Optimization of Integrity Measurement Algorithm Based on Multi-thread Pipeline Paralleling

Yinhao Wang[1]
State Key Laboratory of Mathematical Engineering and Advanced Computing
Zhengzhou, Henan, 450002,China
E-mail: `xdwyh1990@163.com`

Zheng Shan, Chao Fan, Fei Xue
State Key Laboratory of Mathematical Engineering and Advanced Computing
Zhengzhou, Henan, 450002,China

With the continuous development of trusted computing, the integrity measurement has gradually been practical; but the popularization and application of the existing measurement tools are still restricted by their own measurement algorithm's performance. Considering the insufficient of existing measurement tools' performance, we propose an optimization method of integrity measurement algorithm based on multi-thread and pipeline paralleling. We analyze shortage of the existing measurement tools and provide an optimized method based on multi-thread technology and pipeline paralleling technology. Experiments show that this method can significantly improve the performance of SHA-1 and is also applicable to a variety of hash algorithms.

CENet2015
12-13 September 2015
Shanghai, China

[1]Speaker

1. Introduction

With the development of the era and the progress of science, the computing systems have been gradually moved towards the pattern of ubiquitous computing; however, the computer will be faced with a series of security threats such as malicious code attacks, illegal access to information and unlawful destruction of data and systems. Malicious code attacks aiming at the user's private information have become the largest security threat [1]. In order to effectively address these security threats, the trusted computing arises at the historic moment and has become a hot research topic in the field of information security. The integrity measurement mechanism, as the significant technology of trusted platform, builds a trust chain of trusted computing platform so as to ensure the credibility of computing platform. According to the level of protection, the integrity measurement can be divided into two types: the hardware level and the software level [2].

The addition of hardware module can improve the signature algorithms' performance, however it not only increases the cost of system, but also can't adapt to all computing platform.

The integrity measurement tools in the software level mostly use hash algorithm as the integrity measurement algorithm, but the algorithm seriously affects the performance of computer when doing the run-time measurement; therefore, we propose the optimization of integrity measurement algorithm based on the multi-thread pipeline paralleling in this paper.

2. Related Work and Contribution of this Paper

With the development of integrity measurement technology, a series of integrity measurement tools have come out in software level such as TAMU [3], Hobgoblin [4] and Tripwire [5]; but they have the same problem, the performance bottleneck; besides, the measurement algorithm seriously affects the performance of measurement tools, thus the bottleneck affects the promotion and application of measurement tools as well.

Study shows which measurement method is used by the key factors influencing the performance of trusted computing platform. Measurement method includes three elements: the measurement time, the measurement objects and the hierarchy where measurement is implemented [6]. The measurement time refers to the moment of measurement; the measurement objects refer to the manifestation of measured entity's certain status in the system and the hierarchy of implementing measurement indicates which measures taken by us as to the measurement objects. In terms of study of measurement method, proposed a method to measure the integrity of the operating system kernel[7]. The measurement process is divided into three stages and the paper selects appropriate measurement time and the measurement objects to realize the integrity measurement of the operating system kernel and proofs the security with formal proof method, while it does nothing about the key hierarchy where measurement is implemented. PEDIAMA makes use of embedded strategy to bind the strategy to the measurement objects [8]. This architecture can save the cost of query and maintenance, while improving the efficiency of executing the measurement framework; nevertheless, the calculation process of measurement algorithm has to be optimized.

In conclusion, these existing tools' measurement algorithms mostly need just a single execution, but it cannot meet the demands in the environment that computer needs higher security level. Under this circumstance, it needs do frequent measurement for the system, which can bring serious impact on computer performance. In order to maintain the security of the computer and guarantee the performance of computer, we put forward a kind of integrity measurement algorithm based on multi-thread and parallel optimization by introducing the multi-thread technology and the pipeline paralleling technology. Finally, we've optimized the algorithm.

