International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 20 (2017) pp. 9038-9045

Eviction Bit and Inclusive Cache Based Replacement Policy for Side Channel Attack

[1]Adi Maheswara Reddy G, [2]K Venkata Rao and [3]JVR Murthy

[1]Research Scholar, Department of Computer Science and Engineering,
Jawaharlal Nehru Technological University, Kakinada, Andhra Pradesh, India.
[2]Department of Computer Science and Engineering, Vignan Institute of Information Technology Visakhapatnam, Andhra Pradesh, India.
[3]Department of Computer Science and Engineering, Jawaharlal Nehru Technological University, College of Engineering, Kakinada, Andhra Pradesh, India.

Abstract

Modern Cloud Computing Architecture shares physical resources throughout diverse programs so as to maximize the efficiency based on area, energy, and cost. In the most widely-used environment, where each core has more than single level private caches, and shares an inclusive lower-level cache with all the other cores. Regrettably, sharing causes a hazard to security, though procedures are completely remote on rational level, employing a physical resource among them generally tells that single procedures resource usage outline could be witnessed by outsiders. Cache replacement policies are one of the prevalently employed countermeasures that considerably impact the performance of side channel attacks in cloud computing. In this paper, a new replacement scheme is suggested to safeguard in contrast to these conflict side channel attack known as Eviction bit and Inclusive Cache based Replacement policy. An Eviction Bit is assigned to each cache line that is targeted for replacement and the detection of inclusive cache is done. These two steps added increased the cache replacement policy performance efficiency. The experimental results are performed on 8 different cores where 7 cores are victim and 1 is a spy core that seeks the confidential information of victim. This showed that the proposed approach has better performance efficiency compared to existing SHARP and Hybrid mitigation technique.

Keywords: Cloud Computing, Security, Side channel attacks, Cache Replacement Policy, Inclusive cache, Temporal Position Prediction, Eviction Bit

INTRODUCTION

Cloud computing is an amenity given through the network as to distribute the computing equipment's that offers huge storage space and numerous services to the customer using negligible cost of setup and improvement. In this era of high speed, Internet cloud computing is gaining more popularity. Significant characteristics of cloud computing comprises of capital expenses, enhancement in consumption of resources, scalability on request, execution is swift and flexible and the utmost significance is it could be retrieved from everywhere. Cloud services such as Amazon's Elastic Compute Cloud (EC2) [16] and Microsoft's Azure Service Platform (Azure) [19] are promptly achieving acceptance since they suggest price effective, accessible and extremely obtainable computing facilities to its customer. These advantages are made probable through sharing extensive computing resources amongst a huge number of customers. Nevertheless, security and confidentiality above the off-loading subtle information make numerous customers, initiatives and government administrations unwilling to implement cloud services [17, 18].

An intelligent kind of susceptibility in the physical execution of else the secure systems is the outflow of data with the help of unintentional (or side) networks. The seeped data is known as side channel data, and attacks manipulating such data are known as side channel attacks. These [9, 10] obtain private information from a system by observing its behavior, rather than by directly gaining access to private information. Such attacks are both popular and often highly effective. Due to their nature, they are hard to prevent with existing software techniques. Any procedure whose memory access outline hinges on trusted data is in threat of dripping this data using cache-aided side channels. There is enormous work on countermeasures in contrast to cache-aided side channel attacks.

Cache replacement policies are one of the prevalently employed countermeasures that significantly influence the efficiency of numerous appliances. As these strategies select which locations to eject from the cache, they have a uninterrupted influence on miss rates that is typically openly associated with performance. From safety viewpoint it is epitome to absolutely eradicate side channels using the cache by a plan, as in different cache replacement policies or to flush the cache amongst the accesses of two dissimilar resources. Regrettably, such traditional methodologies too partly annulled the performance aids of sharing.

There are numerous defending approaches for side channel attacks in cloud computing, among them SHARP [20] is a secure hierarchy conflict cache based side cannel attack employed for cloud computing through cache replacement policies. Even though, this approach efficiently defends against all existing cross-core shared-cache attacks, with less modification in hardware, there still exist certain limitations.

In SHARP, when there is no private cache available for replacement, the process is interrupted in the middle by generating an alarm. This event stops certain significant replacement actions and minimizes the efficiency of the system with increase in the computational time. Therefore, as to address this issue, the paper suggested a novel cache replacement policy for cache aided side channel attacks. In this paper, prior to performing cache replacement policy, an Eviction bit is assigned to each cache line of both the victim and spy process separately. Formerly the detection of Inclusive cache lines is performed using LLC. These two steps aid the replacement policy to replace the victim cache line with minimized threat of spy process. This method is included within the SHARP algorithm and increases the efficiency of the replacement policy when compared to SHARP algorithm.

