TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 04/10 pp550–560
DOI: 10.26599/TST.2018.9010022
Volume 23, Number 5, October 2018

A Scheduling Optimization Technique Based on Reuse in Spark to Defend Against APT Attack

Jianchao Tang, Ming Xu*, Shaojing Fu, and Kai Huang

Abstract: Advanced Persistent Threat (APT) attack, an attack option in recent years, poses serious threats to the security of governments and enterprises data due to its advanced and persistent attacking characteristics. To address this issue, a security policy of big data analysis has been proposed based on the analysis of log data of servers and terminals in Spark. However, in practical applications, Spark cannot suitably analyze very huge amounts of log data. To address this problem, we propose a scheduling optimization technique based on the reuse of datasets to improve Spark performance. In this technique, we define and formulate the reuse degree of Directed Acyclic Graphs (DAGs) in Spark based on Resilient Distributed Datasets (RDDs). Then, we define a global optimization function to obtain the optimal DAG sequence, that is, the sequence with the least execution time. To implement the global optimization function, we further propose a novel cost optimization algorithm based on the traditional Genetic Algorithm (GA). Our experiments demonstrate that this scheduling optimization technique in Spark can greatly decrease the time overhead of analyzing log data for detecting APT attacks.

Key words: Spark; Advanced Persistent Threat (APT); schedule; reuse; Resilient Distributed Dataset (RDD); Directed Acyclic Graph (DAG); Genetic Algorithm (GA)

1 Introduction

Advanced Persistent Threat (APT) attack is an attack pattern whereby attackers adopt advanced attack means to make long-term and persistent network attacks on a particular target[1]. Generally, the attackers in an APT attack have a specific target. In the first step, they invade into the target network by various means of social engineering. Then, they continuously lurk in the target network to gather intelligence rather than immediately launch attacks. This stage can last for a year or several years until an attack network is built. Then, attackers launch a final attack to unnoticeably steal confidential documents of governments and enterprises or to cause serious damage to important infrastructures.

Advanced persistent threat attacks have several characteristics which make them difficult to detect[2]: (1) They are advanced: Typically, the attackers actively invade into the target network by social engineering or zero-day vulnerabilities. Meanwhile, they adopt a wide range of existing attack techniques to build the subsequent attack network. (2) They are hidden: The attackers usually lurk in the target network for a long time without being discovered. (3) They are persistent: The attackers continuously lurk in the target network to gather intelligence in a long term and wait for the final attack. Moreover, APT attacks are extremely devastating since they can result in

• Jianchao Tang, Ming Xu, and Kai Huang are with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: mymailtjc@163.com; xuming@nudt.edu.cn; kai.huang@nudt.edu.cn.
• Shaojing Fu is with the College of Computer, National University of Defense Technology, Changsha 410073, and Sate Key Laboratory of Cryptology, Beijing 100878, China. E-mail: shaojing1984@163.com.
∗ To whom correspondence should be addressed.
Manuscript received: 2017-09-22; accepted: 2017-09-29

the theft of confidential documents of governments and enterprises and the destruction of country's important infrastructures[1–4]. For example, the events of Operation Aurora of Google in 2009[5] and the Stuxnet attack of Iran's nuclear centrifuges in 2010[6].

To address the above issues, four types of security policies against APT attacks have been proposed[2, 3]: (1) Host file protection: In this security policy, the execution of applications is strictly controlled by creating a white list to prevent the execution of malicious codes. (2) Malicious code detection: In this security policy, the propagation of malicious code is examined repeatedly. (3) Network intrusion detection: The security policy involves analyzing and detecting the command and control channels of APT attacks by deploying network intrusion detection systems to network borders. (4) Big data analysis: The security policy is a kind of network forensics. First, the original flow of network equipment and the logs of terminals and servers are comprehensively collected. Then, the data is stored in a centralized way and subjected to deep analysis. When any clue of APT attack is detected, the whole attack scenarios are reconstructed as early as possible. This security policy is the most effective policy to defend against APT attacks because its analysis covers the whole attack process. Therefore, we adopt the big data analysis policy to defend against APT attacks.

For the big data analysis policy, a big data processing framework is necessary to analyze massive amounts of log data. As a distributed data processing framework based on memory, Spark[7–11] has characteristics of high efficiency, high fault tolerance, and high scalability, and it is suitable framework for data analysis. Therefore, Spark is selected as the log processing framework in this study and the system model is as shown in Fig. 1. The log data of terminals and servers are uploaded to the Spark system. Then, Spark analyzes the input data and sends the results to the output panels for decision-makers to make the next decision.

However, in application scenarios of multiple servers and multiple terminals, Spark cannot meet the performance requirements. In fact, the calculation results of some Spark applications can be reused by other applications to reduce the execution time of these applications[12]. Considering that Spark is a distributed data processing framework, there are many reusable calculation results when huge log data is analyzed. If these calculation results can be effectively reused, it would tremendously improve Spark performance. However, for the existing Spark mechanism, applications cannot reuse the same calculation results.

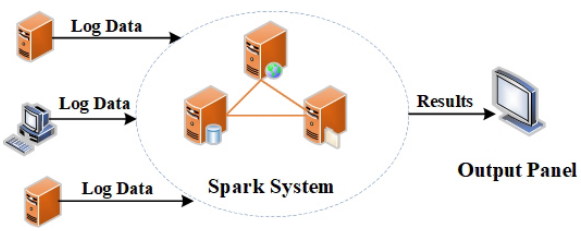


Fig. 1 Big data analysis detection model.

