https://doi.org/10.26682/sjuod.2023.26.2.39

Journal of University of Duhok., Vol. 26, No.2 (Pure and Engineering Sciences), Pp 416 - 426, 2023

(Special Issue)

SINGLE-THREADING BASED DISTRIBUTED-MULTIPROCESSOR-MACHINES AFFECTING BY DISTRIBUTED-PARALLEL-COMPUTING TECHNOLOGY

DILDAR **M**ASOOD **A**BDULQADER[*], **S**UBHI **R**. **M.** ZEEBAREE[**], **R**IZGAR **R**. **Z**EBARI[***], **S**AGVAN **A**LI **S**ALEH[****], **Z**RYAN NAJAT **R**ASHID[*****]and **M**OHAMMED **A**. **M.S**ADEEQ[******]

[*]IT Dept., Technical College of Informatics Akre, Duhok Polytechnic University, Kurdistan Region-Iraq

[**]Energy Eng. Dept., Technical college of Engineering, Duhok Polytechnic University, Kurdistan Region-Iraq

[***]Computer Science Dept., College of Science, Nawroz University, Duhok, Kurdistan Region-Iraq

[****]Electrical and Computer Engineering Dept., College of Engineering, University of Duhok, Kurdistan Region-Iraq

[*****]Computer Network Dept., Sulaimani Polytechnic University, Sulaimani, Kurdistan Region-Iraq

[******]ITM Dept., Technical College of Administration, Duhok Polytechnic University, Kurdistan Region-Iraq

(Accepted for Publication: November 27, 2023)

ABSTRACT

The objective of this study is to propose a methodology for developing a distributed memory system with multiple computers and multicore processors. This system can be implemented on distributed-shared memory systems, utilizing the principles of client/server architecture. The presented system consists of two primary components: monitoring and managing programs executed on distributed-multi-core architectures with 2, 4, and 8 CPUs in order to accomplish a specific task. In the context of problem-solving, the network has the capacity to support multiple servers along with one client. During the implementation phase, it is imperative to consider three distinct scenarios that encompass the majority of design alternatives. The proposed system has the capability to compute the Total-Task-Time (TTT) on the client side, as well as the timings of all relevant servers, including Started, Elapsed, CPU, Kernel, User, Waiting, and Finish. When designing User Programs (UPs), the following creation scenario is carefully considered: The term "single-process-multi-thread" (SPMT) refers to a computing paradigm where a single process is executed by multiple threads The results unequivocally indicate that an augmentation in processing capacity corresponds to a proportional enhancement in the speed at which problems are solved. This pertains specifically to the quantity of servers and the number of processors allocated to each server. Consequently, the duration required to finish the assignment increased by a factor of 9.156, contingent upon three distinct scenarios involving SPMT UPs. The C# programming language is utilized for the coding process in the implementation of this system.

KEYWORDS: *Distributed Systems, Single-Threading, Multiprocessor-Machines, Parallel-Computing, Process, Thread.*

1. INTRODUCTION

Distributed systems consist of computers and other components that may work independently. Whether a "node" was a physical component or a computer software method, they had to function together. Both required collaboration [1-3]. High-performance distributed computing systems are also essential. Cluster computing's central processing units, or core machines, are a set of workstations and other computers with high-speed LAN

dildar.masood@dpu.edu.krd , subhi.rafeeq@dpu.edu.krd rizgar.khuder@nawroz.edu.krd ,sagvan.saleh@uod.ac zryan.rashid@spu.edu.iq , mohammed.abdulrazaq@dpu.edu.krd

connections. Each node runs the same OS [4, 5]. Due to their ubiquitous use in contemporary computing, client-server architectures have grown in popularity. The client and server are the two main components of a client-server system. This architecture has the client always submitting requests and the server continually responding. Both are continuing. Client-server architecture lets processes exchange data. Design allows this [6], [7].

Desktop computers, portable laptops, and corporate servers now have several cores in their central processor units, improving their processing speed. These systems must be lightweight, power-efficient, and heat-efficient [8, 9]. Also, some of these systems are required to achieve these requirements in order to be certified. Traditional central processing units (CPUs), also known as CPUs, are not completely suited to fulfill these requirements [10, 11]. Multi-core processors have more computing power per ounce, watt, and square inch than traditional CPUs. One CPU is inadequate for many tasks, causing poor performance and delayed response times in many systems [12], [13]. The system's efficiency and fairness depend on how resources are distributed among competing accounts. Resource allocation is competitive. Even if we divide the computational burden well across all CPUs, certain scenarios need a system with several processors. Thus, it is crucial to break the work down into smaller, more manageable portions and organize the order in which they will be done [14, 15]. However, multi-processor computers may execute specialized applications better and provide a more secure environment. Thus, multiprocessor planning is constantly studied. Each work runs on its own processor, similar to single-processor, central multiprocessor, and distributed scheduling. Real-time planning requires load complexity estimations. Multiprocessor systems make planning algorithms harder to create [16], [17]. This is because the system has more processors. Multithreading means a software or process may run many code threads at once. Thus, a single-CPU computer may now do the process in parallel [18], [19]. Multithreading lets users break software operations into independent "threads," expanding multitasking. This enables multitasking. The operating system divides its resources across all programs and each app's threads. Many threads on proprietary hardware may enhance CPU usage, reducing application runtime [20], [21].

