(10) Publication No: **IN202641088096 A1**
(43) Publication Date: 24-07-2026
(22) Date of Filing: 19-07-2026
Journal No: 30/2026

(19) Intellectual Property Office, India

(12) INDIAN PATENT APPLICATION

(54) Title: **System and method for cognitive load-aware distributed task scheduling using hardware-bound processing control and predictive task migration**

(51) International Classification:
G06F 9/48, G06F 9/50, G06N 5/04, G06N 20/00 G06F 9/46

(21) Application No: **202641088096**

(31) Priority Document No: --

(32) Priority Date: --

(86) International Application No: --
Filing Date: --

(87) International Publication No: --

(61) Patent of Addition to Application No: --
Filing Date: --

(62) Divisional to Application No: --
Filing Date: --

(71) Name of Applicant(s):
Vellore Institute Of Technology,

(72) Name of Inventor(s):
1. Senthil Kumar T
2. Shagun Khandelwal
3. Sarthak Singhal
4. Advika Pundir
5. Anshul Panigrahi

(57) Abstract:

A cognitive load-aware distributed task scheduling system is disclosed for optimizing computational resource allocation based on user cognitive engagement. The system passively collects user interaction telemetry including keyboard activity, mouse movements, application switching behavior, and idle duration to generate a Cognitive Load Score without requiring physiological sensors or biometric devices. Computational tasks are assigned disruption scores based on user-interface interference, notification potential, processor utilization, memory consumption, and resource requirements, wherein human-perceptual disruption factors are weighted more heavily than hardware-related factors. A scheduling engine correlates the Cognitive Load Score with task disruption classifications and routes, defers, throttles, or migrates tasks across distributed execution nodes using a predefined routing matrix. The system further issues hardware-level processor frequency control commands in response to cognitive-state changes, predicts impending cognitive threshold breaches for pre-emptive task migration, protects sustained focus periods by restricting disruptive workloads, and adaptively personalizes scheduling policies through feedback-driven learning mechanisms. Fig 1

FORM 2

THE PATENTS ACT, 1970

(39 of 1970)

&

The Patents Rules, 2003

COMPLETE SPECIFICATION

(see section 10)

System and Method for Cognitive Load-Aware Distributed Task Scheduling Using Hardware-Bound Processing Control and Predictive Task Migration

Applicant : Vellore Institute of Technology,
Nationality : Indian
Address : Vellore Campus, Tiruvalam Road, Katpadi, Vellore - 632014,
Tamil Nadu, India

The following specification particularly describes the invention and the manner in which it is to be performed

FIELD OF INVENTION

The present invention generally relates to the field of distributed computing, task scheduling, resource orchestration, and human-computer interaction systems. More specifically, the invention describes a cognitive load-aware distributed task scheduling system configured to optimize computational resource allocation based on user cognitive engagement, task disruption potential, and predictive workload management.

BACKGROUND OF THE INVENTION

Modern operating systems employ scheduling mechanisms that allocate processor resources primarily according to computational parameters such as process priority, thread priority, execution history, fairness metrics, input/output activity, and system utilization. Conventional scheduling frameworks, including widely deployed operating system schedulers, are designed to maximize computational efficiency and fairness among competing processes. However, such schedulers operate without awareness of the cognitive state, attention level, or mental engagement of the user interacting with the computing device.

As a result, resource-intensive background processes such as software compilation, data analytics, database rebuilding, indexing operations, machine learning workloads, backup procedures, and synchronization tasks may compete with foreground applications for processor, memory, storage, and network resources regardless of whether the user is actively engaged in cognitively demanding activities. During periods of intensive user concentration, such competition can degrade application responsiveness, increase input latency, reduce rendering performance, generate delayed system responses, and otherwise interrupt productive workflows.

Research in human-computer interaction and cognitive science has demonstrated that interruptions occurring during periods of sustained concentration can adversely affect user productivity and cognitive performance. Such interruptions may disrupt a user's flow state, which is generally characterized as a period of deep focus and efficient task execution. Once disrupted, substantial time may be required for the user to regain the previous level of concentration and task engagement.

Various attempts have been made to address aspects of this problem. Certain systems utilize physiological monitoring devices such as electroencephalography (EEG) sensors, heart-rate monitors, biometric wearables, eye-tracking systems, or other specialized hardware to estimate user cognitive load. Although such approaches may provide useful measurements under controlled conditions, they generally require dedicated sensing equipment, increase deployment complexity, impose additional costs, and are unsuitable for widespread adoption in conventional computing environments.

Other approaches focus on reducing visual or auditory interruptions through notification management frameworks, focus modes, do-not-disturb mechanisms, or attention management systems. While such solutions may suppress alerts, messages, and notifications, they do not influence the underlying allocation of computational resources. Consequently, resource-intensive background workloads continue to execute according to conventional scheduling policies and may still negatively impact the responsiveness of foreground applications used by the actively engaged user.

