algorithm in the IP address range on ABC types networks based on the QoS of the user was proposed in order to consider a better resource distribution for operators by Haydar *et al.* [6]. Alexandros *et al.*, [7] verified the core

diameter size should change and modify with the lightweight and scalability of tasks from many experiments. A cooperative information integration platform was studied by Qin and Li *et al.*, [8–11], which consists of the intelligent mobile terminals, software systems, integrated 3&4G, including global positioning system (GPS), general packet radio service (GPRS) or code division multiple access (CDMA), Internet (Intranet), and remote video monitor and M-DMB (mobile digital media broadcast) networks. Miguel *et al.*, [12] proposed a collaborative decision method to process tasks in lower power.

During advanced logistics service lifecycle, application tasks from cloud terminals are highly concurrent with big data and information. For real-time customer service, dynamic symmetric resource dispatch and load balancing on multi-cores play a very important role for high performance cluster computing ability in low power [13].

According to this motivation, a resource symmetric dispatch model for Internet of Things on advanced logistics service lifecycle is studied, in Section 2, a symmetric integration cooperative system structure is proposed, in Section 3, resource symmetric dispatching among clustering servers is studied, including load balancing among multi-cores, in Section 4, experiment and analysis of load balancing algorithm are provided. Finally, an application case using a prototype system for the cloud platform on advanced logistics service will be reported as a result.

2. The Level Symmetric Structure for Logistics Business with Internet of Things

According to the logistics business application service process, the vehicles' GPS information, driver request messages, sensor information (for example RFID, temperature, pressure, stress, angle), and application tasks in all the terminals are sent to the cloud platform. On the other hand, the control center commands are sent to application terminals by the integrated WAN on 3&4G and Internet (Intranet). Different types of information, including messages, data files and video stream can be sent and received freely and safely via internet. The details of logistics business structure can be seen in Figure 1. The logistics business structure includes Web service & Electronic business layer, management layer, operation layer, and device execution layer. Communication of the different layers depends on the information and data bus to receive and deliver information and data from the different business modules in the cloud service platform.

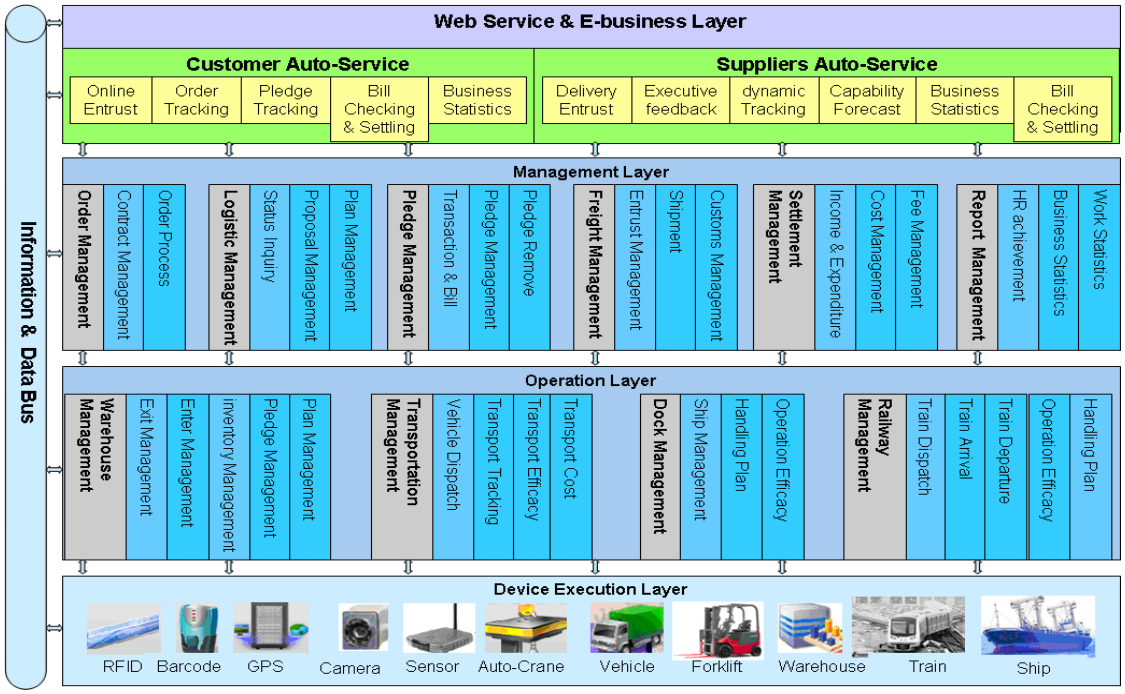

Figure 1. Business structure on advanced logistics service.