Parallel programming processes are evaluated using floating-point operations per second, execution density, and memory latency. Hardware counters capture program-running monitoring data. Performance monitoring collects data on system and application use to help identify bottlenecks. Whether the processor unit has one core or several cores affects thread (process) monitoring systems [22]. To solve performance concerns like data placement, process compatibility, and load distribution, this concept must be grasped. Performance improvement requires data analysis. We provide a solution for thread and process migrations and application tracking. To improve an application that employs shared memory and parallel objectives, this instrument gathers information about all system processes and threads [23]. In 2018, V. Weinberg and colleagues [24] supplied two well-known distributed computing and GPU computing methods to help parallel computing beginners. We will explore the requirements for designing a SIMD-based matrix multiplication algorithm and demonstrate an application in this article. Parallel processing may be enhanced by using data parallelism and parallel control, among other methods. GPUs execute quicker than other computer components due to their specialized hardware and software. In 2019, Y. Xu, et al. [25] faced a new and serious issue: the CPU takes too long to apply computational marine hydrodynamics tokens when private clouds in containers have limited cloud resources. Cloud resources were limited. By matching task resources, the scheduling method improves service provider-end user communication. This saves time. The simulation showed that our technique outperforms the foundation algorithms in task planning time, resource requirements, and performance.

Bianco et al. [26] in 2020, proposed this architecture. This arrangement's ability to speed packet delivery is attracting researchers. This contributes to this growing interest. They suggested adding routes and priority categories to strengthen their plan. The NIC's resource distribution unit may change roles throughout the scheduling algorithm's response phase. This makes a more sophisticated justice policy possible.

Z. Lv, D. Chen, and A. K. Singh [27] in 2021, Lexicographic breadth-first search, centre-division zone distance-based shortest route length approximation technique, area and principal highway axis-vertex, and integrated link and attribute were used. According to analysis, four CPU threads (514.63 ms) calculate the framework best. CPU cores linearly speed up framework calculation. Each pair of processor cores does one task. The arithmetic scale factor should be 0.06 for simple networks and 0.2 for complex ones. This method has the quickest processing, average query, and total query times across datasets at 49.67 milliseconds and 5.12 milliseconds, respectively. 2021 L. M. Haji and his colleagues began developing a parallel processing system with shared memory [28]. This technique provided users complete control over computer processes and threads. Good news: the system works with several multi-core architectures. This effort's algorithms provided server data, verified all running processes, and ran all of a user's application's threads and processes. These steps finish the user's application.

2. METHODOLOGY

The Process/Threads Monitoring and controlling distributed system (PTMCDS) that has been described is made up of two primary components—the client and the servers—and it operates in three distinct scenarios in order to deal with the load that is produced by the clients. The client and the servers are the basic components. Figure 1 reveals that this particular component acts as the controller for the system that is being suggested. Figure 2 depicts the data message, and Figure 3 shows how the system that has been proposed operates. Moreover, Figure 2 depicts the data message. Throughout the course of a communication session between a client and a server, it is not uncommon for both control and data-type messages to be sent back and forth between the two entities.

