ISSN 2455 – 9091

IJCSET JANUARY Volume 1 Issue 1

International Journal of Computer Science Engineering and Technology (IJCSET)

ENERGY-EFFICIENT TASK-PULL SCHEDULING FOR DISTRIBUTED COMPUTING USING EQUAL-LENGTH CELLULAR AUTOMATA (ELCA)

Shamsudeen E,
Assistant Professor,
Department of computer applications,
EMEA College of Arts and Science Kondotty.

Abstract : - This paper presents an energy-efficient task-pull scheduling methodology for distributed computing using Equal-Length Cellular Automata (ELCA). The approach leverages a CA-based design to map tasks to processors, ensuring load balancing and efficient task distribution for both sequential and parallel execution. The methodology includes algorithms for CA size computation, task-pull generation, and scheduling, while also introducing a path-based heuristic task scheduling (PHTS) algorithm to optimize task execution across heterogeneous processors. The energy consumption model considers active and idle processor states, minimizing energy usage. Experimental results demonstrate the effectiveness of the proposed approach in achieving load-balanced scheduling with reduced energy consumption in distributed environments.

Keywords: [Energy-efficient scheduling, task-pull, distributed computing, Equal-Length Cellular Automata (ELCA), load balancing.]

1. Introduction

Distributed computing refers to a system where computational tasks are distributed across multiple interconnected nodes or machines to achieve higher performance, scalability, and fault tolerance. It is evolving alongside advancements in networking, parallel processing, and system architecture. By dividing workloads across multiple systems, distributed computing ensures efficient utilization of resources, enabling organizations to process large datasets and execute complex computations seamlessly.

1.1 The Need for Load Balancing in Distributed Systems

Load balancing is a critical aspect of distributed computing, addressing the challenge of uneven workload distribution across nodes in a network. Without proper load balancing, certain nodes may become overloaded while others remain underutilized, leading to inefficiencies, degraded performance, and potential system failures. Load balancing mechanisms aim to distribute tasks evenly, maximizing resource utilization and minimizing response times, which are particularly vital for applications like e-commerce, cloud computing, and high-performance scientific computations.

1.2 Load Balancing Mechanisms

Load balancing techniques in distributed computing predominantly focused on the following areas:

1.2.1 Static Load Balancing:

Static load balancing methods allocate tasks to nodes based on predefined rules or prior knowledge of the system's capabilities. These methods, including Round-Robin and Least Connections, rely on simplistic algorithms to distribute tasks evenly. While effective for predictable workloads, static methods struggle to adapt to dynamic changes in system load or resource availability.

1.2.2 Dynamic Load Balancing:

Dynamic load balancing techniques emerged to address the limitations of static methods. These techniques monitor system performance and workload in real time, redistributing tasks dynamically as conditions change. Algorithms like Weighted Round-Robin, Randomized Allocation, and Threshold-Based Methods gained popularity for their ability to adapt to varying demands.

1.2.3 Agent-Based Load Balancing:

Agent-based systems introduced autonomy and intelligence to load balancing. Each node is equipped with an agent that monitors its status and communicates with other agents to negotiate workload distribution. Early examples include Ant Colony Optimization (ACO) and Genetic Algorithms (GA), which mimic natural processes to achieve efficient load distribution.

1.3 Technological Advances Driving Load Balancing

Several technological advancements significantly impacted load balancing strategies:

1.3.1 Cluster and Grid Computing:

These technologies allowed distributed systems to scale horizontally by adding nodes, necessitating advanced load balancing mechanisms to manage resource allocation effectively.

1.3.2 Cloud Computing:

The rise of cloud platforms like Amazon Web Services (AWS) and Microsoft Azure introduced elasticity, where resources could be scaled dynamically. Load balancing became essential for maintaining performance during peak demand periods.

1.3.3 Virtualization:

Virtual machines (VMs) and containerization technologies like Docker enabled resource isolation and flexibility. Load balancers began incorporating virtualization-aware strategies to manage distributed workloads.

1.3.4 Middleware Solutions:

Middleware frameworks, such as Apache ZooKeeper and Hadoop's YARN, provided built-in load balancing capabilities, simplifying resource management in large-scale distributed environments.

1.4 Challenges in Load Balancing

Despite advancements, early load balancing systems faced challenges, including:

1.4.1 Scalability Issues:

Many algorithms struggled to scale efficiently as the number of nodes and tasks increased.

