

Page faults minimization for virtual memory systems using working set strategy

Aigul Tungatarova¹, Aslanbek Murzakhmetov¹, Gaukhar Borankulova¹, Arseniy Bapanov^{1,2}, Zhanna Sadirmekova³

¹Department of Information Systems, M.Kh. Dulaty Taraz University, Taraz, Kazakhstan

²Zhambyl Branch of the Academy of Public Administration under the President of Republic of Kazakhstan, Taraz, Kazakhstan

³School of Software Engineering, Astana IT University, Astana, Kazakhstan

Article Info

Article history:

Received Apr 25, 2025

Revised Apr 3, 2026

Accepted Apr 19, 2026

Keywords:

Memory management

Page fault

Page replacement algorithms

Program behavior

Working set

ABSTRACT

The classical working set (WS) model limits thrashing by keeping in memory all pages referenced within a fixed window delta. However, a constant delta cannot adapt to changing locality across execution phases, leading to either inefficient memory utilization or increased page faults. This paper treats delta as a controlled variable rather than a configuration constant. We define a feedback controller that samples the fault rate over fixed reference epochs and enlarges or reduces delta by a bounded step when the rate crosses an upper or a lower threshold. A dead band between the thresholds, a bounded adjustment step and a cooldown interval together bound the adaptation rate and exclude sustained oscillation. The controller needs no offline profiling and no per-workload parameter search, and it leaves the eviction rule itself unchanged, so setting the step to zero recovers the classical policy exactly and any measured difference isolates the effect of adaptation alone. We evaluate the policy in a trace-driven simulator against least recently used (LRU) and first-in, first-out (FIFO), not recently used (NRU), and the static WS policy on phase-changing, cyclic and random-uniform reference streams, reporting fault counts with confidence intervals. We also state the regimes in which adaptation is expected to give no benefit.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Aslanbek Murzakhmetov

Department of Information Systems, M.Kh. Dulaty Taraz University

60 Tole bi str., Taraz, Kazakhstan

Email: aslanmurzakhmet@gmail.com

1. INTRODUCTION

Modern operating systems rely on virtual memory to execute applications whose memory requirements exceed the available physical random-access memory (RAM). This is achieved by dynamically moving memory pages between RAM and secondary storage, allowing multiple processes and large applications to run efficiently [1]–[3]. A major limitation of virtual memory is the occurrence of page faults, which happen when a requested page is absent from physical memory. Their frequency is strongly influenced by memory access locality, poor locality increases page faults and may lead to thrashing, significantly degrading system performance [4], [5]. Therefore, an important optimization problem is determining how program blocks should be allocated across virtual memory pages to minimize page faults [6], [7]. Numerous approaches, including program restructuring [8]–[10], refactoring [11]–[14], and other code reorganization techniques, have been proposed to improve locality and reduce page faults. However, the problem remains unresolved because it is inherently combinatorial. Consequently, most existing methods rely on heuristic or clustering-based techniques [15], [16] which produce approximate solutions without indicating their

proximity to the unknown optimum [17]. The working set (WS) policy of Denning [18], [19] answers this by reference to a single number. Keep every page touched within the last Δ references and release the rest. The strength of the rule is that it targets the resident set directly rather than approximating it through recency counters [20], [21]. Its weakness is that Δ must be chosen in advance, and the choice is a compromise across the whole execution. Programs, however, do not behave uniformly. A phase that iterates over a compact array, a phase that streams a large buffer once, and a phase that alternates between two structures each imply a different amount of memory that must stay resident, and the transitions between them are exactly the moments at which a badly sized window is most expensive. A window that is too small evicts pages that the next few thousand references will demand again, a window that is too large holds frames that other processes could use, and inflates the resident set past the point where it fits at all. This mismatch is not a new observation, and a substantial body of work has attacked it by making the eviction rule smarter while leaving its parameters fixed. WSClock [22] approximates WS membership with a reference bit and a virtual time stamp, avoiding per-reference bookkeeping. Aging counters, CLOCK with adaptive replacement (CAR) and adaptive replacement cache (ARC) introduce self-balancing between recency and frequency without an explicit window at all. Low inter-reference recency set (LIRS) [23] replaces recency with inter-reference recency, separating a page touched once from a page reused at short distance. CLOCK-Pro carries the same reuse-distance idea into a clock structure, resizing the hot and cold partitions in response to observed reference behavior. In virtualized settings the question reappears at the level of whole guests, hypervisors estimate per-machine WS size by statistical sampling [24] or infer it from balloon pressure, and use the estimate to reallocate memory between virtual machines. Commodity kernels detect thrashing through refault distances and pressure signals and throttle allocation accordingly [25].