In the business layer structure, the users can do anything with client personal computers or cloud terminals, including mobile phones and PAD. The users are vehicle drivers, operators, and managers from groups, manufacturers, factories, branch companies, procurement buyers, suppliers, banks, steel factories, management department, warehouse centers *etc.* The users can dispatch vehicles to transport the goods, monitor and manage status of the goods, and finish the financial settlement among the different organizers in the product life cycle at any-time and any-where by all available networks in the cloud platform.

For any-time and any-where business service, a symmetric service platform consists of a large set of intelligent mobile cloud terminals with software systems, integrated 3&4G units, remote video monitors, and different kinds of communication networks such as GPS, GPRS (CDMA), Internet (Intranet), RFID, CANBUS, wireless network, Bluetooth and so on. This symmetric method from business service to data and information service has been greatly promoting business to business, business to development and speeding up circulation of commodities among customers.

2.1. Symmetric Structure for the Internet of Things

In the cloud platform, there are management, operation, and device execution layers in the application layer. There are the mobile cloud terminals, B/S (Browser/Server structure) clients, C/S (Client/Server structure) clients in the application layer. The network layer comprises wireless communication network, Internet, Intranet, and wireless sensor network. The mobile cloud terminals and executive devices are connected to the cloud service platform with 3G or 4G and internet. The B/S clients are connected the cloud platform through Internet and Intranet, also the C/S clients are connected the cloud platform with the Intranet. The logical layer includes the platform group tool-wares and middle wares to deal with parallel application cases among servers. There are software tools for communication, application, geographic information system (GIS), Web Server and the interface in the cloud platform. There are two important supports of reliability and real-time in the platform, one is support tools, and the other is resource symmetric dispatch among clustering servers, including load balancing among multi-cores in one server. The support tools are composed of XML data protocol transfer ware, Java database connectivity (JDBC), NIO (Non-blocking Input/Output), RMI (Remote Method Invocation), load balancer, and so on. The data layer includes all the databases which contain data, video, and audio. The details can be seen in Figure 2.

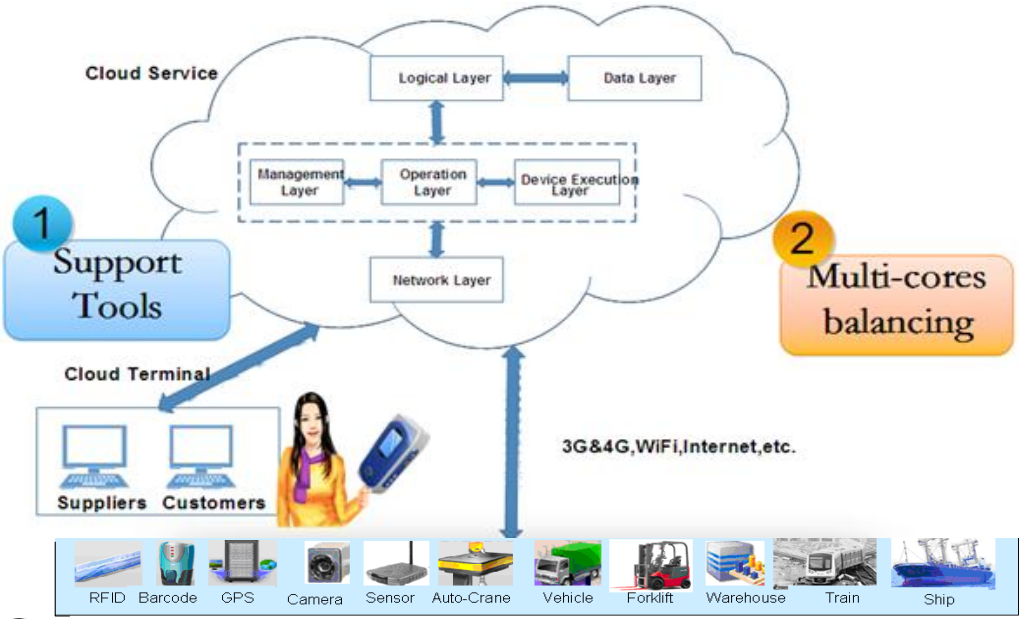

Figure 2. Layer structure of logistics service cloud platform.

2.2. Hierarchical Business Task Dispatch in the Internet of Things

There are many application tasks from smart mobile phones, C/S & B/S clients, and auto-hardware devices including mobile application (APP), electronic business application, and enterprise resource management application. Because information is collected from auto-hardware devices and commands are delivered to execution devices, procedure of data and information are highly concurrent and throughput capacity is very big at any time. In order to improve service performance of cloud platform just in time, a hierarchical structure for business task dispatch is proposed, the first layer is used for assigning application task among servers in the same service cluster; the second layer is charged with dispatching computation resource for one task among multi-cores in a same server. The details of the hierarchical structure for application task dispatch can be seen in Figure 3.