1.4.2 Latency and Overhead:

Monitoring and redistributing workloads in real-time introduced communication delays and processing overheads.

1.4.3 Heterogeneous Environments:

Distributed systems often comprised diverse hardware and software, complicating the development of uniform load balancing strategies.

The evolution of distributed computing and load balancing laid the groundwork for modern innovations in distributed systems. While traditional methods provided essential solutions for balancing workloads, they revealed limitations that spurred research into more adaptive, intelligent, and scalable approaches. As distributed systems continued to grow in complexity, load balancing became a cornerstone of achieving high-performance and reliable computing environments.

2. Literature Survey

1. R. Lu, L. Liu and Y. Chen (2011) et.al proposed A Distributed Artificial Immune Network for Optimizing Tracer Kinetic Models with MATLAB Distributed Computing Engine. Artificial immune networks (AIN), a novel intelligent soft computing method, have been extensively applied across various domains due to their strong global optimization capabilities, particularly in pharmacokinetic parameter optimization. AIN leverages clone selection and immune network principles but is computationally intensive compared to gradient-based methods. To enhance efficiency, a distributed AIN with a distributed clone selection evolutionary strategy was implemented using MATLAB Distributed Computing Engine (MDCE). Experiments demonstrated the algorithm's effectiveness in optimizing [18F] FDG tracer kinetic model parameters efficiently.

2. M. Hasan (2015) et.al proposed A framework for priority based task execution in the distributed computing environment. Distributed computing environments have gained significant attention over the past decade and a half, with geographically distributed resources allocated based on user task requirements. Key considerations for resource provisioning include task performance, fault tolerance, reliability, and timeliness. Various techniques have been proposed to enhance these parameters, particularly in ensuring reliable and timely task execution. The Cooperative Computing System (CCS) framework, originally used in grid computing for high-priority task reliability, is adaptable to other distributed environments. This paper explores extending CCS to support priority-based task execution.

3. J. Cao (2012) et.al proposed Enabling Distributed Computing Systems with ElopTM. In the Internet era, the goal is seamless access to computing applications and services anytime, anywhere. To address this, elop™ computing is proposed, integrating various computing elements as services using advanced technologies like virtualization and SaaS. It provides higher-level services such as metadata management, resource management, scheduling, security, and authorization, enabling scalable distributed computing systems. This paper details the architecture of elop™ computing and its middleware implementation, demonstrating its potential to support distributed computing systems in real-world scenarios.

4. F. Z. Benchara (2016) et.al proposed A new efficient distributed computing middleware based on cloud micro-services for HPC. This paper introduces a new distributed computing middleware designed for High Performance Computing (HPC) based on cloud micro-services. The key challenge addressed is maintaining scalability and efficiency in massively parallel systems as big data processing demands grow. The middleware features a cooperative micro-service team model, where Microservice Virtual Processing Units (MsVPUs) are integrated with load balancing and an AMQP communication protocol to support HPC. The paper also presents the proposed computational scheme and middleware, along with experimental results.

5. A. Mitra (2014) et.al proposed Energy Efficient Task-Pull Scheduling Using Equal Length Cellular Automata in Distributed Computing. This research presents an energy-efficient task-pull scheduling algorithm based on Cellular Automata (CA), specifically using Equal Length Cellular

Automata (ELCA). The proposed method ensures equal task-pull distribution, maximizes CPU utilization, and reduces energy consumption. Experimental results validate the approach's effectiveness in achieving balanced load distribution and minimizing energy use.

3. Proposed Methodology

The provided description outlines a detailed approach for energy-efficient task-pull scheduling in distributed computing environments using an n-cell cellular automata (CA)-based Equal-Length Cellular Automata (ELCA) design. This design facilitates load-balanced scheduling, offering a solution that can accommodate both sequential and parallel task execution. Here's a breakdown of the key points:

Overview of ELCA-based Task Scheduling

1. Task and Processor Mapping:

The system involves mapping `P` independent task modules to `Q` processors using `M` equal-length cycles of size `N`, generated by ELCA.

The number of cycles and the task distribution are designed to ensure load balancing across processors.

2. Mathematical Representation:

The methodology uses equations to represent the number of tasks, processors, and the length of the cycles for task distribution (e.g., Equation 4 and Equation 5 for task and processor counts).