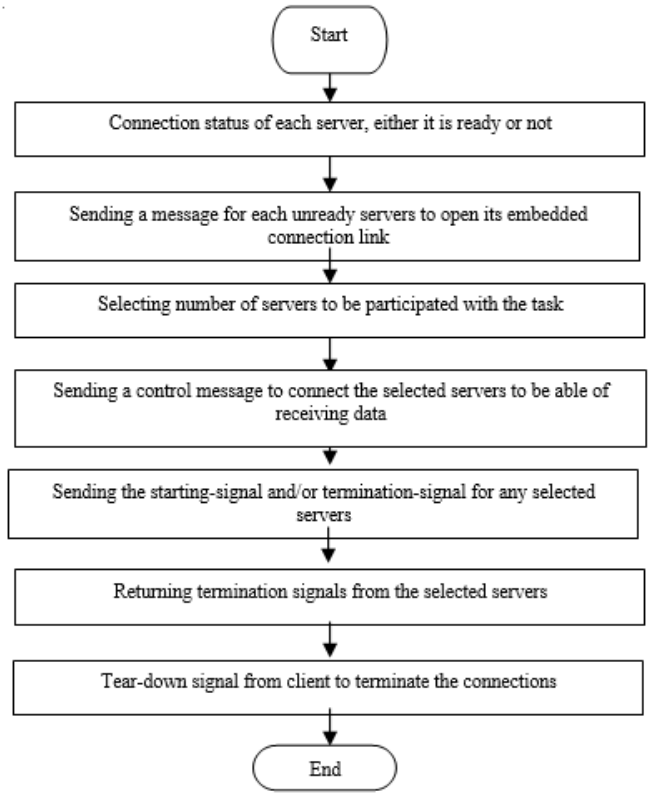

Fig.(1):- Block Diagram of the control message

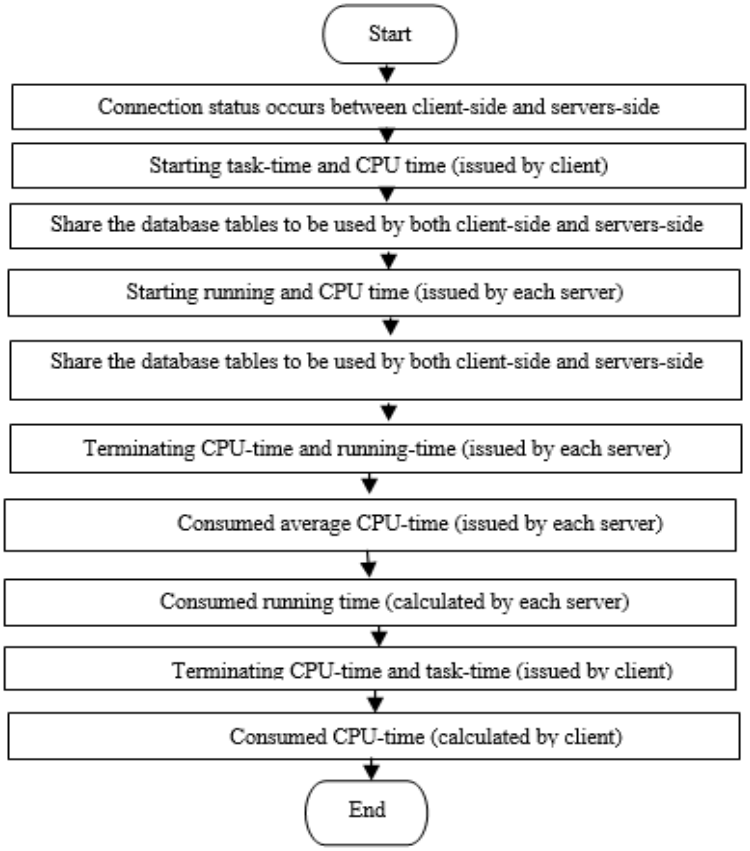

Fig.(2):- Block Diagram of data-messages

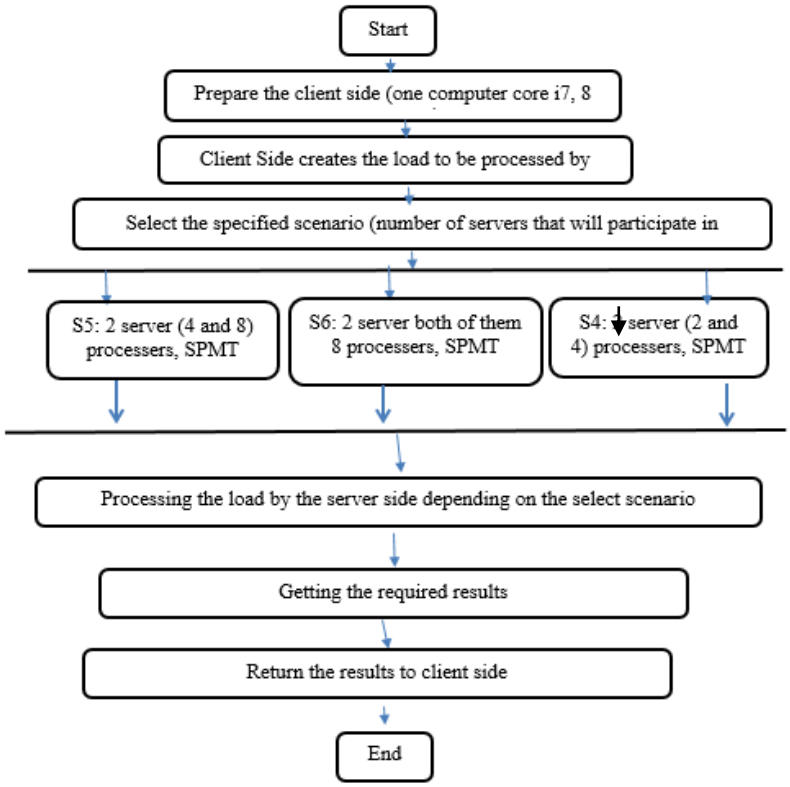

Fig.(3):- General diagram of proposed (PTMCDS) mechanism

2.1 Scenarios of the Proposed System

In order to illustrate the advantages of using parallel processing methods, researchers typically employ one or more scenarios that explain parallel processing algorithms and obtain findings faster than a single processing approach. The procedures for sorting numbers can also be used as an ideal case study application to highlight the effects of Parallel Processing on the amount of time it takes to complete a processing technique and how efficiently it does so. This can be done by comparing the amount of time it takes to complete the technique with the amount of time it takes to complete the procedures for sorting numbers. This investigation makes use of four distinct methods for sorting numbers, all of which are dependent on a large number of server computers. The purpose of this investigation is to demonstrate the benefits of the Parallel Processing strategy in comparison to the more traditional sequential processing method. The Selective sorting method, the Insertion sorting method, the Bubble sorting technique, and the Shakar sorting technique are some examples of the procedures that fall under this category.