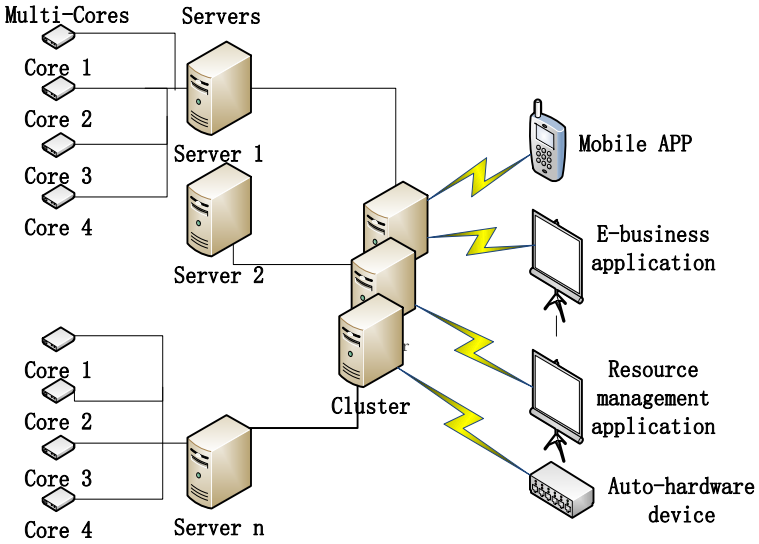

Figure 3. Hierarchical structure for application task dispatch.

In order to improve parallel processing performance, the cloud service platform makes use of the reactor design pattern and events are selected by the selector at set intervals. When a non-blocking method is dealing with Input/Output (I/O) operations, it might return without waiting for the I/O to finish in threads pool, the platform would achieve higher performance and greater parallel processing capacity. The details of work mechanism of NIO can be seen in Figure 4. A Server-Socket-Channel is first registered to the selector as an acceptor or a request handler. Then the acceptor handles the client requests passed from the selector and spawns Socket-Channels to receive or send data. The selector polls all channels and dispatches client requests or I/O operations at set intervals. Of course, under multi-core architecture, thread pool is often used to contain many threads.

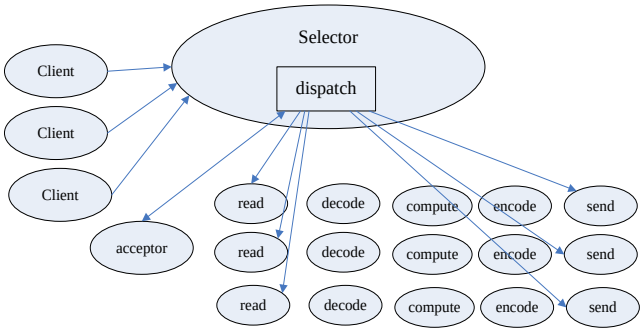

Figure 4. Work mechanism of NIO in clustering servers.

3. Resource Symmetric Dispatching among Clustering Servers in Cloud Platforms

For high performance service just in time, there are clustering servers in cloud platform. Resource symmetric dispatch and load balancing by multiple thread service is the key technical solution in communication server cluster, Web-Service application server cluster and Database server cluster. In order to symmetrically dispatch resource and balance load, there are two layers, one is resource dispatch among clustering servers, and the other is load balancing for multi-cores of server, which can balance the tasks of the clustering servers and the computation resource among multi-cores of server.

3.1. Resource Dispatching Algorithm among Servers

Hypothesis: N is the number of computer process units in the communication servers. E is the whole time cost without loader in one unit. L is the loader of only one unit. The work load of the others is 0. If the communication server can be paralleled, the time cost is E/N without loader in the system [21,22]. If there is a loader L in one unit, then its computing ability is $1/L$, so if loader 1 is added to the unit, its computing ability will be $1/(L+1)$, then the whole time cost C is as follows:

$$C = \frac{E}{(N-1) + \frac{1}{1+L}} = \frac{E(1+L)}{NL+N-L} \tag{1}$$

The Equation (1) is a theoretical result in a theoretical condition. In fact, the whole time cost C' can approximately be expressed in load balanced condition of cluster server as follows:

$$C' = C + \delta(N) + T_{comm}(N) \tag{2}$$

$T_{comm}(N)$ is the whole communication time, which can be determined by experience, $\delta(N)$ is the estimated number of units. The details of resource symmetric dispatch algorithm can be expressed as follows:

Algorithm:

Input: Graph G(V,E) and its sub-graph Gcluster (Vclusterp, Eclusterp) where V are logic processors (LPs), the E's are empty links, Vclusterp is the set of LPs assigned to all processors, s ∈ Clusterp, and Eclusterp is the set of empty links belonging to each processor or q ∈ Clusterp; Clusterp is the set of neighbor-processors to p, LSTv, Tminu, Tsminu, v, Tw, u such that u ∈ Clusterp, v ∈ V and w ∈ E(V-Vclusterp).

