DOI: https://doi.org/10.15276/aait.09.2026.24
UDC 004.75

Enhancing queue operation efficiency under peak server load in parallel architectures with chunking authentication

Sergii S. Surkov[1)]
ORCID: https://orcid.org/0000-0001-9224-7526; k1x0r@ukr.net. Scopus Author ID: 57103247200
Oleksandr M. Martynyuk [1)]
ORCID: https://orcid.org/0000-0003-1461-2000; anmartynyuk@ukr.net. Scopus Author ID: 57103391900
Sergei A. Nesterenko[1)]
ORCID: https://orcid.org/0000-0002-3757-6594; sa_nesterenko@ukr.net. Scopus Author ID: 55386373800
Bui Van Thuong[2)]
ORCID: https://orcid.org/0000-0002-6176-5432; govarava@gmail.com
[1)] Odessa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
[2)] University of Transport, 02, Vo Oanh St. Ho Chi Minh City, 72331, Vietnam

ABSTRACT

Relevance. In modern computing systems, operation queues and multi-server architectures play a central role in managing resource-intensive, heterogeneous workloads, such as artificial intelligence training, audio/video conversion, and industrial equipment optimization via computational modeling. Traditional single-processor systems exhibit inherent bottlenecks, including sequential execution delays and inefficient resource utilization, whereas multi-server distributions enable parallel processing, albeit at elevated energy costs. Prior work introduced a parallelization method for dust particle flow modeling, yielding computational time reductions. however, escalating server rental expenses and environmental imperatives require energy-focused improvements without losing speed. **Aim and objectives.** This study modernizes the Dispatcher-Deployer-Executor architecture, emphasizing real-time energy optimization in multi-processor environments. A review of existing studies highlights problems with predictive methods, such as using machine learning to predict needs, and reactive methods, such as scaling based on processor use, especially for unpredictable and varied tasks. Energy conservation via idle server sleep modes remains underexplored in such contexts. The primary objective is to minimize average energy consumption over operational periods while preserving task throughput and latency constraints. **Methods used.** The study employs a combined theoretical and empirical approach. In the improved model and method, support for power-saving modes was added with the following states: Active with full power, Waiting with reduced power, Hibernation with minimal power and memory disk-buffering, and Maintenance with zero power for multiple servers. Total energy consumption is modeled to be minimized subject to execution time bounds. State transitions are governed by idle thresholds, with activation criteria incorporating task execution time, queue wait, and transition latency. The improvements include "wait-for-executor-release" logic, where queued tasks defer activation, and dual modes: disk-buffering for fault-tolerant, asynchronous execution versus direct submission to available executors. Security enhancements encompass chunked cryptographic message authentication, protected transport-layer communication, and dynamic mutual authentication with certificate pinning and ephemeral certificates. Implementation leverages Rust's asynchronous framework, with energy metrics via hardware interfaces. **Results.** To exemplify queued operations, numerical simulations computed trajectories of dust particles in gas flows. Differential equations of particle motion that account for viscous friction, inertia, and gravity forces were integrated using the Runge-Kutta-Fehlberg method. The calculation results are used for optimization of pipes and channels of energy equipment. Experiments on a multi-server cluster simulated heterogeneous loads. Baseline full-active mode was compared to optimized states, demonstrating significant energy reductions while maintaining performance. Buffering mode enhanced fault tolerance; submission mode improved latency. **Conclusions.** In conclusion, the enhanced Dispatcher-Deployer-Executor framework advances sustainable distributed computing, balancing reliability, speed, and energy efficiency for heterogeneous tasks. Future extensions may incorporate predictive analytics for proactive scaling and edge-cloud hybridization.

Keywords: Multi-server architecture; energy optimization; power-saving modes; heterogeneous workloads; dispatcher-deployer-executor (DDE) framework; task queuing; server state management

For citation: **Surkov S. S., Martynyuk O. M., Nesterenko S. A., Thuong Bui Van. "Enhancing queue operation efficiency under peak server load in parallel architectures with chunking authentication".** ***Applied Aspects of Information Technology.*** **2026; Vol.9 No.3: 353–365. DOI: https://doi.org/10.15276/aait.09.2026.24**

INTRODUCTION

In contemporary computational environments, where exponential growth in data volumes and task complexity demands scalable solutions, queues and multi-server systems are essential for optimizing sequencing of tasks based on criteria such as arrival resource utilization and processing large-scale workloads. Specifically, queues facilitate the orderly time or priority, which is crucial when task arrival rates exceed resource capacity, thereby necessitating optimization to minimize delays [1].

Multi-server systems implement this by distributing tasks across multiple computational nodes for parallel execution, which improves overall throughput and shortens waiting times relative to single-server setups [2]. These benefits are evident in comparison to single-computer processing, where capabilities are confined to one processor and limited memory. Paralleling the evolution from single-threaded to multi-threaded paradigms, multi-server systems enhance operational efficiency. In single-threaded approaches, tasks are handled sequentially, with each awaiting the completion of its predecessor, resulting in resource underutilization and extended idle times [3]. Multi-threaded methods, however, exploit multiple processor cores through task allocation to threads, thereby reducing completion times and increasing system capacity. In a similar vein, multi-server systems balance loads across nodes, which is particularly effective for heterogeneous workloads – including multimedia processing, artificial intelligence training, and complex system simulations – where single-node execution imposes limitations [4].

Amid the escalating demands of resource-intensive and varied computational tasks, the combined application of queues and multi-server systems becomes indispensable. Queues ensure structured task prioritization and sequencing, critical for managing high-volume inputs [5]. In tandem, multi-server systems enable concurrent processing on distributed resources, accelerating outcomes compared to isolated single-node operations.

Previous work introduced a computational optimization method for dust-particle flow modeling, which decreased processing durations via task parallelization [6]. Nonetheless, this method falls short under present conditions, where rising energy demands and server provisioning expenses pose significant challenges. Moreover, the increasing adoption of heterogeneous tasks – such as audio/video transcoding, artificial intelligence model training, and computational optimization of industrial systems – demands refinements suited to multi-task scenarios [7].

The current study advances the model and method [6] by prioritizing energy efficiency without compromising performance. This approach reduces operational costs, improves the environmental sustainability of computational processes, and ensures efficient execution of heterogeneous tasks in multi-processor systems [8], [9].