To demonstrate how the mixed parallel processing approach affects the growth of the amount of work that needs to be completed, we will sort the data using a variety of different orders. This will allow us to demonstrate how the method expands the amount of work that needs to be completed. Core i7 computers have been used throughout the whole of this investigation, both as clients and hosts (8 processor). As a consequence of this, we are only going to talk about the characteristics of the server hosts themselves, which include the following:

S1: 2 UP, SPMT on two servers, each with a Core 2 Duo and a Core I3 CPU (total of 4 processors).

Core i3 servers (with four processors) and Core i7 servers provide the computing capability for S2's 2 UP SPMT setup (with eight processors).

S3 is running with 2 UP and SPMT on two servers, both of which have Core i7 processors (8 CPUs).

2.2 Monitoring Implementation and Results

The system that was being considered at the time was one that could work on anywhere from one to one hundred UPs, each of which may have either a single process or numerous processes, and each of those processes may have either one thread or many threads. Using the system's MI component is one way to protect all of the genuinely important parts of the system. This may be achieved if the component is configured properly. The results of running three distinct scenarios are analyzed, and the findings are presented in appropriate formats (Tables and Plots), as can be seen in the accompanying Tables (1 to 3) and Figures (4 to 6). In each of these tables and figures, the outcomes of running the scenarios are presented in suitable formats (Tables and Plots) (4 to 6).

2.2.1 Scenario-1: Two servers, core two duo (2 processors) and core i3 (4 processors), used for (SPMT).

The following configurations are available on the server: CPU-Type = 2Duo; RAM = 4 Gigabytes; CPU-CoreCount = 2; CPU-Frequency = 2.2 GHz; CPU-Type = i3-2350; RAM = 4 Gigabytes; CPU-CoreCount = 4; CPU-Frequency = 2.3 GHz; CPU-Type = 2Duo; RAM = 4 Gigabytes; CPU-CoreCount = 4; CPU-Frequency = 2.3 GHz; CPU The CPU-Table provides the monitoring information that was gathered during the whole of the execution of a single UP process. This process makes use of four threads. Figure 4 is a scatterplot that compares the length of time that has elapsed to the amount of time spent waiting by the user, by the kernel, and by the total CPU.

dildar.masood@dpu.edu.krd ,
subhi.rafeeq@dpu.edu.krd
rizgar.khuder@nawroz.edu.krd
,sagvan.saleh@uod.ac
zryan.rashid@spu.edu.iq ,
mohammed.abdulrazaq@dpu.edu.krd

Table (1): -Results of Scenario-1, two servers, core two duo and core i3

	Client	Server1	Server2
IP	192.168.1.1	192.168.1.2	192.168.1.3
N_core	8	2	4
Proccess_Name	startproccess	SPMT11	SPMT12
Start_time	01:21:26:017	01:21:24:781	02:08:01:931
Elapsed_Time (ms)	8356802	8357082	3741903
Kernel_Time (ms)	2218	171	109
User_Time (ms)	7325532	8355978	3741658
Total_CPU Time (ms)	7327750	8356149	3741767
Waiting_Time (ms)	1029052	933	136
End_time	03:40:42:819	03:40:41:863	03:10:23:834

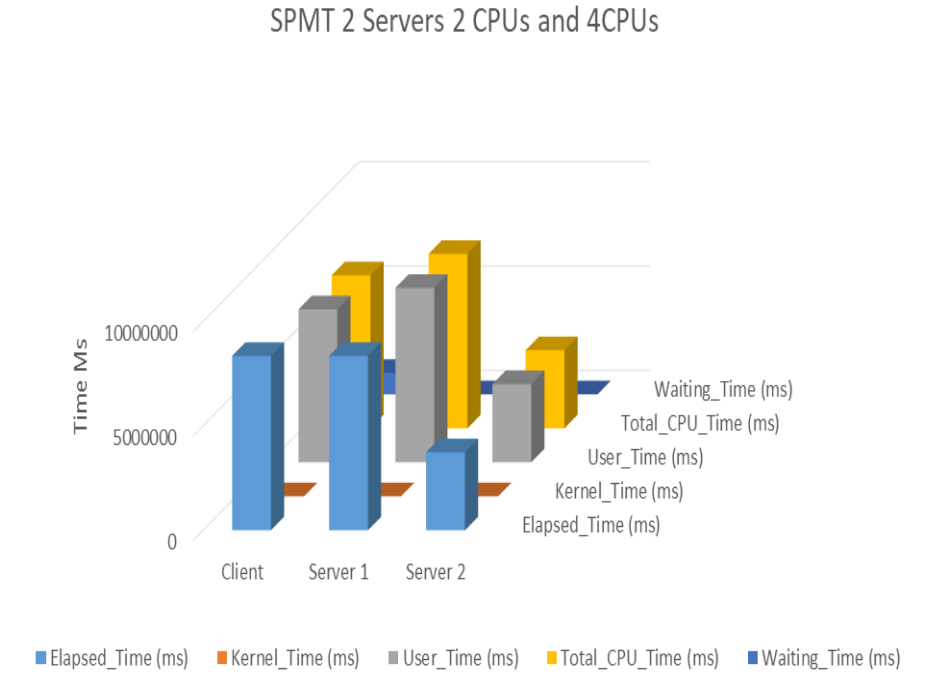

Fig.(4):- Plotted results of scenario-1 (SPMT).

