International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-2, July 2019

Integrated Cache Scheduling Replacement Algorithm to Reduce Cache Pollution

Kavita Kalambe, Sunita Rawat, Nilesh Korde, Gaurav Kawade

Abstract*: Cache memory management has recently become one of the most crucial research topics in the area of high performance computing. The cache memory has gained popularity due to its capability to offer faster access compared to main memory so processor can access data directly through cache very efficiently. As processor execution speed is faster than accessing memory so the goal is to keep programs rapidly accessible for processing. Cache memory helps not only to access data quickly but also improves the performance of the system. Various cache management policies have been developed to improve the performance of the system. In this paper we focus the issues related with the existing replacement policies and present a new algorithm to overcome these issues. In this paper, we proposed Integrated Cache Scheduling Replacement Algorithm (ICSRA) that will remove the object on the basis of integration of all the parameters like size, frequency and recency if the cache is full. This algorithm consider all size of object to replace with the condition of frequency and timestamp parameter to reduce the cache pollution [1]. Cache pollution means the cache unnecessary waste its space by holding such an object which is no longer be accessed. ICSRA helps to enhance the utilization of cache memory in an efficient manner as well as maximize the cache hit ratio to improve the performance of the system. A detailed classification has also been included based on different parameters which are depending on the analysis of the existing techniques.

***Index Terms*: Cache memory, Cache hit ratio, Cache pollution, Cache Replacement Algorithm, ICSR.**

I. INTRODUCTION

Due to rapidly advanced computing technology, computer systems are much more powerful with increasing computing capability but at the same time complexity in system architecture to maintain the performance of memory as well as to handle the consistent data storage on hierarchical levels of memory gets increased. Fig. 1 shows the typical hierarchical view of memory structure. In this fig. memory is organized in an hierarchical form by keeping three main factor in mind i. e. memory access speed, storage capacity and memory cost. As the memory size grows, memory access speed decreases. As speed of CPU is very fast, so providing data to CPU must be as so fast as possible. CPU can access data from closer memory as much as faster than remote memory. The Process of fetching data from main memory is slower than executing instruction in the processor. So the concept of high speed cache memory is introduced. But the storage capacity of cache memory is very low. Due to limited size of cache memory its space needs to be utilized properly. For this various cache management policies has been developed over last few decades to improve the performance of the system. Overall Performance of system is mostly depends on memory performance and memory performance depends on how efficiently and quickly data is provided to the processor for processing.

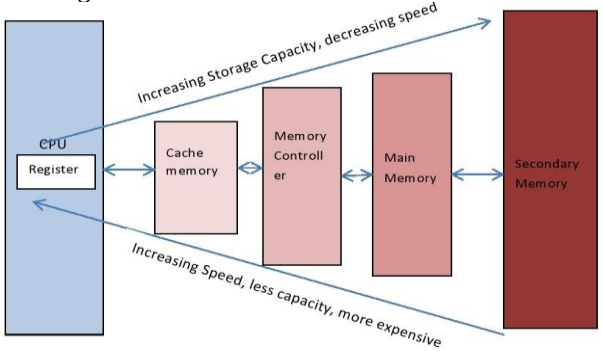

Fig 1. Hierarchical View of memory Structure

There are some parameters affecting the cache performance. The most important parameters are accessing frequency, the last access time, cache block size and cache object size [2]. There is a huge amount of information stored on higher levels of memory because of its capacity, but while accessing this information CPU requires a more searching time to search the data into this remote area of memory. Accessing data from closer small capacity of memory requires less time than remote large capacity of memory. Due to limited capacity of cache memory, it cannot hold whole data in a memory at a time, so cache evict and replace data on demand of user. For this mechanism there are several cache replacement algorithm developed by many researchers like FIFO (First In First Out), LRU (Least Recently Used), LFU (Least Frequently Used) [3], Optimal, SIZE, Function Based[15], Random, Weighting based algorithm[14] and Weighting size and cost algorithm[1] etc.

Revised Manuscript Received on 30 July 2019.
***** Correspondence Author
Kavita Kalambe*, Assistant Professor, Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management, Nagpur, India.
Sunita Rawat. Assistant Professor, Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management, Nagpur, India.
Nilesh Korde. Assistant Professor, Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management, Nagpur, India.
Gaurav Kawade, Assistant Professor, Computer Science and Engineering Department, Shri Ramdeobaba College of Engineering and Management, Nagpur, India.