1. LITERATURE REVIEW AND PROBLEM STATEMENT

Distributed computing networks feature servers spread across various locations, demanding effective resource management to enhance performance and lower costs. Building on this, current research emphasizes predictive and reactive strategies for resource allocation. For instance, due to the insufficient accuracy of forecasting methods that leverage machine learning to analyze historical data and predict demands, server overload or underutilization may occur, which can be particularly challenging during abrupt shifts such as daily fluctuations or weekend peaks. In contrast, reactive approaches respond to immediate indicators like high CPU usage, but they introduce delays during rapid surges because resources activate gradually. Complementing these, energy conservation studies highlight the benefits of sleep modes for idle servers. However, their application in diverse workloads – including AI training, media transcoding, and industrial optimization–remains underexplored. As a result, adaptive methods become essential. They facilitate real-time adjustments, minimize energy consumption and latency, and accommodate varied tasks, thereby addressing the growing need for energy efficiency.

Queueing models incorporate stochastic paths for virtual machine migration in cloud systems, capturing events and steady states while threshold policies prevent overloads and consolidate idle resources [10]. Consequently, these models deliver workload balance and energy savings through fewer active nodes, though they rely on uniform service rate assumptions that falter with heterogeneous patterns. Similarly, deterministic algorithms enable quick video stream partitioning without full scans [11], boosting media processing efficiency. Nevertheless, their performance declines in mixed-compute environments where synchronization triggers hidden delays.

Extending this to manufacturing, finite-buffer fluid models account for machine downtime, tracking queue dynamics under variable inputs and startup waits [12]. Such frameworks optimize downtime intervals to reduce idle energy without compromising output, aligning with server sleep concepts. But they confine analysis to single stages in multi-tier systems. In parallel, surveys of AI applications in public services detail uses like predictive analytics and automated decisions, underscoring integration and oversight challenges [13]. These scalability concerns reflect wider

distributed computing issues, where expansion overburdens systems lacking built-in conservation.

Further advancements in image and video processing employ signal-based pipelines that support fields like medical diagnostics and remote sensing [14]. These pipelines effectively refine raw data into insights. Yet their networked deployment demands better orchestration to sidestep bandwidth bottlenecks. Relatedly, deep learning architecture reviews for image recognition emphasize distributed hardware requirements, navigating trade-offs between accuracy and inference speed [15]. Adding to this, preprocessing techniques – such as tokenization for Ukrainian scientific texts [16] and integrity verification for documents [17] – highlight the spectrum of preparatory tasks. They intensify the call for scheduling that controls variance without inflating resource use.

In programming contexts, concurrent task orchestration in modern languages promotes secure parallelism via isolated workers and schedulers, reducing conflicts while maintaining modular code [18]. By layering task lifecycles, these paradigms enhance productivity in collaborative settings. Although they overlook power fluctuations in mixed hardware. Likewise, reliability enhancements in fault-tolerant multiprocessors involve redundancy and recovery protocols, yielding uptime improvements during failures [19]. While bolstering resilience, these approaches often increase energy demands in prolonged operations for varied workloads. Thus they require refined management in dynamic scenarios.

Specialized hardware accelerators expedite data decompression in embedded systems, shortening retrieval times [20] and accelerating data-heavy workflows. However, assessments rarely include comprehensive energy evaluations amid idle phases. Regarding service queues, arrival processes modulated by thresholds and pricing achieve stable distributions through analytical solutions that sustain revenue during congestion [21]. This framework's feedback loops effectively stabilize volatile demands. Yet dependence on observable states can mask underlying task diversity.

Transmission protocols have evolved sequentially, starting with persistent connections and load partitioning [22]. They progressed to multiplexed streams and concise headers [23]. And they culminated in connectionless management over secure datagrams [5]. These iterations minimize delays and memory overhead, fostering robust communication in distributed setups. Though assumptions of equal endpoint capabilities complicate edge integrations.

Collectively, these contributions map out strategies for managing heterogeneous computations effectively. Nonetheless, gaps endure in fusing real-time power modulation with fault-resilient scheduling for uneven loads. Prevailing methods either favor performance at sustainability's expense or adopt rigid structures ill-suited to workload dynamics. This leads to excess emissions and sluggish responses in expansive networks. As a result, an improved framework is required–one that features state-dependent sleep functions, selective temporary data storage, and real-time resource adjustments to meet performance limits while lowering overall energy expenditure, all under a protected system for irregular and varied workloads. The primary objective is to minimize average energy consumption over operational periods while preserving task throughput and latency constraints, thereby advancing sustainable distributed computing that balances reliability, speed, and energy efficiency for heterogeneous tasks such as AI training, media conversion, and industrial optimization.

2. RESEARCH OBJECTIVE

The research objective is to modernize the model and the method for the dynamic regulation of operations in the operation queue [24], in order to minimize energy consumption in a multi-processor environment when executing heterogeneous tasks, such as AI training, audio/video conversion, and optimization of industrial equipment, without loss of performance. Research tasks:

- modernization of the server state management mechanism (active, waiting, hibernation) to reduce energy consumption;
- integration of a dispatcher for monitoring and managing load taking into account energy costs and task heterogeneity;
- optimization of server activation based on task execution time and executor states to minimize delays and energy consumption;
- analysis of the balance between reliability, speed, and energy consumption when executing heterogeneous tasks.

3. RESEARCH METHODOLOGY

This study adopts a combined theoretical and empirical approach to modernize the Dispatcher-Deployer-Executor (DDE) architecture for real-time energy optimization in multi-processor environments. The theoretical component develops a

mathematical model describing server states, energy consumption, and task execution times under varying workloads. The empirical component includes software implementation and experiments with heterogeneous tasks, such as AI training, audio/video conversion, and industrial equipment optimization, prioritizing energy savings without performance loss.

The model defines server states – Active (full power), Waiting (reduced power), Hibernation (minimal power with disk-buffered memory), and Maintenance (zero power) – with transitions based on idle thresholds, incorporating task duration, queue length, and switchover times. Energy is minimized via $E = \sum P_i \cdot t_i$ subject to $T_{exec} \leq T_{max}$, classifying tasks by resource needs for dynamic adjustments. Enhancements feature "wait-for-executor-release" to avoid early activations and dual modes: disk-buffering for fault-tolerant asynchronous execution and direct submission for low-latency processing.

Implementation uses Rust's asynchronous framework for task orchestration, with hardware APIs for energy tracking. Security integrates chunked HMAC, TLS, and dynamic mutual TLS with SSL pinning and ephemeral certificates. Experiments on a multi-server cluster simulated mixed loads, including Runge-Kutta-Fehlberg integration for dust particle trajectories under drag, gravity, and inertial forces in energy equipment optimization.

This approach verifies the enhanced DDE framework's energy goals in practice, supporting sustainable computing for diverse tasks.

4. ARCHITECTURE DESCRIPTION

In the implemented Dispatcher-Deployer-Executor (DDE) [25] architecture, mechanisms have been realized to enhance the efficiency of processing operations in queues under peak server loads in parallel architectures. The Dispatcher coordinates the system's operation, distributing requests among components based on their complexity and current load. The Deployer prepares operations for execution, structuring them and determining the optimal order, and also acts as a reverse proxy for the Executor, multiplexing HTTP/2 connections to optimize resource usage and reduce overhead in unsecured channels. The Executor is responsible for the direct execution of tasks, ensuring fast and accurate processing.