3. Pipeline Paralleling Design of Integrity Measurement Algorithm

This section firstly describes the existing measurement algorithm, and then introduces how to realize the design of parallel optimization with related technologies.

3.1Hash Algorithm Overview

```
Algorithm 1: integrity measurement algorthm
Input: Archive
Output: Hash
1 Begin
2 Toread=m;Readsize=m;Init_hash
3 while( Readsize == ToRead )do
4     Readsize= read( byte[m], ToRead )
5    Init_hash=update( byte[m], Readsize, Init_hash )
6 endwhile
7 End
```

Figure 1. Algorithm Description

We use hash algorithm as the measurement algorithm generating data extracts for integrity measurement and describe the measurement algorithm in Fig. 1.

Archive is the target data to be measured, and *get* its hash value. Firstly, the algorithm uniformly slices *Archive* for the size of *m*; then reads the block of data into the buffer *byte[m]* in turn, according to the initialization hash value *Init_hash* which calculates the hash value of the block of data in the buffer; later this result serves as the initialization hash value for the next loop calculation until *Archive* is completely processed; the algorithm calculates the hash value of *Archive* that is *Init_hash* after the last loop.

3.2 Concurrent Design of Measurement Algorithm

The measurement algorithm can be paralleled by introducing multi-thread technology and pipeline paralleling technology just as it shows in Fig. 1. The multi-thread can obtain less costs when they are created , make full use of multi-core processors; however, the technology brings large synchronous load and scheduling overhead as well as some other problems such as deadlock, competition and priority inversion, etc[9]. The pipeline paralleling technology assigns each task of loops to different threads with each thread executing pipelining to obtain parallelism[10]; and the data-dependence among loops is maintained in some synchronous way.

The parallel implementation of the algorithm designed in this paper is showed in Fig. 2. We divide the measurement algorithm into two threads parallel execution, the read thread and the update thread. We firstly set the fixed-size buffer array as sharing space of such two threads; at meanwhile, we initialize the count signal for synchronization between threads; then we create the Read thread and the Update thread. The target data is uniformly written into the buffer array by the Read thread by means of data pre-fetching [11]; at meanwhile, the data that has been sequentially written into the buffer array will be processed by Update thread. When the data is written into buffer by Read thread until the buffer is full, the count signal *nread* also accumulates unceasingly; when the update thread sequentially processes the data, the update thread won't stop the data processing until the buffer is empty. In this way, we've realized the thread synchronization. Fig. 3 shows the description of the parallel algorithm:

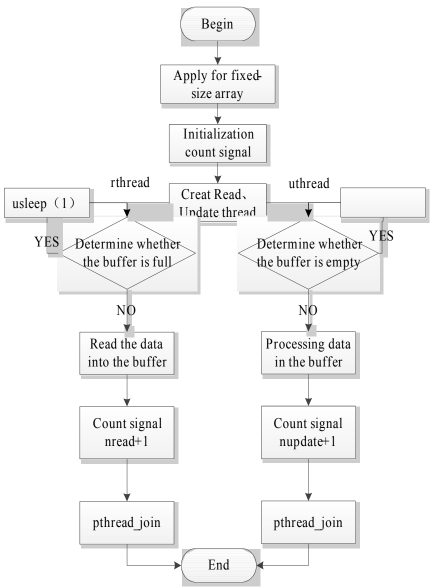

Begin
Apply for fixed-size array
Initialization count signal
rthread
uthread
usleep（1）
Creat Read、Update thread
YES
Determine whether the buffer is full
YES
Determine whether the buffer is empty
NO
NO
Read the data into the buffer
Processing data in the buffer
Count signal nread+1
Count signal nupdate+1
pthread_join
pthread_join
End

