Algorithms **2015**, *8*, 407-414; doi:10.3390/a8030407

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Implementation of a Parallel Algorithm Based on a Spark Cloud Computing Platform

Longhui Wang [1,2], Yong Wang [1,2,*] and Yudong Xie [1,2]

[1] School of Mechanical Engineering, Shandong University, Jinan 250061, China; E-Mails: jaywlh@126.com (L.W.); henryalink@126.com (Y.X.)

[2] Key Laboratory of High-efficiency and Clean Mechanical Manufacture, Shandong University, Ministry of Education, Jinan 250061, China

[*] Author to whom correspondence should be addressed; E-Mail: meywang@sdu.edu.cn; Tel.: +86-531-8839-2539.

Academic Editor: Henning Fernau

Received: 19 March 2015 / Accepted: 23 June 2015 / Published: 3 July 2015

Abstract: Parallel algorithms, such as the ant colony algorithm, take a long time when solving large-scale problems. In this paper, the MAX-MIN Ant System algorithm (MMAS) is parallelized to solve Traveling Salesman Problem (TSP) based on a Spark cloud computing platform. We combine MMAS with Spark MapReduce to execute the path building and the pheromone operation in a distributed computer cluster. To improve the precision of the solution, local optimization strategy 2-opt is adapted in MMAS. The experimental results show that Spark has a very great accelerating effect on the ant colony algorithm when the city scale of TSP or the number of ants is relatively large.

Keywords: cloud computing; MAX-MIN Ant System; TSP; MapReduce; Spark platform

1. Introduction

The ant colony algorithm is a heuristic search algorithm and shows excellent performances on solving various combinatorial optimization, task scheduling and network routing problems [1–3]. However, its time complexity is high and it is seriously time consuming when solving large-scale problems. In order to deal with the problem, many researches are focused on the parallel ant colony algorithm. Up to now, the main parallel ways of applying the ant colony algorithm are achieved by

using multi-core CPUs [4,5]. However, the programming models of these ways are complex and their excellent performances are limited by the number of CPU cores. With the development of cloud computing methods, researchers use Hadoop, which is the first generation of cloud computing platform based on MapReduce framework [6], to accelerate ant colony algorithm [7,8]. In this way, the programming model allows the user to neglect many of the issues associated with distributed computing: splitting up and assigning the input to various systems, scheduling and running computation tasks on the available nodes and coordinating the necessary communication between tasks. Since the intermediate calculation results of Hadoop MapReduce are saved on a hard disk, the tasks take a long time to be started. Therefore, the ant colony algorithm, which needs a lot of iterations, cannot obtain a good speedup. Spark [9] overcomes the drawbacks of Hadoop. The intermediate calculation results of Spark are saved in memory and tasks start fast. The programming model of Spark MapReduce is quite different with Hadoop MapReduce. In this paper, firstly we introduce Spark MapReduce and then combine it with the ant colony algorithm.

In the previous studies, the precision of solutions for TSP is ignored in [7,8]. In order to obtain optimal solutions, the ant colony algorithm needs to be optimized by other algorithms. A composite algorithm is proposed based on Hadoop in [10], which combined simulated annealing (SA). Although the composite algorithm improves the precision of solutions, its speedup is much lower than that of the above two papers. Therefore, it is very important to choose an optimization algorithm which is suitable for the MapReduce framework. We adapted local search strategy 2-opt [11] into ant colony algorithm to speed up convergence rate, in order to improve the efficiency of solving large-scale problems.

2. Principle of MAX-MIN Ant System Algorithm

The ant colony algorithm is an optimization algorithm that was inspired by the behavior of real ants. It is one of the important methods of solving TSP. The basic ant colony algorithm needs a long search time and is prone to premature stagnation. In order to overcome these disadvantages, researchers continue to optimize the basic ant colony algorithm. In 1997, Stützle *et al.* proposed MAX-MIN ant algorithm (MMAS) [12]. There are two main differences between MMAS and the basic ant colony algorithm. The first one is that MMAS only updates pheromone for the shortest path in the current loop, which ensures that the algorithm searches in the vicinity of the shortest path then finds out the global optimal solution gradually. The second one is that the pheromone on each side is controlled in a fixed range (τ_{min}, τ_{max}) to avoid the algorithm converging to the local optimal solution too fast. τ_{min} and τ_{max} are the minimum value and the maximum value of pheromone, respectively.