2.2.2 *Scenario-2: Two servers core i3 (4 processors) and Corei7 (8 processors) used for (SPMT)*

The following is an itemized list of the characteristics of the server: In addition to their respective 4 gigabytes of memory, four cores, and four total threads, the (CPU-Type= i3-2350) and the (CPU-Type= i7-6600) both have a processing speed of 2.3 gigahertz (GHz). The research was conducted to determine how long it takes for a single UP process to do its job when employing four threads. Table 2 presents the data that was gained from this study. Figure 5 depicts a scatter plot of the user, kernel, total CPU, and waiting period's vs the amount of time that has already elapsed. This figure is given in relation to the total CPU use.

Table (2): -Results of Scenario-2, two servers core i3 and core i7 (SPMT)

	Client	Server1	Server2
IP	192.168.1.1	192.168.1.3	192.168.1.4
N_core	8	4	8
Proccess_Name	start process	SPMT11	SPMT12
Start_time	12:08:10:697	12:08:01:931	16:44:46:845
Elapsed_Time (ms)	3741997	3741903	2813290
Kernel_Time (ms)	1187	109	265
User_Time (ms)	3738453	3741658	2811225
Total_CPU_Time (ms)	3739640	3741767	2811490
Waiting_Time (ms)	2357	136	1800
End_time	13:10:32:694	13:10:23:834	17:31:40:279

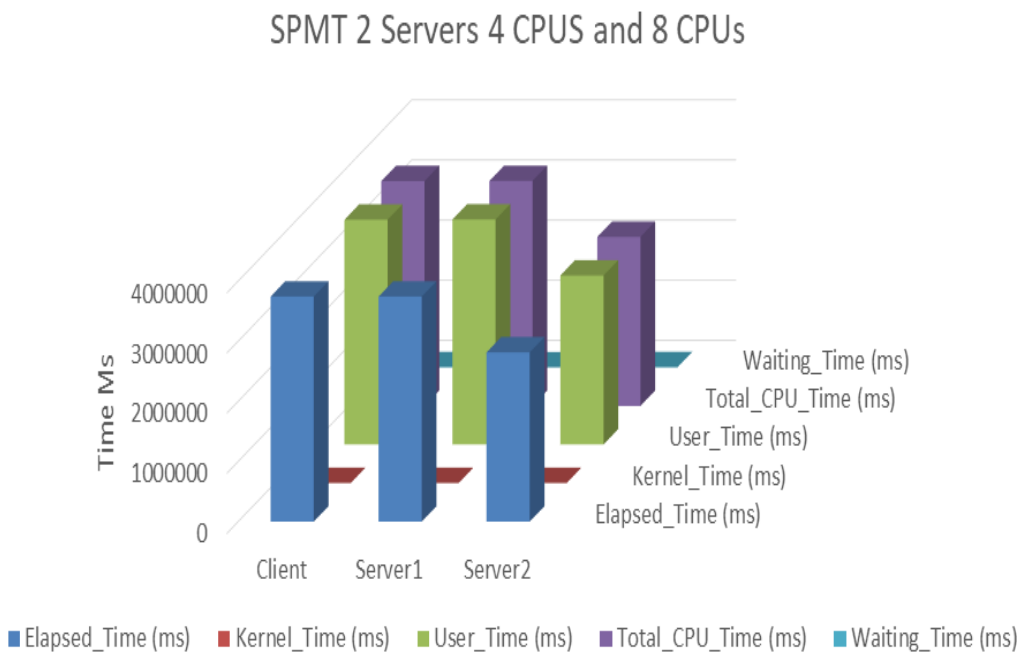

Fig.(5):- Plotted results of scenario-2 (SPMT)

2.2.3 Scenario-3: *Two servers, both of them core i7 (8 processors), used for (SPMT).*

Table (3): -Results of Scenario-3, two servers both of them core i7 (SPMT)

	Client	Server 1	Server 2
IP	192.168.1.1	192.168.1.4	192.168.1.5
N_core	8	8	8
Proccess_Name	start process	SPMT11	SPMT12
Start_time	13:34:56:071	13:44:46:845	11:39:46:845
Elapsed_Time (ms)	2815211	2813290	2793434
Kernel_Time (ms)	256718	265	212
User_Time (ms)	830234	2811225	271103
Total_CPU_Time (ms)	1086953	2811490	2791315
Waiting_Time (ms)	1728258	1800	2119
End_time	14:21:51:277	14:31:40:279	12:26:20:279