These algorithms have their own different mechanism to decide which object should have to be replaced. The main goal of using the cache replacement algorithm is to increase the cache hit ratio and utilize the cache space efficiently. In this paper we propose a new algorithm named Integrated Cache Scheduling Replacement Algorithm (ICSRA) to improve the cache performance. In this algorithm we are integrating the advantageous concept of existing algorithm to make a decision to replace the proper object. ICSRA will find the probability of object using the most important parameters like frequency, size, and last access time. If the probability of object is high then that object needs to be keep in the cache because it's chances to occur again in the near future is high. The consequence is this algorithm will give the better cache hit ratio compared to existing algorithm.

This paper presented as follows : The section II reviews literature review, and section III describes the proposed methodology, section IV goes with the design and working of proposed algorithm i. e. Integrated Cache Scheduling Replacement Algorithm (ICSRA). Finally, Section V concludes the paper with a short summary.

II. LITERATURE REVIEW

Previous work investigated the cache memory management techniques and evaluating the performance of the system in the field of high performance computing. In this section we summarizes the previous related work on various cache replacement algorithm and which strategy used by that algorithm to replace the object. When CPU sends request for object which is not found in the cache and cache is full. It needs to be fetched from higher levels of memory and create a space for storing a new object by replacing irrelevant object. So to determine which object should be removed to make space for the new objects [5], cache replacement algorithm must be applied. An excellent replacement algorithm is that in which less number of replacement process is done so the cache hit ratio is increase. Nowadays many researchers are diverting towards the developing enhanced cache replacement policy [9] to improve the speed of computing. At present, most widely used cache replacement algorithms are: First in First out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU), Randomized-based Algorithm (RANDOM), SIZE Algorithm, Function-based Algorithm, and Weighting-based Algorithm [14], Weighting Size and Cost Replacement algorithm (WSCRA) [1].

A.First In First Out (FIFO):

FIFO is a very simple replacement policy and easy to implement policy like queue, and is also known as a round robin. This algorithm doesn't consider any parameter like frequency, time, memory access cost, size etc. to determine which object should have to be replaced. When cache is full, it simply replace the object which come first with the requested object and the next replacement will be the second object placed in the cache and so on….Although FIFO is simple and easy to implement but its consequence is very poor. It is failed to achieve better cache hit ratio.

B. Least Recently Used(LRU):

LRU algorithm is the most widely used cache replacement algorithm [6,8]. It involves maintaining queue data structure of the objects in the current requested set. Thus, the MRU (Most Recently used) object is placed at one end and the LRU (Least Recently Used) [6] object which is ready to replace at the other end. If any requested object is existing in the cache, it is moved at the MRU position in the queue and rest of the object is shifted to LRU side. If requested object is not exist and cache is full, then the object at the LRU position is evicted and the new object is inserted into the MRU position [4]. This algorithm only considers the time parameter to make a choice of object for replacement. In other words the object in the cache that hasn't been used for the longest time is replaced. The advantage of this policy is that it has very little memory overhead to decide which object has to be replaced and takes O(1) to evict and replace the object from a list. But the disadvantage is that it has high memory access cost because it doesn't consider frequency and size parameter [1].

C.Least Frequently Used (LFU):

Least Frequently Used cache replacement algorithm which removes object which have least frequently used and then place a new requested object in free space [5,8]. In this algorithm, frequency factor is the primary concern to select the object for replacement. In this algorithm, cache maintains reference counters which will count how many times the object is requested by user and replace the object whose frequency is minimum. It assumes that chances of the object having maximum frequency will be requested in the near future again are more. It is very simple and easy to implement and have a better performance also[8]. But if the largest size of object having high frequency before a long time ago, it will be occupy more space in the cache although it will no longer be accessed. So the consequence of this algorithm is ''cache pollution'' [1].

D.Randomized-based Algorithm:

This algorithm doesn't consider any parameter while making a choice of object to replace. If there is no space in the cache to store the new requested object, it will choose any random object to replace for creating space to fit the new object inside the cache[14]. This may lead to remove popular data also and may decrease the cache hit ratio. The advantages of this algorithm are that it is easy to implement and also reduces the overhead of the cache while making a selection of object to replace. But the disadvantage is that it failed to give better performance and hence its cache hit ratio is also poor [15].