A WO patent WO2015167847A1 titled “System and method for providing a user cognitive load service” describes a cognitive load service that passively estimates a user's cognitive state by monitoring interaction parameters such as typing speed, text-entry delays, scrolling behavior, gaze fixation, device movement, orientation changes, and application usage patterns. The system computes a cognitive load score by comparing short-term user behavior against long-term behavioral baselines and provides the score to subscribed applications. Applications utilize the score to adapt user interfaces, delay notifications, modify controls, or request confirmation for actions during periods of high cognitive load.

An US patent US11036552B2 titled “Cognitive scheduler” describes a cognitive scheduler for distributed computing clusters that utilizes machine learning models to predict an optimal resource allocation configuration for submitted jobs. The system analyzes historical execution data, job characteristics, cluster resource availability, and performance objectives to recommend resource allocations that improve metrics such as execution time, queue waiting time, throughput, or overall system efficiency. The scheduler dynamically tunes CPU, memory, storage, and other cluster resources by selecting an optimized configuration within a multidimensional resource allocation space before executing a job.

Certain prior systems have further proposed integrating cognitive-state information with task management functions. However, such systems are generally limited to individual computing devices, lack mechanisms for evaluating the disruptive potential of computational workloads from a human-centric perspective, and do not provide coordinated task routing across multiple computing nodes. Existing solutions also fail to incorporate predictive techniques capable of anticipating cognitive threshold breaches before they occur and proactively adjusting workload distribution to preserve user focus.

Therefore, there exists a need for an improved scheduling and workload orchestration system capable of passively estimating user cognitive load without requiring specialized physiological sensors, determining the potential disruption associated with computational tasks, and dynamically allocating workloads based on both factors. There is a further need for a system capable of distributing workloads across multiple computing resources, issuing hardware-level processor control commands, predicting impending cognitive overload conditions, preserving sustained focus periods, and continuously refining scheduling decisions through adaptive feedback mechanisms.

OBJECTIVES OF THE INVENTION

A primary objective of the present invention is to provide a cognitive load-aware distributed task scheduling system capable of intelligently allocating computational workloads while preserving user concentration, productivity, and workflow continuity.

An objective of the present invention is to infer a user's real-time cognitive load from passive interaction telemetry including keyboard activity, mouse movement patterns, cursor behavior, typing characteristics, and application engagement metrics without requiring physiological sensors, cameras, wearable devices, or biometric monitoring equipment.

A further objective of the present invention is to evaluate and assign a disruption score to computational tasks based on their potential impact on user attention and concentration, thereby enabling human-centric scheduling decisions.

A further aspect of the invention seeks to prioritize human-perceptual disruption factors, including user-interface blocking, notification generation, foreground interference, and workflow

interruption, over hardware resource consumption factors such as processor utilization, memory usage, storage access, and input/output activity.

It is an objective of the invention to enforce a disruption-weighting policy in which human-facing effects are consistently assigned greater significance than hardware-related effects through a non-overridable system invariant.

A significant objective of the present invention is to route, defer, execute, or migrate computational tasks across a distributed cluster of execution nodes based upon a combined assessment of user cognitive state and task disruption classification.

A further aim of the invention is to employ a deterministic scheduling policy matrix that correlates cognitive load levels with disruption classes to generate predictable, transparent, and explainable workload allocation decisions.

It is also an objective of the present invention to generate measurable hardware-level resource management effects by issuing processor control directives to an operating system, firmware layer, or hardware management subsystem in response to detected changes in user cognitive state.

A related objective is to dynamically regulate processor frequency allocation, execution priority, and computational resource distribution between foreground and background workloads in accordance with user cognitive engagement levels.

A further aim of the invention is to predict impending cognitive threshold breaches through analysis of behavioral trends, interaction histories, and cognitive load trajectories, thereby enabling proactive resource management.

Another aspect of the invention is directed toward performing pre-emptive migration of disruptive computational tasks from a local execution environment to one or more remote computing nodes before a predicted cognitive overload condition occurs.

It is a further objective to protect sustained periods of deep concentration by automatically establishing a focus-protection state upon detection of extended cognitive engagement patterns without requiring manual activation by the user.

A complementary objective of the invention is to temporarily restrict, defer, throttle, or relocate disruptive computational activities during the focus-protection state in order to preserve workflow continuity and minimize concentration interruptions.

Yet another objective of the present invention is to personalize disruption scoring and scheduling behavior through a closed-loop feedback mechanism that observes post-scheduling cognitive outcomes, productivity indicators, and user interaction responses.

A further objective is to continuously adapt task-weighting parameters, disruption vectors, migration thresholds, and scheduling policies on a per-user basis to improve long-term scheduling effectiveness and focus preservation.

An overarching objective of the invention is to provide an integrated framework that combines passive cognitive-state estimation, disruption vectorization, predictive workload migration, hardware-level resource control, adaptive learning, and distributed task orchestration within a unified scheduling architecture.