Output: time-of-next Tu, v and select an LP to process a new event

Begin

PQ is initial and empty/*PQ is the priority queue, a data structure */

For all (u,v) u ∈ Vclusterp, v ∈ (V-Vclusterp) do Tv = 0;

For all v ∈ Vclusterp do

Temp = Minw ∈ E(V-Vclusterp)(Tminv,Tw,u);

Tv = Max(Temp,LSTv); Tu = Minv ∈ PQ(Tv); /*LST is the stimulated time, T is the min timestamp of LP */

Insert (v,PQ);

End for;

While (Not finished) do

If (Non-blocking(Prj))/*unblock any process in Prj*/

Select an LP to process a new event/*setup a logic processor to process a new event */

End if

End while

Because data and information of application tasks from mobile APPs, electronic business applications, resource applications, and auto-hardware devices to clustering servers are very highly concurrent, this algorithm is put forward to dispatch application and balance work load from different application terminals among servers in same cluster. This method can promote the ability and efficiency of cluster and make the servers efficient run with low power.

3.2. Load Balancing among Multi-Cores

More and more servers have used the multi-core architecture. However, there exist some problems in utilizing the multi-core processors. Firstly, it is known to us that I/O operation is much slower than the CPU processing speed, thus the traditional blocking I/O keeps CPU waiting simultaneously instead of doing any practical jobs [14,15]. Secondly, tasks are distributed unevenly to threads which lead to workloads unbalancing among multi-cores [16].

In order to solve these problems, a multi-core load balancing model based on the Java NIO (Java New I/O) framework is proposed, which utilizes both the high-performance non-blocking I/O in Java NIO framework and parallel processing ability of multi-core processors. In JDK1.4 API, Java NIO was designed to provide access to the low-level operations of modern operating systems and the intent is to facilitate an implementation that can directly use the most efficient operations of the underlying platform [17,18], which provides buffer-based multi-channel non-blocking I/O methods in Java language.

Work load is balanced among multi-cores in a user-level way rather than a kernel way, including the type of load balancing involved in Java Fork/Join framework which is the basis of the model proposed, namely, a multi-core load balancing model and an overall framework with NIO mechanize [19,20]. There exist two problems of multi-core load unbalancing because tasks distributed unevenly to multi-threads [21].

Firstly, from the version of JDK1.2, Java thread is implemented by native thread which means how operating system supports native thread decides the implementation of Java Virtual Machine (JVM) threads. Both the Windows version and the Linux version of Sun Java Development Kit (JDK) use the 1:1 mapping to implement Java thread which means a Java thread is mapped to a LWP (light-weight process) and could be regarded as a native thread. This problem is studied in Linux, of course, it can be extends to other operating systems. Secondly, after comparing different combination of CPU workload factors, Kunz pointed out that the number of tasks in running queue is the best factor for evaluating the CPU workload in Linux [22]. In Linux, a task is a thread. As it is mentioned, a Java thread is a native thread or a kernel thread, therefore, a 1-slice Java thread and a 10-slice Java thread are equivalent when scheduled to cores by the Linux kernel. Thus multi-core load unbalancing emerges.

When developing an application under multi-core architecture, distributing tasks evenly to cores by using multi-threading techniques is an effective way to enhance system performance [23]. A task scheduling model is proposed to solve the problem, which is based on the Java Fork/Join framework in JDK1.7. Fork/Join framework is a classic way of dealing with parallel programming. Though it could not solve all these problems, it is able to utilize multi-cores and make them cooperatively finish heavy tasks within its applicable scope.

In Fork/Join framework, if a task could be divided into some subtasks and the result could be obtained by combining these subtasks, then the task is fit to be solved with Fork/Join pattern as Figure 5. The task is dependent on the subtasks in a low level, so the task 0 cannot get the result until all the subtasks return. Other problems relating parallelism such as load balancing and synchronization could also be solved with the framework.

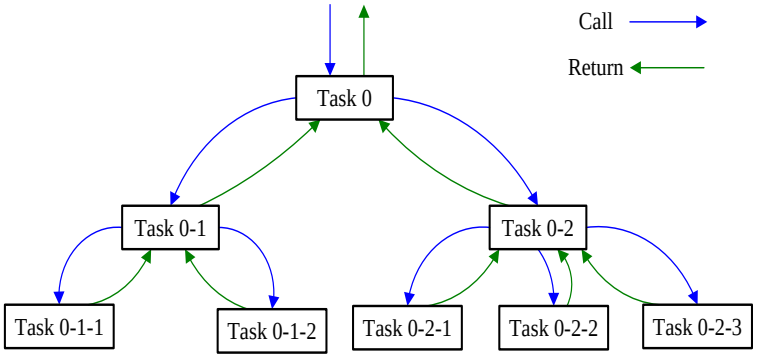

Figure 5. Fork/Join Framework on NIO.