E. SIZE algorithm

This algorithm is designed in order to keep in mind the space limitation of cache[16]. It considers the Size parameter of object for the replacement. If new requested object is not found in the cache and cache is full then it will replace the largest size of object without considering the frequency and recency of object. The advantage of this algorithm is that it will create a more space to fit the maximum number of smallest size of object at a time but the disadvantage of this algorithm is that cache holds the smallest size of object whose frequency is less and it is no longer be accessed. So unnecessary wasting the space. Also the largest size of object which is having more frequency and is most recently used, if it is replaced then next time when it is requested then it suffers from high memory access cost for fetching largest size of object.

It considers time as the second parameter when the two objects have same largest size. So the consequence of this algorithm is also create a ‘‘cache pollution’’ [1].

F. Function-based Algorithm:

This algorithm determines the object value for each object in the cache [2] by using the factor like object size, access latency, cost and etc. then it removes the object which have lowest object value when the cache is nonempty. Greedy dual-Size (GD-SIZE) algorithm [2] is used to calculate object size, access latency, cost and etc. factors. Original GD-SIZE algorithm is designed by Young. It deals with the case if two object have the same size but it takes different cost for accessing documents from the secondary storage. These algorithms only consider the size and cost parameter. The object value is calculated as:

$$H(P) = C(P)/S(P) + L \qquad \ldots\ldots (1)$$

Where P is the object in the cache, C(P) is the cost of retrieved object P, S(P) is the size of object P, L is the offset value. It doesn't consider the frequency and other parameter.

G. Weighting-based Algorithm :

This algorithm calculates the weight of each object on the basis of frequency and recency of the object. When the cache is full, the object having highest weight value is removed [14]. The weight is calculated by the following formula is:

$$W(P) = \frac{L(P)}{F(P) \times \Delta T(P)} \qquad \ldots\ldots (2)$$

Where L(P) is the last referenced time of object P, F(P) is the frequency of object P, T(P) is the average interval time between every two consecutive request.

This algorithm works with the combination of LRU and LFU. It considers both the parameter frequency and time. It works well when caching the smaller size of object. But the object having largest size then it will create again the cache pollution.

H. Weighting Size and Cost replacement Algorithm:

In this algorithm, researchers use the existing Weighting based algorithm in[14] is named Weighting Replacement algorithm(WRP), to remove its limitation by adding the size and cost factor. WSCRA recalculates the weight of cache object by using the formula is as follows:

$$WC(P) = \frac{L(P)}{F(P) \times \Delta T(P)} \times \frac{S(P)}{csv(P)} \qquad \ldots\ldots (3)$$

Where L(P) is the recency of the object P in the cache, F(P) is a frequency which means the total number of times that object P has been referenced and the S(P) is the size of object P. ΔT(P) is the difference between the last referenced time of object and the time gap between two consecutive request. Csv(P) is the cost saving value calculated by the formula is as:

$$csv_{(P)} = c_{(P)}\, s_{(P)}\, h_{(P)} / \sum_{P=1}^{n} c_{(P)}\, s_{(P)}\, r_{(P)} \qquad \ldots\ldots (4)$$

Where $c_{(P)}$ is the network traffic consumption by object P, $s_{(P)}$ is the size of object P, $h_{(P)}$ is how many valid copy of P is found in the cache, and $r_{(P)}$ is the total times a data was requested.

This algorithm every time when the object is requested it recalculates the weight WC of every object and make a decision to replace the object on the basis of highest WC value. This algorithm has a better cache hit ratio and good performance also. But this is very complicated to implement because for calculating WC value it requires to compute a lot of factors such as recency of object, frequency of object, time difference, the most important factor is cost saving value. The factors like recency, frequency, time difference and size is easy to compute but to compute the cost saving value requires to search the entire cache to get the number of valid copy of object and the network traffic consumption of object and the summation of the product of network traffic, size and the total times the object was requested. To compute and maintain this information of each requested object is very difficult and time consuming process. Whenever the object is requested the value of all these factors may change so this algorithm recalculates the WC value. So the complexity of this algorithm is very high. In order to reduce the memory access time we use the cache replacement algorithm but this algorithm will take more time to decide which object should have to be replaced.