SUMMARY OF THE INVENTION

The following summary is provided to facilitate a clear understanding of the new features in the disclosed embodiment and it is not intended to be a full, detailed description. A detailed description of all the aspects of the disclosed invention can be understood by reviewing the full specification, the drawing and the claims and the abstract, as a whole.

The present invention relates to a cognitive load-aware distributed task scheduling system configured to incorporate human cognitive engagement as a scheduling parameter for computational resource allocation.

The invention provides a framework that continuously monitors passive user interaction signals and utilizes such signals to influence task execution, workload placement, resource allocation, and scheduling decisions within a distributed computing environment.

In one aspect, the invention provides a cognitive load assessment module configured to collect passive interaction telemetry including keyboard activity, mouse movement characteristics, application switching behavior, idle duration, interaction frequency, and user engagement

indicators, wherein the collected data is processed to generate a Cognitive Load Score representative of the user's current cognitive state without requiring physiological sensors, wearable devices, cameras, biometric monitoring equipment, or other specialized hardware.

In another aspect, the invention provides a task disruption evaluation module configured to determine a disruption score for each computational task, wherein the disruption score is derived from disruption factors including user-interface interruption potential, notification generation characteristics, foreground application interference, processor utilization, memory consumption, storage activity, network activity, and input/output requirements, and wherein human-perceptual disruption factors are assigned greater weighting than hardware resource consumption factors to ensure human-centric scheduling behavior.

In a further aspect, the invention provides a multidimensional scheduling framework configured to correlate the Cognitive Load Score with the disruption classification of incoming tasks, wherein a routing matrix is utilized to determine whether a task is executed locally, deferred, throttled, migrated, or routed to a selected execution node within a distributed cluster environment.

In another aspect, the invention provides a hardware resource control mechanism configured to issue processor control commands to an operating system, firmware layer, virtualization platform, or hardware management interface, thereby dynamically modifying processor frequency allocation, execution priority, power management parameters, and computational resource distribution in response to changes in the user's cognitive state.

In a further aspect, the invention provides a predictive analytics engine configured to analyze cognitive load trends, interaction histories, and behavioral patterns for identifying an impending cognitive threshold breach, wherein disruptive workloads are proactively migrated to one or more remote execution nodes before the threshold is reached to minimize workflow interruption.

In yet another aspect, the invention provides a flow-state protection mechanism configured to detect sustained periods of focused user engagement, wherein upon detection of a predetermined focus condition, disruptive computational tasks are automatically restricted, deferred, throttled, or migrated to alternative execution resources without requiring manual user activation.

In a further aspect, the invention provides a feedback-based adaptation engine configured to monitor post-scheduling outcomes, user interaction responses, productivity indicators, and cognitive-state variations, wherein disruption-weighting parameters, migration thresholds, scheduling policies, and cognitive-state models are dynamically adjusted to generate individualized scheduling behavior.

In an additional aspect, the invention integrates cognitive-state estimation, disruption vectorization, predictive workload migration, hardware-level processor control, focus-state preservation, adaptive learning, and distributed task orchestration within a unified scheduling architecture capable of simultaneously optimizing computational efficiency and human productivity.

BRIEF DESCRIPTION OF THE DRAWINGS

The manner in which the present invention is formulated is given a more particular description below, briefly summarized above, may be had by reference to the components, some of which is illustrated in the appended drawing It is to be noted; however, that the appended drawing illustrates only typical embodiments of this invention and are therefore should not be considered limiting of its scope, for the system may admit to other equally effective embodiments.

Throughout the drawings, the same drawing reference numerals will be understood to refer to the same elements and features.

The features and advantages of the present invention will become more apparent from the following detailed description a long with the accompanying figures, which forms a part of this application and in which:

Fig. 1 illustrates the system architecture of the present invention;

Fig. 2 illustrates the CLS Computation Pipeline;

Fig. 3 illustrates the Disruption Score Vector Decomposition;

Fig. 4 illustrates the CPU Governor State Machine;

Fig. 5 illustrates the Flow State Lock Timeline;

Fig. 6 illustrates the Predictive Migration Sequence;

Fig. 7 illustrates the Adaptive Weight Calibration Loop;

Fig. 8 illustrates the Multi-Tenant TeamCLS Aggregation;

REFERENCE NUMERALS

101 – Telemetry Agent

102 – Cognitive Load Service

103 – Task Annotator

104 – Scheduler Core

105 – Cluster Nodes

DETAILED DESCRIPTION OF THE INVENTION

The present invention will now be described in detail with reference to the accompanying drawing. The following description is provided to illustrate the invention and should not be construed as limiting the scope of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to a cognitive load-aware distributed task scheduling system configured to incorporate human cognitive engagement into computational resource allocation decisions. The system, referred to herein as the Cognitive Load-Aware Distributed Task Scheduler (CLADS), comprises a plurality of cooperating software services including a Telemetry Agent 101, a Cognitive Load Service 102, a Task Annotator 103, a Scheduler Core 104, and one or more Cluster Nodes 105.

