TELKOMNIKA, Vol.14, No.4, December 2016, pp. 1552~1558
ISSN: 1693-6930, accredited A by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v14i4.3606

An Optimized Model for MapReduce Based on Hadoop

Zhang Hong*[1], Wang Xiao-Ming[2], Cao Jie[3], Ma Yan-Hong[4], Guo Yi-Rong[5], Wang Min[6]
[1,2,5,6]College of Electrical & Information Engineering, Lanzhou University of Technology, Lanzhou730050, China
[1,3]College of Computer & Communication, Lanzhou University of Technology, Lanzhou730050, China
[4]State Grid Gansu Electric Company, Lanzhou730030, China
*Corresponding author, e-mail: zhanghong@lut.cn

Abstract

Aiming at the waste of computing resources resulting from sequential control of running mechanism of MapReduce model on Hadoop platform, Fork/Join framework has been introduced into this model to make full use of CPU resource of each node. From the perspective of fine-grained parallel data processing, combined with Fork/Join framework, a parallel and multi-thread model, this paper optimizes MapReduce model and puts forward a MapReduce+Fork/Join programming model which is a distributed and parallel architecture combined with coarse-grained and fine-grained on Hadoop platform to Support two-tier levels of parallelism architecture both in shared and distributed memory machines. A test is made under the environment of Hadoop cluster composed of four nodes. And the experimental results prove that this model really can improve performance and efficiency of the whole system and it is not only suitable for handling tasks with data intensive but also tasks with computing intensive. it is an effective optimization and improvement to the MapReduce model of big data processing.

Keywords*: Hadoop, MapReduce, Fork/Join, distributed, parallel*

1. Introduction

Hadoop developed by the Apache fund is one of the most popular and stable platforms for large scale data processing in distributed environments, of which the MapReduce, a distributed and parallel programming model, greatly simplifies the design of parallel programming, and the design concept of migrating computation instead of data greatly alleviated the I/O access bottleneck which is a difficult problem of big data processing. Through assigning the massive data to the clusters to parallel processing, it enhances the performance of analyzing big data [1]. The framework also takes care of the low-level parallelization and scheduling details. It is used successfully in several implementations for various scenarios. However MapReduce targets distributed compute cluster not multi-core single node to realize parallelism [2]. This framework does not make full use of the resources of multi-core CPU. For the processing of both IO intensive and CPU intensive, it is powerless. But at present, there are multi-cores in a computer. What's more, in the background of Moore's law, computer hardware has been developed rapidly and the computing power of cluster's single node is also improved. So the resources of each node in the clusters need to be made further mined and used, and effectively using CPU resources to improve whole processing performance of Hadoop platform is the further optimization and improvement to the Hadoop platform. Therefore, it is a hot problem worthy of study to how to make this framework initially designed for low-cost nodes to play its due performance on high performance nodes.

2. Related Work

Up to now, many scholars have done research on optimization for MapReduce/Hadoop itself. These research results mainly can be divided into the following two aspects, which are improvement of MapReduce parallel programming model, optimization of MapReduce scheduling strategy [3]. The typical research results on improvement of MapReduce programming model include barrier-less MapReduce [4], MapReduce-merge [5], Dache [6], MARCO [7]. The typical research results on optimization of MapReduce scheduling strategy

Received March 22, 2016; Revised July 22, 2016; Accepted August 12, 2016

include coupled scheduling [8], shuffle joint scheduling [9], research on localization of data and computing [10], resource aware scheduling [11]. However, these research results are only aimed at the deficiency of MapReduce/Hadoop, and they have not been widely used. Many scholars have done research on MapReduce framework based on third-party technology. For example, R.M.Yoo put forward MapReduce framework Phoneix based on multi-core CPU [12]. W.Fang realized MapReduce framework Mars running on GPU based on NvidiaCUDA [13]. Y.L Zhai studied the collaborative computing between CPU and GPU on Hadoop [14]. These studies are trying to achieve high performance MapReduce framework, but they have no study problems on Hadoop platform nor study the performance of a single node. There are also many researches on hybrid processing models, which combine distribution and multi-core parallelism, such as the hybrid combination of MPI and OpenMP for parallel C programming [15], a two layered parallelism model for CloudHaskell [16].