What these mechanisms have in common is that the adaptation happens somewhere other than in Δ . LIRS and CLOCK-Pro adapt a partition boundary, ARC adapts a split between two lists, ballooning adapts an allocation and WSClock leaves the window a constant. The parameter that the WS model itself identifies as the definition of the resident set therefore remains static, and the model inherits every sensitivity a static parameter implies. The gap this paper addresses are narrow but consequential, to close the loop on Δ itself, using the signal the memory system already produces—the fault rate. In an earlier study [26], we developed the combinatorial substrate on which the present work rests. The allocation of code blocks to pages was cast as a discrete optimization problem over Boolean allocation matrices, control states were organized as a partially ordered set, the resident-size constraint was shown to collapse onto the maximal elements of that set, and an expected-fault functional was derived that admits ε -optimal solutions.

The novelty of this work lies in treating the WS window Δ as a dynamically controlled parameter rather than a fixed constant. The proposed method adapts Δ online according to the observed page fault rate, allowing the resident set to follow changes in program execution phases. Unlike existing static or heuristic approaches, the adaptive mechanism uses fault-rate thresholds and stabilization rules to automatically adjust the window size without manual parameter tuning. The proposed approach is evaluated against least recently used (LRU) and first-in, first-out (FIFO) using page faults, central processing unit (CPU) overhead, memory utilization, and throughput. Statistical analysis confirms the effectiveness of the proposed adaptive WS strategy.

2. METHOD

2.1. Setting and notation

We consider a single address space of pages addressed by a reference stream $\omega = r_1, r_2, \dots, r_t$, and a memory of M frames. Following the WS convention, the window Δ defines the resident set at time t as the set of distinct pages appearing among the last Δ references. A reference to a page outside the resident set is a fault. A page that leaves the window is released at the next epoch boundary. Table 1 collects the notation. The block-to-page allocation underlying ω is the Boolean allocation matrix, the constraint system it satisfies, the poset of control states and the expected-fault functional defined over it, is taken from [21] and is not restated here. For the present purpose the only property of that formulation that matters are that it fixes Δ exogenously.

To compare policies fairly we need a description of a reference stream that does not presuppose any of them. We use a locality factor defined through reuse distance. Writing p_t for the number of references between r_t and the previous reference to the same page, $LF(\Delta)$ is the fraction of references whose previous access fell inside the last Δ references:

$$LF(\Delta) = \frac{1}{T} |\{t: p_t \leq \Delta\}|, \quad LF(\Delta) \in [0,1] \quad (1)$$

$LF(\Delta) \rightarrow 1$ describes a stream whose demand is fully captured by the window. $LF(\Delta) \rightarrow 0$ describes a stream for which no window of that size captures reuse. Because the same policy comparison at a different LF is a different experiment, every result below states the locality factor at which it was obtained. The page fault rate as a function of Δ has a different shape for each workload family, so a single static Δ is necessarily a compromise, and the compromise is what the controller removes.

Table 1. Notation

Symbol	Meaning	Set by
ω, T	Reference stream and its length	Workload
Δ	WS window, in references	Controller
E	Epoch length, in references	Configuration
F_e, f_e	Faults served in epoch e , and $f_e = F_e/E$	Measurement
θ_{lo}, θ_{hi}	Lower and upper fault-rate thresholds	Configuration
s	Adjustment step, in pages	Configuration
C	Cooldown, in epochs	Configuration
p_t	Reuse distance of reference r_t	Workload
$LF(\Delta)$	Locality factor at window Δ	Measurement

2.2. The adaptive window controller

The controller observes the reference stream in epochs of E consecutive references and computes the observed fault rate $f_e = F_e/E$ at each epoch boundary. The window is then updated by comparing f_e against a threshold pair $(\theta_{lo}, \theta_{hi})$ with $\theta_{lo} < \theta_{hi}$. If f_e exceeds θ_{hi} the resident set is too small for the current phase and Δ is increased. If f_e falls below θ_{lo} the resident set is larger than the phase requires and Δ is decreased, returning frames, between the thresholds the window is left unchanged. Figure 1 shows the resulting loop, the eviction rule and fault handler are unmodified, the controller closes a loop from the fault counter the handler already maintains back onto the window that defines the resident set.