III. PROPOSED METHODOLOGY

Going through the literature review of existing cache replacement algorithm**,** we will find that every algorithm is having some advantages as well as some disadvantages. So we proposed a new algorithm named Integrated Cache Scheduling Replacement Algorithm (ICSRA) algorithm. This algorithm uses the mixing features of existing cache replacement algorithm and we add some new concepts in this algorithm to overcome the disadvantages of that existing algorithm. Our algorithm is typically based on priority of the parameters of object to decide which object should have to be replaced. Our Cache Replacement Algorithm associates with the integration of factors like frequency, last accessed time and size to find the least effective object in the cache for replacement. This algorithm will considers both the size of object i. e. largest size of object and smallest size of object having two parameters frequency and timestamp but the priority given to the largest size of object. So that it will release more space to occupy next object. Timestamp is the latest time of requesting the object. Frequency is the counter for how many times the object is requested. This algorithm not only decreases the memory access time but also reduce the memory access cost.

IV. ICSRA ALGORITHM

There are two main phases in this algorithm: CPU process request and evict and replace object using Integrated Cache Scheduling Replacement Algorithm.

A.CPU PROCESS REQUEST

In this phase, when user sends the request for any object, it will process by the CPU. CPU will search the requested object into the cache first. If it finds then it will easily fetch from cache and processed it, it is called as cache hit. But if not found in the cache then it incurs cache miss. It will search into the main memory and so in the higher levels of memory if it not found in there also. And fetch it into the cache first. In this algorithm, it will maintain the information of the object like size of object, frequency and recency of the object when the object is fetch into the cache. This algorithm will stored the objects in the cache in an decreasing order of the Object size. Whenever new object gets inside the cache or the cache hit occurs, it will update the frequency and timestamp block of the database.

//Defining Variables and Set Initial Values
$OBJ_{TIMESTAMP} \leftarrow 0$, $OBJ_{FREQ} \leftarrow 1$
$CACHE_{MAXSIZE}$ – Maximum Cache Size
$CACHE_{SIZE}$ – Total size occupied by cache object
$CACHE_{CURRENTCAPACITY}$ – Remaining space in the cache
OBJ_{SIZE} – Currently requested object size
$CACHE_{CURRENTCAPACITY} = CACHE_{MAXSIZE} - CACHE_{SIZE}$

Algorithm 1: CPU Process Request

1. function **INSERT_INTO_CACHE**(OBJ, request_time)
2. if (Object ∈ Cache) then
3. Hit_Count = Hit_Count +1
4. OBJ ← Cache[object] // CPU get requested object
5. $OBJ_{TIMESTAMP} \leftarrow$ request_time
6. $OBJ_{FREQ} \leftarrow OBJ_{FREQ} + 1$
7. else if (Cache is EMPTY OR $OBJ_{SIZE} \leq CACHE_{CURRENTCAPACITY}$) then
8. $CACHE_{SIZE} = CACHE_{SIZE} + OBJ_{SIZE}$
9. Cache[object]← OBJ
10. SORT(Cache , OBJ_{SIZE})
11. else
12. Call **EVICT_AND_REPLACE** (Cache, OBJ)
13. endif
14. UPDATE Cache Database
15. Call **INSERT_INTO_CACHE**
16. endif

B. Integrated Cache Scheduling Replacement Algorithm

There are some cases to decide which object should be keep ready for replacement if the cache is nonempty or having not sufficient space to fit the new requested object into the cache. In this algorithm first give priority to the largest size of object, and then check the frequency of object and timestamp If, the requested object is not found in the cache, the system fetch it from the main memory or from secondary storage. In addition, when cache miss occure, it triggers the ICSRA algorithm to evict and replace the old object and also it calls if the requested object does not fit in the current capacity of cache or if cache is full. Calculate Midium Frequency value MID_{FREQ} and Average time from overall cache database to apply the condition for object replacement.

$$MID_{FREQ} = \frac{(MAX_{FREQ} + MIN_{FREQ})}{2} \quad \ldots..1$$

$AVG_{TIME} = avg(OBJ_{(i)TIMESTAMP}, OBJ_{(i+1)TIMESTAMP}, \ldots. OBJ_{(n)TIMESTAMP}), \forall OBJ_{(i)\ TIMESTAMP} \in Cache$

Case 1 : Largest size of object with minimum frequency or maximum timestamp.