Working Principle

The Cognitive Load-Aware Distributed Task Scheduler (CLADS) operates as a continuous closed-loop scheduling framework comprising a Telemetry Agent (101), a Cognitive Load Service (102), a Task Annotator (103), a Scheduler Core (104), and a plurality of Cluster Nodes (105). The Telemetry Agent (101) continuously captures passive user interaction signals including keyboard

activity, typing rhythm, typing variance, idle intervals, tab-switching events, and application focus changes using standard operating-system event monitoring mechanisms. The captured interaction data is transmitted to the Cognitive Load Service (102), which normalizes the received data and computes a Cognitive Load Score (CLS) representative of the user's current cognitive engagement level. The Cognitive Load Service (102) further classifies the Cognitive Load Score into predefined cognitive states comprising LOW, MEDIUM, and HIGH, while a hysteresis control mechanism validates state transitions using multiple consecutive measurements to prevent rapid oscillation between states. Upon confirmation of a cognitive-state transition, the Cognitive Load Service (102) generates processor frequency control instructions that modify operating-system-level CPU frequency allocation and resource distribution.

Concurrently, computational tasks are processed by the Task Annotator (103), which evaluates each task according to multiple disruption parameters including user-interface blocking potential, notification generation characteristics, processor utilization requirements, memory consumption, and storage or network input/output activity. Based on the evaluated parameters, the Task Annotator (103) generates a Disruption Score and classifies each task into LOW, MEDIUM, or HIGH disruption categories. The disruption-scoring mechanism prioritizes human-perceptual interruption factors over hardware-resource factors through a predefined weighting policy.

The Scheduler Core (104) receives the cognitive-state classification from the Cognitive Load Service (102) and the disruption classification from the Task Annotator (103) and determines an execution strategy using a multidimensional routing matrix. Depending on the selected matrix outcome, tasks are executed locally, deferred, throttled, or migrated to one or more Cluster Nodes (105). The Scheduler Core (104) further incorporates predictive analytics for forecasting cognitive threshold breaches and initiating pre-emptive workload migration, a flow-state protection mechanism for restricting disruptive workloads during sustained focus periods, and an adaptive calibration mechanism that continuously adjusts disruption-weighting parameters based on observed user responses and scheduling outcomes.

System Architecture

The CLADS architecture comprises five cooperating microservices communicating through application programming interfaces and shared data services.

The Telemetry Agent 101 is configured to execute on a user device and capture interaction telemetry including keyboard events, mouse events, application focus changes, idle intervals, and user activity indicators.

The Cognitive Load Service 102 is configured to receive telemetry data from the Telemetry Agent 101, calculate Cognitive Load Scores, manage hysteresis-based state transitions, generate predictive cognitive trends, issue processor control directives, and maintain focus-state information.

The Task Annotator 103 is configured to receive task descriptors, evaluate disruption characteristics, apply disruption-weighting policies, generate Disruption Scores, and classify tasks according to disruption categories.

The Scheduler Core 104 is configured to retrieve cognitive-state information from the Cognitive Load Service 102, retrieve disruption classifications from the Task Annotator 103, execute routing decisions using the scheduling matrix, initiate predictive task migration, perform adaptive weight calibration, and coordinate distributed scheduling operations.

The Cluster Nodes 105 comprise a plurality of execution resources organized into local, balanced, and remote execution tiers. The Cluster Nodes 105 receive tasks from the Scheduler Core 104 and execute workloads according to the assigned routing decisions.

Accordingly, the invention provides an integrated scheduling architecture that combines passive cognitive-state estimation, disruption-aware task classification, predictive workload migration, hardware-level processor control, adaptive learning, flow-state protection, and distributed workload orchestration within a unified human-centric computing framework.

Telemetry Agent and Cognitive Load Computation

The Telemetry Agent captures seven interaction parameters at five-second intervals: keystroke count, average inter-key interval, typing variance (the spread of timing between keystrokes, which indicates typing rhythm regularity), idle duration since the last interaction, tab-switch count, application focus-change count, and context-switch rate. These are transmitted as structured JSON batches to the CLS Engine. No physiological hardware is used at any stage - the agent reads only standard OS-level input events.

The CLS Engine normalises all features to the range [0, 1] using min-max scaling over a sliding historical window, then computes the Cognitive Load Score as a weighted sum:

$$\text{CLS} = 0.25\,\text{idle}_{\text{norm}} + 0.25\,\text{variance}_{\text{norm}} + 0.20\,\text{tab_switch}_{\text{norm}} + 0.20\,\text{context_switch}_{\text{norm}} + 0.10\,\text{focus_change}_{\text{norm}}$$

Scores below 0.40 are classified as LOW (the user is idle or lightly engaged), scores between 0.40 and 0.70 as MEDIUM (moderate engagement), and scores at or above 0.70 as HIGH (deep concentration). State transitions pass through a hysteresis controller that requires three consecutive windows confirming the new state before committing. This is a direct application of the dead-band control technique used in thermostat design - it prevents the system from switching states because a user paused for a few seconds to think.

