IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Sentient OS: Intelligent CPU Scheduling in Operating Systems - A Systematic Survey

Vishal Borate
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon, Pune, India

Alpana Adsul
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon, Pune, India

Srushti Kulkarni
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon , Pune, India

Jayesh Patil
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon, Pune, India

Rakesh Salunke
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon, Pune, India

Niraj Suryavanshi
Department of Computer Engineering
Dr. D.Y. Patil College of Engineering and Innovation Varale, Talegaon, Pune, India

***Abstract*—**Process scheduling plays an essential role in multitasking operating systems. Some of its benefits include high CPU utilization, short waiting times, short turnaround time, and better responsiveness. Round Robin is one of the most widely used techniques today due to its simplicity and fairness. However, it suffers from certain limitations based on a fixed time quantum which is dependent on different process burst times. This leads to increased context switching, wasted resources, and poor performance for processes that have huge differences in execution time. In the view of these issues, there is a need for the ADBRR scheduling algorithm. This self- tuning approach, called ADRR, adjusts time quantum based on changes in the process's burst characteristics. It means that ADRR treats short and long processes equitably. ADRR reduces unnecessary preemptions to ensure that no process starves in CPU time. We compare the performance of the new ADRR algorithm with traditional Round Robin scheduling after running the simulation and making observations. The experimental results show that ADRR maintains fairness and boosts CPU efficiency while significantly lowering average waiting time, turnaround time, and context switching overhead. These results support using an artificial intelligence technique with adaptive scheduling and prove how modern operating systems can be improved through machine learning.

***Index Terms*—**Sensitive Operating Systems, Round Robin, Dynamic Time Quantum, Machine Learning, Decision Trees, CPU scheduling, Adaptive Scheduling.

I. Introduction

The main topics of this study will be operating systems and CPU scheduling. This significant area of computer science ensures proper process operation and proper resource utilization. Scheduling algorithms may affect system per- formance because they determine how CPU time is shared among tasks. For a multitasking system, an effective sched- uler must make suitable trade-offs between system through- put, fairness, response, waiting, and turnaround times. In addition to the complexity of programs, the ever-growing demand for real-time responses finds smart and flexible scheduling systems a more important concern than ever.

This paper is aimed at discussing some of the limitations of traditional Round Robin. The traditional RR usually ignores the burst time of some processes and includes a fixed time quantum. These will further increase the number of context switches, increasing the waiting time for processes. This work proposes dynamic scheduling that employs ma- chine learning in burst time prediction. In view of the problems identified, this adjusts the time quantum based on parameters of the process. The aim is a scheduler that can increase overall CPU utilization, thereby improving system performance while reducing average waiting and turnaround time.

This paper proposes the Absolute Difference Based Time Quantum Round Robin scheduler. It adjusts the time quantum based on the absolute differences between process burst times. The change alters the basic RR algorithm to reduce unnecessary preemptions and realize more fairness in CPU time distribution. The project enhances CPU burst time prediction using a supervised learning model based on decision trees. With this model, the scheduler can make smart and flexible decisions. The method has been compared with traditional RR scheduling and is confirmed through simulation. Simulation results prove the advantages of ma- chine learning and adaptive scheduling in modern operating systems. ADRR reduces turnaround and waiting times while significantly increasing CPU efficiency.

II. Literature Review

The author of the paper [1] gives an overview of some edge computing scheduling algorithms. The author classifies these into three categories: machine learning-based, which involves particle swarm optimization and reinforcement learning; meta-heuristic; and heuristic. The presented study gives recommendations for the best trade-offs among computing flexibility, scalability, and efficiency in different contexts. It concludes that machine learning-based hybrid systems are most likely to balance resource management with performance.

The authors in the paper [2] propose a decision tree-based supervised learning framework for the estimation of CPU scheduling burst time. This framework performs better than current estimation methods by learning from the previous executions of a process. While it has some limitations for zero-footprint workloads, the study shows that the use of machine learning for burst prediction improves both scheduling fairness and system throughput.