Reuse techniques[13–15] are traditionally used as database optimization methods, which involve reuse of the same query results. Therefore, in this study, a method of combining Spark with reuse techniques is investigated to improve Spark performance. To achieve the maximum reuse between multiple applications, a scheduling optimization technique based on reuse is proposed and implemented. Besides, some experiments are conducted to prove its feasibility. The innovations of this optimization technique are listed as follows:

(1) The concept of the reuse degree of Spark applications is defined and a formula to calculate it is proposed.

(2) Two novel concepts which are the relative location and the redundancy operation based on the reuse degree are defined. To obtain the optimal application sequence, a global optimization function based on the two concepts is proposed.

(3) To obtain the global optimal solution, a cost optimization algorithm is designed and implemented based on the traditional genetic algorithm.

The remaining parts of this paper are organized as follows. The related works are introduced in Section 2. In Section 3, we define and formulate the reuse degree. The global optimization function is introduced in Section 4. In Section 5, we propose and implement the cost optimization algorithm based on Genetic Algorithm (GA). In Section 6, we conduct experiments to prove the feasibility of our optimization technique, and in Section 7, we present our conclusion about the study.

2 Related Work

2.1 Data structure of Spark

Spark has a unique data structure: Resilient Distributed Dataset (RDD)[16], which is a kind of read-only data structure. The read-only property is embodied so that operating an RDD will generate a new RDD rather than change the contents of original RDD. The new

and original RDDs form a dependent relationship: the original RDD is named father RDD, the new one is named son RDD. Each RDD records its dependency with other RDDs in its dependency list. From the first RDD to the last RDD, all RDDs form a Directed Acyclic Graph (DAG)[17] that can be directly transformed from Spark applications. Therefore, we can analyze DAGs instead of Spark applications to achieve the same purpose. Figure 2 shows a DAG with several different RDDs.

In Fig. 2, RDDs are denoted as uppercase alphabets from A to G. The different RDD operations are denoted by the words such as map, union, etc. RDD operation refers to a function that acts on an RDD to generate a new RDD. These operations can be divided into transformation operation and action operation. Here, transformation operation is an inert operation, which means the transformation operation between two RDDs

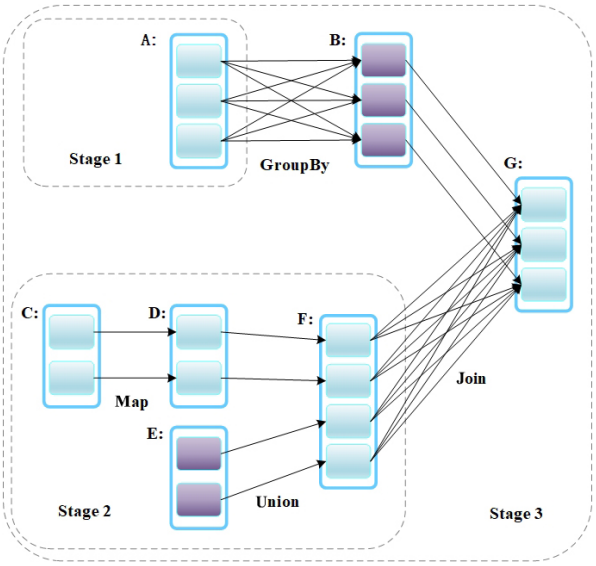


Fig. 2 A DAG with several RDDs.

is not executed immediately, but waits to be triggered by an action operation. In addition, action operation triggers the submission of Spark jobs. Table 1 shows common RDD operations.

2.2 Reuse mechanism of Spark

Reuse refers to an effective utilization of the original information. In the fields of software development and databases, reuse techniques are widely adopted, which can greatly reduce the overhead of data processing by avoiding data to be repeatedly uploaded, built or calculated. In most databases, reuse is generally implemented by caching mechanisms, such as data blocks, pages, tables, tuples, and semantic cache[14, 18].

Until now, the reuse of RDDs in a single application is the only reuse technique in Spark. To store RDDs for resue, users need to perform manual programming by using the built-in cache function or persist function in Spark. Then, users can retrieve RDDs from the cache to reuse them by utilizing unpersist function. However, in real cases, it is difficult for users to decide when to save RDDs for resue and how to organize, manage and utilize these reusable RDDs. These characteristics restricts reuse in Spark.

2.3 Scheduling mechanism of Spark

In distributed clusters, cluster resources are not usually enough to use. Thus, the scheduling problem[19] still exists. Common scheduling algorithms in distributed clusters are First-In First-Out (FIFO)[20], round-robin scheduling[21], priority scheduling[22], and fairness scheduling[23].

Until now, the scheduling mechanism of Spark can be divided into two levels: The first level is scheduling between different applications and the second is scheduling within an application. They adopt different scheduling algorithms. The FIFO scheduling algorithm

Table 1 The common RDDs operations.

Transformation	map($f : T => U$) : RDD[T] => RDD[U]
	filter($f : T => Bool$) : RDD[T] => RDD[U]
	flatMap($f : T => Seq[U]$) : RDD[T] => RDD[U]
	groupByKey() : RDD[(K, V)] => RDD[$(K, Seq(V))$]
	reduceByKey($f : (V, V) => V$) : RDD[(K, V)] => RDD[(K, V)]
	union() : (RDD[T], RDD[T]) => RDD[T]
	join() : (RDD[(K, V)], RDD[(K, W)]) => RDD[$K, (V, W)$]
Action	$count()$: RDD[T] => $Long$
	$collect()$: RDD[T] => $Seq[T]$
	$reduce(f : (T, T) => T)$: RDD[T] => T
	$save(path : String)$: $Outputs$ RDD to $storage$ $system$, e.g., $HDFS$;

is adopted in the first level: Applications are placed into an application queue according to their submission orders. Then, the cluster resource scheduler tests the cluster resources. If the current cluster resources can meet the need of the first application in queue, this application will obtain the corresponding resources and be immediately executed; if not, the resource scheduler will continue to test other applications behind the first one in the queue.