The CLS Engine also maintains a rolling buffer of the last ten CLS values and fits a linear regression line through them. This produces a trend slope (whether CLS is rising or falling), an estimated time to threshold breach in seconds, and a probability estimate for reaching HIGH state. These predictive metrics feed into the pre-emptive migration engine

Task Disruption Scoring and the Weight Invariant

Every task entering the scheduling queue is annotated with a Disruption Score (Dk) that measures how much it would interrupt a human user. The score combines five components: UI blocking impact (how much it interferes with the foreground application), notification output (how many visible alerts it generates), CPU intensity, memory footprint, and I/O load. Each component is scored from 0 to 1 and multiplied by its respective weight.

The default weight vector is [0.35, 0.25, 0.20, 0.12, 0.08], assigning UI blocking the highest weight (0.35) and I/O the lowest (0.08). The critical design rule is the weight invariant: the sum of the two human-facing weights (UI blocking + notifications = 0.60) must always exceed the sum of the three hardware weights (CPU + memory + I/O = 0.40). The system checks this condition at startup and refuses to initialise if it is violated.

This means no configuration change can accidentally make the system prioritise hardware metrics over human disruption - the scheduling philosophy is architecturally locked in.

$$D_k = \beta_1 U_{\text{block}} + \beta_2 N_{\text{out}} + \beta_3 CPU + \beta_4 MEM + \beta_5 IO$$

$$\boldsymbol{\beta} = [0.35,\ 0.25,\ 0.20,\ 0.12,\ 0.08]$$

$$\beta_1 + \beta_2 = 0.60 \ > \ \beta_3 + \beta_4 + \beta_5 = 0.40$$

Each annotated task carries a full decomposed disruption vector showing the per-component weighted contribution, plus a human-perceptual weight ratio (0.60 / 0.40 = 1.50), which is always greater than 1.0 by construction. Tasks are then classified into LOW, MEDIUM, or HIGH disruption classes based on their total Dk score.

Two-Dimensional Routing Policy Matrix

The Scheduler Core pairs the user's current CLS state (LOW, MEDIUM, HIGH) with the task's disruption class (LOW, MEDIUM, HIGH) and looks up the result in a 3x3 routing matrix:

	Dk: LOW	**Dk: MED**	**Dk: HIGH**
CLS: LOW	Local	Local	Balanced
CLS: MEDIUM	Local	Balanced	Background
CLS: HIGH	Balanced	Background	Background (Deferred)

When CLS is LOW and the task is not disruptive, it runs locally with no overhead. As either dimension increases, tasks migrate outward toward balanced and background nodes. The most extreme case - HIGH CLS with HIGH disruption sends the task to the furthest background node and marks it as deferred, meaning it will not execute until the user's cognitive load drops or the task's priority deadline forces execution.

This is a joint two-dimensional scheduling decision. One axis captures the user's biological state (how focused they are), and the other captures the task's interruption profile (how disruptive it would be). No prior art identified in our search combines these two axes in a formal routing matrix for distributed cluster scheduling**.**

	Task Dk: LOW	Task Dk: MEDIUM	Task Dk: HIGH
USER CLS: LOW	Local Node	Local Node	Balanced Node
USER CLS: MEDIUM	Local Node	Balanced Node	**Background Node**
USER CLS: HIGH	Balanced Node	**Background Node**	**Background Node (Deferred)**

CPU Frequency Governor Control

On every confirmed CLS state transition, the CPU Frequency Governor Controller issues directives to the operating system's CPU frequency scaling interface. Each directive specifies a governor policy (performance, on demand, conservative, or power save), a minimum and maximum frequency in MHz, the set of CPU cores affected, and whether the directive targets foreground or background processes.

The mapping is straightforward:

1. When CLS is LOW, both foreground and background get the "performance" governor - full speed for everything, since the user is not deeply engaged.
2. When CLS is MEDIUM, foreground switches to "ondemand" (scaling up when needed) and background switches to "conservative" with a 2000 MHz cap.
3. When CLS is HIGH, foreground gets "performance" with a raised floor of 3200 MHz (ensuring maximum responsiveness for the application the user is actively working in), while background drops to "powersave" with an 800 MHz ceiling - physically throttling background processes at the hardware level.

In production, these directives are written directly to the Linux sysfs virtual filesystem at /sys/devices/system/cpu/cpu{N}/cpufreq/scaling_governor. In the prototype, identical directive records are persisted to a MongoDB collection with full timestamps, constituting structured evidence of the intended hardware control mechanism.

Flow State Protection Lock

The Flow State Controller tracks how long each user sustains a LOW cognitive load state. It maintains a counter in Redis (flow_streak:{user_id}) that increments for every consecutive LOW-CLS window. When this counter reaches 180 windows - equivalent to 15 minutes at 5-second sampling- the controller sets a boolean flag: flow_state_locked = true.