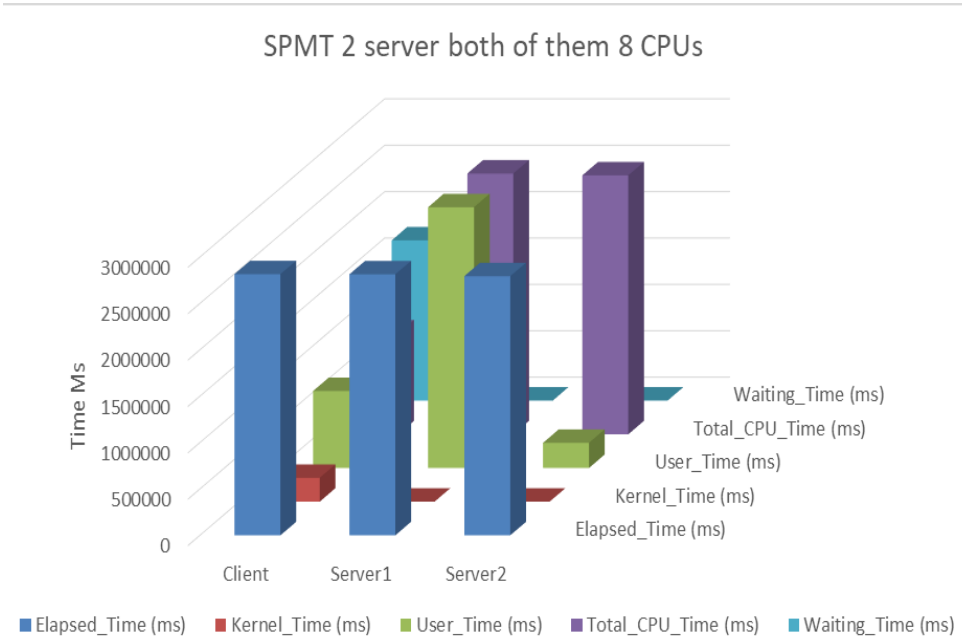

Fig.(6):- Plotted results of scenario-3 (SPMT)

The hardware of the server has the following configurations: (CPU-Type=i7-10700, RAM=8 GB, No. of Core=8, CPU-Frequency=3.8 GHz) and (CPU-Type=i7-6600, RAM=8 GB, No. of Core=8, CPU-Frequency=2.7 GHz) respectively. Both of these configurations are for eight cores. A total of eight cores may be found throughout both configurations. A study was carried out to ascertain the amount of time required for a single UP process to complete its task while making use of four threads. The findings of this research are summarized in Table 3, which can be seen below. The results of a time-series analysis that was performed on waiting times, user times, kernel times, and total CPU times are presented in Figure 6.

3. Discussion of Obtained Results from PTMCDS Scenarios

It is clear from looking at Tables (1), (2), and (3) that the TTT of a single program executing SPMT on a single server grows quicker as the number of cores available on that server increases. The total time to complete Scenario 1 was calculated to be 8,356,802 milliseconds while utilizing a server with two cores.

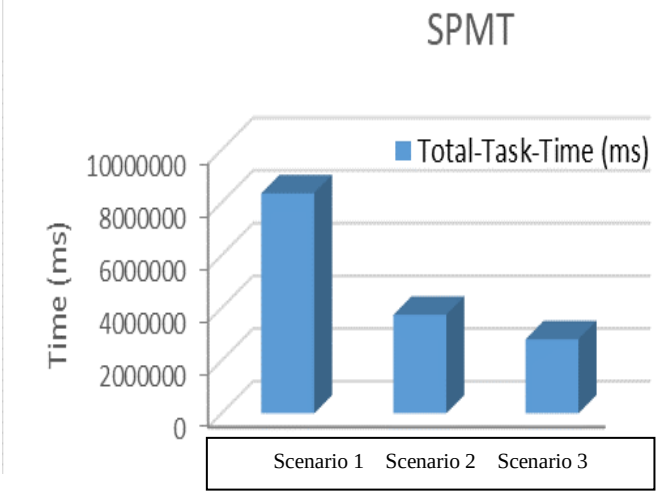

Fig.(7):- The TTT of Scenarios (1, 2, and 3) for SPMT

In the meantime, the whole transaction time in Scenario 2 with a core i3 server took 3,741,997 milliseconds to complete. When Scenario 3 was run on a server equipped with a core i7 CPU, the total transaction time took 2,815,211 milliseconds to complete. The TTT (from Scenario 1 to Scenario 3) was reduced by the same amount, as indicated in Figure 7, despite the fact that the processing speed was raised by 9.156 times.

I. COMPARISON WITH PREVIOUS WORKS

Based on the characteristics that are presented in Table 4, one would be able to deduce the following major differences between the two groups:

Ref.	Technique	Operating System And Cores	Programing Language	Significant Results
[24]	matrix multiplication	Linux, Dual core	Visual basic	The number of process- 50, the execution time in system - 0.158426 seconds, the execution time in the cluster - 1.232802 seconds. The executed program in the system group took less time in single system.
[25]	scheduling algorithm	Windows server2016, Core (4 processor)	java	The result is a 6 times improvement, a 44% volume reduction, and an average defined satisfaction rate, respectively
[26]	scheduling algorithm	Linux, PCI-X core	LOGIC design	It is processed and managed by the operating system in software, reducing load performance.
[27]	N-SPFA algorithm	Linux, PCI-X core	C# language	The processing time, average query time, and total query time of the algorithm are the shortest, being 49.67 ms, 5.12 ms, and 94.720 ms, respectively.
[28]	Sorting Algorithm	Windows 10, core i5, core i7	C# language	Controlling and monitoring Single/Multiple Processes/Threads in multicomputer parallel processing system. Reduce execution time for CPU.
This System	Sorting Algorithm	Windows 10, Core 2 Duo, core i3, core i7	C# language	Controlling and monitoring Single/Multiple Processes/Threads in multicomputer parallel processing distributed system. Reduce the Client Total-Task-Time, server (elapsed and CPU) times.