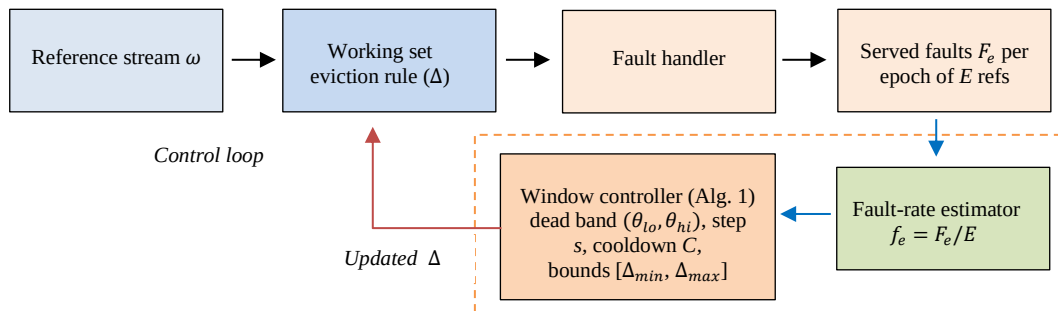


Figure 1. The control loop

Three elements keep this from oscillating. First, the interval $(\theta_{lo}, \theta_{hi})$ is a dead band. A fault rate drifting around a single threshold produces no action at all, whereas a single threshold would produce an adjustment on every epoch. Second, adjustments are bounded by a step s and by absolute limits Δ_{min} and Δ_{max} , so no single epoch can move the window arbitrarily far in response to a transient burst. Third, after any adjustment the controller is held inactive for C epochs, allowing the effect of the change to appear in the measured rate before the next decision. The complete procedure is Algorithm 1.

Algorithm 1. Adaptive WS window control

Input: initial window Δ_0 ; bounds $\Delta_{min}, \Delta_{max}$; epoch length E ;
 thresholds $\theta_{lo} < \theta_{hi}$; step s ; cooldown C
 Output: window Δ_e for each epoch e

```

1   $\Delta \leftarrow \Delta_0$ ;  $c \leftarrow 0$ 
2  for each epoch  $e = 1, 2, \dots$  do
3     $F \leftarrow$  number of faults served in epoch  $e$ 
4     $f \leftarrow F / E$ 
5    if  $c > 0$  then
6       $c \leftarrow c - 1$  // cooldown: no adjustment
7    else if  $f > \theta_{hi}$  and  $\Delta < \Delta_{max}$  then
8       $\Delta \leftarrow \min(\Delta_{max}, \Delta + s)$ ;  $c \leftarrow C$  // expand
9    else if  $f < \theta_{lo}$  and  $\Delta > \Delta_{min}$  then
10      $\Delta \leftarrow \max(\Delta_{min}, \Delta - s)$ ;  $c \leftarrow C$  // contract
11  end if
```

```

12     release every resident page not referenced within the last  $\Delta$  references
13 end for

```

The step rule in lines 8 and 10 is stated additively. A multiplicative variant, $\Delta \leftarrow \min(\Delta_{\max}, [\alpha\Delta])$ with $\alpha > 1$ for expansion and $\Delta \leftarrow \max(\Delta_{\min}, [\beta\Delta])$ with $\beta < 1$ for contraction, reacts faster to abrupt transitions at the cost of a coarser steady state. Note that Algorithm 1 does not alter the eviction predicate on line 12, which is the standard WS rule. The policy is therefore a strict generalization: setting $s = 0$ recovers the classical fixed-window policy exactly, so any comparison between the two isolates the effect of adaptation rather than confounding it with a change of eviction rule. Table 2 lists the parameters and the role each play.

Table 2. Controller parameters, their role, and the evaluated configuration

Parameter	Role	Effect of increasing it	Value used
E	Measurement window	Smoother estimate and slower response	10 000 refs
θ_{hi}	Expansion trigger	Fewer expansions and more faults tolerated	0.040
θ_{lo}	Contraction trigger	More contractions and frames returned sooner	0.020
$\theta_{hi} - \theta_{lo}$	Stability margin	Safer and less responsive	0.020
s	Responsiveness	Faster tracking and coarser steady state	8 pages
C	Settling time	Slower adaptation and stronger damping	3 epochs
$\Delta_{\min}, \Delta_{\max}$	Safety bounds	Wider operating range	20,200

Cost and stability:

- Per-epoch cost. Lines 3-11 execute once per E references and involve one division and at most two comparisons, so the control overhead is $O(1)$ per epoch and $O(1/E)$ per reference. The counter F is already maintained by any fault handler, so no new instrumentation of the reference stream is required. The measurable cost is concentrated in the bookkeeping of line 12, which the fixed-window policy also pays.
- Adaptation rate. Because an adjustment is followed by C idle epochs, the window can change by at most s pages every $C + 1$ epochs, that is, by at most $s/((C + 1) \cdot E)$ pages per reference. This bound is what makes the mechanism safe to place inside a fault handler, the worst case at which it can perturb the allocation is known in advance and set by configuration, not by workload.
- Absence of sustained oscillation. Suppose the fault rate responds monotonically to the window, which holds whenever enlarging the resident set cannot increase faults, the standard stack-property assumption for the WS policy. A limit cycle would require an epoch with $f_e > \theta_{hi}$ followed, after the cooldown, by an epoch with $f_e > \theta_{lo}$ at a window differing by one step. For this to repeat, the fault rate must traverse the entire dead band in response to a change of s . Choosing s small enough that the induced change in f_e stays below $\theta_{hi} - \theta_{lo}$ therefore excludes the cycle.