Organization of the Chapter

A brief introduction to the cache aided side channel attacks in computing along with the motivation for the suggested approach is given in this section. A brief discussion of Existing approaches in side channel attacks using cache and other security issues in cloud computing in section 2. Dissimilar Cache Replacement policies for the cloud computing is briefly given in section 3. The proposed Eviction Bit and Inclusive Cache based Replacement Policy as to defend against side channel attack is briefly explained in section 4. The experimental results and Its analysis for the suggested approach are given in section 5. The conclusion for the suggested approach along with the reference is given in section 6 and section 7 respectively.

LITERATURE SURVEY

Various approaches are being suggested to protect in contrary to cache based side channel attacks and cloud computing security. Some of them are given in this section. Godfrey [13] implements process-based cache partition using page coloring on Xen. Even though this scheme can successfully prevent side channel attacks, it has been shown that it suffers from significant performance degradation when aiding a high number of parts. SecDCP [14] is a way-partitioning outline where every appliance is allotted to a security class. Based on the security classes of the applications running concurrently, the scheme dynamically adjusts the partition layout to ensure an application cannot attack another application with a higher security class. However, when appliances are in the similar security class, the scheme is forced to use static partitioning. In both of the previous schemes, selective cache flushing of partitions is required when the number of processes outdoes the number of partitions available. In addition, both schemes must disable both de-duplication and the use of shared libraries.

CacheBar [15] periodically and probabilistically configures the maximum number of ways that a security domain can occupy within each cache set. However, since an attacker can use multiple cooperating threads, CacheBar must limit the number of ways for all unknown processes. This tends to result in unfairness and performance degradation. Moreover, this scheme cannot efficiently support a large number of security domains. STEALTHMEM [11] uses page coloring. It reserves several stealth pages via special colors for security-sensitive data. In their scheme, the operating system ensures that these pages are not evicted by normal cache accesses. Dissimilar to the prevailing up-to-date defensing techniques, STEALTHMEM functions with prevailing commodity hardware and do not need intense alternatives to appliance software. It enacts 5.9% of performance overhead on the SPEC 2006 CPU standard, and amongst 2% and 5% overhead on protected AES, DES and Blowfish, demanding merely amongst 3 and 34 lines of code fluctuates from the unique executions.

Catalyst [129] leverages Intel's CAT (Cache Allocation Technology) hardware mechanism to divide the cache into secure and non-secure partitions, and uses software page coloring within the secure partition to block interference between processes requesting protection. Cache line locking [12] allows processes to exclusively use the cache at the granularity of a cache line. In [7], the negligible methodological improving of cloud are made and prolonged its performance research through employing 15 days in combination with 5 days. The 15-days consequences tend to a significant outcome in the realm of heterogeneous multiple systems caching that is Cloud can offer optimum byte-hit and hit performances at the similar time. In [8], a novel group of further extends one-month traces was employed in contrast with 15-day traces as to witness Clouds patterns in elongated runs. This was to assure its performance practicality and constancy in much genuine atmosphere to offer user confidence in using cloud caching facilities.

EXISTING CACHE REPLACEMENT POLICIES FOR CLOUD COMPUTING

a) Least Recently Used (LRU) Approach: This approach is one of the utmost well-known cache replacement algorithms. It uses the principle of temporal locality. It works on the timestamp by replacing the page which has not been used for the long time. This algorithm is useful only for the conventional system rather than for Internet services because while taking replacement decisions it takes into consideration the time since last access and not access frequency. In Internet services, different user requests are need to be handled which are not inter dependent and has different access characteristics hence optimal performance is not obtained in case of Internet services.[1] LRU is simple and easy to use in conventional system. Has constant time and space overhead, and captures regency or clustered locality of reference. In case of cloud, it does not provide an optimal performance. There is a Lock contention problem, to move a page to most recently used position on every hit it incurs, an overhead which is unacceptable, and access

frequency is not taken into consideration. By scanning, LRU can be easily polluted.

b) Least Frequently Used (LFU) Approach: In LFU, history of accesses is employed for forecasting the probability of upcoming references. The frequency of access counts of all the lines are maintained by the cache and whenever there is a new access, cache replaces the line which has been used least frequently. In Internet environment, if LFU is used there is a higher chance that one or the other document will always be replaced by a new document with high frequency without the considerations of the access probabilities of the replaced and the new documents. When videos and others such large documents are accessed they fill out the cache completely which might destroy the proper functioning of the cache. [1] LFU is simple and easy to use in conventional system. LFU does not offer an optimal performance for Internet systems.