Table (4): -Comparison with Previous Works.

- The PTMCDS that has been presented has support for four different sorting algorithms; this gives it the ability to manage an almost unimaginably vast spectrum of task loads (and the flexibility to support any number of new sorting algorithms).
- The fact that Microsoft Windows is not an open-source computer operating system was the most difficult obstacle that the planned PTMCDS system had to get through in order to be successful. This problem has been fixed, and the system has been adjusted so that it may work correctly while still adhering to the requirements of the operating system being used. This is in spite of the fact that a significant portion of the tasks described above were accomplished through the use of open-source computer operating systems.
- PTMCDS supported four distinct core types, in contrast to prior implementations, which only supported one or two cores each. These core types were i7, i5, and i3, respectively. Previous implementations only supported one or two cores each. In previous versions, only CPUs with a single core were supported. Moreover, PTMCDS does not set any limitations on either the number of servers or the kind of cores that can work together in order to automatically carry out jobs. Users get access to both of these aspects of the product.
- The capability of the proposed PTMCDS to make full use of the efficacy of shared memory systems on each individual server, in addition to taking advantage of the
- Capacity of distributed systems for performing parallel processing, is an additional important feature that sets it apart from the works that have already been discussed. This ability distinguishes the proposed PTMCDS from the works that have been discussed previously. This is one of the ways in which it stands out from the other works, and it is one of the ways in which it distinguishes itself. The key factor in PTMCDS's improved performance was the use of this novel

dildar.masood@dpu.edu.krd ,
subhi.rafeeq@dpu.edu.krd
rizgar.khuder@nawroz.edu.krd
,sagvan.saleh@uod.ac
zryan.rashid@spu.edu.iq ,
mohammed.abdulrazaq@dpu.edu.krd

method, which had not been used in any of the earlier research. It is essential to keep in mind that determining whether or not parallel processing is effective is necessary due to the widespread use of this strategy for the purpose of reducing the amount of time spent on the resolution of difficult issues. Because of this, it is essential to keep in mind that determining whether or not parallel processing is effective is necessary.

4. CONCLUSIONS

This work proposes using a Process/Threads Monitoring Controlling Distributed System (PTMCDS) to monitor and control UPs. "Process Monitoring Controlling Distributed System" is abbreviated "PTMCDS." In the following paragraphs, the possible inferences or conclusions from this thesis will be broken down into their component parts, then presented in their entirety. This will help readers grasp probable implications and conclusions. Each control mode may affect the CPU and report real-time events. These control modes are versatile. Mixing effects and logs is feasible. Software developers no longer needed to work with several operating system versions and distributions to solve the issue. This resolved it. This technique should work well with all Windows versions. This technology is also expected soon. Second, OpenMP Communication (shared memory) and MPI Communication (distributed memory) should be integrated into the system for effective parallel processing. This enabled efficient parallel processing. This ensured effective parallel processing. This was done to ensure the system could do parallel processing. These measures ensured that the computer system could parallel process without issues. Scenario 1 outperforms Scenario 3 in client-side task processing by 9.156.

Acknowledgments

The authors would like to express full thanks to Information Technology department at Technical College of Informatics Akre, for preparing the specific laboratories to perform the real implementation for this research.

REFERENCES

Z. N. Rashid, S. R. Zeebaree, R. R. Zebari, S. H. Ahmed, H. M. Shukur, and A. Alkhayyat, “Distributed and Parallel Computing System Using Single-Client Multi-Hash Multi-Server Multi-Thread,” in 2021 1st Babylon International Conference on Information Technology and Science (BICITS), 2021, pp. 222–227.

Z. N. Rashid, S. R. Zeebaree, M. A. Sadeeq, R. R. Zebari, H. M. Shukur, and A. Alkhayyat, “Cloud-based Parallel Computing System Via Single-Client Multi-Hash Single-Server Multi-Thread,” in 2021 International Conference on Advance of Sustainable Engineering and its Application (ICASEA), 2021, pp. 59–64.

H. Shukur, S. Zeebaree, R. Zebari, D. Zeebaree, O. Ahmed, and A. Salih, “Cloud Computing Virtualization of Resources Allocation for Distributed Systems,” Journal of Applied Science and Technology Trends, vol. 1, no. 3, pp. 98–105, 2020.

Z. N. Rashid, S. R. Zeebaree, and A. Shengul, “Design and Analysis of Proposed Remote Controlling Distributed Parallel Computing System Over the Cloud,” in 2019 International Conference on Advanced Science and Engineering (ICOASE), 2019, pp. 118–123.

S. R. Zeebaree, K. Jacksi, and R. R. Zebari, “Impact analysis of SYN flood DDoS attack on HAProxy and NLB cluster-based web servers,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 19, no. 1, pp. 510–517, 2020.