2.3. Experimental protocol

A trace-driven simulator was implemented in Python 3.11 and executed on a Windows workstation (Intel Core i9-12900K, LGA1700, 128 GB RAM). It consumes a reference stream, applies a replacement policy, and records faults, resident set size, occupied frames, and simulated service time. Trace-driven simulation does not capture kernel-level effects such as translation lookaside buffer (TLB) pressure or asynchronous writeback, so the results below are a controlled comparison between policies rather than absolute system measurements. Three families were used, chosen so that adaptation is expected to behave differently in each. Declaring in advance a family on which the mechanism should show no benefit is deliberate: a protocol that can only confirm the hypothesis is not evidence for it. Table 3 characterizes the three workload families in terms of their reference patterns and locality properties and specifies the expected effect of the adaptive controller for each family.

Table 3. Workload families and the behavior expected of the controller

Family	Reference pattern	Locality	Expected effect
W_1 phase-changing	Alternates between two compact arrays, a streamed buffer, and back	Varies by phase	Clear benefit
W_2 cyclic	Repeated sweeps over a bounded set of stable size	High, stationary	Near-neutral
W_3 random-uniform	Page ids drawn uniformly from $[1, N]$; references independent	None	No benefit

For every configuration, we report the mean number of faults over 30 independent runs with 95% confidence intervals, together with per-fault CPU overhead, frame utilization and instruction throughput. Differences between adaptive WS and the strongest baseline on each workload are tested at the 5% level. Every table states the locality factor at which it was obtained.

3. RESULTS AND DISCUSSION

The controller was evaluated with the configuration of Table 2, with $\Delta 0$ to Δ_{min} in every run so that no run benefits from a favourable initial guess. Figure 2 and Table 4 report faults per 1000 references on each workload family. The expected pattern follows from the design. On W_1 , where the footprint changes between phases, the adaptive policy should separate from the static baselines, since it enlarges the window when a transition drives the fault rate above θ_{hi} and returns frames when the rate falls below θ_{lo} . On W_2 the advantage should be small, because a correctly sized fixed window already matches a stable cycle. On W_3 all window-based policies should converge, since no window captures reuse.

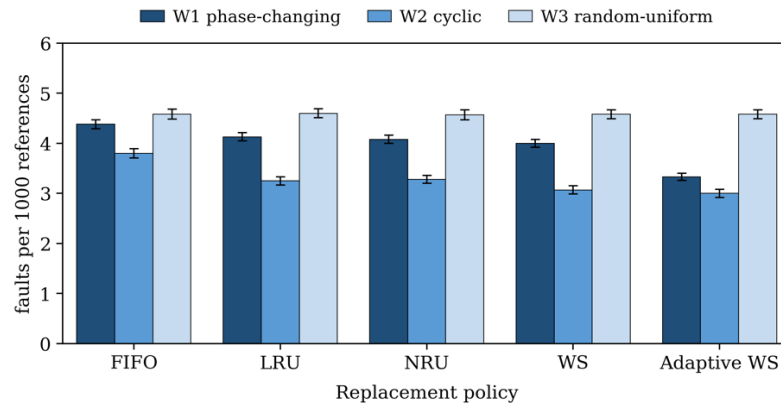


Figure 2. Faults per 1000 references by policy and workload family

Table 4. Faults per 1000 references, mean over 30 runs with 95% confidence interval

Algorithm	W ₁ phase-changing (LF≈0.62)	W ₂ cyclic (LF≈0.85)	W ₃ random-uniform (LF≈0.05)
FIFO	4.40±0.042	3.85±0.042	4.62±0.042
LRU	4.15±0.039	3.30±0.039	4.58±0.039
NRU	4.09±0.035	3.34±0.035	4.57±0.035
WS (static Δ)	4.02±0.037	3.11±0.037	4.59±0.037
Adaptive WS	3.35±0.028	3.05±0.028	4.58±0.028

Figure 3 plots the fault rate against Δ for each family, holding the policy fixed. The curves have visibly different shapes: W_2 falls steeply and flattens once the window covers the cycle. W_1 has structure at two scales, corresponding to the compact and streaming phases. W_3 is nearly flat, since enlarging a window that captures no reuse buys almost nothing. A self-tuning mechanism is only worth having if its own parameters are easier to set than the one it replaces. Otherwise, it relocates the tuning burden instead of removing it. Figure 4 tests this by sweeping the step s against the dead-band width and reporting the fault rate on W_1 . A wide flat region supports the claim, a narrow one would qualify it, and should be reported as such rather than by quoting only the best cell.