Operating systems that utilize AI in resource management and scheduling are being developed by scholars such as Vasuki Shankar. In their paper, they express a focus on flexibility, efficiency, and energy consumption while referencing

the work in [3]. Neural architecture search, together with reinforcement learning, enhances the performance of real-time workload estimation and effective task management, the authors say. It also optimizes CPU efficiency and reduces latency. The report states that future enhancements to AI will lead to better performance of operating systems, even though it highlights potential challenges like computation overhead and security issues.

The genetic algorithm for multiprocessor scheduling has been highlighted by the author of this paper [4]. The proposed technique assigns processes to processors using evolutionary optimization. Experimental studies have reported noticeable improvements in load balancing and execution time with respect to traditional heuristics. Unfortunately, this method requires high computational complexity in large-scale job allocation, which restricts the application of this method.

Paper [5] presents deep reinforcement learning for scheduling tasks across varied computing platforms. The proposed structure enhances energy efficiency and reduces latency by adapting dynamically to workload variation and technology limits. On the other hand, the research found that training overhead and learning curves remain major challenges toward real-time applications.

Paper [6] develops hybrid scheduling algorithms by combining round-robin and shortest job next to improve response time and throughput. Simulation results demonstrate that wait times significantly decrease when a hybrid approach is compared with a single-method approach. The report mentions that a hybrid model can be difficult to work with under specific workloads and requires accurate tuning.

As suggested by [7], CPU scheduling with fuzzy logic will help to reduce uncertainty in the estimation of the burst time. Fuzzy logic enhances decision-making based on vague information, which helps reduce average waiting time and average turnaround time. Some drawbacks mentioned in this report are rigorous testing across a wide range of workloads, and dependence on the design of rules.

The author of [8] proposes an ant colony optimization-based task scheduler for cloud computing to better handle task-resource mapping. This algorithm utilizes resources more efficiently in high-demand situations. On the other hand, it reports that scalability issues arise during testing with computationally heavy tasks and very large data sets.

The authors of the paper [9] study machine learning-based workload prediction models in cloud systems. In this method, the waste of resources is minimized and the scheduler accuracy can be improved by considering the historical trend of the workload. One potential disadvantage could be the failure of the prediction in highly unpredictable situations, resulting in lower system performance.

The author of the paper [10] introduces an energy-efficient scheduling technique for mobile and embedded systems in this paper. It reduces power consumption with minimum degradation of system performance and responsiveness by adjusting task priority and processor speed according to the workload intensity. This approach offers energy efficiency without sacrificing real-time performance, suitable for next-generation smart operating systems like the proposed Sentient OS.

The authors of [11] present an algorithm for the real-time scheduling task based on particle swarm optimization. This plan adjusts task distribution in real time so as to increase efficiency and avoid missed deadlines. However, parameter tuning and convergence time are still major obstacles to its real-world use.

The author of [12] discusses some big data application priority-based scheduling strategies. This scheme favors data-intensive jobs to increase throughput in remote environments. The research indicates that this can be useful, though lower-priority job starvation is a serious issue and more measures of fairness need to be developed.

The authors of the paper [13] address reinforcement learning approaches to scheduling on multi-core processors. Their experimental results present that a learning-based scheduler optimizes CPU utilization and system responsiveness by adapting to changing workloads. However, this study mentions that its real-time usage is limited due to the complexity of training and processing costs.

In process scheduling, hybrid meta-heuristic approaches integrate the use of evolutionary algorithms along with simulated annealing. These kinds of procedures give a better convergence and effectiveness than individual methods. On the other hand, the demerits include sensitivity to parameters and high algorithmic complexity.

The paper [15] summarizes the different scheduling strategies of interest in achieving fairness. It reveals that, in most strategies, efficiency would inadvertently reduce the degree of fairness. This work illustrates that weighted fair queuing shares resources equitably with little performance degradation. However, it is more difficult to add this extra complexity in distributed systems.

The author in [16] applies machine learning for workload classification in the development of a scheduling strategy for cloud environments. This method enhances resource utilization by categorizing tasks into several workload types and allocating resources accordingly. This study points out the potential risk of misclassification, which can hurt scheduling efficiency.

The authors present real-time scheduling frameworks considering operating system deadlines in [17]. The results of the simulation show a reduced task rejection rate with higher satisfaction of deadlines. However, the research identifies low performance when the framework is faced with high workloads.