Fork/Join framework can give full play to the advantages of multi-core computer resources [17]. For the same tasks, it takes less time than multi-thread does [18]. In addition, Fork /Join framework greatly simplifies the programming of parallel procedure and effectively reduces the workload of developers [19]. The most important thing is that java7 begins to incorporate this framework and Hadoop platform is developed in java. So it is feasible and valuable to combine MapReduce architecture with Fork/Join framework to study the optimized programming model of two-tier parallelism. It is a hybrid programming approach which makes full use of advantages of MapReduce and Fork/Join. R. Stewart compared and evaluated the advantages of fork/join and MapReduce framework [20]. But there are few literatures merging these two frameworks into one model.

This paper extends the MapReduce framework and studies the MapReduce+Fork/Join computing model based on Hadoop platform. Through using Fork/Join framework in map and reduce functions, it realizes parallel computing of sharing and distributed memory on each node of Hadoop cloud platform to speed up the computation and improve the performance of processing data. We will analyze the process of MapReduce of Hadoop platform and point out its shortcomings (section 3). In section 4.1, we will analyze the process of Fork/ Join framework. In section 4.2, we will build MapReduce+Fork/Join hybrid model. In section 5, we will do experiment and analyze the results. Section 6 is the conclusions.

3. MapReduce Model of Hadoop Platform

The process of MapReduce is transparent to users. It can be divided into such stages as map stage, combine stage and reduce stage[21], which can be expressed in the following steps.

1. Map: (K1,V1)→list(K2,V2)
2. Combine: list(K2,V2)→(K2,list(V))
3. Reduce: (K2,list(V))→list(K3,V3)

Their input and output are all key-value pairs. Generally, we need to program two functions which are map function in mapper class and reduce function in reducer class. If the input of a map function is (K1, V1) and the output is list (K2, V2), then system shuffle the output and get such output as (K2, list (V)) which is the input of reduce function. If the output of reduce function is list (K3, V3), then list (K3, V3) or its deform may be the final results. The list (K3,V3) can also be the input of another map function and realizes a recursive implementation. Next, we can construct Job and Configuration objects and initialize them, and define the input and output path. And then we can call run Job method in JobClient class to commit this job and so complete a simple running of a distributed task.

When a job is submitted to the system, it is performed with MapReduce model whose process is described in Figure 1. After JobTracker receives the new job, it will pass the job to job scheduler, and initialize it, and create an object to encapsulate operation, status and progress for the job. Job scheduler gets block information of the input data, and creates a map task for each block and assigns an ID number to each map task. JobTracker assigns tasks for each survival TaskTracker which uses a heartbeat to interact with JobTracker. When TaskTracker receives a new task, it will copy relative JAR files from HDFS and duplicate all the files required by application to local disk from distributed caches and new a TaskRunner instance to run the task.

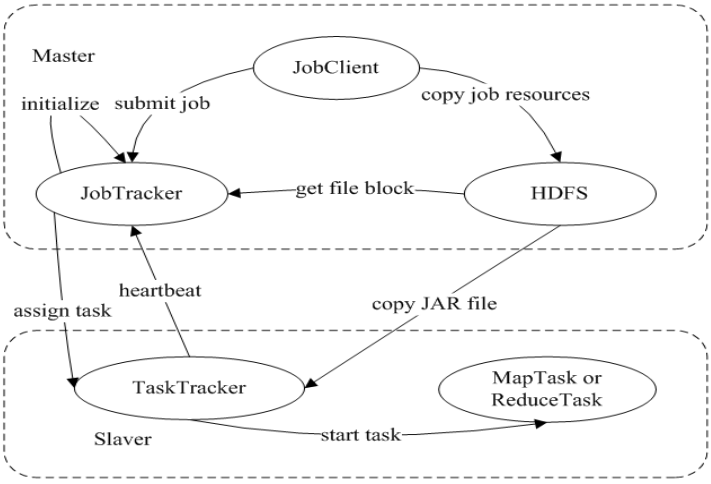

Figure 1. Data processing flow of MapReduce

It can be seen from Figure 1 that JobTracker is mainly used for scheduling task, handling machine fault, managing communications with TaskTracker, monitoring implementation of Job and Task and TaskTracker is used to complete Map and Reduce task and it is the real executor of a task. So the obvious advantage of this model is that it divides the massive data into small blocks and calls a group of distributed computer nodes to perform a job in parallel to improve performance of big data processing. This is a kind of coarse-grained high performance computing. But the computing resources of each node do not be fully used. The Map, Combiner and Reduce are running in sequential order. Combine and Reduce stages must wait for Map stage. And until Map stage completes its task, they can begin works of summary. If the Map task are division uneven or error occurs, Reduce will have been waiting for a long time relatively. This will waste the computing resources. So making full use of resources of each node and introducing Fork/Join framework [22] into it and studying MapReduce+Fork/Join programming model are the purport of this paper, which is a computing model of high performance combined with coarse-grained and fine-grained.