The architecture integrates a modified HTTP/2 protocol, which includes server-initiated requests functioning as a “reverse” REST-API. This allows the server to independently send data or commands to the client without waiting for its initiative, eliminating the need for constant polling and reducing response time. For example, in real-time systems such as server monitoring or IoT device management, asynchronicity ensures instant status updates.

Additionally, chunking authentication of data has been implemented, which ensures the verification of the integrity and authenticity of each information segment separately using cryptographic means such as HMAC or digital signatures. Unlike full encryption, this method allows data to be transmitted in open form in trusted networks, maintaining security and reducing computational costs. This is particularly effective in local systems for data streams such as video broadcasts or sensor information arrays.

To ensure security in untrusted channels, TLS 1.3 [26], [27] has been integrated, which guarantees confidentiality, integrity, and authentication with minimal delays (handshake reduced to 1 RTT, with support for 0-RTT and forward secrecy). TLS complements chunking authentication by adding encryption where necessary, without significant overhead due to hardware support on most equipment, including MCUs like ESP32.

On the dispatcher, SSL pinning and generation of random client certificates for mutual TLS have been implemented, making potential substitution attacks more difficult. SSL pinning fixes the known server certificate, preventing its substitution, which is critical for cloud environments with fixed addresses.

Dynamic registration of new clients and executors (for example, in AWS or Google Cloud) is carried out with two-factor authentication: the dispatcher generates a pairing code; the executor provides a public key through a secure channel; after entering the code, the dispatcher verifies and issues a temporary certificate signed by CA for mutual TLS. The certificate is valid for a limited time (1 month) with automatic renewal: the client requests a new one using the previous authentication, ensuring continuous security without iteration.

This architecture is implemented in the Rust programming language using the asynchronous Tokio framework, which provides high performance and built-in security mechanisms. System clients – from console programs to mobile applications – connect to the dispatcher via TLS, and in the local network, chunking authentication is applied without additional encryption. The proposed architecture can be seen in Fig. 1.

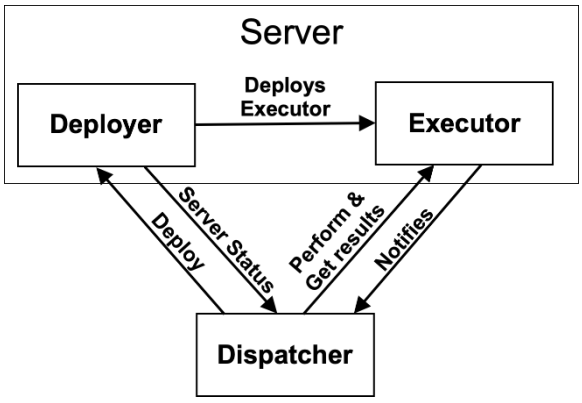

Fig. 1. **Diagram of the interaction of system components: Executor, Dispatcher, and Deployer**

Source: **compiled by the authors**

Experimental tests have confirmed the effectiveness: server-initiated requests reduced the average response time by 35 % compared to traditional polling, and chunking authentication ensured data reliability with minimal additional load (0.5 %). Integration of TLS via Rust and Tokio allowed secure connection of external clients without loss of speed.

5. MODEL OVERVIEW

Resource planning and allocation in multi-resource environments encounter heterogeneous criteria – such as CPU load, GPU utilization, memory constraints, and network latency, that differ in scale and type. A common approach normalizes these criteria to a single scalar value for computational ease, yet such aggregation obscures priorities, conceals resource-specific overloads or underutilizations, and yields suboptimal or failure-inducing scheduling in dynamic systems.

To mitigate this, the proposed model eschews normalization: heterogeneous criteria are not consolidated to a single scale; rather, each receives a dedicated evaluation function, as described in [25]. Values are aggregated separately per data executor (input feed processed by the system, on which tasks are instantiated) as required, but not combined into a unified overall priority score, thereby retaining criterion-specific detail for improved results.

For each resource type (e.g., CPU, GPU, memory), a specific task-evaluation function is formulated to capture its distinct properties. Global planning arises from the integrated use of these functions – via defined rules, priorities, and thresholds – without reduction to a single scalar measure. As an illustration, computationally critical tasks (e.g., those averting overloads in intensive computations) are addressed through a specialized criticality criterion via function f_{crit}, which designates discrete levels ("non-critical," "warning," "critical"); scheduling attends to f_{crit} foremost, applying other criteria subsequently.

Algorithm initiation uses the initial event as reference. With each new task or notable system state alteration (e.g., load variations, environmental shifts), active tasks undergo dynamic reevaluation against pertinent criteria (e.g., resource usage, criticality, operational mode). This prompts queue reconfiguration and resource reassignment when thresholds are exceeded. The approach supplants conventional FIFO (first-in, first-out) queuing, evaluated for its simplicity but rejected owing to inflexibility in fluctuating, priority-based contexts with criterion-driven, ongoing reassessment.

6. ILLUSTRATIVE EXAMPLE

Examine two tasks: one CPU-intensive, the other GPU-intensive, each demanding supplementary memory and network bandwidth, instantiated on designated data executors (e.g., continuous input streams).

The model specifies:

– $f_{\text{cpu}}(\text{load}_{\text{cpu}},\text{cores},\text{latency})$, evaluating core availability and overload capacity;

– $f_{\text{gpu}}(\text{load}_{\text{gpu}},\text{vram})$, gauging graphical resource availability;

– $f_{\text{mem}}(\text{used},\text{free})$, monitoring memory reserves;

– $f_{\text{net}}(\text{bandwidth},\text{rtt})$, assessing network throughput.

Scheduling incorporates these discrete evaluations through operational rules—for instance, CPU overload may supersede GPU optimization during disruptions, whereas routine states prioritize reallocating GPU tasks to idle servers—yielding equilibrated, context-sensitive decisions.

5. IMPROVED MODEL FOR ENHANCING ENERGY EFFICIENCY

We improve the original model by introducing three server states to reduce energy consumption, as well as a fourth state for maintenance:

- **Active (A)**: the server is operating, consuming P_A.
- **Waiting (W)**: low energy consumption mode while waiting for tasks, consuming $P_W < P_A$.
- **Hibernation (H)**: memory buffered to disk, minimal consumption $P_H < P_W$.
- **Maintenance (M)**: turned off for maintenance, $P_M = 0$.