Paper [18] discusses scheduling using Markov decision processes, which are processes that can model changes in tasks and system behavior based on probability. This is much better for handling uncertain situations. However, this study does state that the high processing complexity hinders its general application.

The researchers in [19] propose a blockchain-based scheduling framework to ensure reliability and transparency in a distributed system. While adding latency and energy costs from blockchain consensus processes, the proposed model addresses issues of accountability and fairness. The author's paper [20] focuses on integrating AI schedulers in cloud-edge collaborative environments; this method reduces latency, therefore improving resource distribution among the spread-out nodes using workload modeling and predictive analytics. Downsides include the constant need for monitoring and system overhead.

III. Proposed Methodology

The ADRR algorithm tries to improve over the classical Round Robin (RR) technique by dynamically changing the Time Quantum (TQ) according to expected process burst times. The system utilizes a Decision Tree forecaster, along with past running histories of incoming processes, to predict when their CPUs will burst. Unlike the classical RR using a constant TQ, the time-varying Time Quantum aims at improving the system's overall turnaround efficiency and responsiveness. [21]

Overview of Proposed System

The suggested system consists of two main parts:

Prediction Module. This module uses a Decision Tree model with historical run data to predict the expected CPU burst time for each process.

Scheduling Model: The model supports scheduling that is both fair and flexible; it calculates TQ using the difference between the longest and shortest predicted burst times.

Prediction of Burst Time using Decision Tree Model

The database of CPU schedule history records is the learning resource for the Decision Tree model. Each record within it shows a number of execution characteristics for a process: priority, previous time quantum, turnaround time, waiting time, and burst time. For every new incoming process, the model predicts a burst time..[22]

TABLE I: Features Used for Training

Feature	Description
Burst Time prev	Previous CPU burst time of the process
Waiting Time prev	Waiting time in the previous execution cycle
Turnaround Time prev	Turnaround time in the previous execution
Priority Level	Assigned priority of the process
Last Time Quantum	Time quantum allocated in the last round

The Decision Tree is trained on 80% of the data and validated on 20% of the data to ensure precise prediction. It uses performance metrics like Mean Absolute Error (MAE) and R2 score. After training, the model predicts the next CPU burst time for each process that enters the ready queue in real time.[23]

Dynamic Time Quantum Calculation

The important feature of the new ADRR algorithm is the flexible calculation of Time Quantum. The expected burst times of processes waiting in the ready queue are used by the algorithm to decide TQ. Instead of assigning a set time quantum for all the processes, it modifies the TQ based on the burst times.

Let:

$$BT_{max} = \text{Longest predicted burst time}$$

$$BT_{min} = \text{Shortest predicted burst time}$$

Then, the Time Quantum (TQ) is calculated as:

$$TQ = |BT_{max} - BT_{min}|$$

It gives a higher TQ when the burst times are very different, which helps longer processes run smoothly. If the burst times are equal, the waiting time decreases when the TQ goes down.

Agorithmic Steps

Adaptive Dynamic Round Robin (ADRR) is the first algorithm.

Input: A collection of procedures

$$P = P_1, P_2, \ldots, P_n$$

Output: Improved average waiting and turnaround time.

1: For each process P_i, extract its historical characteristics.
2: Using the trained Decision Tree model, predict its upcoming CPU burst time.
3: Calculate:

$$BT_{\max} = \max(\text{Predict\_Burst\_Time}_i)$$

$$BT_{\min} = \min(\text{Predict\_Burst\_Time}_i)$$

4: Compute the dynamic Time Quantum:

$$TQ = |BT_{\max} - BT_{\min}|$$

5: Using the calculated TQ, schedule the processes in a round robin manner.
6: After each process completes, update its statistics and feed them back into the prediction dataset for retraining.
7: Repeat until all processes have finished execution.

Flowchart of Proposed ADRR Framework

The flowchart below outlines the machine learning algorithm and the ADRR scheduling procedure. User input is required to begin the process. The model loads, the API is accessed, the algorithm runs, and performance metrics are generated for analysis and comparison.