4. MapReduce+Fork/Join Hybrid Model

MapReduce+Fork/Join computing model based on Hadoop platform is described in Figure 2. The whole system is Master-slave structure. The interaction between JobTracker and TaskTracker, task scheduling, the file block and so on are same as the description in Figure 1. The main difference is when slaver nodes perform tasks, Fork/Join framework has been introduced into map or reduce function to make full use of the advantages of shared memory and parallel multi-thread running [23]. That is to say it is these slaver nodes which will use optimized multi-thread to execute tasks and achieve parallel task execution fine-grained in each node [24]. Before a job is submitted to the system, we should set such parameters as path, class name, THRESHOLD and so on according to the real problems. We can choose distributed execution with single thread, only if we set the value of THRESHOLD one. When a JobTracker receives a new job, it will pass the job to job scheduler and assigns tasks for each survival TaskTracker. TaskTracker will copy relative JAR files from HDFS and duplicate all the files required by application to local disk from distributed caches and new a TaskRunner instance to run the task. TaskTracker is the real executor and its execution has the characters of localization. TaskTracker start-ups Fork/Join framework on each computer node and performs parallel tasks with multi-core chips under distributed environment.

4.1. Fork/Join Framework

Fork/Join framework provided in Java7 targets the parallelism on multi-core single computer node. It adopts the divide and conquer algorithmic skeletons and design concept and can dynamically divide a big task into many small tasks called sub tasks for separate executing

in parallel [25], which is shown in Figure 3. At first, Fork operation divides a task into many sub tasks and creates a new branch task. These sub tasks can be divided into smaller sub tasks and performed with multi-thread in parallel. Then Join operation blocks the current tasks and merges the results of sub task execution until obtains the final results. We can set a threshold THRESHOLD to vary the granularity of task execution. The setting of threshold THRESHOLD decides execution time of Fork/Join framework