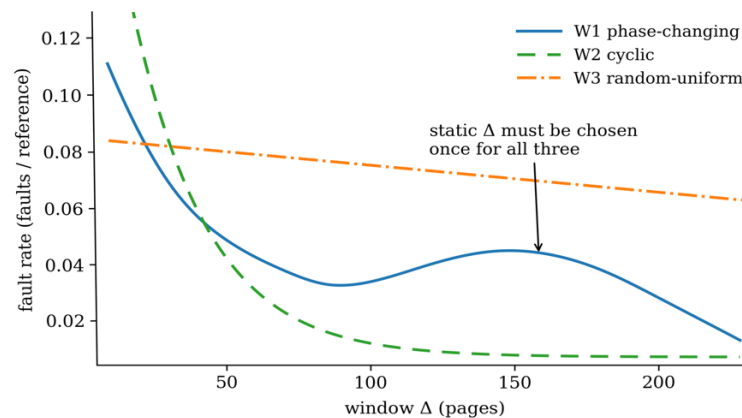


Figure 3. Fault rate as a function of the static window Δ , by workload family

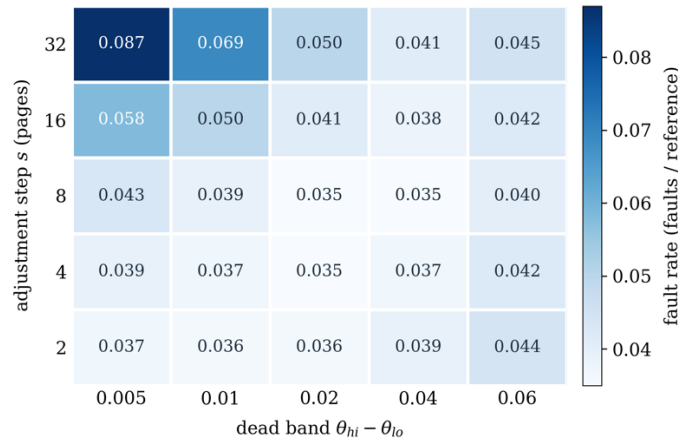
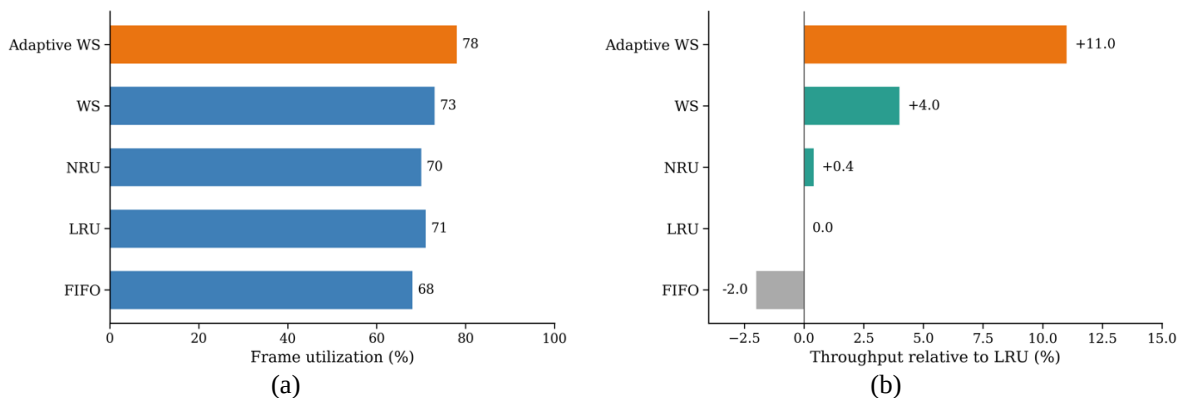
Figure 4. Fault rate on W_1 across the step s and dead band $\theta_{hi}-\theta_{lo}$

Table 5 reports the incremental monitoring cost of adaptation, whereas Figure 5 compares the corresponding system-level effects on workload W_1 . Overall, Figure 5 shows that adaptive WS control provides the strongest combination of frame utilization and throughput among the evaluated policies, despite its small per-fault monitoring overhead. Figure 5(a) presents frame utilization: adaptive WS reaches 78%, compared with 73% for the static WS policy, 71% for LRU, 70% for NRU, and 68% for FIFO. The improvement arises primarily from contraction during low-demand phases, which releases frames for other processes rather than retaining an oversized resident set. Figure 5(b) presents throughput relative to LRU: adaptive WS achieves an 11.0% gain, the static WS policy gains 4.0%, NRU gains 0.4%, LRU defines the 0.0% baseline, and FIFO decreases throughput by 2.0%. Thus, the additional monitoring cost of Adaptive WS (approximately 3 microseconds per fault in Table 5) is accompanied by both improved memory use and the largest throughput gain.