Figure 2: Parallel Flowchart

```
Algorithm 2: parallel algorithm
Input: Archive
Output: Hash
1 Begin
2 Toread=m;Readsize=m;nread=nupdate=0;Init_hash
3 /*Read rthread*/
4 while( Readsize == ToRead )do
5   if((nread+1)%n==nupdate) then usleep(1)
6   endif
7   Readsize= read(byte[m], ToRead )
8   nread=(nread+1)%n
9  endwhile
10 /*Update uthread*/
11 while( Readsize== ToRead )do
12   if(nupdate==nread ) then usleep(1)
13   endif
14   Init_hash=update( byte[m], Readsize, Init_hash )
15   nupdate=(nupdate+1)%n
16 endwhile
17 End
```

Figure 3: Parallel Algorithm Description

3.3 Performance Analysis of Parallel Algorithms

According to Amdahl's law, the algorithm's speedup is:

$$S(p,f)=\frac{p}{1+(p-1)f} \tag{3.1}$$

p presents the number of physical threads those execute calculating, f refers to the proportion that the serialized execution part accounts in the total workload.

Algorithm 1 shows that there is data dependence between loops in the computation section of the measurement algorithm, namely, the serial components of the measurement algorithm. We assume that the system's I/O rate is $s_r = f_s(m)$, the calculating rate of the measurement algorithm is s_u per second. According to Formula (3.1) the speedup of Algorithm 2 is:

$$S_{th}(p,f)=\frac{p}{1+(p-1)f}$$

$$\Rightarrow S_{th}(p,f)=\frac{p}{1+(p-1)\bullet(s_u\bullet t)/[(s_u+s_r)\bullet t]}$$

$$\Rightarrow S_{th}(p,f)=\frac{p}{1+(p-1)\bullet s_u/(s_u+f_s(m))} \tag{3.2}$$

p_r is the number of Read thread, p_u is the number of the update thread, the existing data dependence results is $p_u = 1$.

As to Formula (3.2), we put forword that the theoretical speedup of Algorithm 2 is:

$$S_{th}(p_r,f)=\frac{p_r+1}{1+p_r\bullet s_u/(s_u+f_s(m))} \tag{3.3}$$

When considering the synchronization overhead, the actual speedup is:

$$S_{ac}(p_r,f)=\frac{p_r+1}{1+p_r\bullet s_u/(s_u+f_s(m))+T_0/T} \tag{3.4}$$

In which, T_0 refers to addition overhead of the algorithm which includes communication, synchronization and idle time, etc. T is the total overhead of Algorithm 1.

Compared with Algorithm 1, Algorithm 2 increases the number of instructions and increases the scheduling overhead, synchronization overhead and the buffer size, but it reduces the frequent accessing to the shared buffer and hides the operations of memory accessing, finally it improves the efficiency of measurement algorithm.

The main factors affecting the expandability of algorithm involve: serial component of program; the system overhead includes communication, synchronization, idle time and load imbalance.

The proportion that the serialized part of the algorithm accounts in the total workload is:

$$f = s_u / (s_u + f_s(m)) \tag{3.5}$$

According to Formula (3.5), we can adjust *f* by adjusting the pipeline granularity.

Overhead T_0 increases with the increasing of the number of threads, increasing workload can offset part of overhead that is produced by the increasing number of threads.

There is only an update thread in the algorithm which leads to non-uniform task allocation.

In summary, Algorithm 2 obtains expandability, but its expandability is restricted by unbalanced load.

4. Experiment and Results Analysis

4.1Quantification of Factors Affecting Algorithm Performance

We assume that the size of target file is *S*, the block size is *m*, the overhead is T_r that Read thread writes one block into the buffer, the overhead is T_u that Update thread processes one block in the buffer, the total workload of Algorithm 1 is T_{total}, the total workload of Algorithm 2 is $T_{parallel}$.