The main process of using MMAS to solve TSP is as follows:

(1) Initializing ants, cities and tabu list. At first, place m ants randomly in n cities. Pheromone values of every path between cities are initialized to the maximum value τ_{max}. For each ant, there is a tabu list to record the cities which the ant has gone by. The initialization of the tabu list is the city where the ant is situated. The values of pheromone released by ants in each path are initialized to 0.

(2) Constructing the ant paths. An ant moves from city i to j with probability

$$P_{ij} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}(t)]^{\beta}}{\sum_{j \subset allowed_k} [\tau_{ij}(t)]^{\alpha} \times [\eta_{ij}(t)]^{\beta}} & j \subset allowed_k \\ 0 & otherwise \end{cases} \quad (1)$$

where $\tau_{ij}(t)$ is the amount of pheromone deposited for transition from city i to j in time t, α is a parameter to control the influence of $\tau_{ij}(t)$, $\eta_{ij}(t)$ is the desirability of city transition ij (*a prior* knowledge, typically $1/d_{ij}$, where d is the distance) in time t and β is a parameter to control the influence of η_{ij}. τ_{ij} and η_{ij} represent the attractiveness and trail level for the other possible city transitions. $Allowed_k$ is the set of cities which are not in the tabu list. Equation (1) shows that ants will not choose cities in the tabu list to ensure the legitimacy of solutions.

(3) Operating the pheromone. In iteration, one ant is to complete a traversal of all cities. When all ants tabu lists have been filled, path constructions are finished. Compute the length of the path that each ant has passed by. Compare all paths to find the shortest one, and then save it, which is denoted as C^{best}. The pheromone is updated by

$$\begin{cases} \tau_{tj} = (1-\rho)\tau_{ij} + \Delta\tau_{ij} \; if \; ij \; is \; in \; C^{best} \\ 0 \; otherwise \end{cases} \quad (2)$$

where ρ is the pheromone evaporation coefficient and $\Delta\tau_{ij} = 1/C^{best}$. The minimum value and the maximum value of pheromone are respectively τ_{min}, τ_{max}. Go to step 2 after the pheromone operation until meeting the termination condition of the algorithm: result converging or reaching the maximum number of iterations.

3. Spark MapReduce Overview

3.1. The MapReduce Framework

In April 2004, Google researchers found that most distributed computing can be abstracted as a MapReduce operation. A MapReduce program is composed of a Map procedure that performs filtering and sorting and a Reduce procedure that performs a summary operation. MapReduce can be used for processing massive data and parallel algorithms with high time complexity.

The operation process of the MapReduce framework is shown in Figure 1. In the stage of Map, the input is split into a plurality of parallel parts and Map tasks are generated. Then they are assigned to each computing node and processed in the same way. In the stage of Reduce, Reduce Tasks merge the results calculated by Map Tasks, and then output final results.

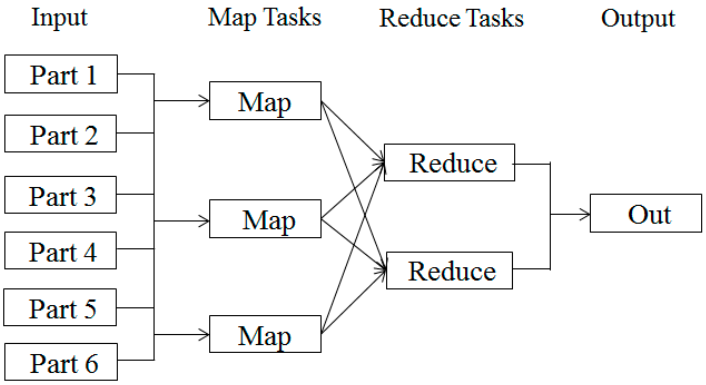

Figure 1. The operation process of MapReduce framework.

MapReduce is one of the basic Could Computing technologies of Google. Programs based on MapReduce framework can be run on thousands of common computers in a parallel way. Programmers

can take use of the power of a computer cluster even though they don't have any distributed system design experience.