c) First in First Out (FIFO) Algorithm: The system preserves list of entire pages in the memory where initial page at head is the oldest one whereas page at the end is the recent page being added. Whenever there is a request for a page and a miss occurs, system removes a page from the head of the list and adds the new page at the end of the list. This process is known as First in First out (FIFO). But this method fails if it removes the data which is important but not recently used. Hence, FIFO is rarely used in its pure form. Low-overhead paging algorithm. It is not very effective. System needs to keep track of each frame. Sometimes, it behaves abnormally. This behavior is called Beladys anomaly and might throw out important pages. Same problems are faced in case of Internet systems.

d) Adaptive Replacement Approach*:* This approach, retain the track of both recurrently employed pages and freshly employed pages and maintain balance between the features of workload. Workloads contain changing frequency, temporal locality, sequential I/Os, etc. ARC is easy to implement, and its running time per request is essentially independent of the cache size. As compared to LRU it has a performance gain with respect to hit ratio for variable cache sizes. It is a low-overhead algorithm that responds online to changing access patterns. [2] Features like regency and frequency of workload are exploited in a self-tuning fashion, time and space complexity is low, but for wide range of workload and cache size it outperforms LRU. It has a cache hit serialization problem.

e) Greedy Dual-Size with Frequency (GDSF): This methodology allots a smallest priority significant value for a utility function like cost function to the object and using the object size and frequency of the object, the key value is calculated. Largest object gets assigned with lower priority key and object with smaller priority then gets replaced if they are not getting referenced in the near future. [3] Advantages of this algorithm are good hit ratio for both large and small cache size, less cache Pollution and byte hit ratio is lower than other policies.

f) CERA- A Cost Effective Replacement Approach*:* CERA is suggested and developed in [4]. During a cache miss, price is gained to obtain the cache. The price could be measured in two diverse manners: (i) the quantity of traffic produced to obtain the disappeared elements, and (ii) the downloaded potential obtained through the customers to access the lost elements. Price could be specified as the price obtained pertaining to the download potential and traffic latency. In CERA, a benefit value (BV) is allotted to every element pertaining to the priority. Element having the lower BV is substituted whenever the cache is filled. The BV comprises of three partitions such as regularized price, re-accessing likelihood and energetic aging. This approach employs miss price, elements dimension, and access frequency. CERA whenever matched with other approaches like LRU, LFU for dissimilar measures such as hit-ratio, byte-hit-ratio and price minimization rate, it outstrips amongst the entire. In network traffic, it minimizes the entire accessible price. Its efficiency hinges on the assigned features. For high dimensional cache magnitude, it has the finest byte hit ratio, however for not smaller caches.

g) Optimal Cache Replacement Approach: This approach is suggested and developed in [6] and depends on the byte accessible likelihood, b. Files in the cache are allotted having the byte access possibility and are arranged in the non-descending order. Files with maximum possibility is reserved in the bottommost list. List ought to be updated recurrently if the value b of files alters at a quick pace or the list is fixed one. Whenever replacement is needed, formerly the novel files with accessible possibility b and dimension is obtained, the novel files would substitute the initial X documents on the topmost cache only if the byte employs the possibility of novel document is higher compared to the total byte access possibility of entire documents where X is the minute amount documents that has complete dimension higher compared to the novel document. In case of smaller cache dimension, OCR approach outstrips LRU through providing 50% enhancement in cache hit rate. For the higher cache dimension, it do not provide much enhancement in the performance i.e., merely 18%.

h) C-Aware Approach: This methodology is suggested and developed in [5]. In this approach, each demand is not cached however the caching is accomplished depending on the present accessible atmosphere. This strategy comprises of three elements: the native cache controlling approach, C-Aware core, employs the history and the heuristics expectancy, monitors the caching and a tracer that archives the working period for every request. Depending on the previous data, C-Aware core heuristically expects the forthcoming access condition and executes the judgment of cache information. Execution of C-Aware core is performed distinctively for

every cache replacement approach. It chooses whether to cache information chunk essential present access prior to the replacement approach. Once the caching is accomplished, it uses a conventional replacement approach to place the information chunks into cache, or else, it might not cache the information chunk and C-Aware would evade cache and openly obtain the information from the network in the subsequent accesses. The efficiency minimized pertaining to time-taking read functions, for high dimensional cache size and storage server.