The second level in Spark schedules different jobs decomposed from an application by constructing scheduling pools which are like containers. A scheduling pool has three main parameters to implement its scheduling:

(1) Scheduling mode: This parameter supports FIFO scheduling algorithm and FAIR scheduling algorithm. Here, FAIR algorithm adopts round-robin to guarantee every job fairly obtains cluster resources.

(2) Weight: This parameter reflects that one scheduling pool owns the amount of the cluster resources compared with other scheduling pools. Its default value is 1, which means that all the scheduling pools obtain the same amount of cluster resources. Users can set this parameter in the configuration file.

(3) MinShare: This parameter reflects that every scheduling pool can obtain the amount of minimum resources. In FAIR scheduling algorithm, every scheduling pool must obtain the minimum resources according to the minShare value.

2.4 Optimization problems and optimization algorithms

In practical applications, we often need to find the maximum or minimum value of a function from all feasible values; this problem is called an optimization problem. Common optimization problems include solving the objective function to obtain the maximum or minimum value in a linear programming problem[24] and solving the shortest travel distance in a traveling salesman problem[25].

Optimization problems can be solved by a certain algorithm, called the optimization algorithm. The common optimization algorithms include hill climbing algorithm[26], simulated annealing algorithm[27], and GA[28]. Here, GA simulates biological evolution. After several iterations, GA can usually obtain the optimal solutions of problems. In each iteration, the best individuals are selected from parent population to form a progeny population by GA selection function, which guarantees that the progeny population is superior to the parent population. In addition, some individuals in the progeny population are forced to cross and mutate to produce some new individuals. There, GA can jump out of the local optimal solution and get the global optimal solution in GA. Compared to hill climbing algorithm and simulated annealing algorithm, GA usually has better solutions for optimization problems.

3 Reuse Degree

From this section, we discuss the proposed scheduling optimization technique based on the reuse of DAGs in Spark, and we begin by defining the reuse degree, which is the first part of the technique.

In Section 2, we mention that a DAG composed of many RDDs can be transformed from a Spark application. From Fig. 3, we can find two different DAGs and each DAG has an RDDC. If the RDDC generated in Fig. 3a is stored into the cache in advance, we would only need to directly read the RDDC results from the cache when the DAG in Fig. 3b is executed. This is the reuse technique investigated in this study.

3.1 Definition of reuse degrees

The execution of RDDs transformation operations requires time. We define this execution time as the execution cost of these operations. The formula for

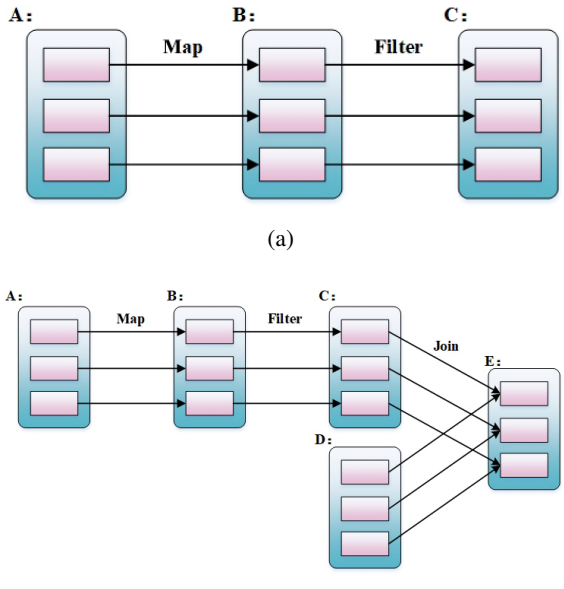


Fig. 3 Two different DAGs.

calculating the execution cost is shown as Eq. (1).

$$OP_i = S_{di} \times W_{ti} \tag{1}$$

Here, OP_i denotes the execution cost of the i-th transformation operation of RDDs, S_{di} denotes the input data size of the i-th transformation operation, and W_{ti} denotes the relative weight of the i-th transformation operation compared with the map operation, which we view as the basic transformation operation. To calculate the relative weight of the i-th transformation operation $operation_i$, we propose a simple algorithm as shown in Algorithm 1.

In Algorithm 1, the size of input data remains unchanged to calculate the execution time T_i, and the algorithm is run on the same hardware configuration. We test the T_i severally and calculate the average, denoted as $\overline{T}_i$. Then, we can obtain the relative weight of transformation operations by comparing the average execution times. The formula to calculate the relative weight is shown as Eq. (2).

$$W_{ti} : W_{tj} : \cdots : W_{tk} = \overline{T}_i : \overline{T}_j : \cdots : \overline{T}_k \tag{2}$$

Here, W_{ti}, W_{tj}, and W_{tk} denote the relative weight of the i-th, j-th, and k-th transformation operations, respectively. Likewise, $\overline{T}_i$, $\overline{T}_j$, and $\overline{T}_k$ denote the average execution time of the i-th, j-th, and k-th transformation operations, respectively. Table 2 shows the relative weight of some common RDDs transformation operations (approximate ratio).