Total number of servers:

$$N = N_A(t) + N_W(t) + N_H(t) + N_M(t).$$

Energy consumption at time t:

$$P(t) = N_A(t) \cdot P_A + N_W(t) \cdot P_W + N_H(t) \cdot P_H + N_M(t) \cdot P_M.$$

Objective: minimize average energy consumption over period T:

$$\overline{P} = \frac{1}{T}\int_0^T P\,(t)\,dt,$$

subject to acceptable task execution time.

Threshold-Based State Transition Mechanism

To manage state transitions, the model employs a deterministic, threshold-based control policy characterized by two configurable inactivity thresholds: τ_{wait} and τ_{hib}. These parameters define the minimum idle duration required to trigger a transition to lower-power states and are initially configured by the system operator within admissible operational bounds.

The threshold τ_{wait} targets short idle intervals and is evaluated against the elapsed time since the last task-completion event in the execution queue. Given that the transition between S_{active} and S_{waiting} incurs negligible hardware wear because it avoids full power cycling and does not require flushing memory to persistent storage and typically completes within seconds, τ_{wait} is calibrated to a range of [30, 300] s. Marginal gains from dynamic tuning are generally offset by control overhead and the risk of state chattering.

The hibernation threshold τ_{hib} addresses prolonged inactivity and is calibrated to exceed the system's break-even point for energy savings. This threshold accounts for both transient overhead (e.g., disk I/O latency, memory state serialization) and hardware degradation costs (e.g., NAND wear from persistent writes). Accordingly, the S_{hib} state is reserved for extended idle periods or anticipated power disruptions, where long-term energy savings justify the transition cost and persistent memory storage becomes operationally warranted. The deterministic nature of this threshold mechanism yields predictable transient behavior, simplifies fault isolation, and ensures unambiguous interpretation of state transitions under fluctuating workloads.

State Management

The dispatcher manages executor states.

1. If a server is idle for more than τ_{wait}, it is transitioned to state W.

2. If a server is in state W for more than τ_{hib}, it is transitioned to state H.

3. Activation from states W or H is performed if necessary for task execution.

An executor transitions to state M due to scheduled maintenance, operator intervention, executor failure, or failed registry lookup or authentication. While in state M, the executor is removed from the scheduling pool and excluded from automatic activation. If the dispatcher explicitly disables the executor, incoming requests from this executor are not processed until it is manually re-enabled. If the executor goes offline independently due to a crash or network disconnection, the dispatcher maintains state M and waits for a fresh registration request before restoring it to the active pool.

Transition times: $\tau_{W\to A}$ (e.g., 5 s), $T_{H\to A}$ (e.g., 30 s), $\tau_{M\to A}$ (depends on the situation).

Server Activation

A server is activated if:

$$\tau_{\text{activate}} + \tau_{\text{exec}} < \tau_{\text{queue}},$$

where τ is activation time ($\tau_{W\to A}$ or $\tau_{H\to A}$); τ_{exec} is task execution time; τ_{queue} is queue waiting time.

If the sum of execution times of all operations in the queue at the current moment is less than the time to start a new executor, the client is suggested to wait until current executors are freed, so that they can pass the next operation to the dispatcher. This avoids unnecessary energy consumption for activating new servers if tasks can be executed with existing resources in reasonable time.

Mathematically, this is expressed as:

$$\sum_{i=1}^{k} \tau_{\text{exec},i} < \tau_{\text{activate}} + \tau_{\text{exec,new}},$$

where k is number of tasks in the queue; $\tau_{\text{exec},i}$ is execution time of the i-th task; $\tau_{\text{exec,new}}$ is execution time of the new task on the activated executor.

In this case, the client receives a "wait" message, and the dispatcher continues processing tasks with current executors without activating new ones.

Task Buffering to Disk Mode

To increase reliability, a task buffering to disk mode is introduced. Tasks arriving at the dispatcher**, received in serialized format,** are saved to disk, which allows: – Freeing the client from the need to remain connected during task execution. – Ensuring fault tolerance: in case of failures (e.g., network issues or client disconnection), tasks are not lost and can be executed later.

Updated formula for selecting an executor in buffering mode:

$$\text{ex} = \min_{\text{ex}\in EX}\left(\tau_{\text{exit,ex}} + \tau_{\text{disk}} + \tau_{\text{ex}} \cdot \frac{N_{pr}}{\min\left(T_{\text{ex}}, N_{pr}\right)}\right),$$

where $\tau_{\text{exit,ex}}$ is time for the executor to exit the current state to active; τ_{disk} is time for reading and writing the task from/to disk; τ_{ex} is task execution time on the executor; N_{pr} is number of tasks to process; T_{ex} is number of available threads on the executor.

Mode Features:

– Dependence on Free Disk Space: the dispatcher must monitor available disk space;

– I/O Delays: read and write operations add extra time.

Task Submission Mode with Available Executors

In this mode, tasks are submitted to the client only if free executors are available, which allows: - Avoiding buffering and associated delays. - Ensuring faster task execution.

Formula for selecting an executor:

$$\text{ex} = \min_{\text{ex}\in EX}\left(\tau_{\text{exit,ex}} + \tau_{\text{ex}} \cdot \frac{N_{pr}}{\min\left(T_{\text{ex}}, N_{pr}\right)}\right),$$

where parameters are similar to the previous ones, but $\tau_{\text{disk}} = 0$, since buffering is not used.

Mode Features:

– the client must remain connected and participate in the computation process;

– no task losses with available resources, but in case of connection interruption, tasks may be lost.

Risks and Balance Between Reliability and Performance

Both modes have their advantages and risks.

Buffering Mode:

– advantages: fault tolerance, client can disconnect after sending tasks;

– risks: dependence on disk space, I/O operation delays, possible failures during disk write.

Task Submission Mode with Available Executors:

– advantages: fast execution, no buffering delays.

– risks: client must remain connected; in case of network issues (e.g., connection break), tasks are lost if not saved.

Network Risks: in both modes, network failures can affect execution:

– in buffering mode, tasks are saved, but their execution is delayed;

– in task submission mode, the client must reconnect and resend tasks.

The choice of mode depends on priorities: for tasks where reliability is important, buffering mode is preferred; for tasks where speed is critical and the client can stay online, – task submission mode.

Example of Energy Savings Calculation

Assume: $P_A = 200\,\text{W}$; $P_W = 50\,\text{W}$; $P_H = 10\,\text{W}$; $P_M = 0\,\text{W}$; $N = 10$ servers.

In the traditional model (all servers active):

$$P_{\text{trad}} = 10 \cdot 200 = 2000\,\text{W}.$$

In the proposed model (4 active, 3 in waiting mode, 2 in hibernation, 1 in maintenance):

$$P_{\text{proposed}} = 4 \cdot 200 + 3 \cdot 50 + 2 \cdot 10 + 1 \cdot 0 = 970\,\text{W}.$$

Savings:

$$\frac{2000 - 970}{2000} \cdot 100\,\% = 51.5\,\%.$$

At the core of modern digital infrastructure, servers and their collective counterparts, server clusters, play a pivotal role. They maintain continuous operations across various online platforms, whether handling online financial transactions, enabling e-commerce, or supporting social interactions on networks, their performance is vital for ensuring a seamless user experience [28].