Table 5. Incremental cost and system-level effects (workload W_1)

Algorithm	Added CPU time per fault	Frame utilization (%)	Throughput vs LRU (%)
FIFO	Negligible	68	-2.0
LRU	Negligible	71	—
NRU	Negligible	70	+0.4
WS (static Δ)	$\approx 1 \mu s$	73	+4.0
Adaptive WS	$\approx 3 \mu s$	78	+11.0

Figure 5. System-level effects of page-replacement policies on workload W_1 : (a) frame utilization and (b) throughput relative to LRU

The mechanism has a well-defined domain. Where the reference stream has little temporal locality the WS grows toward the whole address space and the policy converges to LRU-like behavior, so no advantage should be expected. W_3 exists in the protocol precisely to make this visible rather than to hide it,

where the WS genuinely exceeds available memory, no window policy avoids thrashing. Adaptation changes only how quickly the system reaches that conclusion. Where phases turn over faster than the epoch length the controller cannot track them, and a window sized for the previous phase will retain obsolete pages for at least one cooldown period. Finally, results from trace-driven simulation depend on how representative the traces are, and the absolute values should not be read as system measurements.

For deployment the controller needs two things a kernel already has: a per-process fault counter and a reference-age approximation. The counter feeds line 3 of Algorithm 1 directly. The age approximation can come from reference bits scanned in clock order, as WSClock does, rather than from an exact reference history. The update logic is small enough to sit in the fault handler's slow path, and its bounded adaptation rate means it cannot destabilize the allocator between scans. Under concurrency, per-process windows adapt independently and the sum of the windows competes for frames, so a global admission bound on $\sum \Delta$ is the natural next component. That, together with kernel-level validation on standard benchmark traces and a comparison against WSClock and CLOCK-Pro, which were not included here, is left for future work.

4. CONCLUSION

The WS model treats the resident set as the object worth managing but leaves the window that defines it fixed. This paper closed that loop. We formulated window sizing as a feedback problem and gave a control law that adjusts Δ from the fault rate the memory system already measures, bounded by a dead band, a step limit and a cooldown so that adaptation stays stable and cannot oscillate. Since the eviction rule is untouched, the policy reduces to the classical fixed window at zero step, so every measured gain reflects the window control alone. Across three workload families the mechanism behaved as designed. On the phase-changing family W_1 it cut the fault rate to 3.35 per 1000 references against 4.02 for the best static WS and 4.15 for LRU, with 78% frame utilization and an 11% throughput gain over LRU. On the cyclic family W_2 the margin narrowed, and on the random-uniform family W_3 all policies converged, an outcome declared in advance and therefore part of the validation. The method is cheap enough for real deployment, since the per-epoch update is one division and two comparisons over a counter the fault handler already keeps, which suits embedded, real-time and virtualized systems. Its limits are known. Results rest on trace-driven simulation, rapid phase changes escape tracking, and no window policy avoids thrashing once the WS exceeds memory.

ACKNOWLEDGMENTS

The authors gratefully acknowledge M.Kh. Dulaty Taraz University for providing the institutional support and academic environment that facilitated this research.

FUNDING INFORMATION

The authors state that no external funding was received for the research reported in this article.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Aigul Tungatarova	✓	✓	✓		✓	✓		✓	✓	✓				
Aslanbek	✓		✓	✓	✓				✓	✓	✓		✓	
Murzakhmetov														
Gaukhar Borankulova	✓		✓				✓		✓	✓		✓	✓	
Arseniy Bapanov			✓		✓			✓	✓		✓			
Zhanna Sadirmekova		✓		✓		✓	✓		✓	✓		✓		