PROPOSED EVICTION BIT AND INCLUSIVE CACHE BASED REPLACEMENT POLICY FOR SIDE CHANNEL ATTACK

In this paper, a novel cache replacement policy is introduced to protect against cache aided side channel attacks. Unlike SHARP [20] cache replacement policy, the triggered process not interrupted if the cache is line is present in private cache of spy and victim process. Instead, an Eviction Bit (EB) and presence of inclusive cache is detected distinctively for the spy process and victim processes. The proposed approach focused on attacks leveraging a shared cache in an inclusive cache without interrupting the process. Generally, the victim process and the spy process (es) altogether run on different cores. Each core has more than single level of private caches, and finally shares a lower level of cache known as Last Level Cache which is farther from the CPU with all the other cores. The block diagram for suggested approach is given in fig 1.

Since, a private higher-level cache contains a subset of the lines held in the shared cache level [2]. These are known as inclusion victim lines. These are lines that need to be evicted from a private cache because they are being displaced from the shared cache due to conflicts there. Thus, a novel approach is suggested to mitigate against cache-based side channel attacks that are highly effective and induced negligible average performance degradation. The proposed Eviction Bit based Cache Replacement Policy initially employs two steps prior to replacement algorithm. They are:

i. Determination of Eviction Bit (EB)
ii. Presence of Inclusive Cache

a) Determination of Eviction Bit (EB):

The EB indicates the level of eviction priority of each block. For this purpose, Temporal Positional Predictor (TPP) is employed to predict the position in the shared cache. If prediction has poor location, its EB is considered to be 1 to specify its higher eviction significance other it is 0. On a miss, amongst entire victim applicants in the consistent group, cache lines whose EPBs are 1 are validated initially. TPP begins the exploration from the initial physical manner to the last physical manner. For a victim applicant, TEP recognizes whether it is in interior caches or not. The EB is determined separately for both the victim process and spy process and are listed in a table.

Temporal Positional Prediction: On the cache miss, TEP employs a Replacement Table (RT) to achieve a group of the incoming lines (IB) and victim lines (VB) of spy or a victim system that is chosen through the standard replacement approach. The program counter (PC) which employs IB is also archived. Further pertaining to the below given re-usage series on IB-VB group, the temporal locality of IB is defined as:

- Case 1: if IB is used prior, it signifies that IB has better position.
- Case 2: if VB is used prior, it signifies that IB has bad position.
- Case 3: if IB is not re-used prior eviction, it likewise signifies that IB has bad position.

The initial and third circumstances are flexible to comprehend, due to a re-employed chunk is predicted to have better position, and no re-employed chunks would have bad position locality. The hypothesis of the second circumstance is that victim block (VB) is predicted to have the worst position amongst the entire chunks in the cache. If VB is used prior to IB, it signifies that the position of IB is worse compared to that of VB. Consequently, IB have a bad position. A drenching counter table known as Prediction Table (PT) is employed to know and forecast the temporal position of chunks. PT understands the position depending on the PC of the instruction that employs the chunk, since prior study has represented that the forecasting employing PC is further efficient when matched with other approaches [10], [12], [15]. The whole counters in the PT are initialized to 0. Whenever the position of an IB-VB group is known in the RT, the achieved PC of that group is employed to index an equivalent counter in the LPT. If that chunk exhibits better position, the counter is raised by 1 else lowered by 1. On a miss, the IB refers to the PT along its PC to forecast the position. If its equivalent counter is lesser than 0, it is anticipated to have bad position, and its EB is considered to 1 else set to 0.

Algorithm 1 (Temporal Locality Prediction):

On an access to block X

$for\ each\ valid\ entry\ A\ in\ corresponding\ set\ of\ RT$
  $if\ x.tag == A.IB.Tag$
    $PT[A.PC]++$
    $Invalidate\ A$
  $Else\ If\ x.tag == A.VB.Tag$
    $PT[A.PC]—$
    $Invalidate\ A$
$if\ x\ misses\ in\ the\ cache$
  $y = Pick_{Victim}.Candidate$
$for\ each\ valid\ entry\ A\ in\ corresponding\ set\ of\ RT$
    $if\ y.tag == A.IB.Tag$
      $PT[A.PC]--$
      $Invalidate\ A$
  $if\ RT.Record_x == true$

$B = RT.Select_{victim}(x)$
$B.PC = x.PC$
$B.IB.Tag = x.tag$
$B.VB.Tag = y.tag$
$if\ PT[x.PC] < 0$
$x.EB = 1$
$else\ x.EB = 0$

b) Presence of Inclusive Cache:

The proposed Cache Replacement Algorithm must be aware of what lines within the shared cache are present in the private caches. This is determined using following approaches.