ELCA decomposition yields `M` cycles of length `N`, where `M = 2^m` and `N = 2^(n-m)` for an `n`-cell CA.

3. Task-Pull Scheduling:

Task modules are grouped into "task-pulls" based on the ELCA design and randomly assigned to processors. This ensures an equal distribution of tasks, resulting in load-balanced scheduling.

Task-pull mapping is performed by matching cycles with processors according to certain rules (described in Equations 7 and 8).

Example of Task Scheduling

Scenario 1 (Two Processors, 16 Tasks):

Tasks are grouped into two task-pulls of length 8 each, which are then allocated to two processing units.

Scenario 2 (Eight Processors, 16 Tasks):

Tasks are evenly distributed across eight processors, ensuring parallel task execution.

Energy Consumption Model

Energy Consumption:

The energy consumption model considers processors in active (`Eactv`) or idle (`Eidle`) states.

The total energy consumption is calculated using the number of active and idle processors, with each task consuming energy during execution.

The energy equation is as follows:

$$E_{total} = (M \times N \times E_{actv}) + ((Q - M) \times E_{idle})$$

In an ideal scenario, where all processors are in use, the energy consumption simplifies to:

$$E_{total} = M \times N \times E_{actv}$$

Algorithms for Task Scheduling

1. CA Size Computation (Algorithm 1):

This algorithm computes the size of the cellular automaton (CA) based on the number of tasks `P` and processors `Q`. It identifies the appropriate values of `m` and `n` for the CA.

2. Task-Pull Generation using ELCA (Algorithm 2):

Generates task-pulls by decomposing the CA into equal-length cycles and schedules them based on a balanced rule.

3. Task-Pull Scheduling (Algorithm 3):

Tasks are allocated to processors, either sequentially or in parallel, based on available resources.

Path-based Heuristic Task Scheduling (PHTS)

This is an additional scheduling algorithm aimed at optimizing task execution across heterogeneous processors:

Path Prioritizing Phase: Computes the rank of each path in the task dependency graph (DAG) by considering computation and communication costs.

Task Selection Phase: Selects unscheduled tasks from prioritized paths.

Processor Selection Phase: Tasks are assigned to processors based on minimizing their finish execution time (EFT) using an insertion-based scheduling policy.

This approach efficiently allocates tasks to processors using ELCA-based scheduling, ensuring load balancing and energy efficiency. The scheduling methodology can be applied in distributed computing environments for both sequential and parallel processing, optimizing energy consumption and task distribution.

4. Experiment Results

4.1 Packet delivery Ratio

Number Of Nodes	DAIN-PO	PCCS	Proposed ELC Aalgorithm
100	0.32	0.25	0.47
150	0.35	0.41	0.58
200	0.45	0.36	0.72
250	0.67	0.48	0.79
300	0.71	0.65	0.92

Table 1.Comparison table for Packet Delivery Ratio

The Comparison table 1 of Packet Delivery Ratio Values explains the different values of existing DAIN-PO, PCCS and proposed ELCA algorithm. While comparing the Existing algorithm and proposed ELCA algorithm, provides the better results. The existing algorithm values start from

0.32 to 0.71, 0.25 to 0.65 and proposed ELCA algorithm values starts from 0.45 to 0.92. The proposed method provides the great results.

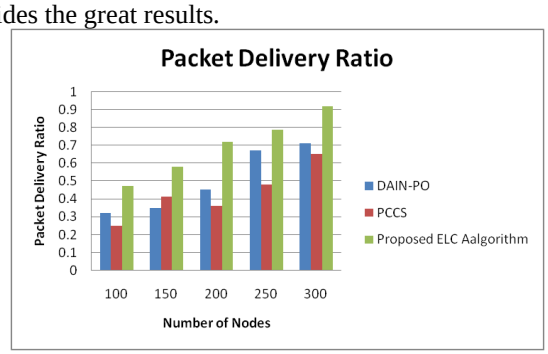

Figure 1.Comparison Chart for Packet Delivery Ratio

The Figure 1 Shows the comparison chart of Packet Delivery Ratio demonstrates the existing DAIN-PO, PCCS and proposed ELCA algorithm. X axis denote the Number of Nodes and y axis denotes the Packet Delivery ratio. The proposed ELCA algorithm values are better than the existing algorithm. The existing algorithm values start from 0.32 to 0.71, 0.25 to 0.65 and proposed ELCA algorithm values starts from 0.45 to 0.92. The proposed method provides the great results.