The overhead that read thread writes one block into the buffer is:

$$T_r = f_r(m, s_r) = m/s_r = m/f_s(m) \tag{4.1}$$

The overhead that update thread processes one block in the buffer is:

$$T_u = f_u(m, s_u) = m/s_u \tag{4.2}$$

Additional overhead of Algorithm 2 is:

$$T_0 = T_{parallel} - \frac{S}{s_u} \tag{4.3}$$

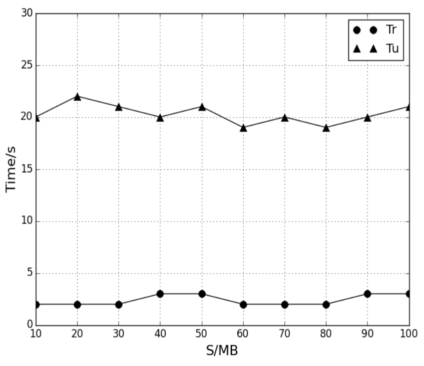
4.2 Experiment

We use Sugon Tiankuo I420-G10 server as the testing platform, the processor is Intel Xeon E5-2407 with four cores, each core's main frequency is 2.2GHz, its memory is 16.0GB, here the platform is configured with the operating system of Redhat Enterprise 5 and the versions of the operating system's kernel is 2.6.32. The testing files respectively are 10MB, 20MB, 30MB…… 80MB, 90MB, 100M.

We take SHA-1 as the testing algorithm. Firstly, we test the performance of the original algorithm; then we test the performance of the algorithm modified according to Fig. 3 did the test for the efficiency of reading and writing by selecting different size of *m*, its conclusion is: when *m* equals to the size of file system's block, we relatively achieve higher efficiency of reading and writing; at meanwhile, T_{total} is smaller[12] ; thus we decide that *m* equals to 4KB, the testing results are shown in table1 and the coordinate Fig. 4. According to Formula (4.1), (4,2): $s_u = 2.048 \cdot 10^8$ (byte/s); $s_r = f_s(m) = 2.048 \cdot 10^9$ (byte/s); then according Formula(3.3): theoretical speedup is: $S_{th} \approx 1.83$

S(MB)	10	20	30	40	50	60	70	80	90	100
T_r (μs)	2	2	2	3	3	2	2	2	3	3
T_u (μs)	20	22	21	20	21	19	20	19	20	21

Table1: The Test of System I/O Rate and Algorithm's Rate



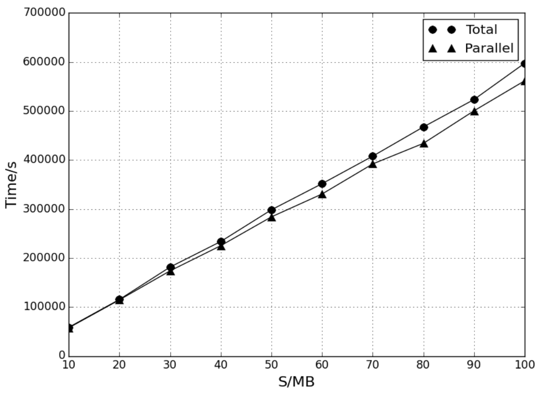


Figure 4: I/O Rate and Algorithm's Rate **Figure 5:** Parallel Test Pattern

We show the original algorithm's overhead and the parallel algorithm's overhead in Table 2. With Formula (4.2), (4.3): the actual speedup of algorithm is Sac≈1.72.

S(MB) / **T**(μs)	10	20	30	40	50	60	70	80	90	100
T_{total}	58537	115313	181456	233564	298243	351582	407712	467338	523281	597413
$T_{parallel}$	57517	114531	173745	225072	283818	330495	391863	433820	500271	561525
T_0	6317	1891	12465	20272	14756	38655	33463	44700	39471	23925

Table 2: Parallel Test Results

4.3 Algorithm Security and Performance

In view of the existing measurement models considering less about the measurement algorithm's optimization and seriously affecting the computer performance, we analyze the original measurement algorithm and uniformly slice the target data in this paper. Then we take advantage of multi-thread's feature that different threads of the same process share the address space and divide the measurement process into two threads; at meanwhile, we make use of pipeline paralleling technology to parallel processing fetch part and calculate part of algorithm. Finally, we realize memory accessing hidden and improve algorithm's performance. As the operation process of the algorithm itself is still running sequentially, we get the correct measurement results. We guarantee unidirectional and collision resistance of the improved algorithm. With the introduction of the multi-thread technology, we've effectively avoided the occurrence of the deadlock by setting the appropriate synchronization signals and ensured the correctness of the algorithm.