- Using Core Valid Bits (CVB) – Every line in the shared cache is augmented with a bitmap with as many bits as cores. The bit for core i is set if the line is present in core i's private cache. These are the Presence bits used in directory-based protocols [26]. If bit i is set, core i may have the line in its private cache, while if bit i is clear, core i is guaranteed not to have the line in its private cache. Such conservatism stems from silent evictions of non-dirty lines from private caches. These evictions do not update the CVB bits. As a result, the CVB bits will still show the evicting core as having a copy of the line in its private cache. However, correctness is not compromised.
- Using Queries - A shortcoming of the previous design is that it often ends-up assuming that shared cache lines are present in more private caches than they really are. As a result, a process may unnecessarily fail and end-up victimizing its own lines. To solve this problem, whenever a victim applicant is selected, TPP directs its information to internal caches to enquiry whether it is within. Then internal caches finds information and return the reply. To evade directing numerous queries to improve the network traffic, a threshold is set for the maximal queries permitted in single replacement. If any single replacement of LLC is done, the amount of queries has attained threshold, the subsequent applicant would be picked as the victim deprived enquiring internal caches.
- Using Core Valid Bits and Queries - A limitation of the previous design is that it does not scale well. For multicores with many cores, the latency of the queries may not be hidden by the cache miss latency. Moreover, the traffic induced by the queries may slow down other network requests. Consequently, we present a third design that reduces the number of queries.

Cache Replacement Approach:

The Cache Replacement Approach is employed using three different stages slightly similar to SHARP [20]. They are:

- In stage 1, proposed approach considers each line of the set at a time, in the order based on its replacement priority. For each line, it checks if the line is in any private cache. As soon as a line is found that is not in any private cache, it is used as the replacement victim. Victimizing this line will not create any inclusion victim. If no such line is found, the algorithm goes to Step 2.
- In stage 2, proposed approach considers again each line of the set at a time, in the order based on its replacement priority. For each line, it checks if the line is present only in the private cache of Spy. As soon as one such line is found, it is used as the replacement victim. Evicting this line will at worst create an inclusion victim in R. No other process will be affected. If no such line is found, the algorithm goes to Step 3.
- In stage 3, proposed approach considers again each line of the set at a time, in the order based on its replacement priority along with the EB bit. For each line, it checks if the cache line belongs to spy process or victim process. If the cache line belongs to spy process, then checks EB bit and inclusive cache of spy process. If there is no inclusive cache for the bit and EB bit is 1, then this cache line is used as the replacement victim only for the spy cache line replacement. Similarly, for the victim process also, EB bit and inclusive cache is detected. If block is not inclusive cache and EB bit of victim process is 1, then this cache line is used as the replacement victim only for the victim cache line replacement. If the both the condition does not satisfy, then the process interrupt is triggered for this cache line.

Algorithm 2 (Proposed Methodology):

A. Initially, Eviction Bit (EB) is determined using Algorithm 1 separately for Spy process and Victim Process.
B. Presence of the type of the Inclusive Cache is determined using core value bits and Queries.
C. If the selected cache line is not present in the private cache line victim process
    Then selected as replacement victim
D. Else If the selected cache line is not present in the private cache line spy process
    Then selected as replacement victim
E. Else
    a. If cache line is in Spy process
        i. If EB=1 && not inclusive cache
            Then selected as replacement victim
    b. Else if cache line is in Victim process
        i. If EB=1 && not inclusive cache
            Then selected as replacement victim
    c. End if
F. End If