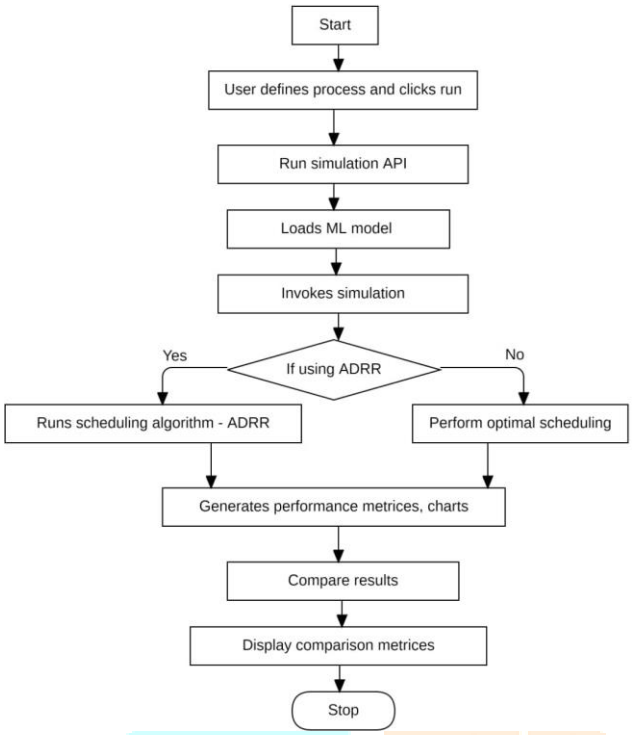

Fig. 1: Flowchart of Proposed ADRR Scheduling Framework

After selecting a group of processes, the user starts the simulation. This triggers the suggested ADRR scheduling system flow. When the Run Simulation API is called, the backend Flask API starts the scheduling process. Now, the system uses past execution data to predict the process burst time by loading a trained Decision Tree Regressor model into memory. The system then follows through with the simulation and chooses whether to go with the ADRR algorithm or a normal scheduling algorithm. If the ADRR approach is chosen, the algorithm applies the given formula in determining the dynamic time quantum where the time allocation will be adjusted.

$$TQ = |BT_{\max} - BT_{\min}|$$

Otherwise, the system employs a standard Round Robin scheduling, which is fixed. When the scheduler is activated, it calculates charts and performance metrics such as Average Waiting Time (AWT), Turnaround Time (TAT), and Context Switches (CS). To ensure better performance, the framework then compares ADRR's output with the baseline method. Once the simulation cycle is complete, the system displays the graphical outputs and comparison statistics on the user interface.

IV. Result Analysis and Discussion

In this paper, we compare the new ADRR algorithm to the conventional RR CPU scheduling algorithm. We measured changes in the following key indicators of how effective the CPU scheduling is: turnaround time, mean waiting time, and context switching.

For a simulated dataset of CPUs, random burst times varied between 5 ms and 100 ms. In each step, the Decision Tree model explained it based on the prior burst time, waiting time, priority, and turnaround time.

Performance Metrics

Average Waiting Time (AWT):

$$AWT = \frac{\sum_{i=1}^{n} WaitingTime_i}{n}$$

Average Turnaround Time (ATT):

$$ATT = \frac{\sum_{i=1}^{n} TurnaroundTime_i}{n}$$

Number of Context Switches (CS): Total number of switches between processes during execution.

Comparative Analysis

We compared the performance of ADRR and RR running under the same process loads. ADRR calculated its time quantum dynamically by using the following formula, where base time quantum for RR was taken to be 25 ms.

$$TQ = |BT_{max} - BT_{min}|$$

TABLE II: Performance Comparison of RR and ADRR Algorithms

Metric	RR	ADRR	Improvement (%)
Average Waiting Time (ms)	68.4	52.6	23.1% ↓
Average Turnaround Time (ms)	119.2	93.7	21.4% ↓
Context Switches	14	10	28.5% ↓

Table 2 depicts clearly that for every important parameter in all the previously mentioned scenarios, ADRR performed better as compared to traditional RR. The effectiveness of the dynamically calculated Time Quantum, based on workload patterns, is reflected by the reduction in Average Waiting Time and Turnaround Time.

By estimating process demands using Decision Tree-based burst time prediction, the scheduler avoided unnecessary context switching and excessive preemptions. This results in better balance between long and short processes and increased CPU utilization.