3.2. Spark Platform

Spark platform is developed in the UC Berkeley AMP lab. Spark is a fast and general-purpose cluster computing system based on MapReduce framework. Spark uses Scala language for development, which is concise and efficient.

The fundamental programming abstraction of Spark is called Resilient Distributed Datasets (RDD), a logical collection of data partitioned across machines. By allowing user programs to load data into a cluster's memory and to query it repeatedly, Spark is well suited to iterative algorithms, such as ant colony algorithm.

The working principle of Spark platform is shown in Figure 2. Spark divides a computer cluster into a master node and multiple worker nodes. The master node is responsible for task scheduling, resource allocation and error management. Worker nodes process Map tasks and Reduce tasks in parallel. User program distribution and input data splitting are automatically completed by the platform.

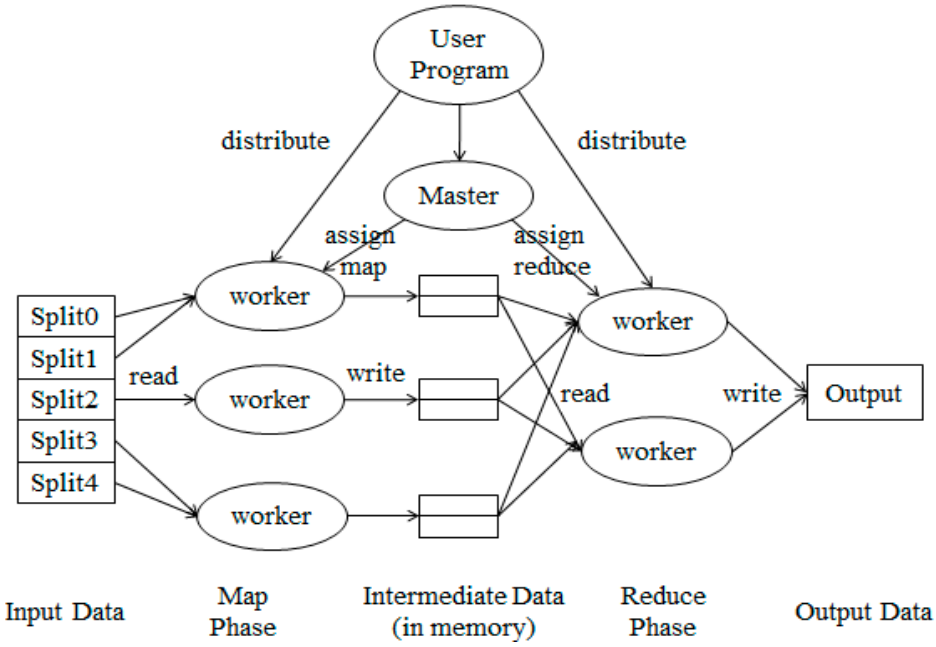

Figure 2. The working principle of Spark.

4. Implementation of MMAS Based on Spark Platform

The ant colony algorithm is essentially a parallel algorithm. In every iteration, the ant colony algorithm can be divided into two stages. In the first stage, each ant iterates through all cities independently. Calculate the walking distance and save the current path. In the second stage, compare all the walking distances of ants to get the shortest one and then operate the pheromone. These two stages can be realized respectively by Map and Reduce, so the MapReduce framework is suitable for the ant colony algorithm. The implementation of MMAS based on MapReduce framework and Spark platform is as follows:

(1) Data parallelism. At first, initialize the city distance matrix and the pheromone matrix. Let the number of ants be *m* and establish an array of size *m*. Use *parallelize*() of Spark to convert the array to a distributed dataset, then use this dataset as the input of *map*() to start *m* Map tasks. The array that has no effect on results is only used for starting Map tasks, and the elements in it can be any value.
(2) Map Tasks. Because all Map tasks will use the same algorithm, we only need to complete the process that one ant constructs a path. Construct path according to MMAS and use the 2-opt algorithm for optimization. Then output the walking distance and path of each ant. The calculations of the algorithm are mainly concentrated in Map stage. The outputs of *map*() are terms of key-value pairs, in which the distance is key and the path is value. Sort the distances using *sortByKey*().
(3) Reduce Tasks. Use *take*(1) to find the shortest distance and its corresponding path.
(4) Iteration: Update the pheromone matrix according to the results of Reduce tasks, and then save the current minimum distance and optimal path. Iterate Map tasks and Reduce tasks to obtain the final result.