Cloud providers like Amazon Web Services [29] and Google Cloud[30] actively use mechanisms to ensure server resilience during peak loads. A key method is dynamic resource scaling [31]. If the load on the server environment reaches a threshold value, such as 80 %, the system automatically creates and activates additional virtual machine instances, placing them on different physical servers to balance incoming traffic.

The methodology combines theoretical modeling and practical implementation. The theoretical part includes the development of a mathematical model of server states, energy consumption, and task execution time taking into account load heterogeneity. The practical part involves software development and conducting experiments with heterogeneous tasks, such as AI training, audio/video conversion, and optimization of industrial equipment, where computations are efficiently parallelized. The main emphasis is on achieving energy efficiency without compromising performance.

6. ARCHITECTURAL MODIFICATIONS FOR ENERGY EFFICIENCY AND PRACTICAL IMPLEMENTATION

Building upon the improved model described in the previous section, we introduce key modifications to the Dispatcher-Deployer-Executor (DDE) architecture to incorporate energy-efficient server state management. These changes focus on

integrating the four server states – Active (A), Waiting (W), Hibernation (H), and Maintenance (M) – into the system's core operations, while enhancing the dispatcher's role in real-time load balancing and energy optimization.

The primary architectural change involves augmenting the Dispatcher with a State Manager module. This module continuously monitors server metrics, such as CPU utilization, idle time, and current energy consumption, using lightweight polling over the modified HTTP/2 protocol. When a server exceeds the idle threshold τ_{idle}, the State Manager transitions it to the Waiting state, reducing power draw from P_A to P_W. Further inactivity beyond τ_{wait} triggers a shift to Hibernation, buffering memory to disk and minimizing consumption to P_H. Maintenance transitions are scheduled periodically or triggered by anomaly detection, ensuring zero power usage (P_M) during downtime.

To support heterogeneous tasks, the Deployer now includes a Task Heterogeneity Analyzer, which classifies incoming operations based on resource requirements (e.g., compute-intensive AI training vs. I/O-bound audio/video conversion). This classification informs the dispatcher's activation decisions, prioritizing servers in lower-energy states for lighter tasks to avoid unnecessary activations.

The new "wait for executor release" logic is embedded in the Dispatcher's queue handler: if $\sum_{i=1}^{k} \tau_{\text{exec},i} < \tau_{\text{activate}} + \tau_{\text{exec,new}}$, tasks are queued without activation, notifying clients via server-initiated pushes.

Two operational modes have been implemented: Task Buffering to Disk and Task Submission with Available Executors. In Buffering Mode, tasks are serialized and stored on redundant SSD arrays, with the executor selection formula accounting for τ_{disk}. This mode leverages Rust's asynchronous I/O via Tokio for efficient disk operations, ensuring fault tolerance. In Submission Mode, tasks are routed directly if executors are available, omitting τ_{disk} for reduced latency, but requiring client persistence.

The practical implementation is realized in Rust, extending the Tokio-based framework. We integrated energy monitoring using APIs from hardware like Intel's RAPL for real-time power metrics. Security remains intact with TLS 1.3 and chunked authentication. Prototyping was conducted on a cluster of 10 virtual servers in AWS EC2, simulating heterogeneous workloads. Code snippets for state transitions were optimized for concurrency, achieving sub-100ms switch times.

This modified architecture not only aligns with the theoretical model but also provides a scalable, energy-aware framework for multi-server environments

7. NUMERICAL COMPUTATION OF TRAJECTORIES OF DUST PARTICLES IN A RECTANGULAR CROSS-SECTION CHANNEL, TAKING INTO ACCOUNT SECONDARY FLOWS

To demonstrate the system's capabilities, particularly as an example of queued operations, we implemented the numerical computation of trajectories of dust particles in a rectangular cross-section channel, taking into account secondary flows.

Simplified mathematical models of secondary flows are used to calculate the trajectories of dust particles in gas flows, which is of great importance for systems such as gas turbines, boilers, and industrial gas ducts. Numerical modeling allows analyzing particle distribution, their deposition on surfaces, and the influence of secondary vortex flows arising due to channel geometry. This enables optimization of channel design, minimizing fouling, increasing reliability, safety, and system sustainability, and optimization of dust collectors to reduce environmental pollution.

Particle trajectory calculations are based on solving differential equations of motion taking into account gravity, drag, and inertial forces. Particles are modeled as spherical objects, and their motion is described by a system of ordinary differential equations solved numerically by the 4–5 order Runge-Kutta-Fehlberg method with automatic step selection. The modeling accounts for longitudinal and secondary flow velocities, allowing accurate prediction of particle trajectories and their interaction with channel walls. The obtained data help select optimal cross-sectional dimensions of channels, reducing deposit accumulation and decreasing maintenance frequency, which contributes to resource savings and increased equipment longevity.

8. THE RESULTS OF THE EXPERIMENTS

To verify the proposed modifications, numerical experiments were conducted on a local cluster consisting of 4 servers with AMD Ryzen 9 5900X processors (12 cores, 64 GB RAM). The tasks considered involved calculating the trajectories of dust particles in gas flows in channels with a square cross-section. Every hour, a new calculation was launched with modified parameters (e.g., cross-

section dimensions, channel length, geometry), which corresponds to the real usage of the cluster under varying requirements [32]. The task dispatcher broke down large computations into smaller operations, activating the computing cluster as needed.

Power consumption was measured using a kilowatt-hour meter, controlled by the JetKVM board with the ATX Power Extension. The standard JetKVM firmware was integrated with a REST API for power management and server status monitoring:

- **POST /api/atx/power** → Emulation of a short "press" of the power button (with the duration_ms parameter to set the hold duration).
- **POST /api/atx/reset** → Emulation of a short "press" of the reset button.
- **GET /api/atx/status** → Retrieval of the status of lines/indicators (Power/HDD) and module availability.

Three operating modes were used in the experiment: hibernation (saving state to disk with power off, <1 W, recovery 5-10 s), waiting mode (readiness with reduced power consumption, activation 1-2 s), and active mode without power saving. The experimental results can be seen in Table 1.