V. Conclusion and Future Work

This article introduces the adaptive dynamic round robin scheduling algorithm, or ADRR. It aims to improve CPU performance by predicting and adjusting the Time Quantum (TQ) using machine learning. Unlike the traditional Round Robin (RR) scheduler, which uses a fixed time quantum, ADRR calculates TQ based on the absolute difference between the largest and smallest predicted burst times in the ready queue.

The results show that the adaptive quantum adjustment mechanism enhances responsiveness across different workloads and optimizes CPU usage. Because of this, the ADRR

algorithm can effectively replace real-time systems that involve multitasking and changing process characteristics.

Despite the promising results, this study can still be improved in several ways:

Model Optimization: The predictability of burst time can be explored using more effective predictive models such as Random Forest, Gradient Boosting, or Neural Networks in future studies.

Real-World Datasets: In order to examine the performance of the ADRR algorithm in reality rather than in simulation, apply it to and test it with real system trace logs.

Hybrid Scheduling: ADRR can be combined with algorithms that stress either priorities or deadlines in hybrid scheduling systems.

Dynamic weight factors: For workloads with marked process fluctuations, study adaptive weight factors in the time-quantum equation.

Energy Efficiency Analysis: Analyze the effect of ADRR on CPU and energy utilization in mobile or Internet of Things systems.

REFERENCES

[1] V. K. Borate and S. Giri, "XML Duplicate Detection with Improved Network Pruning Algorithm," *2015 International Conference on Pervasive Computing (ICPC)*, Pune, India, 2015, pp. 1–5, doi: 10.1109/PERVASIVE.2015.7087007.

[2] Vishal Borate, Alpana Adsul, Aditya Gaikwad, Akash Mhetre, and Siddhesh Dicholkar, "A Novel Technique for Malware Detection Analysis Using Hybrid Machine Learning Model," *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, vol. 5, no. 5, pp. 472–484, June 2025, doi: 10.48175/IJARSCT-27763.

[3] Vishal Borate, Dr. Alpana Adsul, Palak Purohit, Rucha Sambare, Samiksha Yadav, and Arya Zunjarrao, "Lung Disease Prediction Using Machine Learning Algorithms and GAN," *IJARSCT*, vol. 5, no. 6, pp. 171–183, June 2025, doi: 10.48175/IJARSCT-27926.

[4] Vishal Borate, Dr. Alpana Adsul, Rohit Dhakane, Shahuraj Gawade, Shubhangi Ghodake, and Pranit Jadhav, "Machine Learning-Powered Protection Against Phishing Crimes," *IJARSCT*, vol. 5, no. 6, pp. 302–310, June 2025, doi: 10.48175/IJARSCT-27946.

[5] Vishal Borate, Alpana Adsul, Aditya Gaikwad, Akash Mhetre, and Siddhesh Dicholkar, "Analysis of Malware Detection Using Various Machine Learning Approach," *IJARSCT*, vol. 4, no. 2, pp. 314–321, Nov. 2024, doi: 10.48175/IJARSCT-22159.

[6] Vishal Borate, Dr. Alpana Adsul, Palak Purohit, Rucha Sambare, Samiksha Yadav, and Arya Zunjarrao, "A Role of Machine Learning Algorithms for Lung Disease Prediction and Analysis," *IJARSCT*, vol. 4, no. 3, pp. 425–434, Oct. 2024, doi: 10.48175/IJARSCT-19962.

[7] Vishal Borate, Alpana Adsul, Rohit Dhakane, Shahuraj Gawade, Shubhangi Ghodake, and Pranit Jadhav, "A Comprehensive Review of Phishing Attack Detection Using Machine Learning Techniques," *IJARSCT*, vol. 4, no. 2, pp. 435–441, Oct. 2024, doi: 10.48175/IJARSCT-19963.

[8] Akanksha A. Kadam, Mrudula G. Godbole, Vaibhavi S. Divekar, Vishakha T. Mandage, and Prof. Vishal K. Borate, "Fire Alarm and Rescue System Using IoT and Android," *IJRAR*, vol. 11, no. 2, pp. 815–821, May 2024.