3.3. Balanced Task Scheduling Algorithm among Multi-Cores

Service requests from clients are likely to handle both heavy tasks and lightweight tasks. If one heavy task is distributed to one thread, it would cause load un-balanced. Analogously, too many lightweight tasks that each one occupies a thread would increase high system cost [24,25]. Thus tasks executed in threads should be controlled in a suitable size. A method of task size controlling is used to balance load of multi-cores with parallelizing heavy tasks and serializing light-weight (LW) tasks as Figure 6. Because serialization of the lightweight tasks is relatively easy to attain, it just waits for enough tasks to arrive and fetches a thread from the thread pool to execute the tasks, thus we only discuss the heavy task parallelization. The task scheduling algorithm is based on Java Fork/Join framework and classic task scheduling algorithm.

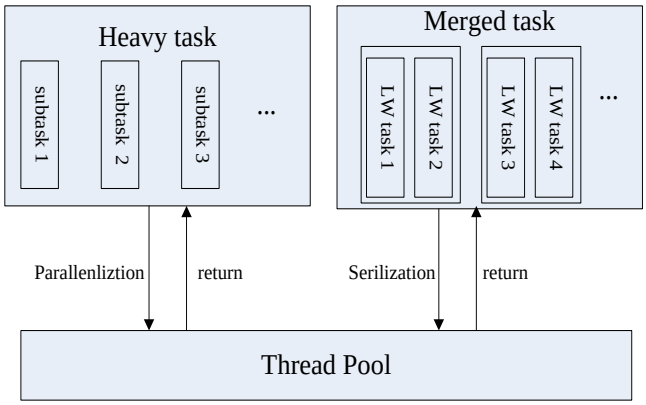

Figure 6. Task parallelization and serialization.

The task scheduling algorithm in Figure 7 is described as follows:

Condition: (1) The server has n processors P_1, P_2, ... P_n, and each processor P_i has a running task queue Q_i which the subtasks run on; (2) There are also n worker threads W_1, W_2, ... W_n and W_i executes the tasks on Q_i; (3) There are k tasks T_1, T_2, ... T_k whose priority ranges strictly from high to low. T_j could be divided into m parallel subtasks $t_{j,1}$,$t_{j,2}$ ... $t_{j,m}$ (m may differ for different tasks) which inherit the priority from their parent task by fork operation.

The steps of the algorithm are the following:

I. For task T_j, the parallel subtasks $t_{j,1}$,$t_{j,2}$ ... $t_{j,m}$ are distributed evenly to all task queues, so each queue has m/n parallel tasks. Tasks with higher priorities are queued earlier than other tasks. Thus the distribution of parallel tasks is obtained and is presented in Figure 8.

II. If Q_i is not empty, W_i dequeues a task from it to execute; else go to Step III.

III. W_i searches the system for the processor which has the longest task queue, locates the task that has the highest priority and migrates it to Q_i. Go back to Step II.

IV. If the migration in Step III is failed, W_i tries to migrate lower-priority tasks or search for other processors to repeat the migration. If all these failures and Q_i is still empty, W_i blocks and waits for new task to awake.

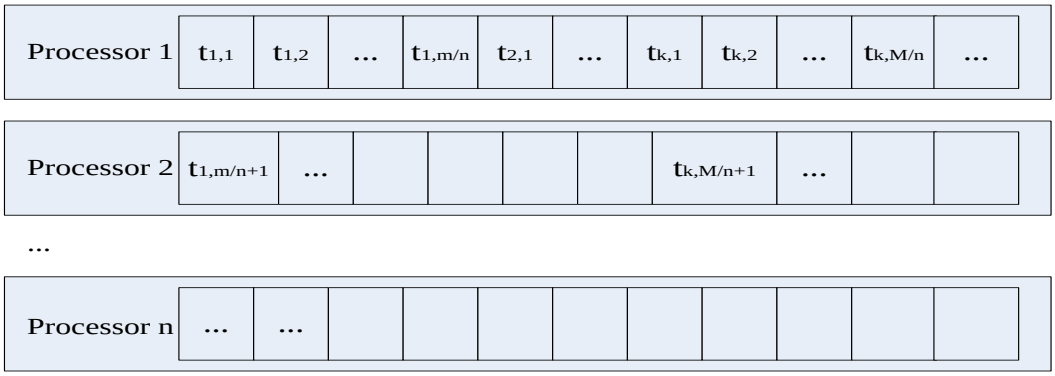

Figure 7. Dispatch of subtasks on multi-cores.