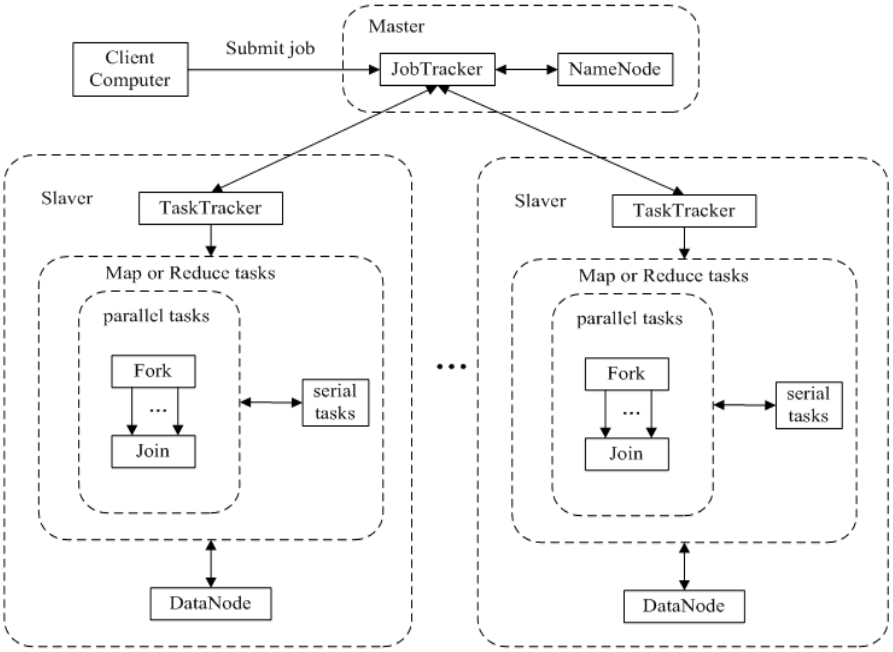

Figure 2. MapReduce+Fork/Join model based on Hadoop

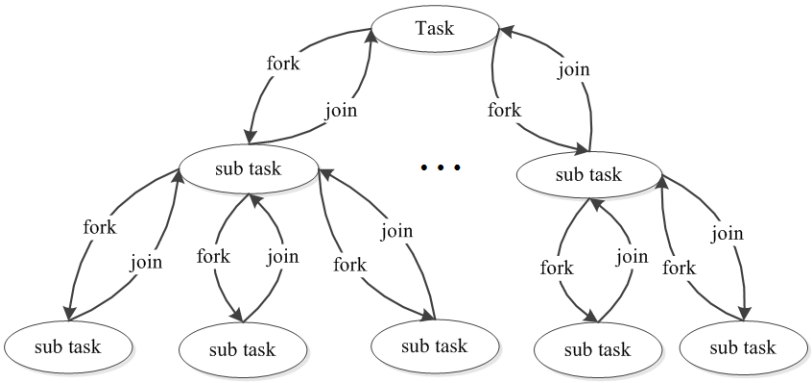

Figure 3. Fork/Join framework

Compared with other parallel architectures, fork/join framework allows inter-task communication for shared-memory systems with multi-core chips. Also this framework uses a algorithm called work stealing, which has been shown to be an effective load balancing strategy for parallel runtime systems [26]. Each worker thread maintains a double-ended task queue. When a worker thread exhausts its local queue or completes itself tasks, it will steals a task from the tail of another thread's queue to perform tasks. By this algorithm, the framework can make full use of multi-thread to perform tasks in parallel and reduce the waiting time of the empty thread, and shorten the time of program executing.

4.2. Implementation Pattern of MapReduce+ Fork/Join

The Fork/Join framework mainly uses two classes to realize parallel computing. These two classes are ForkJoinTask, which is used to make fork and join operation, and ForkJoinPool, which is a threadpool used to storage task queue [27]. When creating a Fork/Join framework and using multi-thread to realize a parallel operating, it needs to new a subclass of RecursiveAction used to define tasks without return values or Recursive Task used to define tasks with return values, both of which are subclass of ForkJoinTask. No matter which classes the new class inherits from, it needs to reconstruct computer method and set the value of THRESHOLD used to set the threshold of sub task?

As mentioned above, MapReduce model based on Hadoop platform needs to complete map and reduce method. So a method implementing MapReduce+Fork/Join model is that when constructing map or reduce function, we create ForkJoinTask task, set parameters of Fork/Join framework and call written computer method. A important problem is to set key-value pairs. In the following experiment, a line of the *.plt file will be looked as the values and its offset value as the key. Thus it can achieve fine-grained and thread level tasks processing in parallel on Hadoop platform based on MapReduce distributed mode.

5. The Experiment

In order to prove the correctness and effectiveness of above described theory, a cluster of four nodes is constructed [28], information of each node is show in Table 1. The hardware specification of each node is Intel (R) Core (TM) i5-3210M CPU @2.50GHZ 2.50 GHZ, 4G memory, Windows 7 Ultimate Edition 64. The virtual infrastructure is VMWare Workstation 10. CentOS 6.0 are the virtual operating system. Hadoop is 1.2.1 version. The Java VM is version 1.7.0-02-b13. We make an experiment to process a taxi GPS data file of 128.7MB(.plt), filtering out the data that the velocity values are greater than 80km/h and less than or equal to 0km/h, then matching the data to a real map [29]. The experimental results show that the MapReduce model takes 1.8 minutes, while MapReduce+Fork/Join model (4 threads which is the number of cores in each node) takes 1.1 minutes. Compared with the MapReduce model, it get a 1.64 speedup.

Table 1. Node information of cluster

Name	IP address	Functions
Master	192.168.91.128	NameNode and JobTracker
Slave1	192.168.91.131	DataNode and TaskTracker
Slave2	192.168.91.132	DataNode and TaskTracker
Slave3	192.168.91.133	DataNode and TaskTracker

In above cluster environment, we make an experiment same as above mentioned with MapReduce+Fork/Join model. As number of threads changes, the running time is shown in Table 2. And the corresponding time variation is shown in Figure 4. While a single thread processed the file, the time consumed is 1.82 minutes which was slightly more than one of MapReduce model. Later, the time consumed was rapidly decreased and then gradually decreased. While the number of threads war four, the time consumed was 1.1 minutes when it got the maximum speedup. Then the time taken was gradually increased. While the number of threads is ten, the time consumed was 1.76 minutes and changed slowly.