Algorithm 1 The algorithm to the execution time of *operation*$_i$

1: *rdd1* <- *sc.textFile(path1)*
2: *rdd2* <- *rdd1.saveAsTextFile(path2)*
3: T_1 <- $T_{rdd1} + T_{rdd2}$
4: *rdd1* <- *sc.textFile(path1)*
5: *rdd3* <- *rdd1.operation*$_i$
6: *rdd2* <- *rdd3.saveAsTextFile(path2)*
7: T_2 <- $T_{rdd1} + T_{operation_i} + T_{rdd2}$
8: T_i <- $T_2 - T_1$

Table 2 The relative weight of different operations.

Operation	Relative weight
Map	2
Fliter	2
FlatMap	3
Union	4
Distinct	4
Join	5
ReduceByKey	6
GroupByKey	7

When the relative weight of transformation operations is obtained, we can calculate their execution cost by Eq. (1). The reuse degree is defined based on the execution cost: the sum of execution cost of reusable RDDs between two different DAGs. For example, the reuse degree of DAGs in Fig. 3 is the sum of the execution cost of map operation and filter operation.

3.2 Calculation of reuse degree

Once the reuse degree is defined, the formula for calculating it can be derived from its definition. For arbitrary two DAG$_i$ and DAG$_j$, the formula for calculating the reuse degree between them is shown as Eq. (3).

$$C_r = \sum_{i=1}^{n} C_i \tag{3}$$

Here, C_r denotes the size of the reuse degree between the two DAGs, and C_i denotes the execution cost of the i-th reusable RDD of the two DAGs, which can be calculated as Eq. (4).

$$C_i = \sum_{i=1}^{n} OP_i \times n_i \tag{4}$$

Here, OP_i can be calculated by Eq. (1), and n_i denotes the number of the i-th transformation operation appearancing in the i-th RDD. For example, for the two DAGs in Fig. 3, assuming the size of input data is 10 MB, referring to Table 2, the size of the reuse degree between the two DAGs is $C_r = OP_{map} + OP_{filter} = 2 \times 10 + 2 \times 10 = 40$.

However, our goal is to obtain the reuse degree of a DAG sequence. For a DAG sequence, the total size of reuse degree can be calculated by the formula shown as Eq. (5).

$$C_{ra} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} C_{rij} \tag{5}$$

Here, C_{rij} denotes the size of reuse degree between DAG$_i$ and DAG$_j$, which can be calculated by Eq. (3), n denotes the number of DAGs in the DAG sequence, and C_{ra} denotes the total size of reuse degree of the DAG sequence. Our goal is to maximize the C_{ra}.

4 Global Optimization Function

The global optimization function is the second part of the scheduling optimization technique, which is designed to obtain the maximized C_{ra}. To achieve this goal, we propose two novel concepts: relative location and redundant operation.

4.1 Relative location

To reuse different DAGs, the reusable calculation results must first be stored into the reuse warehouse of Spark. Obviously, this warehouse cannot be infinite. The warehouse may not have enough capacity to store reusable calculation results when very many DAGs are processed simultaneously. In addition, the cached results in the reuse warehouse have to be replaced by some replacement strategies.

For two DAGs with reusable parts, the reuse results are different for different locations in the DAG sequence: If they are close to each other, the reuse probability is relatively higher. Otherwise, it is much likely that the reusable parts in the reuse warehouse have been replaced by some other calculation results when the second DAG is executed, which means the probability of reusing the two DAGs is much smaller. Therefore, to realize reuse, the location of DAGs in a DAG sequence must be considered.

We propose the concept of relative location to denote the distance between two DAGs in a DAG sequence. The formula for calculating the relative location is shown as Eq. (6).

$$L_{ij} = |j - i| \tag{6}$$

Here, i denotes the location subscript of DAG_i in the DAG sequence, j denotes the location subscript of DAG_j in the DAG sequence, and L_{ij} denotes the relative location of the two DAGs. For two DAGs, the larger the size of their reuse degrees, the smaller their relative location. We define the size of relative location to reflect this trait. The formula for calculating the size of relative location is shown as Eq. (7).

$$C_{lij} = C_{lij} / L_{ij} \tag{7}$$

Here, C_{lij} denotes the size of the relative location between DAG_i and DAG_j; C_{rij} and L_{ij} denote the size of reuse degree in Eq. (3) and relative location in Eq. (6) between DAG_i and DAG_j, respectively. From Eq. (7), we can find that the size of reuse degree and the relative location between two DAGs are both considered in the size of relative location. When C_{rij} is large and L_{ij} is small, then, C_{lij} will be large. This suggests the two DAGs should be reused, which is reasonable.

4.2 Redundant operation

As Fig. 4 shows, the two different DAG sequences have the same DAGs but different DAG orders. In Fig. 4a, the second DAG can reuse the calculated RDDB results of the first DAG, and the third DAG can reuse the RDDC results of the first DAG. The reuse process is as follows: The first DAG calculates RDDB and RDDC and caches the two results into the reuse warehouse of Spark. The second and third DAGs directly read the calculated RDDB and RDDC results from the reuse warehouse, respectively.