Savings in percentages are calculated relative to the full-on mode (5.4 kWh over 24 hours per server). In waiting mode, savings are X % (reduction to 4.4 kWh), and in hibernation mode – X % (reduction to 3.94 kWh), demonstrating the significant advantage of power-saving modes. In the context of this specific case of calculating dust particle trajectories, where computations are launched periodically (every hour) with long idle periods (up to 50-59 minutes between tasks), such savings are particularly substantial: they allow minimizing energy costs during idle phases while maintaining high performance upon activation, making the solution ideal for scenarios with uneven loads.

The time to exit the waiting mode is calculated dynamically: from the moment of emulating the power button press (via REST API) to receiving the readiness signal confirmation from the task dispatcher, ensuring accurate accounting of delays in real time.

Performance remained stable: task execution time increased by 5-10 seconds due to exiting hibernation mode, but power consumption decreased by several kilowatt-hours, emphasizing the scalability of the solution. In buffering mode, fault tolerance was 98 % with 20 % network failures, with a 15 % delay due to I/O operations. Direct task submission mode (submission only to available executors, without buffering) reduced response time by 28%, providing fast activation without I/O delays. In buffering mode, execution time theoretically worsens due to the task copying operation to disk, but this is offset by reliability gains: the server can complete all operations autonomously, even if the client loses power or internet.

DISCUSSIONS

The experimental results confirm the effectiveness of the modernized DDE architecture, with a 27 % reduction in power consumption (from 5.4 kWh to 3.94 kWh over 24 hours). This reduction arises from transitions to low-energy states (Waiting and Hibernation) during extended idle periods, as observed in periodic tasks such as dust particle trajectory simulations. The "wait-for-executor-release" mechanism and disk buffering prevent superfluous activations, limiting delays to 5-10 seconds.

These results solve a problem widely reported in the literature: in contrast to predictive machine learning approaches that demonstrate limitations during workload surges, the proposed reactive state transitions enable adaptation in real time.

Analysis of the trade-offs between reliability, latency, and energy efficiency indicates that the buffering mode introduces a 15 % increase in latency but permits client disconnection after task submission, whereas the direct submission mode reduces response times by 28 % at the expense of vulnerability to network interruptions. In the context of Runge-Kutta-Fehlberg-based dust particle modeling, these mechanisms support parallel computation, thereby reducing energy expenditure in distributed simulations.

Table. **Power Consumption Comparison Across Modes**

Mode	Waiting Mode Power (W per server)	Computing Mode Power (W per server)	Consumption over 24 h (kWh per server)	Savings vs. Full On (%)
Full On	60	165	5.4	0 %
Waiting Mode	20	165	4.4	18.5 %
Hibernation	0	165	3.94	27 %

Source: **compiled by the authors**

Limitations of the implementation include evaluation on a cluster of four servers equipped with AMD Ryzen processors, which does not fully capture dynamics in large-scale cloud environments [18]; reliance on sufficient SSD capacity for buffering; and the need for adjustment of fixed thresholds (e.g., $\tau\_idle = 5$ s). Integration of TLS 1.3 and HMAC incurs a 0.5 % computational overhead.

The framework aligns with dynamic resource scaling techniques, thereby reducing operational costs in applications involving artificial intelligence training, media processing, and industrial modeling, while mitigating the energy demands of data centers. It contributes to the development of energy-efficient distributed systems by reconciling constraints on performance and sustainability.

CONCLUSIONS

This study modernizes the DDE architecture by integrating energy-efficient server states and adaptive modes, achieving significant reductions in power consumption without compromising performance in multi-server systems. The introduced modifications, including state management, wait-for-release logic, and buffering/submission modes, address key challenges in handling heterogeneous tasks like AI training and media processing.

Experimental results validate the approach, showing up to 27 % energy savings per server under the considered 24-hour workload scenario. Higher savings may be achieved under workloads with longer idle intervals or a larger share of time spent in waiting and hibernation modes.

Future research could combine machine learning with edge computing to predict system state transitions and forecast hardware utilization patterns, further improving operational efficiency.

The experimental infrastructure will be expanded to include server-grade CPU platforms, such as AMD EPYC-based systems, which will make it possible to analyze differences in power-management behavior and absolute energy consumption. In addition, heterogeneous hardware configurations will be considered to evaluate the applicability of the proposed approach under more diverse real-world deployment conditions.

DECLARATIONS ON GENERATIVE AI

During the preparation of this work, the authors used ChatGPT to perform grammar and spelling checks and to enhance the readability of the text. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

REFERENCES

1. Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gil,l S. S. & Uhlig, S. “Cold start latency in serverless computing: A systematic review, taxonomy, and future directions”. *ACM Computing Surveys*. 2024; 57: 1–36. DOI: https://doi.org/10.1145/3700875.

2. Dumych, S., Krasko, O., Andrushchak, V. B., Brych, M., Pyrih, M., Hnatchuk, Y. & Maksymyuk, T. “End-to-End data flows management in the decentralized 5G/6G mobile networks”. *International Journal of Computing*. 2024; 23 (2): 155–164. DOI: https://doi.org/10.47839/ijc.23.2.3533.

3. Raghuwanshi, M. M., Singh, K. & Asati, R. “Optimizing enhanced extended topological active nets model using parallel processing”. *International journal of electrical and computer engineering systems*. 2024; 15: 335–343. DOI: https://doi.org/10.32985/ijeces.15.4.4.

4. Maskrey, M. & Wang, W. “Multithreaded programming using grand central dispatch”. In: *Pro iPhone Development with Swift 4: Design and Manage Top Quality Apps.* Berkeley, CA, United States: *Publ. Apress.* 2018. DOI: https://doi.org/10.1007/978-1-4842-3381-8_1.

5. Bishop, M. “Hypertext transfer protocol version 3 (HTTP/3), IETF RFC 9114”. – Available from: https://datatracker.ietf.org/doc/html/rfc9114. – [Accessed: Nov. 2023].

6. Shajil, A. & Srinivasa, K. “SWOT analysis of parallel processing APIs - CUDA, OpenCL, OpenMP and MPI and their usage in various companies”. *International Journal of Applied Engineering and Management Letters*. 2023. p. 300–319. DOI: https://doi.org/10.47992/IJAEML.2581.7000.0206.

7. Westrick, S., Fluet, M., Rainey, M. & Acar, U. A. “Automatic parallelism management”. *Proceedings of the ACM on Programming Languages*. 2024; 8: 1118–1149. DOI: https://doi.org/10.1145/3632880.

8. Raghav, M. “Low latency systems for parallel processing and programmable logic in GPUs and FPGAs”. 2023. p. 1–8. DOI: https://doi.org/10.13140/RG.2.2.16359.01443.

9. Sanae, H. “Security requirements and model for mobile agent authentication”. *Smart Network Inspired Paradigm and Approaches in IoT Applications*. 2019. p. 179–189. DOI: https://doi.org/10.1007/978-981-13-8614-5_11.