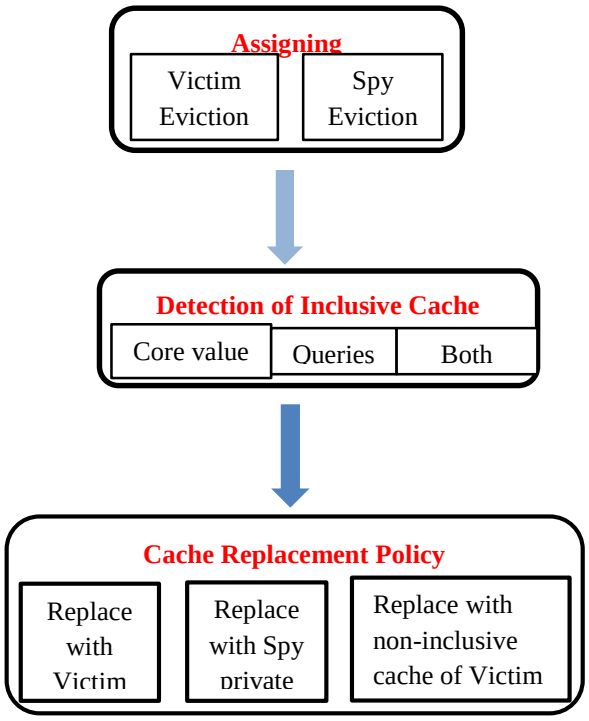

Figure 1: Block Diagram of the Proposed Cache Replacement Policy

EXPERIMENTAL RESULTS AND ITS ANALYSIS

The experimental result for the proposed Eviction Bit and Inclusive Cache based replacement policy is carried out using 8 core processors in which 7 are victim processes and 1 is a spy process that shared a single Last Level Cache Hierarchy. Every core process has also has its own private L1 caches where some of them are also inclusive cache. The inclusive cache shares the cache line along with the LLC. The Experimental setup for the proposed approach is implemented using a local Open Stack Icehouse deployment for the test bed armed with 2.50 GHz 64-bit Intel Xeon CPU L5420 processor having 8-cores, 16 GB RAM, 500 to 1000GB disks, and two network interfaces with 100Mbps and 1Gbps speed. Every System executes on Ubuntu 14.04. The power sites are aligned to execute the CPU continuously at complete speed so as to minimize the measurement noise. The virtual machines are employed in the 64-bit version of Windows 7 Enterprise Edition and have 8 GB of RAM. This is suggested minimal quantity of memory for SPEC 2006 CPU standard.

Metric for Side Channel Leakage

1. Side-Channel Vulnerability Factor (SVF): SVF is a parameter and method for evaluating a side channel's leakiness. It depends on the surveillance that there are two appropriate parts of data in a side channel attack: the data that an invaders is attempting to attain (delicate information), and information that an invaders could truly acquire. To estimate leakiness, it merely ought to calculate the correlation amongst these two parts of datasets.
2. Signal-to-noise ratio (SNR): the SNR of the signal x as is given as $\frac{x(k)-\bar{x}}{\sqrt{Var(x)}}$, where $x(k)$ is the value of the signal taken at the key, and $\bar{x}$ and $Var(x)$ are the mean and variance of x correspondingly. SNR is used to compare the level of the desired signal to the level of the background noise. It measures how complex it is for the invaders to obtain beneficial knowledge from the noise.

Table 1: Comparison of Defending Techniques for Side Channel Attacks using SVF and SNR

Defending Techniques	SVF	SNR
Hybrid Mitigation	0.15	0.268
Dynamic VM placement Policy	0.135	0.215
SHARP	0.12	0.197
Proposed Approach	0.115	0.19

Algorithm Efficiency Evaluation

The performance efficiency of the algorithm is analyzed with five different performance metrics for side channel attacks in cloud computing. Better Approaches provide better efficiency for more number of performance measures. They exhibit an outcome to the tradeoff presented through dissimilar performance measures. They are Hit rate, lose rate, resource loss, byte hit rate and delay saving ratio. Three of them are defined as follows:

- Hit-rate: This is the mostly employed cache performance measure. Techniques that indulge smallest elements such as dimension, function-aided policies with a robust weighting of object magnitude optimizes for hit-rate. This performance measure is exciting for customers that want to have a higher proportion of localized hits.
- Byte-hit-rate: Policies that tend to eradicate higher elements enhance the hit rate and minimizes the byte-hit-rate. The dissimilar weighting of elements magnitude effects the tradeoff. This performance measures is stimulating for ISPs whenever they attempt to diminish the download bulk from the Internet.