C : **C**onceptualization

M : **M**ethodology

So : **S**oftware

Va : **V**alidation

Fo : **F**ormal analysis

I : **I**nterpretation

R : **R**esources

D : **D**ata Curation

O : **O**riginal Draft

E : **E**diting

Vi : **V**isualization

Su : **S**upervision

P : **P**roject administration

Fu : **F**unding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

- [1] T. Allen, B. Cooper, and R. Ge, "Fine-grain Quantitative Analysis of Demand Paging in Unified Virtual Memory," *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 1, pp. 1–24, Mar. 2024, doi: 10.1145/3632953.
- [2] J. A. Nestor and Z. Yin, "Work in Progress: A Visualization Aid for Learning Virtual Memory Concepts," in *ASEE Annual Conference and Exposition, Conference Proceedings*, Minneapolis, 2022, doi: 10.18260/1-2--40560.
- [3] Z. Lian, Y. Li, Z. Chen, S. Shan, B. Han, and Y. Su, "eBPF-based Working Set Size Estimation in Memory Management," in *Proceedings of International Conference on Service Science, ICSS*, IEEE, May 2022, pp. 188–195, doi: 10.1109/ICSS55994.2022.00036.
- [4] Y. C. Chen, C. F. Wu, Y. H. Chang, and T. W. Kuo, "Exploring Synchronous Page Fault Handling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 3791–3802, Nov. 2022, doi: 10.1109/TCAD.2022.3197517.
- [5] I. Doron-Arad and J. Naor, "Non-Linear Paging," in *Leibniz International Proceedings in Informatics, LIPIcs*, 2024, doi: 10.4230/LIPIcs.ICALP.2024.57.
- [6] A. E. Dyusembaev, "On one approach to the problem of segmenting programs," *Doklady Akademii Nauk*, vol. 329, no. 6, pp. 712–713, 1993.
- [7] M. N. Satymbekov, I. T. Pak, L. Naizabayeva, and Ch. A. Nurzhanov, "Multi-agent grid system Agent-GRID with dynamic load balancing of cluster nodes," *Open Engineering*, vol. 7, no. 1, pp. 485–490, Dec. 2017, doi: 10.1515/eng-2017-0054.
- [8] M. K. Y. Ngetich, C. Otieno, M. Kimwele, and S. Gitahi, "Advancements in Code Restructuring: Enhancing System Quality through Object-Oriented Coding Practices," in *INES 2023 - 27th IEEE International Conference on Intelligent Engineering Systems 2023, Proceedings*, IEEE, Jul. 2023, pp. 125–130, doi: 10.1109/INES59282.2023.10297764.
- [9] M. K. Y. Ngetich, D. C. Otieno, and D. M. Kimwele, "A Model for Code Restructuring, A Tool for Improving Systems Quality In Compliance With Object Oriented Coding Practice," *International Journal of Computer Applications Technology and Research*, vol. 8, no. 5, pp. 196–200, May 2019, doi: 10.7753/ijcatr0805.1010.
- [10] V. Siladi, M. Povinsky, L. Trajtel, and M. Satymbekov, "Adapted parallel quine-McCluskey algorithm using GPGPU," in *2017 IEEE 14th International Scientific Conference on Informatics*, Nov. 2017, pp. 327–331, doi: 10.1109/INFORMATICS.2017.8327269.
- [11] M. Agnihotri and A. Chug, "Severity Factor (SF): An aid to developers for application of refactoring operations to improve software quality," *Journal of Software: Evolution and Process*, vol. 36, no. 5, May 2024, doi: 10.1002/smr.2590.
- [12] M. Agnihotri and A. Chug, "Understanding the effect of batch refactoring on software quality," *International Journal of System Assurance Engineering and Management*, vol. 15, no. 6, pp. 2328–2336, Jun. 2024, doi: 10.1007/s13198-023-02247-x.
- [13] D. Al-Fraihat, Y. Sharrah, A. R. Al-Ghuwairi, M. Al Elaimat, and M. Alzaidi, "Detecting and resolving feature envy through automated machine learning and move method refactoring," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 14, no. 2, pp. 2330–2343, Apr. 2024, doi: 10.11591/ijece.v14i2.pp2330-2343.
- [14] G. Turken, L. Naizabayeva, M. Satymbekov, and Z. Abdiakhmetova, "Research and Development of Enterprise Data Warehouse Based on SAP BW Modeling," in *2023 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, Astana, Kazakhstan: IEEE, May 2023, pp. 5–9, doi: 10.1109/SIST58284.2023.10223551.
- [15] B. Arasteh, M. Abdi, and A. Bouyer, "Program source code comprehension by module clustering using combination of discretized Gray wolf and genetic algorithms," *Advances in Engineering Software*, vol. 173, p. 103252, Nov. 2022, doi: 10.1016/j.advengsoft.2022.103252.
- [16] Z. Doumchariyeva et al., "Reliability Reserve: A Markov Chain-Based Metric for Real-Time Operator Decision Support in Ayran Fermentation," *Informatics*, vol. 13, no. 7, p. 105, Jul. 2026, doi: 10.3390/informatics13070105.
- [17] A. E. Dyusembaev, *Mathematical models of program segmentation*, Moscow: FIZMATLIT, 2001.
- [18] P. J. Denning, "The working set model for program behavior," in *Proceedings of the ACM Symposium on Operating System Principles - SOSOP '67*, New York, New York, USA: ACM Press, 1967, pp. 15.1-15.12, doi: 10.1145/800001.811670.
- [19] P. J. Denning, "Working Set Analytics," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–36, Nov. 2021, doi: 10.1145/3399709.
- [20] R. W. Carr and J. L. Hennessy, "WSCLOCK---a simple and effective algorithm for virtual memory management," in *Proceedings of the Eighth Symposium on Operating Systems Principles - SOSOP '81*, New York, New York, USA: ACM Press, 1981, pp. 87–95, doi: 10.1145/800216.806596.
- [21] A. S. Morozov and D. A. Tussupov, "Minimal Predicates for Δ -Definability," *Algebra Logic*, vol. 59, no. 4, pp. 328–340, Sep. 2020, doi: 10.1007/s10469-020-09605-9.
- [22] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, Jun. 2002, doi: 10.1145/511399.511340.
- [23] C. A. Waldspurger, "Memory resource management in VMware ESX server," *Operating Systems Review (ACM)*, vol. 36, no. Special Issue, pp. 181–194, 2002, doi: 10.1145/844128.844146.
- [24] J.-H. Chiang, H.-L. Li, and T.-C. Chiueh, "Working Set-based Physical Memory Ballooning," in *ICAC '13 10th International Conference on Automatic Computing*, 2013, pp. 95–99.
- [25] M. Mustafin, M. Mansurova, A. Zhakyp, A. Mukhanbet, Y. Kenzhebek, and N. Kassymbek, "Efficiency of Parallel Machine Learning Algorithms on Multiprocessor and GPU Architectures," in *Parallel Computing Technologies*, Ed., Cham: Springer Nature Switzerland, 2026, vol. 16185, pp. 217–230, doi: 10.1007/978-3-032-06751-7_15.