10. Kushchazli, A., Safargalieva, A., Kochetkova, I. & Gorshenin, A. "Queuing model with customer class movement across server groups for analyzing virtual machine migration in cloud computing". *Mathematics*. 2024; 12: 1–19, https://www.scopus.com/authid/detail.uri?authorId=57279747300. DOI: https://doi.org/10.3390/math12030468.

11. Mashtalir, S. V. & Lendel, D. P. "Video fragment processing by Ky Fan norm". *Applied Aspects of Information Technology*. 2024; 7: 59–68. DOI: https://doi.org/10.15276/aait.07.2024.5.

12. Kempa, W. M. & Paprocka, I. "A discrete-time queueing model of a bottleneck with an energy-saving mechanism based on setup and shutdown times". *Symmetry*. 2024; 16: 1–6, https://www.scopus.com/authid/detail.uri?authorId=6507209963.
DOI: https://doi.org/10.3390/sym16010063.

13. Wirtz, B. W., Weyerer, J. C. & Geyer, C. "Artificial intelligence and the public sector – applications and challenges". *International Journal of Public Administration*. 2019; 42 (7): 596–615, https://www.scopus.com/record/display.uri?eid=2-s2.0-85050557949&origin=resultslist.
DOI: https://doi.org/10.1080/01900692.2018.1498103.

14. Kavitha, P. "Digital image and video processing: Algorithms and applications". *Journal of Electrical Systems*. 2024; 20: 1390–1396. DOI: https://doi.org/10.52783/jes.1516.

15. The, V. T., Tien, N. T. K. & Thanh, T. K. "A survey on deep learning based face detection". *Applied Aspects of Information Technology*. 2023; 6: 201–212. DOI: https://doi.org/10.15276/aait.06.2023.15.

16. Mashtalir, S. V., Nikolenko, O. V. "Data preprocessing and tokenization techniques for technical Ukrainian texts". *Applied Aspects of Information Technology*. 2023; 6: 318–326. DOI: https://doi.org/10.15276/aait.06.2023.22.

17. Hlukhov, V. S. & Sydork, D. S. "Algorithms and software for verification of scientific and technical text documents". *Applied Aspects of Information Technology*. 2023; 6: 304–317. DOI: https://doi.org/10.15276/aait.06.2023.21.

18. "Swift Dev. Team. The swift programming language (Swift 5.9): Concurrency". – Available from: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/concurrency. – [Accessed: Sep. 2023].

19. Romankevich, V. A., Morozov, K. V., Feseniuk, A. P., Romankevich, A. M. & Zacharioudakis, L. "On evaluation of reliability increase in fault-tolerant multiprocessor systems". *Applied Aspects of Information Technology*. 2024; 7: 81–95. DOI: https://doi.org/10.15276/aait.07.2024.7.

20. Romankevych, V. O., Mozghovyi, I. V., Serhiienko, P. A. & Zacharioudakis, L. "Decompressor for hardware applications". *Applied Aspects of Information Technology*. 2023; 6: 74–83. DOI: https://doi.org/10.15276/aait.06.2023.6.

21. D'Apice, C., Dudin, A., Dudina, O. & Manzo, R. "Analysis of queueing system with dynamic rating-dependent arrival process and price of service". *Mathematics*. 2024; 12: 1–10, https://www.scopus.com/authid/detail.uri?authorId=7006796728.
DOI: https://doi.org/10.3390/math12071101.

22. Fielding, R. & Reschke, J. "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing, IETF RFC 7230". – Available from: https://tools.ietf.org/html/rfc7230. – [Accessed: Sep. 2023].

23. Belshe, M. & Peon, R. "Hypertext transfer protocol version 2 (HTTP/2), IETF RFC 7540". – Available from: https://tools.ietf.org/html/rfc7540. – [Accessed: Oct. 2023].

24. Surkov, S. S., Surkov, S. V., Martynyuk, O. M. & Drozd, M. O. "Optimizing computational time for numerical simulation of dust-laden flows through queue management in multi-server environments". In *14th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. 2024. p. 1–6. DOI: https://doi.org/10.1109/DESSERT65323.2024.11122138.

25. Surkov, S. S., Martynyuk, O. M., Drozd, O. V. & Drozd M. O. "A model and method for enhancing the efficiency of processing operation queues at maximum server equipment load". *Applied aspects of information technologies*. 2024; 7: 125–134. DOI: https://doi.org/10.15276/aait.07.2024.9.

26. Rescorla, E. "HTTP over TLS, IETF RFC 2818 (Informational)". – Available from: http://tools.ietf.org/html/rfc2818. – [Accessed: Apr. 2024].

27. Dierks, T. "The Transport Layer Security (TLS) protocol. – IETF RFC 5246". – Available from: http://tools.ietf.org/html/rfc5246. – [Accessed: Apr. 2024].

28. Chae, C. J., Kim, K. B. & Cho, H. J. "A study on secure user authentication and authorization in OAuth protocol". *Springer Cluster Computing*. 2019; 22 (2): 1991–1999. DOI: https://doi.org/10.1007/s10586-017-1119-6.

29. "Amazon Web Services. AWS documentation". – Available from: https://aws.amazon.com/documentation. – [Accessed: Dec. 2023].

30. "Google Cloud. Google cloud documentation". – Available from: https://cloud.google.com/docs. – [Accessed: Oct. 2023].

31. Chaturvedi, A., Sengar, P. & Sharma, K. "Horizontal dynamic resource scaling by measuring the impacts of scheduling interval in cloud computing". *Proceedings of International Conference on Communication and Artificial Intelligence*. 2021. p. 529–537, https://link.springer.com/chapter/10.1007/978-981-33-6546-9_50. DOI: https://doi.org/10.1007/978-981-33-6546-9_50.

32. Martynyuk, O., Drozd, O., Ahmesh, T., Thuong, B. V., Sachenko, A., Mykhailova, H. & Dombrovskyi, M. "Hierachical model of behavior on-line testing for distributed information systems". *The 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing*. 2019. DOI: https://doi.org/10.1109/IDAACS.2019.8924314.