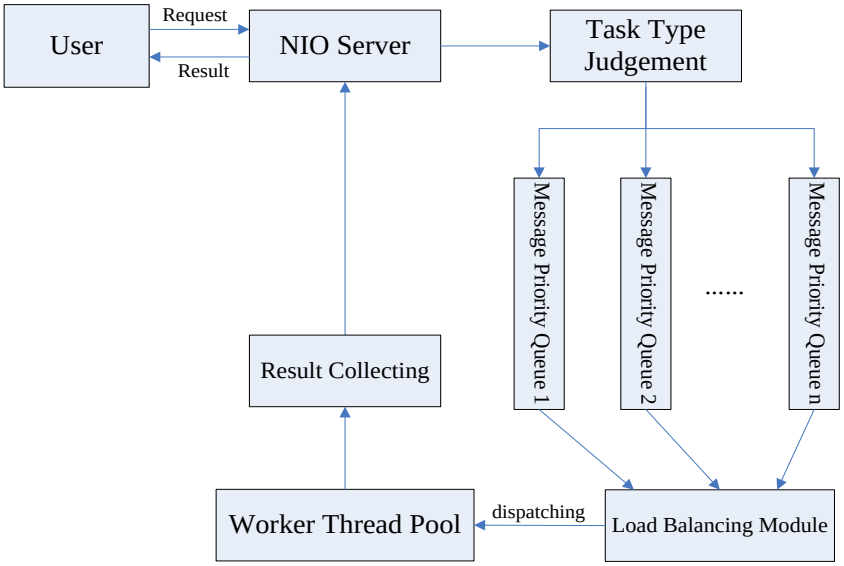

Figure 8. Service task delivering procedure of servers.

On one hand, because of the FIFO task queue structure, later high-priority tasks are queued after earlier low-priority tasks, which assure the chance of execution of low-priority tasks. On the other hand, task migration gets high-priority tasks executed as soon as possible.

3.4. Application Service Delivering Procedure of Servers

The improved task scheduling algorithm with Java NIO is proposed to keep load among overall clustering servers balancing. The procedure is presented in Figure 8.

The service task assigning module plays a very important role for system performance. Some important details of the module are presented as follows:

a. Task type: generally task type determines task size and how tasks are handled, namely, parallelized or serialized. When receiving a task, the server judges its task type at first, and decides

what to do next. For example, a file access task or a mathematical calculation task has more work to do than a submitted form task, so they are better to be parallelized.

b. Message priority queue: in order to handle messages with different priorities, a data structure of message priority queue is introduced to classify the messages. When these queues are polled, more messages in high-priority queues would be chosen and forked.

c. I/O: Java NIO is used in both network I/O and native I/O to minimize the time of waiting for processors in the system architecture.

d. CPU affinity: as a worker thread takes charge of a processor's queue tasks, it is likely to be bound to the processor. However, Java language does not provide CPU affinity, thus we use JNI (Java Native Interface) and set CPU affinity of worker thread by invocating a low-level C language dynamic library.

4. Experiment and Analysis

In order to verify system performance, an experiment is done to test the task scheduling algorithm in load balancing and utilizing multi-core processors. It compares the results of task execution with one core, two cores, and four cores. We adopt three types of classic tasks for test, they are the Fibonnaci program with argument 40 (Fib), quickly sorting of 20,000 integers (Sort) and multiplication of 512-bit and 512-bit integer (Mul). All three types have the same priority. For each type of task, number of subtasks executed on cores and the total execution time was gathered. The tasks are executed 10 times and the results are averaged. The results are presented in Tables 1 and 2 and Figure 9.

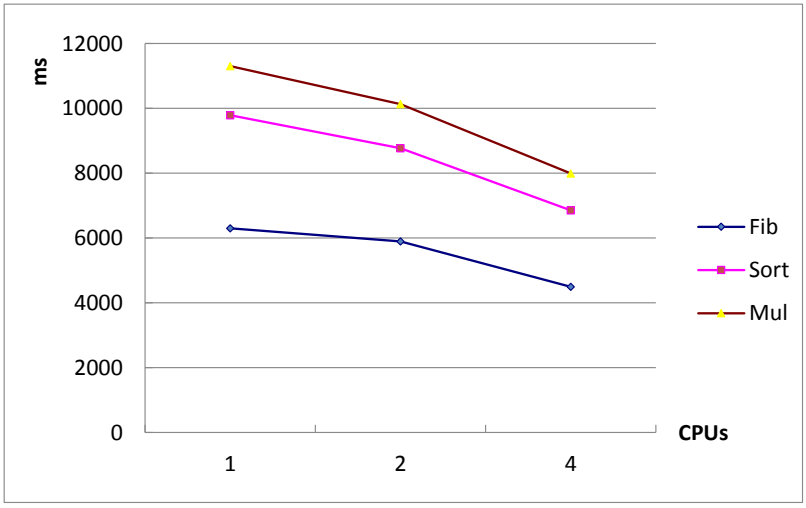

Figure 9. Execution time of tasks.