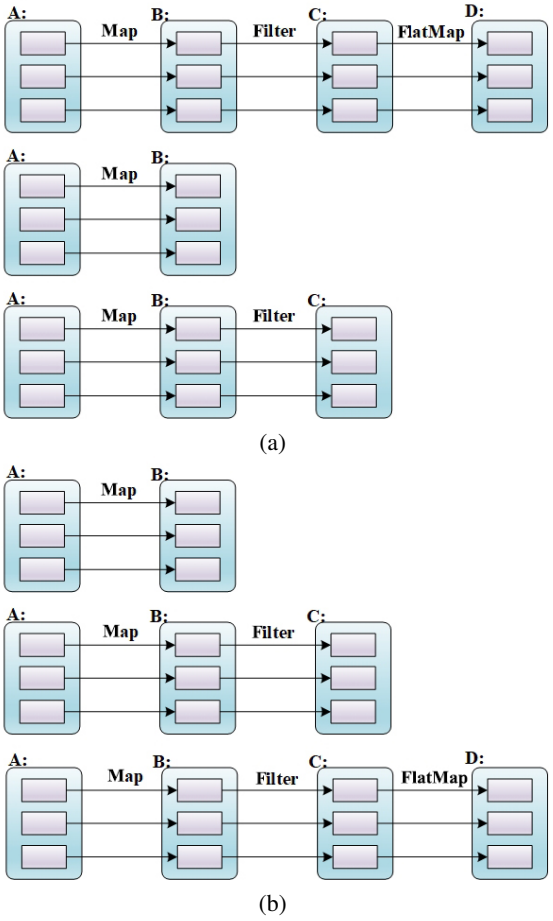


Fig. 4 Two different DAG sequences.

In Fig. 4b, the situation is different. The second DAG reuses the RDDB result in the first DAG, so the first DAG needs to first calculate the RDDB and then cache the result into the reuse warehouse. Meanwhile, the third DAG reuses the RDDC result in the second DAG. However, the first DAG and the second DAG belong to different applications, and the second DAG has to calculate RDDC from RDDA, which will consume much execution time. Therefore, compared with the DAG sequence in Fig. 4a, the DAG sequence in Fig. 4b needs more execution time.

To describe the above phenomenon, we propose the concept of redundant operation, whereby DAG sequence has different DAG orders. To realize the reuse of DAGs, some orders of the DAG sequence need some extra RDD operations; these operations are called

redundant operations. For example, relative to Fig. 4a, the redundant operation in Fig. 4b is the map operation between RDDA and RDDB. We define redundant operation gain based on redundant operation. The redundant operation gain refers to the reduced execution time of some DAG orders of a DAG sequence because the execution of redundant operations is avoided. The formula for calculating redundant operation gain for some parts of a DAG sequence is shown as Eq. (8).

$$C_s = \sum_{i=1}^{n} OP_i \tag{8}$$

Here, OP_i denotes the execution cost of the i-th redundant operations, n denotes the number of redundant operations in this part, and C_s denotes the redundant operation gain for some parts of the DAG sequence. The larger the redundant operation gain of a DAG sequence is, the more its execution time is reduced. The redundant operation gain of the total DAG sequence is shown as Eq. (9).

$$C_{sa} = \sum_{i=1}^{m} C_{si} \tag{9}$$

Here, C_{si} denotes the redundant operation gain of the i-th part of DAG sequence, and C_{sa} denotes the redundant operation gain of the total DAG sequence.

4.3 Design of the global optimization function

The global optimization function is designed based on the relative location and redundant operation. The input of the function is the reuse degree information of a DAG sequence. First, the size of relative location and redundant operation gain of the DAG sequence are calculated by this function according to the reuse degree information. Then, the function seeks out the optimal DAG sequence with the largest size of relative position and redundant operation gain. Finally, the function outputs the optimal DAG sequence. The core part of the global optimization function is to calculate the optimal DAG sequence, which has the least execution time. The implementation of the global optimization function is discussed in next section.

5 Cost Optimization Algorithm Based on GA

To implement the global optimization function, we propose a GA-based cost optimization algorithm, which is the third part of the scheduling optimization technique. GA is chosen as the basic algorithm in this paper for the following reasons:

(1) Compared with hill climbing algorithm, simulated annealing algorithm and other common iterative algorithms, GA can overcome the disadvantage of falling easily into local optimal solutions and obtain the global optimal solution.

(2) Compared with traditional enumerated and heuristic optimization algorithms, GA converges easily, which ensures less execution time.

(3) Considering the studied problem, GA does not require too much, and it is well suitable as a DAG sequence is analogous to a chromosome and the DAGs in the DAG sequence to the genes of a chromosome.

The GA process is described as follows: According to the size of the fitness function, GA chooses a certain proportion of individuals from the parent population to form the progeny population. To ensure that the number of individuals in the progeny population is the same as that in the parent population, these chosen individuals are crossed to produce new progeny individuals according to a certain crossover rule. Then, some individuals in the progeny population are chosen to randomly mutate to produce some new progeny individuals. After the mutation, one iteration is completed. The above process is repeated until the final iteration. Next, We introduce some innovative modules of the GA-based cost optimization algorithm.

5.1 Algorithm encoding

GA cannot directly deal with the actual problem space of parameters. These parameters need to be converted into individuals with a genetic structure that GA can deal with. This process is called encoding. Considering the one-to-one correspondence between DAGs in the DAG sequence and their subscripts, these subscripts are integers. Therefore, we adopt integer encoding to encode the problem space. For example, a DAG sequence encoded with five DAGs is shown as Table 3.

5.2 Fitness function

In GA, the fitness function is used to evaluate the quality of different individuals in a population to select the best. In Section 4, we have discussed that the global optimization function has two factors: relative location and redundant operations. Therefore, we design the

Table 3 An encoded DAG sequence.