While flow lock is active, the scheduling rules change: any task classified as HIGH disruption is unconditionally sent to a remote background node, regardless of whether local hardware is available and idle. The scheduling record includes a flow_state_override flag for audit purposes. The lock breaks immediately when CLS transitions to MEDIUM or HIGH - any sign that the user's deep work period has ended.

This differs from every existing "Focus Mode" feature in a fundamental way: it is not manually activated by the user. The user does not press a button or toggle a setting. The system detects sustained concentration by measuring it, and activates protection automatically. It also operates on computational scheduling - actually deferring CPU-intensive tasks - rather than merely hiding notification popups.

Predictive Pre-emptive Migration

The Predictive Migration Engine uses the trend-slope data from the CLS Engine's linear regression module to act before a cognitive threshold is actually crossed. The trigger conditions are: the user's current state is MEDIUM, the regression model estimates a probability of reaching HIGH that exceeds 0.70, and a HIGH-disruption migratable task is currently running on or queued for the local node.

When all conditions are met, the engine issues a pre-emptive migration command, moving the task to a background node before the user's CLS actually reaches HIGH. This is logged as "preemptive_migrate" to distinguish it from reactive migrations that happen after a threshold breach.

An accuracy tracking coroutine fires 30 seconds after each pre-emptive action. It checks whether CLS did in fact reach HIGH within that window. If it did, the prediction is logged as a true

positive. If it did not, it is a false positive. Precision and recall statistics accumulate in a MongoDB collection, providing empirical evidence of prediction effectiveness.

Adaptive Weight Calibration

The Weight Calibrator personalises the disruption score weights for each user over time. After every scheduling decision, it waits 30 seconds, then observes whether the user's CLS state changed in an unexpected direction. If routing a task locally during MEDIUM CLS consistently results in CLS rising to HIGH, that indicates the disruption weights were too lenient - the system should have treated the task as more disruptive.

The calibrator adjusts weights using an Exponential Moving Average (EMA) update rule with a learning rate of 0.05. When CLS degrades unexpectedly, the human-facing weights (UI blocking, notifications) are increased slightly. When CLS remains stable, they decay slightly. After every update, the weight invariant (human weights > hardware weights) is re-verified. If an update would violate the invariant, the weights are renormalised to preserve it.

Calibrated profiles are stored per-user in MongoDB. Over days and weeks of use, the system gradually learns that one developer is more sensitive to compilation noise than another, and adjusts routing decisions accordingly.

Multi-Tenant TeamCLS Aggregation

In shared infrastructure environments where multiple users share the same cluster, the TeamCLS Aggregator computes a composite cluster-level cognitive index. It collects CLS values from all concurrent users and computes a weighted average, where actively interacting users (currently typing or clicking) receive a weight of 2.0 and idle users receive a weight of 1.0. This prevents idle users from diluting the signal when one or more team members are in deep concentration.

When the composite TeamCLS exceeds the HIGH threshold, the system applies a cluster-wide deferral policy: all HIGH-disruption tasks are pushed to background nodes for every user on the cluster, not just the one with high CLS. This protects shared infrastructure during collective work periods - for example, when an entire development team is simultaneously preparing a release.

EXPERIMENTAL RESULTS

A functional prototype of all system modules was built and tested. The prototype runs as a Docker-based cluster of five microservices (Telemetry Agent, CLS Engine, Task Annotator, Scheduler Core, and three worker nodes) with MongoDB and Redis as persistent state stores and a React-based dashboard for telemetry injection and real-time monitoring.

Experiment 1 - Foreground Latency Under Load

Simulated telemetry sequences representing high-typing-variance, high-tabswitch-rate user behaviour were injected to drive CLS to the HIGH state. At the same time, HIGH-disruption tasks simulating build and deploy operations were submitted to the scheduler. Two modes were compared: BASELINE (tasks scheduled without CLS awareness, using only hardware metrics) and CLADS mode (full cognitive-aware routing).

Under BASELINE, 100% of HIGH-disruption tasks were routed to the local primary node regardless of user state, directly competing with the foreground application for CPU and memory. Under CLADS mode with HIGH CLS, 100% of these tasks were routed to background or remote nodes, achieving complete task isolation from the foreground environment.

Experiment 2 - Predictive Migration Accuracy

Pre-emptive migration events triggered when the predictive engine detected probability_high >= 0.70 during MEDIUM CLS were tracked against actual CLS outcomes 30 seconds later. Over N=32 pre-emptive migration events logged during the test protocol, 28 were followed by CLS actually reaching HIGH within the 30-second verification window (true positives), yielding a precision of 87.5%. Four events were false positives where CLS remained at MEDIUM. No pre-emptive migration events were missed when CLS did breach HIGH (zero false negatives in the observed window), though recall measurement requires a larger longitudinal dataset to confirm.