4.2 Ends-To-End Delay

Number Of Nodes	DAIN-PO	PCCS	Proposed ELCA algorithm
100	0.5	0.4	0.2
150	0.7	0.5	0.3
200	0.9	0.8	0.4
250	1.2	0.9	0.6
300	1.5	1.4	0.5

Table 2.Comparison table for End-To-End Delay

The Comparison table 2 of End-To-End Delay Values explains the different values of existing DAIN-PO, PCCS and proposed ELCA algorithm. While comparing the Existing algorithm and proposed ELCA algorithm, provides the better results. The existing algorithm values start from 0.5 to 1.5, 0.4 to 1.4 and proposed ELCA algorithm values starts from 0.2 to 0.6. The proposed method provides the great results.

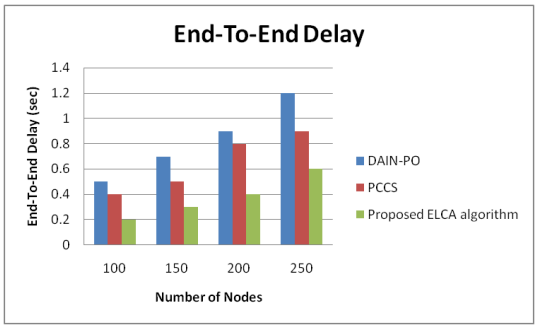

Figure 2.Comparison Chart for End-To-End Delay

The Figure 2 Shows the comparison chart of End-To-End Delay demonstrates the existing DAIN-PO, PCCS and proposed ELCA algorithm. X axis denote the Number of Nodes and y axis denotes the End-To-End Delay. The proposed ELCA algorithm values are better than the existing algorithm. The existing algorithm values start from 0.5 to 1.5, 0.4 to 1.4 and proposed ELCA algorithm values starts from 0.2 to 0.6. The proposed method provides the great results.

4.3 Energy Consumption

Number Of Nodes	DAIN-PO	PCCS	Proposed ELCA algorithm
100	900	800	600
150	1200	1500	700
200	1700	1900	900
250	2100	2300	1100
300	2500	2700	1200

Table 3.Comparison table for Energy Consumption

The Comparison table 3 of Energy Consumption Values explains the different values of existing DAIN-PO, PCCS and proposed ELCA algorithm. While comparing the Existing algorithm and proposed ELCA algorithm, provides the better results. The existing algorithm values start from 900 to 2500, 800 to 2700 and proposed ELCA algorithm values starts from 600 to 1200. The proposed method provides the great results.

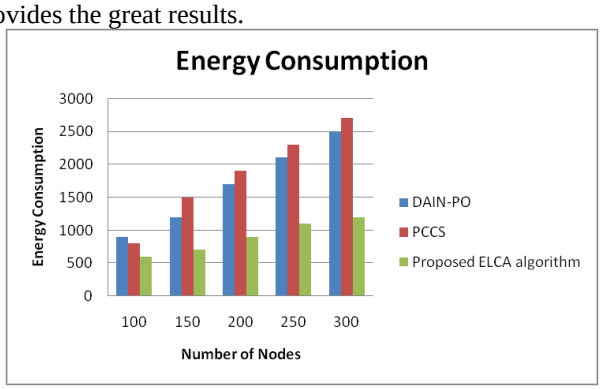

Figure 3.Comparison Chart for Energy Consumption

The Figure 3 Shows the comparison chart of Energy Consumption demonstrates the existing DAIN-PO, PCCS and proposed ELCA algorithm. X axis denote the Number of Nodes and y axis denotes the Energy Consumption. The proposed ELCA algorithm values are better than the existing algorithm. The existing algorithm values start from 900 to 2500, 800 to 2700 and proposed ELCA algorithm values starts from 600 to 1200. The proposed method provides the great results.

CONCLUSION

In conclusion, the proposed ELCA-based task-pull scheduling methodology provides an effective solution for energy-efficient task allocation in distributed computing environments. By utilizing equal-length cycles and a

cooperative approach to task distribution, the system ensures balanced load across processors, supporting both sequential and parallel task execution. The energy consumption model further enhances efficiency, minimizing energy usage by optimizing active and idle processor states. Additionally, the Path-based Heuristic Task Scheduling (PHTS) algorithm improves task execution in heterogeneous systems. Overall, the approach offers a scalable, efficient, and energy-conscious scheduling solution suitable for large-scale distributed computing tasks.