1	2	3	4	5

fitness function with two factors: the size of relative location and the redundant operation gain. To address the problem of the fitness function with two factors, the relative weight is proposed, which considers the weight of the size of relative position in the fitness function denoted as w_l and the weight of the redundant operation in the fitness function denoted as w_s. The formula for calculating the fitness function in the cost optimization algorithm is shown as Eq. (10).

$$C_f = C_{la} \times w_l + C_{sa} \times w_s \tag{10}$$

Here, C_{la} denotes the size of the relative location of the DAG sequence, calculated as Eq. (7); C_{sa} denotes the redundant operation gain of the DAG sequence, calculated as Eq. (9); and C_f denotes the fitness function. The relationship between w_l and w_s is shown as Eq. (11).

$$w_l + w_s = 1 \tag{11}$$

The relative weight reflects the significance of each factor in the fitness function. How to set the relative weight values depends on the real cases. We set w_l as 0.6 and w_s as 0.4 in our subsequent experiments.

5.3 Crossover function

The discrete crossover function, single-point crossover function, and multi-points crossover function are the common crossover functions. However, these crossover functions are too random, and in many cases, their effects are not acceptable. To avoid this problem, we adopt a novel crossover function, named revolving double heuristic crossover function. The detailed process of this function is described as follows:

(1) Two DAG sequences are randomly selected from a population (they must be different, otherwise, the selection is repeated). For example, two different DAG sequences with five DAGs have been selected as shown in Table 4.

(2) The fitness of the first two DAGs of the two DAG sequences are compared. If the fitness between DAG 1 and DAG 3 in the first DAG sequence is larger than that between DAG 4 and DAG 5 in the second DAG sequence, then the first DAG sequence remains unchanged in this round of crossover, and the DAGs in the second DAG sequence are rotated counterclockwise until DAG 1 reaches the head of the second DAG sequence and becomes fixed. Again, the remaining four DAGs are rotated counterclockwise until DAG 3 reaches the second location of the second DAG sequence. This process is shown as Table 5.

(3) The fitness of the second and the third DAGs of the two DAG sequences are compared and the above process is repeated until the last two DAGs in the two DAG sequences finish their crossovers. Crossing is stopped when the two DAG sequences are absolutely the same. The result is shown as Table 6.

(4) One of the original two DAG sequences is randomly replaced with the new DAG sequence produced by the above crossover process. The other original DAG sequence remains unchanged. Therefore, the output of the crossover function is shown as Table 7 (the first DAG is replaced by the new DAG sequence).

At every step of the above crossover process, the DAGs are rotated counterclockwise according to the fitness of DAGs. Therefore, revolving double heuristic crossover function belongs to the heuristic function and its effect is much better than those of the common discrete crossover function, single-point function, and multi-point random crossover function.

6 Experiments

To prove the efficiency of the GA-based cost optimization algorithm and the scheduling optimization technique based on reuse, we discuss the design of two experiments in this section.

Table 4 Two DAG sequences without crossover.

1	3	4	2	5
4	5	3	2	1

Table 5 Two DAG sequences with the first crossover.

1	3	4	2	5
4	5	3	2	1
		⇓		
1	3	4	2	5
1	4	5	3	2
		⇓		
1	3	4	2	5
1	3	2	4	5

Table 6 Two DAG sequences with the last crossover.

1	3	2	5	4
1	3	2	4	5
		⇓		
1	3	2	5	4
1	3	2	5	4

Table 7 The output of crossover function.

1	3	2	5	4
4	5	3	2	1

6.1 Experiment of the cost optimization algorithm based on GA

To produce the initial population of the GA-based cost optimization algorithm, an experiment was carried out, in which some DAG sequences were randomly produced by a random function. After several iteration rounds, the DAG sequence with the largest fitness in final population was outputted. To verify the improvement of the GA based-cost optimization algorithm, the individual fitness of population in each iteration was outputted to a text file and compared with those in other iterations.

The experimental parameter settings are as follows: The size of population is 20, the DAG sequence length is 10, the number of algorithm iterations is 5, selection probability is 0.8, crossover probability is 0.6, mutation probability is 0.02, relative weight w_l is 0.6, and w_s is 0.4. The size of reuse degree of DAGs is randomly set from 0 to 50 by a random function in DAG sequence. The experimental result is shown as Fig. 5.

The horizontal axis in Fig. 5 denotes the individual of a population; the vertical axis denotes the individual fitness. The five colors denote five populations at different iterations as the figure shows. The colored dots denote the individual of a certain population and certain individual fitness. The colored broken lines denote the tendency of individual fitness of different populations. From Fig. 5, we can find that the further the population is, the higher the location of the broken line of the population is in the coordinate system, which means its individual fitness is larger than that of the previous populations. This proves that the individual fitness of different populations is continually improved by the GA-based cost optimization algorithm. We can further see that the largest individual fitness of the last population is more than 140, which is much larger than the largest individual fitness of the first population, whose value is less than 100. Thus, the GA-based cost optimization algorithm can significantly improve the individual fitness of population.

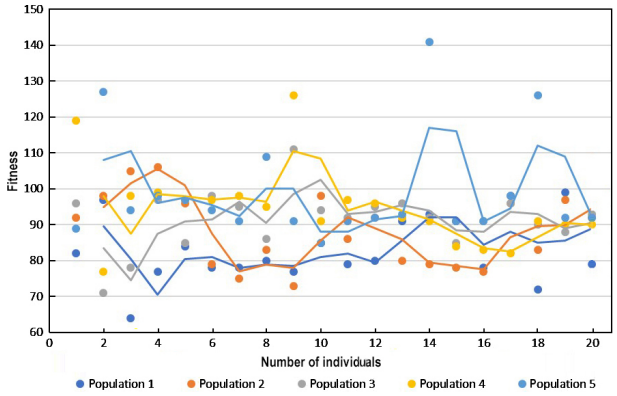


Fig. 5 The exprimental result of the cost optimization algorithm.