Experiment 3 - Hysteresis Effectiveness

Typing pause patterns were injected that would cause rapid state oscillation without hysteresis (e.g., a user pausing for 8 seconds between paragraphs, then resuming). With hysteresis enabled,

no false state transitions were committed. Without hysteresis, the system oscillated between MEDIUM and LOW states an average of 4.3 times per minute during the same input sequence.

Experiment 4 - Adaptive Weight Calibration Convergence

A simulated user profile was initialised with the default disruption weight vector [0.35, 0.25, 0.20, 0.12, 0.08]. Fifty scheduling decisions were executed, with telemetry injected such that local routing of HIGH-disruption tasks during MEDIUM CLS consistently produced CLS degradation to HIGH within 30 seconds. The Weight Calibrator's EMA feedback loop was observed across all 50 cycles. After 50 iterations, the human-perceptual weights shifted measurably upward while hardware weights decreased:

Weight Component	Initial Value	After 50 Cycles
beta_1 (UI Block)	0.3500	0.3814
beta_2 (Notif.)	0.2500	0.2693
beta_3 (CPU)	0.2000	0.1842
beta_4 (Memory)	0.1200	0.1012
beta_5 (I/O)	0.0800	0.0639
Human sum	0.6000	0.6507
Hardware sum	0.4000	0.3493
Invariant held?	Yes	Yes

The weight vector converged toward a more conservative profile for this user, increasing beta_1 by 9.0% and beta_2 by 7.7% over the baseline. The invariant (human sum > hardware sum) was maintained after every single update cycle. This confirms that the EMA calibration mechanism produces measurable, directionally correct adaptation without violating the core scheduling constraint.

Experiment 5 - Multi-Tenant TeamCLS Aggregation

Two concurrent simulated users were connected to the same three-node cluster. User A was injected with HIGH-CLS telemetry (active typing, high variance, frequent tab switches) and User B with LOW-CLS telemetry (idle, no keystrokes). User A received an activity weight of 2.0 (actively interacting) and User B received 1.0 (idle).

$$\text{TeamCLS} = \frac{2.0 \cdot 0.82 + 1.0 \cdot 0.18}{2.0 + 1.0} = \frac{1.64 + 0.18}{3.0} = 0.607$$

With a individual-only policy, User B's LOW CLS would have allowed HIGH-disruption tasks to run on the local node, competing for shared cluster resources. Under the TeamCLS policy, the composite score of 0.607 (MEDIUM range) triggered balanced-node routing for HIGH-disruption tasks cluster-wide, protecting User A's foreground responsiveness.

A second test raised User A's CLS to 0.91 while User B remained at 0.15:

$$\text{TeamCLS} = \frac{2.0 \cdot 0.91 + 1.0 \cdot 0.15}{3.0} = 0.657$$

At this level, the system routed all HIGH-disruption tasks to background nodes for both users. When User A was removed from the cluster (simulating logout), TeamCLS dropped to 0.15 and normal local routing resumed within one hysteresis window cycle. This confirms that the activity-weighted aggregation correctly biases toward actively engaged users and that the cluster-wide deferral policy activates and deactivates as expected.

Results Summary

Metric	BASELINE	CLADS
Avg Latency (LOW CLS, HIGH Dk)	119.0 ms	91.0 ms
Avg Latency (MEDIUM CLS, HIGH Dk)	145.0 ms	105.0 ms
Avg Latency (HIGH CLS, HIGH Dk)	120.0 ms	68.6 ms (42.8% reduction)
Task Isolation at HIGH CLS	0%	100%
Predictive Migration Precision (N=32 events)	N/A	87.5%
False State Transitions (without/with hysteresis)	4.3/min	0/min
Weight Calibration Shift (after 50 cycles)	N/A	beta_1: +9.0%

TeamCLS Cluster Deferral (2-user shared cluster)	N/A	Activated correctly

Cognitive Load State	BASELINE Latency	CLADS Latency (Reduced)
LOW CLS State	119 ms	91 ms
MEDIUM CLS State	145 ms	105 ms
HIGH CLS State	120 ms	**68.6 ms (42.8% Improvement)**

The 42.8% latency reduction at HIGH CLS confirms that cognitive-aware routing produces a measurable improvement in foreground application responsiveness during periods of deep user engagement. The 87.5% predictive precision across 32 pre-emptive migration events validates the regression-based forecasting approach. The complete elimination of false state transitions by the hysteresis controller demonstrates effective noise filtering in real interaction data. The weight calibration convergence over 50 scheduling cycles confirms that the EMA feedback loop produces directionally correct, constraint-preserving adaptation. The TeamCLS aggregation test confirms that activity-weighted composite scoring correctly protects shared infrastructure when at least one user is deeply engaged.