In this case the largest size object having minimum frequency and maximum timestamp occupy unnecessary largest space of cache memory for a longer time, So due to this cache memory cannot hold the more object inside the cache this will decrease the cache hit ratio. Probability of requesting largest size of object having minimum frequency is very low. so in this algorithm, if cache is full and this type of object cross the threshold time limit, it should have to be replace from the cache but keep the reference of that object so that it is helpful when next time user send request for that object then using its reference it will easily find from the next memory level. So this will save its searching time. In this case largest space will be released to cache new request.

Case 2 : Smallest size of object with minimum frequency or maximum timestamp.

Removal of this type of object from the cache overcome the problem occurred in SIZE algorithm. In SIZE algorithm [17], it considers only the largest size of object for replacement. The advantage of this algorithm is that it will create a more space in the cache so that more number of objects can store inside the cache. But also there are two major disadvantages of this algorithm. First is that object of small size resides in the cache for a long time even if its frequency is too small and exceeds its timestamp. Even though the object size is small but it consumes unnecessary space inside the cache for a long duration. Second disadvantage is that every time replace the object of largest size which can lead to increase the memory access cost. Because if after replacing largest size object although its frequency is more and next request come for that object then it needs to be access that object again. Accessing largest size of object requires more cost than accessing smallest size of object.

Algorithm 2: ICSRA Algorithm

//Consider the object stored in the cache in an decreasing order of the Object size

// N is the total number of objects in the cache

1. function **EVICT_AND_REPLACE** (Cache, OBJ)
2. i = 1
3. While (i ≤ N)
4. If (Cache[$OBJ_{(i)FREQ}$] ≤ MID_{FREQ} AND
5. Cache[$OBJ_{(i)TIMESTAMP}$] ≥ AVG_{TIME}) then
6. Cache. REPLACE (Cache[$OBJ_{(i)}$])
7. $CACHE_{CURRENTCAPACITY} = CACHE_{CURRENTCAPACITY} + OBJ_{(i)SIZE}$
8. If ($OBJ_{SIZE} \leq OBJ_{(i)SIZE}$)
9. break loop
10. Else
11. i++ // check for next largest size of object

Retrieval Number: B2868078219/19©BEIESP
DOI: 10.35940/ijrte.B2868.078219
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

12. End while

In this algorithm, objects are placed inside the cache in an decreasing order of the size so in the cache database processed objects from largest size to smallest size. This algorithm first compare the frequency of object i with the mid frequency of the objects in the cache and timestamp of object i compare with the average time of objects in the cache. If both the conditions are true that means object resides in the cache is not called by user from a longer duration then it will choose that object for replacement and after that it will confirm that the size of selected object is greater than or equal to newly requested object. So that it will replace the next object to create sufficient space for it. Then the selected object is replaced but if size is not sufficient to fit the new object then it will replace that object go for next object and repeat the same process. If all conditions are satisfied then it will encounter out of the loop and insert the object in the free space of the cache and update the cache database.

V. CONCLUSION

The Integrated Cache Scheduling Replacement algorithm is the simplest algorithm which will enhance the performance of the system as compared to the existing replacement algorithm. In this algorithm we assume that the probability of requesting largest size of object having maximum frequency and minimum timestamp in the near future is more so this type of object should keep in the cache until it fails to satisfy any one of its condition. This concept will maximize the cache hit ratio and also saves the cost of memory access. This algorithm will first check the condition for largest size object and then second largest size object next third and so on according to the decreasing order of size. This concept will help to give the chance to every object even the smallest one. This will avoid the cache pollution created by smallest size of object having minimum frequency and maximum timestamp.