6.2 Experiment of the scheduling optimization technique based on reuse

To prove the feasibility of the scheduling optimization technique based on reuse to defend against APT attack, our programs are run on three different Spark systems. Then, we can conclude by comparing the average turnaround times of applications and the system throughputs of the three different systems. The three different Spark systems are the native Spark system, which is the existing Spark system without any changes, and it is named as native system; the Spark system with reuse technique, which is the native system with a reuse module to realize DAG reuse but is without any scheduling, and it is named as reuse system; and the Spark system with the proposed scheduling optimization technique based on reuse and it is named as scheduling system.

Our experimental programs are run on a single machine, whose configuration is as follows: Intel core i3-3240 processor, 4 cores, 3.4 GHz clock speed, 12 GB memory, and 500 GB disk capacity. Software includes Ubuntu 14.04, jdk 1.8.0-60, maven 3.3.9, Spark 1.6.1, and VMware Workstation 11. We constructed a distributed Spark system with three nodes by simulating two worker nodes by a virtual machine. The whole Spark system is faced with resource shortage. Our experimental data is downloaded from the big data benchmark data set of AMPLab[29], which is developed by Pavlo and some other researchers to describe the site visits suitable for representing log data in APT attacks. The size of experimental data is 1.1 GB, which is large enough for our Spark system. We design three groups of experiment on the aforementioned three systems. Their corresponding data sizes are 100 MB, 500 MB, and 1 GB. The average turnaround time of applications of the three systems is shown as Fig. 6. Here, the horizontal axis denotes the data size and the vertical axis denotes the average turnaround time of applications. As shown in Fig. 6, the average turnover time of applications of the native system is larger than those of the other two systems regardless of the data size. This suggests that reuse technique

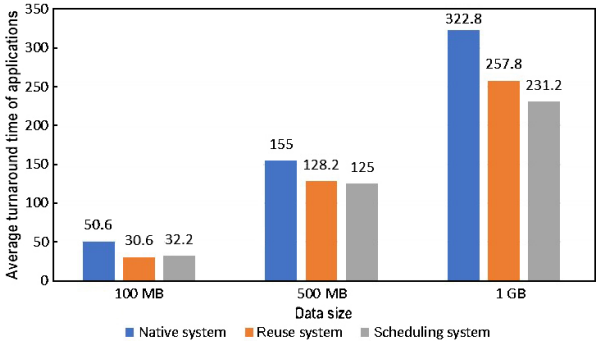


Fig. 6 The average turnaround time of applications.

and the scheduling optimization technique can improve Spark performance in using log analysis to detect APT attack. Meanwhile, when the data size is 100 MB, the average turnaround time of the reuse system is slightly lower than that of the scheduling system. The scheduling system may have considerable extra scheduling overhead at a small data size. However, for the reuse system, with the increase of experimental data size, the average turnover time is gradually larger than that of the scheduling system, which suggests the scheduling system is much more efficient.

The system throughput of the three systems is shown as Fig. 7. Here, the horizontal axis denotes the system throughput and the vertical axis denotes the data size. From Fig. 7, we can find the system throughput of the scheduling system continues to increase with data size and is maximum at 21.63. However, the system throughputs of the other two systems fluctuate to some degree and their values are gradually smaller than that of the scheduling system. This means that the performance of the scheduling system increases with data size. Therefore, the effectiveness of our scheduling optimization technique to detect APT attacks confirmed.

7 Conclusions

In this paper, we propose a scheduling optimization technique based on reuse in Spark to defend against APT attacks. This scheduling optimization technique combines the reuse technique with scheduling technique to improve Spark performance, and it involves three parts: (1) It defines and formulates the reuse degree of Spark DAGs. (2) It designs a global optimization function based on relative location and redundant operation to calculate the optimal DAG sequence with the least execution time. (3) It designs a GA-based cost optimization algorithm to implement the global optimization function. The conducted experiments shows that the scheduling optimization technique is efficient in defending against APT attacks because it greatly reduces the time of log data analysis in Spark.

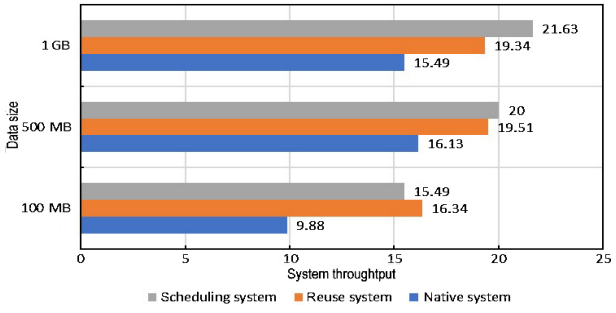


Fig. 7 The throughput of system.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61379144, 61572026, 61672195, and 61501482) and the Open Foundation of State Key Laboratory of Cryptology.

References

[1] P. Chen, L. Desmet, and C. Huygens, A study on advanced persistent threats, in *Proc. 15th IFIP TC 6/TC 11 Int. Conf. Communications and Multimedia Security*, Aveiro, Portugal, 2014, pp. 63–72.

[2] J. Vukalović and D. Delija, Advanced persistent threats detection and defense, in *Proc. 2015 38th Int. Convention on Information and Communication Technology, Electronics and Microelectronics*, Opatija, Croatia, 2015, pp. 1324–1330.

[3] C. Tankard, Advanced persistent threats and how to monitor and deter them, *Netw. Secur.*, vol. 2011, no. 8, pp. 16–19, 2011.

[4] R. Brewer, Advanced persistent threats: Minimising the damage, *Netw. Secur.*, vol. 2014, no. 4, pp. 5–9, 2014.