CLAIMS

We Claim:

1. An adaptive cognitive load-aware distributed task scheduling system, comprising:

a) a Telemetry Agent (101) configured to collect passive user interaction telemetry including keyboard activity, mouse activity, application focus changes, tab-switching events, typing characteristics, and idle-duration information from a user computing device;

b) a Cognitive Load Service (102) communicatively coupled to the Telemetry Agent (101), the Cognitive Load Service (102) being configured to process the collected interaction telemetry and generate a Cognitive Load Score (CLS) representative of a cognitive engagement state of a user, classify the Cognitive Load Score into a plurality of cognitive-state categories, and detect cognitive-state transitions;

c) a Task Annotator (103) configured to receive computational tasks and generate a Disruption Score for each computational task based on a plurality of disruption parameters comprising user-interface interruption potential, notification generation characteristics, processor utilization requirements, memory consumption requirements, and input/output activity requirements;

d) a Scheduler Core (104) configured to determine a task execution action by correlating the cognitive-state category generated by the Cognitive Load Service (102) with a disruption classification generated by the Task Annotator (103) using a multidimensional routing matrix;

e) three Cluster Nodes (105) configured to execute, defer, throttle, or receive migrated computational tasks according to the task execution action determined by the Scheduler Core (104);

f) wherein the Task Annotator (103) assigns a greater weighting to human-perceptual disruption parameters comprising user-interface interruption potential and notification generation characteristics than to hardware-resource disruption parameters comprising processor utilization, memory consumption, and input/output activity, and

g) wherein the Cognitive Load Service (102) is further configured to issue processor frequency control instructions responsive to detected cognitive-state transitions, thereby modifying hardware-level processor resource allocation based on the cognitive engagement state of the user.

2. The system as claimed in claim 1, wherein the Cognitive Load Service (102) classifies the Cognitive Load Score into LOW, MEDIUM, and HIGH cognitive-state categories.

3. The system as claimed in claim 1, wherein the Cognitive Load Service (102) incorporates a hysteresis control mechanism configured to validate a cognitive-state transition only after a predetermined number of consecutive measurements indicate the same target cognitive state.

4. The system as claimed in claim 1, wherein the processor frequency control instructions comprise commands configured to modify processor clock frequency, processor governor settings, execution priority, power-management parameters, or combinations thereof.

5. The system as claimed in claim 1, wherein the Task Annotator (103) is configured to prevent operation when weighting values assigned to the human-perceptual disruption parameters are less than or equal to weighting values assigned to the hardware-resource disruption parameters.

6. The system as claimed in claim 1, wherein the multidimensional routing matrix comprises a plurality of routing policies corresponding to combinations of cognitive-state categories and disruption classifications, each routing policy defining whether a task is executed locally, deferred, throttled, or migrated to a selected Cluster Node (105).

7. The system as claimed in claim 1, wherein the Scheduler Core (104) further comprises a predictive analytics engine configured to analyze historical Cognitive Load Scores and behavioral trends to predict an impending cognitive threshold breach and initiate pre-emptive migration of one or more computational tasks before occurrence of the predicted threshold breach.

8. The system as claimed in claim 1, wherein the Scheduler Core (104) further comprises a flow-state protection mechanism configured to detect a sustained focus condition and automatically restrict, defer, throttle, or migrate disruptive computational tasks during the sustained focus condition.

9. The system as claimed in claim 1, wherein the Scheduler Core (104) further comprises an adaptive calibration engine configured to monitor post-scheduling cognitive outcomes and dynamically adjust disruption-weighting parameters, migration thresholds, and scheduling policies for an individual user.

10. The system as claimed in claim 1, wherein the one or more Cluster Nodes (105) comprise a local execution tier, an intermediate execution tier, and a remote background execution tier configured to receive computational tasks according to the task execution action generated by the Scheduler Core (104).

Name and Signature of the Patent Agent
For Applicant



Ms.Divya Sridharan
Patent Agent(IN/PA-3292)
For Daniel & Gladys.

System and Method for Cognitive Load-Aware Distributed Task Scheduling Using Hardware-Bound Processing Control and Predictive Task Migration

ABSTRACT

A cognitive load-aware distributed task scheduling system is disclosed for optimizing computational resource allocation based on user cognitive engagement. The system passively collects user interaction telemetry including keyboard activity, mouse movements, application switching behavior, and idle duration to generate a Cognitive Load Score without requiring physiological sensors or biometric devices. Computational tasks are assigned disruption scores based on user-interface interference, notification potential, processor utilization, memory consumption, and resource requirements, wherein human-perceptual disruption factors are weighted more heavily than hardware-related factors. A scheduling engine correlates the Cognitive Load Score with task disruption classifications and routes, defers, throttles, or migrates tasks across distributed execution nodes using a predefined routing matrix. The system further issues hardware-level processor frequency control commands in response to cognitive-state changes, predicts impending cognitive threshold breaches for pre-emptive task migration, protects sustained focus periods by restricting disruptive workloads, and adaptively personalizes scheduling policies through feedback-driven learning mechanisms. Fig 1

Date: 19.07.2026

Name and Signature of the Patent Agent
For Applicant



Ms.Divya Sridharan
Patent Agent(IN/PA-3292)
For Daniel & Gladys.

Digitally Signed By:
Divya Sridharan
Date: 19-07-2026 19:35:46