Table 2. Number of threads and the corresponding running time

Number of threads	running time (minute)						average time (minute)
	1	2	3	4	5	6	
1	1.811	1.823	1.815	1.832	1.818	1.835	1.82
2	1.521	1.485	1.523	1.511	1.523	1.514	1.51
3	1.311	1.332	1.324	1.341	1.324	1.317	1.32
4	0.971	1.201	1.163	1.154	1.111	0.982	1.1
6	1.412	1.414	1.421	1.425	1.398	1.389	1.41
8	1.644	1.653	1.637	1.635	1.648	1.656	1.65
10	1.771	1.762	1.772	1.761	1.763	1.758	1.76

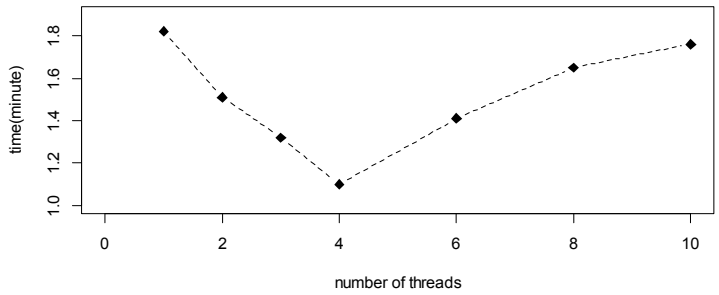

Figure 4. Time variation with the number of threads

Thus it can be concluded that the computing speed of MapReduce+Fork/Join model is more rapid than one of MapReduce model and it is really can improve computational performance although the beginning and final time consumed is almost as same as the original model. But why? The reason may be as follows. At beginning, creating objects, configuring and initializing these parameters take a part of time. When the number of threads is increased to 10, creating threads and communicating among threads take considerable time. How many threads can achieve the highest efficiency on earth? It should be determined according to the file size, the cluster scale and the specific analysis. In above experiment, when number of threads is four, system obtains the highest efficiency.

6. Conclusion

On the basis of analyzing running mechanism and principle of MapReduce model on Hadoop platform, aiming at the waste of computing resources resulting from sequential control of MapReduce computing model, this paper optimizes the model and introduces Fork/Join framework and exploits multi-core parallelism in distributed MapReduce frameworks. A MapReduce+Fork/Join programming model is constructed on the Hadoop platform. This model is a distributed and parallel programming architecture combined with coarse-grained and fine-grained on Hadoop platform, which is not only suitable for handling tasks with data intensive but also tasks with computing intensive because the internal work stealing algorithm on nodes can greatly improve the running efficiency of threads. It has realizes hierarchical structure of distributed and parallel computing on Hadoop. Through the experiment under the cluster environment of four nodes, the results prove that it really can improve the computational efficiency of the whole system and is an effective optimization for MapReduce model on the Hadoop platform. In fact, Hadoop library offers a MultithreadedMapper class extending from the default Mapperclass, which offers thread level parallelism of multi-core chips. But Fork/Join is a more effective and an optimized framework offering load balancing strategy. Because Fork/Join framework hides the lowest details of the parallel implementation, it is easier to be received by the programmer. So this architecture is not only suitable for CPU intensive tasks but also for IO intensive tasks. Next work is to do many environments to test its high performance on large scale clusters.

Acknowledgements

We acknowledge the support from various grant sources: the Natural Science Foundation of Gansu Province (Grant No.148RJZA019), the Scientific and Technological support program Foundation of Gansu Province (Grant No.1304GKCA023).

References

[1] Li Zhang-yong. Optimization and Application Research of MapReduce Computing Model Based on Hadoop. Dissertation. Wuhan: Wuhan University of Science and Technology; 2015.
[2] Sutikno T, Stiawan D, Subroto IMI. Fortifying Big Data infrastructures to Face Security and Privacy Issues. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2014; 12(4): 751-752.