[5] A. Moscaritolo, Transparency: Operation aurora, *SC Magazine: For IT Security Professionals*, vol. 21, no. 3, p. 14, 2010.

[6] T. M. Chen and S. Abu-Nimeh, Lessons from stuxnet, *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[7] Apache Spark^{TM}, Apache spark project, http://spark.apache.org/, 2018.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, Spark: Cluster computing with working sets, in *Proc. 2nd USENIX Conf. Hot Topics in Cloud Computing*, Boston, MA, USA, 2010, p. 10.

[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *Proc. 9th USENIX Conf. Networked Systems Design and Implementation*, San Jose, CA, USA, 2012, p. 2.

[10] M. Zaharia, T. Das, H. Y. Li, S. Shenker, and I. Stoica,

Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters, in *Proc. 4th USENIX Conf. Hot Topics in Cloud Ccomputing*, Boston, MA, USA, 2012, p. 10.

[11] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, Shark: SQL and rich analytics at scale, in *Proc. 2013 ACM SIGMOD Int. Conf. Management of Data*, New York, NY, USA, 2013, pp. 13–24.

[12] N. M. Weber, The relevance of research data sharing and reuse studies, *Bull. Am. Soc. Inf. Sci. Technol*, vol. 39, no. 6, pp. 23–26, 2013.

[13] T. K. Sellis, Multiple-query optimization, *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.

[14] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan, Semantic data caching and replacement, in *Proc. 22nd Int. Conf. Very Large Data Bases*, Bombay, India, 1996, pp. 330–341.

[15] K. Dursun, C. Binnig, U. Cetintemel, and T. Kraska, Revisiting reuse in main memory database systems, in *Proc. 2017 ACM Int. Conf. Management of Data*, Chicago, IL, USA, 2017, pp. 1275–1289.

[16] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis.* California, CA, USA: O'Reilly Media, 2015, pp. 26–30.

[17] L. Wang, Directed acyclic graph, in *Encyclopedia of Systems Biology*, W. Dubitzky, O. Wolkenhauer, eds. New York, NY, USA: Springer, 2013, pp. 1105–1114.

[18] Q. Ren, M. H. Dunham, and V. Kumar, Semantic caching and query processing, *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 1, pp. 192–210, 2003.

[19] Wikipedia, Schedule, https://en.wikipedia.org/wiki/Schedule, 2018.

[20] R. Sakellariou and H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems, in *Proc. 18th Int. Parallel and Distributed Processing Symp.*, Santa Fe, NM, USA, 2004, pp. 111–123.

[21] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, *Job Scheduling for Multiuser Mapreduce Clusters*. Berkeley, CA, USA: University of California, 2009.

[22] U. Schwiegelshohn and R. Yahyapour, Fairness in parallel job scheduling, *J. Schedul.*, vol. 3, no. 5, pp. 297–320, 2000.

[23] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Parallel job scheduling—A status report, in *Proc. 10th Int. Workshop on Job Scheduling Strategies for Parallel Processing*, New York, NY, USA, 2004, pp. 1–16.

[24] T. S. Ferguson, Linear Programming: A concise introduction, https://www.math.ucla.edu/∼tom/LP.pdf, 2000.

[25] M. Dorigo and L. M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[26] D. B. Skalak, Prototype and feature selection by sampling and random mutation hill climbing algorithms, in *Proc. 11th Int. Conf. Machine Learning*, New Brunswick, NJ, USA, 1994, pp. 293–301.

[27] S. Z. Selim and K. Alsultan, A simulated annealing algorithm for the clustering problem, *Pattern Recognit.*, vol. 24, no. 10, pp. 1003–1008, 1991.

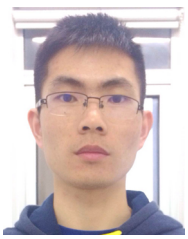
[28] K. De Jong, Learning with genetic algorithms: An overview, *Mach. Learn.*, vol. 3, no. 23, pp. 121–138, 1988.

[29] U. C. Berkeley AMPLab, Big data benchmark, https://amplabcsberkeleyedu/benchmark, 2014.

Jianchao Tang is currently a PhD student at National University of Defense Technology. He received the master degree from National University of Defense Technology in 2016, and BS degree from Beijing Institute of Technology in 2014. He achieved best paper awards in the 11th Chinese Conference on Trusted Computing and Information Security. His current research interests include big data security and cloud computing.

Ming Xu is currently a professor and director of the Network Engineering Department in the College of Computer, National University of Defense Technology. He is a senior member of China Computer Federation (CCF), Institute of Electrical and Electronics Engineers (IEEE), and Association for Computing Machinery (ACM). He was invited to serve as program committee of more than 30 international academic conference proceedings and serve as chairman or the chairman of the procedural committee four times. Meanwhile, he edited 3 volumes of international conference proceedings of Springer. He is also a member of editorial board of *IASTED International Journal of Computers and Applications*, and *Journal of Communication*. His major research interests include mobile computing, wireless network, cloud computing, and network security.

Shaojing Fu received the BS and PhD degrees in Applied Mathematics from National University of Defense Technology, China, in 2005 and 2010, respectively. He has been an associate professor with College of Computer at National University of Defense Technology since December 2015. His research interests include applied cryptography, and security and privacy issues for cloud computing.



Kai Huang received the BS degree and MS degree in Cryptography from Naval University of Engineering, China, in 2007 and 2009, respectively. Currently, he is working towards the PhD degree in the College of Computer at National University of Defense Technology. His research interests include cloud computing and network security.