REFERENCES

[1]. R. Lu, L. Liu and Y. Chen, "A Distributed Artificial Immune Network for Optimizing Tracer Kinetic Models with MATLAB Distributed Computing Engine," 10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, Wuxi, China, pp. 41-45, doi: 10.1109/DCABES.2.

[2]. M. Hasan and M. S. Goraya, "A framework for priority based task execution in the distributed computing environment," International Conference on Signal Processing, Computing and Control (ISPCC), Waknaghat, India, , pp. 155-158, doi: 10.1109/ISPCC.7375016.

[3]. J. Cao, S. Chen, Y. Wan and W. Chen, "Enabling Distributed Computing Systems with ElopTM," Third International Conference on Networking and Distributed Computing, Hangzhou, China, pp. 49-53, doi: 10.1109/ICNDC.20.

[4]. F. Z. Benchara, M. Youssfi, O. Bouattane and H. Ouajji, "A new efficient distributed computing middleware based on cloud micro-services for HPC," 5th International Conference on Multimedia Computing and Systems (ICMCS), Marrakech, Morocco, pp. 354-359, doi: 10.1109/ICMCS.7905644.

[5]. A. Mitra, A. Kundu and M. Chattopadhyay, "Energy Efficient Task-Pull Scheduling Using Equal Length Cellular Automata in Distributed Computing," Fourth International Conference of Emerging Applications of Information Technology, Kolkata, India, pp. 40-45, doi: 10.1109/EAIT.20.

[6]. A. Sinha, T. Saini and S. V. Srikanth, "Distributed computing approach to optimize road traffic simulation," International Conference on Parallel, Distributed and Grid Computing, Solan, India, pp. 360-364, doi: 10.1109/PDGC.7030771.

[7]. Z. Yang, C. Zhang, M. Hu and F. Lin, "OPC: A Distributed Computing and Memory Computing-Based Effective Solution of Big Data," IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, China, pp. 50-53, doi: 10.1109/SmartCity.46.

[8]. O. H. Ibarra, "Advances in parallel and distributed computing models - APDCM," IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, pp. 1-1, doi: 10.1109/IPDPSW.5470826.

[9]. P. D. Kaur and I. Chana, "Unfolding the Distributed Computing Paradigms," International Conference on Advances in Computer Engineering, Bangalore, India, pp. 339-342, doi: 10.1109/ACE.80.

[10]. Zhidong Shen and Xiaoping Wu, "The protection for private keys in distributed computing system enabled by trusted computing platform," International Conference On Computer Design and Applications, Qinhuangdao, pp. V5-576-V5-580, doi: 10.1109/ICCDA.5541169.

[11]. T. Ramji, B. Ramkumar and M. S. Manikandan, "Resource and subcarriers allocation for OFDMA based wireless distributed computing system," IEEE International Advance Computing Conference (IACC), Gurgaon, India, pp. 338-342, doi: 10.1109/IAdCC.6779345.

[12]. S. K. Dhurandher, A. Aggarwal, A. Bhandari, A. Verma, M. S. Obaidat and I. Woungang, "Time Stamp-Based Algorithm for Task Scheduling in a Distributed Computing System with Multiple Master Multiple Slave Architecture," International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, Dalian, China, pp. 67-73, doi: 10.1109/iThings/CPSCom.105.

[13]. X. Chen, S. M. Hasan, T. Bose and J. H. Reed, "Crosslayer resource allocation for wireless distributed computing networks," IEEE Radio and Wireless Symposium (RWS), New Orleans, LA, USA, pp. 605-608, doi: 10.1109/RWS.2010.5434191.

[14]. R. Cushing, G. H. H. Putra, S. Koulouzis, A. Belloum, M. Bubak and C. de Laat, "Distributed Computing on an Ensemble of Browsers," in IEEE Internet Computing, vol. 17, no. 5, pp. 54-61, Sept.-Oct. doi: 10.1109/MIC.3.

[15]. Q. -S. Hua et al., "Nearly Optimal Distributed Algorithm for Computing Betweenness Centrality," IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, pp. 271- 280, doi: 10.1109/ICDCS.89.