Factors mainly affect the computer performance from three aspects: CPU, memory and costs of program. On the basis of the constant development of multi-core technology today, we take full advantage of characteristics of the multi-core architecture by introducing multi-thread technology and realize the effective utilization of resources; we set the length of buffer for *n=100*, doing like this which occupies smaller cache. In conclusion, we have improved the overall performance of algorithm.

Since the majority of hash algorithms themselves lack parallelism, this paper mainly processes the pretreatment process of algorithm in a parallel manner. Finally, this optimization method is also applicable to the CRC32, MD5, SM3, HAVAL, etc.

5. Conclusion

In this paper, considering the integrity measurement algorithm in the trusted computing which affects computer performance, we proposed a method of the optimization of integrity measurement algorithm based on multi-thread and pipeline paralleling. Firstly, we process the pretreatment process of algorithm in a parallel manner by dividing I/O and calculating into two threads, then we select the appropriate block size to obtain better rate of reading and writing; finally we've realize the parallelism of algorithm.

References

[1] Feng Dengguo,Qin Yu, and Wang Dan. *Research on Trusted Computing Technology*[J]. Journal of Computer Research and Development, 48(8):1342-1349(2011)(In Chinese).

[2] Song Ningnan. *Research on the Integrition Protection of Data Stored on Hard Disk*[D]. Shanghai Jiao Tong University, Shanghai ,(2009)(In Chinese).

[3] David R. Safford, Douglas Lee Schales, David K. Hess. *The TAMU security package: An ongoing response to internet intruders in an academic environment*[C]//Proc of the 4th Usenix UNIX Security Symposium.USENIX Association Berkeley, CA, USA :91-118(1993).

[4] Scott Leadly, Kenneth Rich, Mark Sirota. *Hobgoblin: A File and Directory Auditor*[C]//Proceedings of the Fifth Large Installation Systems Administration Conference. LISA V proceedings :199-207(1991).

[5] Gene H. Kim, Eugene H. Spafford. *The design and implementation of tripwire: A file system integrity checker,*CSD-TR-93-071[R]. [s.1.]:Purdue University:18-29(1993).

[6] Liu Ziwen,Feng Dengguo. *TPM-based dynamic integrity measurement architecture*[J]. Journal of Electronics & Information Technology,32(4) : 874-877(2010)(In Chinese).

[7] Li Yu, Zhao Yong, and Lin Li. *Method of Trusted Measrement for Operation System Kernel*[J]. Journal of Chinese Computer Systems, 34(5):997-1002(2013)(In Chinese).

[8] Deng Rui, Chen Zuoning. *Policy embedded dynamic integrity active measurement architecture*[J]. Application Research of Computers, 30(1): 261-264(2013)(In Chinese).

[9] David R. Butenhof. *Programming With POSIX Threads*[M]. Addison-Wesley Professional.Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ,pp.112-114:(1997).

[10] Liu Xiaoxian, Zhao Rongcai, Ding Rui. *Pipelining granularity optimization algorithm based on loop tiling*[J]. Journal of Computer Applications, 33(8):2171-2176(2013)(In Chinese).

[11] Chen Wei, Du Lingxia, and Chen Hong. *Optimization Strategies of Data Processing Algorithms under Multi-Core Architecture*[J]. Journal of Frontiers of Computer Science and Technology,5(12):1057-1075(2011).

[12] Li Mengyu. *Research on Efficiency of I/O and Standard I/O Reading-writing*[J]. Software Guide,9(6):5-7(2010).