Experiment environment: Intel Core i7 Q720 1.6GHz, Linux 2.6.32, JDK 1.7. We use VMware Workstation 8.0 as the running environment to configure the number of cores.

Table 1. Number of subtasks on cores.

Test Program	1 Core	2 Cores		4 Cores			
	CPU0	CPU0	CPU1	CPU0	CPU1	CPU2	CPU3
Fib	35,421	17,635	17,806	8868	8990	8820	8742
Sort	66,670	33,020	33,754	16,253	17,120	16,879	16,445
Mul	87,381	43,912	43,469	21,769	21,773	21,964	21,874

Table 2. Execution time of tasks (unit: ms).

Test Program	1 Core	2 Cores	4 Cores
Fib	6300	5895	4496
Sort	9788	8772	6857
Mul	11,302	10129	7994

Table 1 shows that subtasks are evenly distributed and executed on cores. From Table 2 we can see that with the number of cores increasing time cost of each type of task decreases, but it does not have a strictly inverse relationship with number of cores. According to Figure 9, if the task size is larger, the utilization of multi-core processors can be higher.

5. An Application Case

In the cloud platform on Internet of Things for advanced logistics service, many cameras are integrated to a mobile cloud vehicle terminal with CANBUS or USB interface, which can provide service for about a million mobile cloud terminals and two thousand personal computer clients according to its designed parameter, currently there are three hundred mobile cloud terminals and forty PC clients in the cloud service platform. The video stream data with JPG image format is collected and sent to the platform, which are transferred to a video file, the recognition software deals with the images. The security emulation software analyzes status of the vehicle and alarm according to deviation index of lane line at any time. The day, week, and month reports of goods, warehouses, vehicles, employees, customers, finance and so on will be output in the platform.

The details of the logistics business cloud service and the algorithm application can be seen as Figure 10. There are logistics business operation and management service functions in the left, geographic information system (GIS) map in the middle, and map layer management in the right in Figure 10a. There are real-time video and alarm status in the left, and vehicle running status reports in the right in Figure 10b. The management and monitor service requests from operators and sensors are taken place in high concurrency, which must be dealt with by the task scheduling algorithm among the clustering servers and the load balancing algorithm among multi-cores in the cloud service platform.

(**a**) (**b**)

Figure 10. (**a**) Logistics business cloud service; (**b**) Status & security for vehicle in algorithm application.

6. Conclusions

Many flexible methods should be used for obtaining data and information from sensors and terminals, in order to collect and deliver the structure and bio-structure data with the NIO mechanism.

There exist two layers of computing resource dispatch for high performance application service, one is resource symmetric dispatch among clustering servers for application service, and the other is load balancing method among multi-cores in one server, which are in charge of delivering business application to clustering servers equally and balancing task load to multi-core processors effectively. A load balancing algorithm for cloud service was utilized to balance the computation resource among multi-cores. By experiment, the cloud model and the algorithm were tested and proved to be feasible and efficient. An advanced logistics cloud service platform application case verified high performance of symmetric resource dispatch. In the future, the necessary work will be to promote the system dependability, scalability, and other Quality of Service.

Acknowledgments: The researcher is supported by the national 863 program in Ministry of Science and Technology of P.R.China. Project number: 2013AA040302.

Author Contributions: The correspondence author: Guofeng Qin works in the Department of Computer Science and Technology as an associate professor, Tongji University, Shanghai, 201804, P.R.China, who designed the framework of cloud platform on advanced logistics and resource symmetric dispatch Algorithm. He received a BA from Hunan University in 1995, MA from the Huazhong University of Science and Technology in 2001, and Ph.D. from Tongji University in 2004. Lisheng Wang and Qiyan Li are professors in the Department of Computer Science and Technology, Tongji University, Shanghai, 201804, P.R.China, they provided the test tools and environment for resource symmetric dispatch Algorithm, and analyzed the test data on the different test programs.

Conflicts of Interest: We declare no conflict of interest with other people or organizations.