5. Experiments

The experimental hardware platform is four Linux servers with a total of 64 CPU cores and 128 G memory. Use five standard TSP instances in TSPLIB [13]. Compared with common serial programs, resource scheduling and task allocation of Spark platform consume extra time, so we first compare common program (single core) with the 2-core Spark platform on run time. The experimental parameters and the results are shown in Table 1. Column "opt" represents the length of the known optimal solution of every instance. Columns "MMAS" and "MMAS+2-Opt" represent the best results for every instance achieved by the two algorithms, respectively.

Table 1. Comparison between single core program and Spark platform.

Instance	Opt	MMAS	MMAS + 2-Opt	Number of ants	Iterations	Spark 2-core/min	Single core/min
att48	10,628	10,628	10,628	48	1000	1.1	0.5
st70	675	677	675	96	1000	2.7	2.6
eil101	629	635	629	128	1000	6.5	8.5
rat195	2323	2369	2323	192	1000	45.8	70.2
pr299	48,191	48,287	48,191	320	1000	221.3	396

From Table 1 we can see that MMAS combined 2-Opt based on MapReduce can obtain optimal solutions. The running time of the 2-core Spark platform is longer than common program when city scales are small. As the city scale increases, the acceleration effect becomes greater. So the Spark platform is suitable for large calculations, such as rat195, pr299.

Because the number of ants has a great influence on the solution accuracy, we choose the eil101 instance and change the number of ants to compare common program with 2-core Spark platform. The experimental results are shown in Figure 3.

When the number of ants increases by 64, the time consumptions of common programs increase by 4 min and these of the Spark platform are 2 min. Spark platform starts tasks so fast that adding Map tasks will not bring extra time consumption.

Then we use pr299 and increase the number of CPU cores to test the speedup of the Spark platform. If the number of CPU cores is less than 64, cores are evenly divided among four servers. For example, when we test four cores, the program is executed on four servers (every server uses one core) and cores are LAN-connected. The time consumptions of different CPU cores are shown in Table 2.

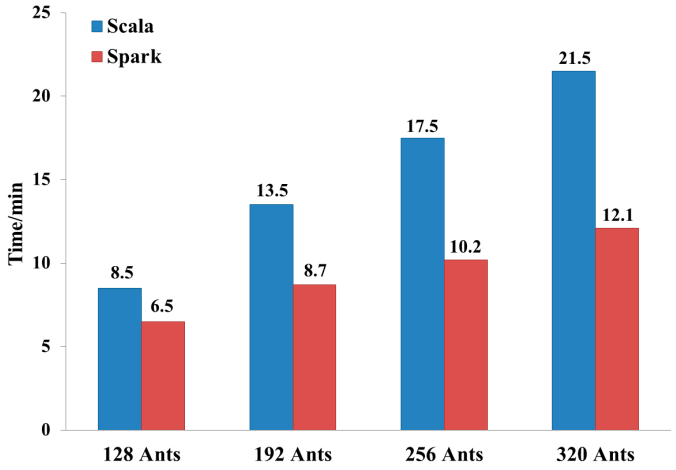

Figure 3. Time consumption of different number of ants.

Table 2. Time consumption of different number of Cores.

Cores	1	2	4	8	16	32	64
Time/min	396	221	112.6	57.4	28.7	17	12.3

For the parallel computing system, the main performance index is speedup. The comparison between Spark and Hadoop is shown in Figure 4.