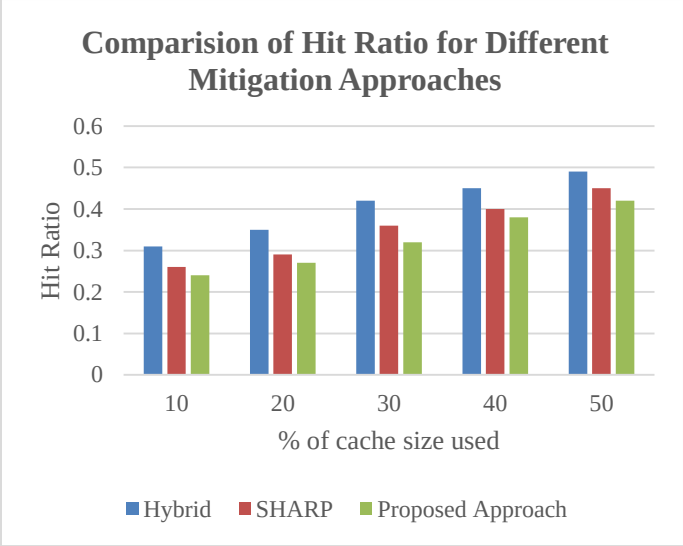

Figure 2: Comparison of Hit Ratio for Mitigations Approaches

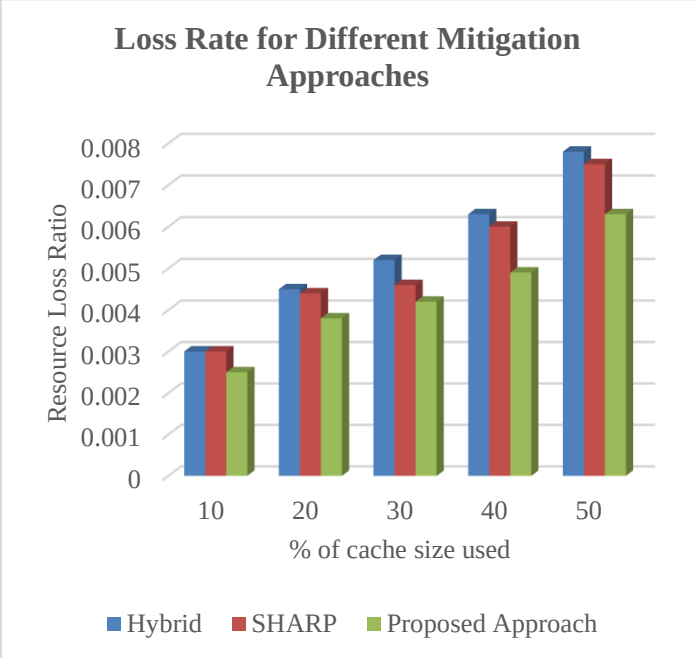

Figure 3: Comparison of Loss Rate for Mitigations Approaches

The comparison of Hit ratio, loss rate, Resource Loss and Byte-Hit ratio for different Mitigation approaches of side channels attacks are given in Fig 2, Fig 3, Fig 4 and Fig 5 respectively. From Fig 2, Fig 3, Fig 4 and Fig 5, it is clearly shown that the proposed cache replacement approach has much smaller hit ratio when compared to the other existing cache based side channel defending technique like SHARP and Hybrid Mitigation Approach. Due to the addition of an eviction and targeting the non-inclusive cache as the replacement victim, the hit ratio of the spy process has minimized through Last Level cache. This intuitively raised to a decrease in loss and resource loss unnecessarily.

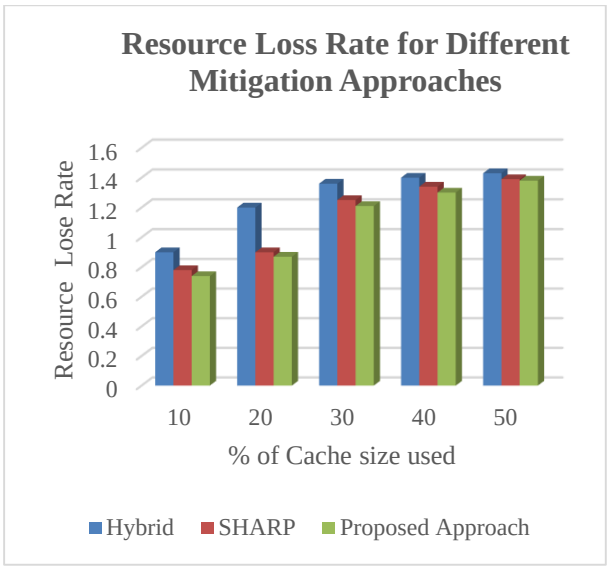

Figure 4: Comparison of Resource Loss for Mitigations Approaches

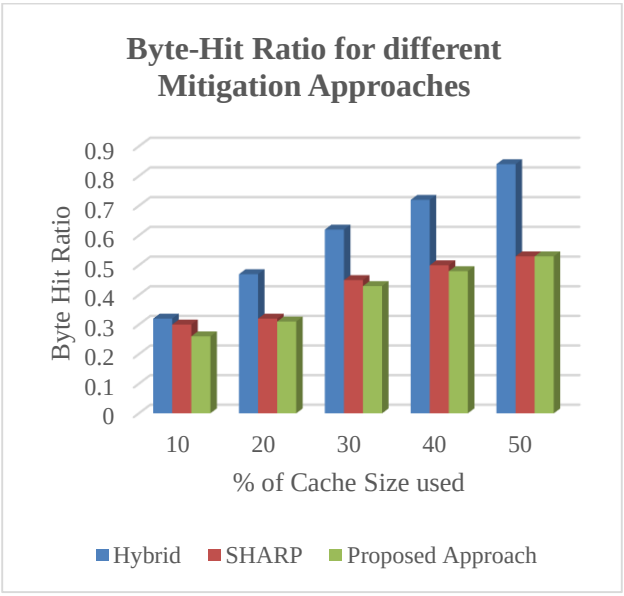

Figure 5: Comparison of Byte-Hit Ratio for Mitigations Approaches

CONCLUSIONS

This paper suggested a novel Cloud cache replacement policy to mitigate against cache aided side channel attacks in cloud computing. This approach intends to improve the cloud efficiency and scalability issues and minimize the threats that attack the Last Level Cache through inclusive cache. An Eviction Bit is assigned is assigned to the spy cache line and victim cache line that determined the replacement of the cache line as victim replacement. Then presence of Inclusive Cache is detected in the LLC with respect to spy and victim process. After this, the cache replacement policy is executed that seeks