References

1. Huang, M.; Liu, H.; Hsieh, W. A Hybrid Protocol for Cluster-based Wireless Sensor Networks. In Proceedings of the 13th Asia-Pacific Computer Systems Architecture Conference, Hsinchu, Taiwan, 4–6 August 2008; pp. 16–20.
2. Ma, Z.; Krings, A.W. Insect sensory systems inspired computing and communications. *Ad Hoc Netw.* **2009**, *7*, 742–755. [CrossRef]
3. Kim, H.-S.; Wang, J.; Cai, P.; Cui, S. Detection Outage and Detection Diversity in a Homogeneous Distributed Sensor Network. *IEEE Trans. Signal Process.* **2009**, *57*, 2875–2881.
4. Vicente-Charlesworth, L.; Galmes, S. On the development of a sensor network-based system for wildfire prevention. In Proceedings of the 8th International Conference on Cooperative Design, Visualization, and Engineering, Hong Kong, China, 11–14 September 2011; pp. 53–60.
5. Saaty, T.L.; Shih, H.-S. Structures in decision making: On the subjective geometry of hierarchies and networks. *Eur. J. Oper. Res.* **2009**, *199*, 867–872. [CrossRef]
6. Haydar, J.; Ibrahim, A.; Pujolle, G. A New Access Selection Strategy in Heterogeneous Wireless Networks Based on Traffic Distribution. In Proceedings of the 1st IFIP Wireless Days Conference, Dubai, United Arab Emirates, 24–27 November 2008; pp. 295–299.
7. Tsakountakis, A.; Kambourakis, G.; Gritzalis, S. A generic accounting scheme for next generation networks. *Comput. Netw.* **2009**, *53*, 2408–2426. [CrossRef]
8. Qin, G.; Li, Q. An Information Integration Platform for Mobile Computing. In Proceedings of the Third International Conference on Cooperative Design, Visualization, and Engineering, Mallorca, Spain, 17–20 September 2006; pp. 123–131.
9. Qin, G.; Li, Q. Strategies for resource sharing and remote utilization in communication servers. In Proceedings of the 4th International Conference on Cooperative Design, Visualization, and Engineering, Shanghai, China, 16–20 September 2007; pp. 331–339.
10. Qin, G.; Li, Q. Dynamic Resource Dispatch Strategy for WebGIS Cluster Services. In Proceedings of the 4th International Conference on Cooperative Design, Visualization, and Engineering, Shanghai, China, 16–20 September 2007; pp. 349–352.
11. Qin, G.; Wang, X.; Wang, L.; Li, L.; Li, Q. Remote Video Monitor of Vehicles in Cooperative Information Platform. In Proceedings of the 6th International Conference Cooperative Design, Visualization, and Engineering, Luxembourg, Luxembourg, 20–23 September 2009; pp. 208–215.

12. Garcia, M.; Lloret, J.; Sendra, S.; Rodrigues, J.J.P.C. Taking cooperative decisions in group-based wireless sensor networks. In Proceedings of the 8th International Conference on Cooperative Design, Visualization, and Engineering, Hong Kong, China, 11–14 September 2011; pp. 61–65.
13. Wang, Y.; Qin, G. A multi-core load balancing algorithm based on Java NIO. *Telkomnika Indones. J. Electr. Eng.* **2012**, *10*, 1490–1495.
14. Wikipedia. Asynchronous I/O. Available online: http://en.wikipedia.org/wiki/Asynchronous_I/O (accessed on 28 March 2015).
15. Li, S.; Liu, N.; Guo, J. Multi-threading workload balance under multi-core architecture. *Comput. Appl.* **2008**, *28*, 138–140.
16. Wikipedia. New I/O. Available online: http://en.wikipedia.org/wiki/New_I/O (accessed on 10 September 2014).
17. Zhou, Z. *Understanding the JVM Advanced Features and Best Practices*; China Machine Press: Beijing, China, 2011; pp. 336–337.
18. Kunz, T. The influence of different workload description on a heuristic load balance scheme. *IEEE Trans. Softw. Eng.* **1991**, *17*, 725–730. [CrossRef]
19. Lea, D. A Java Fork/Join Framework. In Proceedings of the ACM 2000 Conference on Java Grande, San Francisco, CA, USA, 3–5 June 2000; pp. 36–43.
20. Steven, H.; Costin, I.; Filip, B. Load Balancing on Speed. In Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Bangalore, India, 9–14 January 2010; pp. 124–157.
21. Vivek, K.; William, G. Load Balancing for Regular Meshes on SMPs with MPI. In Proceedings of the 17th European MPI Users Group Meeting, Stuttgart, Germany, 12–15 September 2010; pp. 229–238.
22. Shameem, A.; Jason, R. *Multi-core Programming: Increasing Performance through Software Multi-Threading*; China Machine Press: Beijing, China, 2007; pp. 12–13.
23. Gun, Z.; Dai, X. The Fork/Join Framework in JDK 7. Available online: http://www.ibm.com/developerworks/cn/java/j-lo-forkjoin/ (accessed on 23 August 2007).
24. Boukerche, A.; Tropper, C. Parallel Simulation on the Hypercube Multiprocessor. *Distrib. Comput.* **1995**, *8*, 181–190. [CrossRef]
25. Fujimoto, R.M. Parallel Discrete Event Simulation. *Commun. ACM* **1989**, *33*, 30–53. [CrossRef]

© 2016 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons by Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).