REFERENCES

1. Tinghuai ma (member, ieee), yu hao, wenhai shen, yuan tian, and mznah al-rodhaan, “an improved web cache replacement algorithm based on weighting and cost ”, in ieee access , digital object identifier 10.1109/access.2018.2829142, volume 6, 2018.
2. Tinghuai ma (member, ieee), jingjing qu, wenhai shen, yuan tian, abdullahal-dhelaan, and mznah al-rodhaan, “weighted Greedy Dual Size Frequency Based Caching Replacement Algorithm”, in IEEE Access, volume 4, PP(99):1-1,2018.
3. Sreya Sreedharan , Shimmi Asokan, “A cache replacement policy based on re-reference count” in International Conference on Inventive Communication and Computational Technologies (ICICCT) 2017, IEEE.
4. André Pessoa Negrão, Carlos Roque, Paulo Ferreira and Luís Veiga, “An adaptive semantics-aware replacement algorithm for web caching”, Journal of Internet Services and Applications 6:4 https://doi.org/10.1186/s13174-015-0018-4, Springer 2015.
5. M. Gallo et al.,"Performance evaluation of the random replacement policy for networks of caches", Performance Evaluation, vol. 72, pp. 16-36, 2014.
6. W. Meizhen, S. Yanlei, and T. Yue, ‘‘The design and implementation of LRU-based Web cache’’, in 8th International Conference on Communications and Networking in China (CHINACOM) , Guilin, China,pp. 400–404, Aug. 2013, IEEE ***Xplore***.
7. T. S. Warrier, B. Anupama, and M. Mutyam, "An application-aware cache replacement policy for last-level caches," Architecture of Computing Systems ̶ ARCS 2013, pp. 207-219: Springer, 2013.
8. Kapil Arora, Dhawaleswar Rao Ch, “Web Cache Page Replacement by Using LRU and LFU Algorithms with Hit Ratio: A Case Unification”, in International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 5 (3) , 3232 – 3235, 2014.
9. Areej M. Osman and Niemah I. Osman, “Comparison of Cache Replacement Algorithm for Video Services ”, in International Journal of Computer Science & Information Technology (IJCSIT) Vol 10, No 2, April 2018.
10. D. Singh, S. Kumar, and S. Kapoor, ‘‘An explore view of Web caching techniques,’’ Int. J. Adv. Eng. Sci., vol. 1, no. 3, pp. 38–43, 2011.
11. J. Alghazo A. Akaaboune N. Botros , “SF-LRU cache replacement algorithm” in IEEE *Xplore,* ISSN: 1087-4852, DOI: 10.1109/MTDT.2004.1327979,2004.
12. Aline Zeitunlian and Ramzi A. Haraty, “An Efficient Cache Replacement Strategy for the Hybrid Cache Consistency Approach”, in International Journal of Computer and Information Engineering Vol:4, No:3, 2010
13. K.M. AnandKumar , Akash S , Divyalakshmi Ganesh , Monica Snehapriya, “A hybrid cache replacement policy for heterogeneous multi-cores” in International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, IEEE
14. K. Samiee, ‘‘A replacement algorithm based on weighting and ranking cache objects, International Journal of Hybrid Information Technology Vol.2, No.2,pp. 93–104, 2009.
15. Aditya Bhutra, “Reviewing various Cache Replacement Policies”, available at : https://www.researchgate.net/publication/301546620.
16. Waleed Ali, Siti Mariyam Shamsuddin, and Abdul Samad Ismail, “A Survey of Web Caching and Prefetching”, Int. J. Advance. Soft Comput. Appl., Vol. 3, No. 1, March 2011.

AUTHORS PROFILE

Kavita Kalambe Shri. Ramdeobaba College of Engineering and Management, (kalambekb@rknec.edu), Department of CSE, Nagpur, India. K. Kalambe is currently working as Assistant Professor in the Department of CSE at SRCOEM. She received B. E. degree from KDK college of engineering, Nagpur in 2010 and M. Tech degree in CSE specialization from G. H. R. I. E. T. W. Nagpur, Nagpur University in 2015. She has ISTE membership. Her current research interests include Algorithm designing, High Performance Computing Architecture, Wireless Sensor Network, Microprocessor and Microsystems.

Sunita Rawat working as a Assistant professor in Computer Science and Engineering department at SRCOEM, Nagpur, India. She is pursuing PhD in the field of NLP. She has publications in National and International Conferences. And also 3 publications in Scopus Indexed journal. She has ISTE membership.

Nilesh Korde has received Master of Technology degree in Information Technology from YCCE an Autonomous Institute at Nagpur University India in 2014. Presently he is working as an Assistant Professor in Computer Science and Engineering Department at Shri Ramdeobaba College of Engineering and Management Nagpur (An Autonomous Institute). He has published research papers on Parallel Computing, IOT, Cache Management and Clustering. Recently he is working on Natural language Processing, Data Retrieval and Machine Learning.

Gaurav Kawade has received Master of Technology degree in computer Science and Engineering from Visvesvaraya National Institute of Technologyisv.Nagpur in 2015. Presently he is working as an Assistant Professor at Shri Ramdeobaba College of Engineering and Management, Nagpur. His research areas are Wireless Sensor Network, Network Security, Parallel Computing. He has published research papers in National and Internal Journals and has 2 conferences proceeding in Scopus Indexed journal.