for the EB and inner cache status during replacement in the Shared Last Level Cache Lines. The experimental results of the suggested approach showed that, it has improved hit ratio, loss rate, resource loss and byte-hit ratio when compared existing replacement technique such as SHARP and Hybrid Mitigation Techniques.

REFERENCES

[1] Kapil Arora1, Dhawaleswar Rao Ch, "Web Cache Page Replacement by Using LRU and LFU Algorithms with Hit Ratio: A Case Unification" *International Journal of Computer Science and Information Technologies*, Vol. 5 (3), 2014, 3232 3235.

[2] Nimrod Megidd, Dharmendra S. Modha, "Outperforming LRU with an Adaptive Replacement Cache Algorithm", *Published by the IEEE Computer Society*, 2004.

[3] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich and Tai Jin, "Evaluating Content Management Techniques for Web Proxy Caches".

[4] Rassul Ayani, Yong Meng Teo and Peng Chen, "Cost-based Proxy Caching", *Proceedings of the International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, December 2002.

[5] Zhu Xudong, Yin Yang, Liu Zhenjun, Shao Fang, "C-Aware: A Cache Management Algorithm Considering Cache Media Access Characteristic in Cloud Computing", *Mathematical Problems in Engineering,* 2013.

[6] K. H. Yeung and K. W. Ng, "An Optimal Cache Replacement Algorithm for Internet Systems", *IEEE Transaction on Computers*, 1997.

[7] T. Banditwattanawong and P. Uthayopas, "Cloud Cache Replacement Policy: New Performances and Findings," *Proceedings of 1st Annual PSU Phuket International Conference*, 2013.

[8] T. Banditwattanawong and P. Uthayopas, "Improving Cloud Scalability, Economy and Responsiveness with Client-Side Cloud Cache," *Proceedings of 10th IEEE International Conference ECTI-CON 2013*, 2013.

[9] Ralf Hund, Carsten Willems, and Thorsten Holz. " Practical timing side channel attacks against kernel Space ASLR", *In IEEE Symposium on Security and Privacy*. IEEE, 191–205, 2013.

[10] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds" *In Proceedings of the 16th ACM Conference on Computer and Communications Security*, ACM, New York, NY, USA, 199–212, 2009.

[11] Taesoo Kim, Marcus Peinado, and Gloria M. Ruiz, "STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud", *In Proceedings of the 21st USENIX Conference on Security Symposium*, USENIX Association, Berkeley, CA, USA, 2012.

[12] Zhenghong Wang and Ruby B. Lee, "New cache designs for thwarting software cache-based side channel attacks", *In Proceedings of the 34th Annual International Symposium on Computer Architecture*, ACM, New York, NY, USA, 494–505, 2007.

[13] Michael M. Godfrey, "On the prevention of cache-based side-channel attacks in a cloud environment", Master's thesis, Queen's University, 2013.

[14] YaoWang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, G. Edward Suh, "SecDCP: secure dynamic cache partitioning for efficient timing channel protection", *In Proceedings of the 53rd Annual Design Automation Conference*, ACM, New York, NY, USA, 2016.

[15] Ziqiao Zhou, Michael K. Reiter, and Yinqian Zhang, "A software approach to defeating side channels in last-Level caches", *In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, New York, NY, USA, 871–882, 2016.

[16] AMAZON, INC. Amazon Elastic Compute Cloud (EC2). http: //aws.amazon.com/ec2, 2012.

[17] Ion, I., Sachdeva, N., Kumaraguru, P., c Apkun, S. "Home is safer than the cloud! Privacy concerns for consumer cloud storage" *In Proceedings of the Seventh Symposium on Usable Privacy and Security* (2011), pp. 13:1–13:20.

[18] Jansen, W., And Grance, T, "Guidelines on security and privacy in public cloud computing" *NIST Special Publication* 800-144, December 2011.

[19] Microsoft, Inc. Microsoft Azure Services Platform. http: //www.microsoft.com/azure/

[20] Mengjia Yan, Bhargava Gopireddy, Thomas Shull, Josep Torrellas, "Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Attacks", *Proceedings of the 44th Annual International Symposium on Computer Architecture,* Pages 347-360, June 24 - 28, 2017.