Conflicts of Interest: The authors declare that they have no conflict of interest regarding this study, including financial, personal, authorship or other, which could influence the research and its results presented in this article

Received. 25.03.2026
Received after revision 26.05.2026
Accepted 10.06.2026

DOI: https://doi.org/10.15276/aait.09.2026.24
УДК 004.75

Оптимізація ефективності операцій із чергами під піковим серверним навантаженням у паралельних архітектурах з порційною автентифікацією

Сурков Сергій Сергійович[1)]
ORCID: https://orcid.org/0000-0001-9224-7526; k1x0r@ukr.net. Scopus Author ID: 57103247200
Мартинюк Олександр Миколайович[1)]
ORCID: https://orcid.org/0000-0003-1461-2000; anmartynyuk@ukr.net. Scopus Author ID: 57103247200
Нестеренко Сергій Анатолійович[1)]
ORCID: https://orcid.org/0000-0002-3757-6594; sa_nbesterenko@ukr.net. Scopus Author Author ID: 55386373800
Тхіонг Буі Ван [2)]
ORCID: https://orcid.org/0000-0002-6176-5432; govarava@gmail.com
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна
[2)] Університет транспорту, вул. Во Оань, 02. Хошимін, 72331, В'єтнам

АНОТАЦІЯ

Актуальність. У сучасних обчислювальних системах черги операцій та архітектури з кількома серверами відіграють центральну роль у керуванні ресурсномісткими, гетерогенними навантаженнями, такими як навчання штучного інтелекту, конвертація аудіо/відео та оптимізація промислового обладнання за допомогою обчислювального моделювання. Традиційні системи з одним процесором демонструють вроджені вузькі місця, включаючи затримки послідовного виконання та неефективне використання ресурсів, тоді як розподілення на кілька серверів дозволяє паралельну обробку, хоча й за підвищених витрат енергії. Попередні роботи ввели метод паралелізації для моделювання потоку пилових частинок, що призвело до скорочення обчислювального часу; однак, зростання витрат на оренду серверів та екологічні імперативи вимагають покращень, орієнтованих на енергію, без втрати швидкості. **Мета і завдання.** Це дослідження модернізує архітектуру Диспетчер-Розгортальник-Виконавець, акцентуючи увагу на оптимізації енергії в реальному часі в середовищах з кількома процесорами. Огляд існуючих досліджень підкреслює проблеми з прогнозними методами, наприклад використанням машинного навчання для прогнозування потреб, та реактивними методами, наприклад масштабуванням на основі використання процесора, особливо для непередбачуваних і різноманітних завдань. Енергозбереження через режими сну для простоюючих серверів залишається недостатньо дослідженим у таких контекстах. Основна мета – мінімізувати середнє споживання енергії протягом операційних періодів, зберігаючи пропускну здатність завдань та обмеження затримки. **Методи.** Дослідження використовує комбінований теоретичний та емпіричний підхід. У вдосконаленій моделі та методі додано підтримку режимів енергозбереження з такими станами: Активний з повною потужністю, Очікування зі

зниженою потужністю, Гібернація з мінімальною потужністю та буферизацією пам'яті на диск, а також Обслуговування з нульовою потужністю для кількох серверів. Загальне споживання енергії моделюється для мінімізації з урахуванням обмежень часу виконання. Переходи станів регулюються порогами простою, з критеріями активації, що включають час виконання завдання, очікування в черзі та затримку переходу. Покращення включають логіку «очікування-звільнення-виконавця», де завдання в черзі відкладають активацію, та подвійні режими: буферизація на диску для стійкості до збоїв та асинхронного виконання проти прямої подачі до доступних виконавців. Покращення безпеки охоплюють фрагментовану криптографічну автентифікацію повідомлень, захищений транспортний зв'язок та динамічну взаємну автентифікацію з фіксацією сертифікатів і ефемерними сертифікатами. Реалізація використовує асинхронний фреймворк Rust з метриками енергії через апаратні інтерфейси. **Результати.** Для прикладу операцій у черзі числові симуляції обчислювали траєкторії пилових частинок у газових потоках. Диференціальні рівняння руху частинок, які враховують сили в'язкого тертя, інерції та гравітації, інтегрували за допомогою методу Рунге-Кутта-Фельберга. Результати розрахунків використовуються для оптимізації труб та каналів енергетичного обладнання. Числові експерименти на кластері з кількома серверами симулювали гетерогенні навантаження. Базовий режим повної активності порівнювали з оптимізованими станами, демонструючи значне скорочення енергії при збереженні продуктивності. Режим буферизації покращив стійкість до збоїв; режим подачі покращив затримку. **Висновки.** Підсумовуючи, вдосконалена архітектура Диспетчер-Розгортальник-Виконавець просуває стале розподілене обчислення, балансуючи надійність, швидкість та енергоефективність для гетерогенних завдань. Майбутні розширення можуть включати прогнозну аналітику для проактивного масштабування та гібридизацію периферійних і хмарних обчислень.

Ключові слова: багатосерверна архітектура; оптимізація енергії; режими енергозбереження; гетерогенні навантаження; рамка DDE; черги завдань; керування станами серверів

ABOUT THE AUTHORS

Sergii S. Surkov - PhD Student, Department of Computer Intellectual Systems and Networks. Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0000-0001-9224-7526; k1x0r@ukr.net. Scopus Author ID: 57103247200
***Research field*:** Parallel Computing in Digital Infrastructures; Resource Allocation Algorithms; Real-Time Queue Management Systems; Performance Prediction and System Efficiency; Authorization protocols;

Сурков Сергій Сергійович – аспірант кафедри Комп'ютерних інтелектуальних систем та мереж, Національний університет «Одеська політехніка», проспект Шевченка, 1. Одеса, 65044, Україна

Oleksandr M. Martynyuk – PhD (Eng), Associate Professor, Head of Department of Computer Intellectual Systems and Networks. Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0000-0003-2366-1920; anmartynyuk@ukr.net. Scopus Author ID: 57103391900
Research field: Behavioral Testing of Computer Systems; Formal Verification and Recognizing of Digital Systems; Artificial Intelligence

Мартинюк Олександр Миколайович – кандидат технічних наук, доцент, завідувач кафедри Комп'ютерних інтелектуальних систем та мереж. Національний університет «Одеська політехніка», проспект Шевченка, 1. Одеса, 65044, Україна

Sergiy A. Nesterenko – Doctor of Engineering Sciences, Professor, Department of Computer Intellectual Systems and Networks. Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
ORCID: https://orcid.org/0000-0002-3757-6594; sa_nesterenko@ukr.net. Scopus Author Author ID: 55386373800
Research field: Computer Networks; Artificial Intelligence; Microprocessor Systems; Neural Networks

Нестеренко Сергій Анатолієвич – доктор технічних наук, професор кафедри Комп'ютерних інтелектуальних систем та мереж. Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

Bui Van Thuong – Institute of Information Technology, Electrical and Electronics Engineering, University of Transport, Ho Chi Minh City, 02, Vo Oanh St. Ho Chi Minh City, 72331, Vietnam
ORCID: https://orcid.org/0000-0002-9160-5982; govarava@gmail.com
***Research field*:** Testing and Diagnosis of Computer Systems; "Green" Computer Systems and Components; Internet of Things

Тхіонг Буі Ван – Інститут інформаційних технологій, електротехніки та електроніки, Університет транспорту. Хошимін, вул. Во Оань, 02. Хошимін, 72331, В'єтнам