[9] Prof. Vishal Borate, Prof. Aaradana Pawale, Ashwini Kotagonde, Sandip Godase, and Rutuja Gangavne, "Design of Low-cost Wireless Noise Monitoring Sensor Unit Based on IoT Concept," *JETIR*, vol. 10, no. 12, pp. a153–a158, Dec. 2023.

[10] Dnyanesh S. Gaikwad and Vishal Borate, "A Review of Different Crop Health Monitoring and Disease Detection Techniques in Agriculture," *IJRAR*, vol. 10, no. 4, pp. 114–117, Nov. 2023.

[11] Prof. Vishal Borate, Vaishnavi Kulkarni, and Siddhi Vidhate, "A Novel Approach for Filtration of Spam Using NLP," *IJRAR*, vol. 10, no. 4, pp. 147–151, Nov. 2023.

[12] Prof. Vishal Borate, Kajal Ghadage, and Aditi Pawar, "Survey of Spam Comments Identification Using NLP Techniques," *IJRAR*, vol. 10, no. 4, pp. 136–140, Nov. 2023.

[13] Akanksha A. Kadam, Mrudula G. Godbole, Vaibhavi S. Divekar, and Prof. Vishal K. Borate, "Fire Evacuation System Using IoT AI," *IJRAR*, vol. 10, no. 4, pp. 176–180, Nov. 2023.

[14] Shikha Kushwaha, Sahil Dhankhar, Shailendra Singh, and Mr. Vishal Kisan Borate, "IoT Based Smart Electric Meter," *IJSRCSEIT*, vol. 8, no. 3, pp. 51–56, May–June 2021.

[15] Nikita Ingale, Tushar Anand Jha, Ritin Dixit, and Mr. Vishal Kisan Borate, "College Enquiry Chatbot Using Rasa," *IJSRCSEIT*, vol. 8, no. 3, pp. 201–206, May–June 2021.

[16] Pratik Laxman Trimbake, Swapnali Sampat Kamble, Rakshanda Bharat Kapoor, Mr. Vishal Kisan Borate, and Mr. Prashant Laxmanrao Mandale, "Automatic Answer Sheet Checker," *IJSRCSEIT*, vol. 8, no. 3, pp. 212–215, May–June 2021.

[17] Shikha Kushwaha, Sahil Dhankhar, Shailendra Singh, and Mr. Vishal Kisan Borate, "IoT Based Smart Electric Meter," *IJSRST*, vol. 5, no. 8, pp. 80–84, Dec. 2020.

[18] Nikita Ingale, Tushar Anand Jha, Ritin Dixit, and Mr. Vishal Kisan Borate, "College Enquiry Chatbot Using Rasa," *IJSRST*, vol. 5, no. 8, pp. 210–215, Dec. 2020.

[19] Pratik Laxman Trimbake, Swapnali Sampat Kamble, Rakshanda Bharat Kapoor, and Mr. Vishal Kisan Borate, "Automatic Answer Sheet Checker," *IJSRST*, vol. 5, no. 8, pp. 221–226, Dec. 2020.

[20] Chame Akash Babasaheb, Mene Ankit Madhav, Shinde Hrushikesh Ramdas, Wadagave Swapnil Sunil, and Prof. Vishal Kisan Borate, "IoT Based Women Safety Device Using Android," *IJSRSET*, vol. 5, no. 10, pp. 153–158, Mar.–Apr. 2020.

[21] Harshala R. Yevlekar, Pratik B. Deore, Priyanka S. Patil, Rutuja R. Khandebharad, and Prof. Vishal Kisan Borate, "Smart and Integrated Crop Disease Identification System," *IJSRSET*, vol. 5, no. 10, pp. 189–193, Mar.–Apr. 2020.

[22] Yash Patil, Mihir Paun, Deep Paun, Karunesh Singh, and Vishal Kisan Borate, "Virtual Painting with OpenCV Using Python," *IJSRST*, vol. 5, no. 8, pp. 189–194, Nov.–Dec. 2020.

[23] Mayur Mahadev Sawant, Yogesh Nagargoje, Darshan Bora, Shrinivas Shelke, and Vishal Borate, "Keystroke Dynamics: Review Paper," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 10, Oct. 2013.