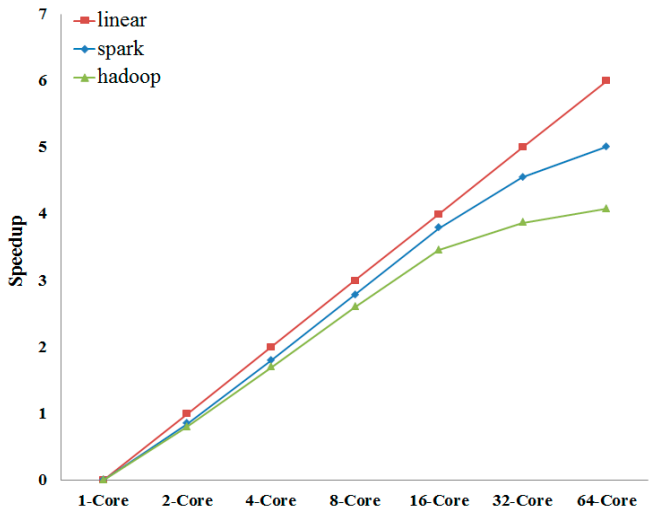

Figure 4. Comparison between the speedups of Spark and Hadoop. In order to make the linear speedup to be a straight line, we let $y = log_2(T_1/T_n)$. T_1 is the running time of common program (single core).

The experimental results show that Spark platform has a very noticeable accelerating effect on ant colony algorithm and performs better than Hadoop. When the CPU core number is 1–16, the speedup is close to the linear speedup and then decreases. By observing the CPU load we find that CPU runs in full load when the CPU core number is 1–16, and CPU load decreases while the CPU core number increases to 32 or 64, which means that the amount of calculation is trivial for the Spark platform. So the parallel ant colony algorithm based on the Spark platform is more suitable for handling large-scale problems.

6. Conclusions

This paper presents the implementation scheme of parallel ant colony algorithm based on the Spark platform. Using the Spark MapReduce framework can improve the efficiency of the ant colony algorithm. At present, cloud computing has been widely used in big data processing while it its potential for speeding up parallel algorithms needs to be developed. Not only the ant colony algorithm but also other parallel algorithms, such as neural network, logistic regression or particle swarm algorithm, can also be sped up by using the Spark platform.

Acknowledgments

This research is supported by Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20110131110042), and National Natural Science Foundation of China (grant No. 51305234).

Author Contributions

The idea for this research work is proposed by Yong Wang. The Scala code is achieved by Longhui Wang, and the paper writing is completed by Longhui Wang.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Ellabib, I.; Calamai, P.; Basir, O. Exchange Strategies for Multiple Ant Colony System. *Inf. Sci.* **2007**, *177*, 1248–1264.
2. Middendorf, M.; Reischle, F.; Schmech, H. Multi Colony Ant Algorithms. *Heuristics* **2002**, *8*, 305–320.
3. Origo, M.; Bianiezzo, V.; Colorni, A. Ant system: Optimization by a colony of coorperating agents. *IEEE Trans. Systems Man Cybern.* **1996**, *26*, 8–41.
4. Su, S.Q.; Liang, S.Z. Parallel Study of Ant Colony Alogorithm. *Jisuanji Yu Xiandaihua* **2009**, *10*, 18–21. (In Chinese)
5. Randalla, M.; Lewisb, A. A parallel implementation of ant colony optimization. *J. Parallel Distrib. Comput.* **2002**, *62*, 1421–1432.

6. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2004**, *51*, 147–152.
7. Zhang, J.; Liu, Y.; Lu, F.L. Parallel Research and Implementation of Ant Colony Algorithm to Solve Problem of TSP. *Comput. Technol. Dev.* **2011**, *21*, 72–75.
8. Tan, Q.; He, Q.; Shi, Z.Z. Parallel Max-Min Ant System Using MapReduce. *Adv. Swarm Intell.* **2012**, *7331*, 182–189.
9. Zaharia, A.M. An Architecture for Fast and General Data Processing on Large Clusters. Ph.D. Thesis, the University of California, Berkeley, CA, USA, 2013.
10. Yang, Q.F.; Mei, D.; Han, Z.-B.; Zhang, B. Ant colony optimization for the shortest path of urban road network based on cloud computing. *J. Jilin Univ. Eng. Technol. Ed.* **2013**, *43*, 1210–1214.
11. Xu, S.H.; Wang, Y.; Huang, A.Q. Application of Imperialist Competitive Algorithm on Solving the Traveling Salesman Problem. *Algorithms* **2014**, *7*, 229–242.
12. Stützle, T.; Hoos, H. MAX-MIN Ant System and local search for the traveling salesman problem. In Proceedings of the IEEE International Conference on Evolutionary Computation, Indianapolis, IN, USA, 13–16 April 1997; pp. 309–314.
13. TSPLIB. Available online: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ (accessed on 6 August 2008).