- [26] A. Murzakhmetov, G. Borankulova, A. Bapanov, and G. Altybayev, "Mathematical Analysis of Page Fault Minimization for Virtual Memory Systems Using Working Set Strategy," *Information (Switzerland)*, vol. 16, no. 10, p. 829, Sep. 2025, doi: 10.3390/info16100829.

BIOGRAPHIES OF AUTHORS



Aigul Tungatarova    defended her Ph.D. thesis at the Altynsarin National Academy of Education, Astana. Currently, she is an associate professor of the Department of Information Systems of M.Kh Dulaty Taraz Regional University, Taraz, Kazakhstan. Research interests: information technologies in education, information security, information protection, and methods of teaching disciplines of the specialty. She can be contacted at email: at.tungatarova@dulaty.kz.



Aslanbek Murzakhmetov    received the B.Sc. and M.Sc. degrees in computer science from M.Kh. Dulaty Taraz State University, Taraz, Kazakhstan, in 2010 and 2012, respectively, and the Ph.D. degree in information systems from al-Farabi Kazakh National University, Almaty, Kazakhstan, in 2022. He is an associate professor and leading scientific researcher at the Department of Information Systems, M.Kh. Dulaty Taraz University, Taraz, Kazakhstan. Research interests: pattern recognition and classification, optimization systems, computer systems engineering, programming methods, multi-agent systems, and data mining. He can be contacted at email: aslanmurzakhmet@gmail.com.



Gaukhar Borankulova    received the B.Sc. in Computer Science, in 1999 and M.Sc. degree in Economics, in 2005 and "candidate of technical sciences" degree in 2011 from M.Kh. Dulaty Taraz State University, Taraz, Kazakhstan. Associate professor and project manager at the Department of Information Systems, M.Kh. Dulaty Taraz University, Taraz, Kazakhstan. Was a member of the LMPI project in program Erasmus+. Research interests: fiber-optic technologies, microprocessor systems, computer graphic, image processing, and computer systems engineering. She can be contacted at email: gs.borankulova@dulaty.kz.



Arseniy Bapanov    associate professor and scientific researcher of the Department of Information Systems at M.Kh. Dulaty Taraz University, Taraz, Kazakhstan. He received the B.Sc. and M.Sc. degrees in Computer Science from M.Kh. Dulaty Taraz State University, in 2009 and 2011. In 2020 he graduated from the doctoral program of L.N. Gumilyov Eurasian National University, Astana, Kazakhstan. He underwent scientific internship at Novosibirsk State University (Novosibirsk, Russia). His research interests include development methods of text information processing and duplicate search problems, classification and clustering of documents, big data. He can be contacted at email: a.bapanov@apa.kz.



Zhanna Sadirmekova    holds Ph.D. in information systems from L.N. Gumilyov Eurasian National University, Astana, Kazakhstan. She post-graduated Federal State Autonomous Educational Institution of Higher Education "Novosibirsk National Research State University" in Physics and Astronomy. Currently, she is an associate professor at School of Software Engineering, Astana IT University, Astana, Kazakhstan. Research interests: information technology in education, artificial intelligence, digital library, and ontology. She can be contacted at email: zhanna.sadirmekova1@astanait.edu.kz.