[3] Li Jian-Jiang, Cui Jian, Wang Dan, et al. Survey of MapReduce Parallel Programming Model. *Acta Electronica Sinica.* 2011; 39(11): 2635-2642.
[4] Verma A, Cho B, Zee N, et al. Breaking the MapReduce stage barrier. *Cluster computing*. 2013; 16(1): 191-206.
[5] Yang HC, Dasdan A, Hsiao RL, et al. *Map-reduce-merge:simplified relational data processing on large clusters*. Proceedings of the 2007 ACM Sigmod International Conference on Management of Data. 2007: 1029-1040.
[6] Zhao Ya-xiong, Wu Jie. *Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework*. Proceedings IEEE INFOCOM. 2013; 19(1): 35-39.
[7] Ahmad F, Lee S, Thottethodi M, et al. MapReduce with communication overlap (MARCO). *Journal of Parallel & Distributed Computing*. 2013; 73(5): 608-620.
[8] Tan J, Meng X, Zhang L. *Performance analysis of Coupling Scheduler for MapReduce/Hadoop*. Proceedings IEEE INFOCOM. 2012; 131(5): 2586-2590.
[9] Chen F, Kodialam M, Lakshman TV. *Joint scheduling of processing and Shuffle phases in MapReduce systems*. Proceedings IEEE NFOCOM. 2012; 131(5): 1143-1151.
[10] Tan J, Meng S, Meng X, et al. *Improving ReduceTask data locality for sequential MapReduce jobs*. Proceedings IEEE NFOCOM. 2012: 1627-1635.
[11] Tan J, Meng X, Zhang L. *Coupling Task Progress for MapReduce Resource-Aware Scheduling*. Proceedings IEEE INFOCOM. 2013; 12(11): 1618-1626.
[12] Yoo R, Romano A, Kozyrakis C. *Phoenix Rebirth: Scalable MapReduce on a Large-Scale Shared-Memory System.* IEEE International Symposium on Workload Characterization. 2009: 198:207.
[13] Fang Wei-bin, He Bing-sheng, Luo Qiong, et al. Mars: Accelerating MapReduce with Graphics Processors. *IEEE Transactions on Parallel and Distributed Systems*. 2011; 22(4): 608-620.
[14] Zhai Yan-long, Luo Zhuang, Yang Kai, et al. High Performance Massive Data Computing Framework Based on Hadoop Cluster. *Computer science*. 2013; 40(3): 100-103.
[15] Lusk EL, Chan A. *Early Experiments with the OpenMP/MPI Hybrid Programming Model.* Proceedings of the International Workshop on OpenMP. 2008: 36-47.
[16] Epstein J, Black AP, Jones SLP. *Towards Haskell in the Cloud*. Proceedings of the Haskell Symposium. 2011: 118-129.
[17] Lea D. *A Java fork/join framework*. Acm Conference on Java Grande. 2000: 36-43.
[18] Agrawal K, Leiserson CE, Sukha J. Helper locks for fork-join parallel programming. *ACM Sigplan Notices*. 2011; 45(5): 245-256.
[19] Cao Yang-jie, Qian De-pei, Wu Wei-guo, et al. Adaptive scheduling algorithm based on dynamic core-resource partitions for many-core processor systems*. Journal of Software*. 2012; 23(2): 240-252.
[20] Stewart R, Singer J. Comparing Fork/Join and MapReduce. *Department of Computer Science, Heriot-Watt University*. 2012.
[21] Sandholm T, Lai K. MapReduce optimization using regulated dynamic prioritization. *ACM Sigmetrics Performance Evaluation Review*. 2015; 37(1): 299-310.
[22] Giacaman N, Sinnen O. Object-oriented parallelization of Java desktop programs*. IEEE Software, Software for the Multiprocessor Desktop: Applications, Environments, Platforms*. 2011; 28(1): 32-38.
[23] Senghor A, Konate K. *A Java hybrid compiler for shared memory parallel Programming*. 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2012: 131-136.
[24] Zhang Hong, Wang Xiao-ming, Cao Jie, et al. *Parallel Computing of Shared Memory Multiprocessors Based on JOMP*. International Conference on Mechatronics, Electronic, Industrial and Control Engineering. 2015; 8: 1628-1632.
[25] Zhang Dong-wen, Liu Chen-guang, Zhang Yang. Fork/Join-oriented software refactoring and performance analysis*. Journal of Computer Applications*. 2015; 359(11): 3172-3177.
[26] Wang Lei, Cui Hui-min, Chen Li, et al. Research on task parallel programming model*. Journal of Software*. 2013; 24(1): 77-90.
[27] Zhang Yang, Ji Wei-xing. Scalable method-level parallel library and its improvement*. The Journal of Supercomputing*. 2012; 61(3): 1154-1167.
[28] Purbasari A, *Suwardi IS, Santoso OS, Andala R.* Data Partition and Communication on Parallel Heuristik Model Based on Clonal Selection Algorithm. *TELKOMNIKA*. 2015; 13(1): 193-201.
[29] Amin MS, Reaz MBI, Nasir SS. Integrated Vehicle Accident Detection and Location System. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2014; 12(1): 73-78.