O. Alzakholi, L. Haji, H. Shukur, R. Zebari, S. Abas, and M. Sadeeq, “Comparison Among Cloud Technologies and Cloud Performance,” Journal of Applied Science and Technology Trends, vol. 1, no. 2, Art. no. 2, Apr. 2020, doi: 10.38094/jastt1219.

H. S. Oluwatosin, “Client-server model,” IOSR Journal of Computer Engineering, vol. 16, no. 1, pp. 67–71, 2014.

H. Shukur et al., “A State of Art Survey for Concurrent Computation and Clustering of Parallel Computing for Distributed Systems,” Journal of Applied Science and Technology Trends, vol. 1, no. 4, pp. 148–154, 2020.

H. Shukur, S. Zeebaree, R. Zebari, O. Ahmed, L. Haji, and D. Abdulqader, “Cache Coherence Protocols in Distributed Systems,” Journal of Applied Science and Technology Trends, vol. 1, no. 3, pp. 92–97, 2020.

R. Craig and P. N. Leroux, “Case study-making a successful transition to multi-core processors,” QNX Software Systems GmbH & Co, 2006.

Z. N. Rashid, K. H. Sharif, and S. Zeebaree, “Client/Servers Clustering Effects on CPU Execution-Time, CPU Usage and CPU Idle Depending on Activities of Parallel-

Processing-Technique Operations “,” INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH, vol. 7, no. 8, pp. 106–111, 2018.

J. Chang and G. S. Sohi, “Cooperative cache partitioning for chip multiprocessors,” in ACM International Conference on Supercomputing 25th Anniversary Volume, 2007, pp. 402–412.

S. Zeebaree and I. M. Zebari, “Multilevel Client/Server Peer-to-Peer Video Broadcasting System,” International Journal of Scientific & Engineering Research, vol. 5, no. 8, Art. no. 8, 2014.

G. P. Acharya and M. A. Rani, “FPGA Prototyping of Micro-Blaze soft-processor based Multi-core System on Chip,” International Journal of Engineering & Technology, vol. 7, no. 2.16, pp. 57–60, 2018.

A. A. Yazdeen, S. R. Zeebaree, M. M. Sadeeq, S. F. Kak, O. M. Ahmed, and R. R. Zebari, “FPGA implementations for data encryption and decryption via concurrent and parallel computation: A review,”Qubahan Academic Journal, vol. 1, no. 2, pp. 8–16, 2021.

M. Ababneh, S. Hassan, and S. Bani-Ahmad, “On Static Scheduling of Tasks in Real Time Multiprocessor Systems: An Improved GA-Based Approach.” International Arab Journal of Information Technology (IAJIT), vol. 11, no. 6, 2014.

S. R. M. Zeebaree et al., “Multicomputer Multicore System Influence on Maximum Multi-Processes Execution Time,” TEST Engineering & Management, vol. 83, no. May-June 2020, pp. 14921–14931, May 2020.

A. S. Y. Subhi Rafeeq Mohammed Zebari, “Improved Approach for Unbalanced Load-Division Operations Implementation on Hybrid Parallel Processing Systems,” Journal of University of Zakho, vol. 1, no. (A) No.2, Art. no. (A) No.2, 2013.

D. M. Abdulqader and S. R. Zeebaree, “Impact of Distributed-Memory Parallel Processing Approach on Performance Enhancing of Multicomputer-Multicore Systems: A Review,” QALAAI ZANIST JOURNAL, vol. 6, no. 4, pp. 1137–1140, 2021.

N. Goel, V. Laxmi, and A. Saxena, “Handling multithreading approach using java,” International Journal of Computer Science Trends and Technology (IJCST), vol. 3, no. 2, pp. 24–31, 2015.

L. Haji, R. R. Zebari, S. R. M. Zeebaree, W. M. Abduallah, H. M. Shukur, and O. Ahmed, “GPUs Impact on Parallel Shared Memory Systems Performance,” International Journal of Psychosocial Rehabilitation, vol. 24, no. 08, pp. 8030–8038, 21, May, doi: 10.37200/IJPR/V2418/PR280814.

O. H. Jader et al., “Ultra-Dense Request Impact on Cluster-Based Web Server Performance,” in 2021 4th International Iraqi Conference on Engineering Technology and Their Applications (IICETA), 2021, pp. 252–257.

O. G. Lorenzo, T. F. Pena, J. C. Cabaleiro, J. C. Pichel, and F. F. Rivera, “Multiobjective optimization technique based on monitoring information to increase the performance of thread migration on multicores,” in 2014 IEEE International Conference on Cluster Computing (CLUSTER), 2014, pp. 416–423.

V. Weinberg, J. Duato, E. El-Araby, and V. Narayana, “An Approach to Parallel Processing,” International Journal on Recent and Innovation Trends in Computing and Communication, vol. 6, no. 2, pp. 126–128.

Y. Xu, P. Liu, I. Penesis, and G. He, “A task-resource mapping algorithm for large-scale batch-mode computational marine hydrodynamics codes on containerized private cloud,” IEEE Access, vol. 7, pp. 127943–127955, 2019.

M. P. R. B. A. Bianco, “HERO: High-speed Enhanced Routing Operation in Ethernet NICs for Software Routers⋆,” 2020.

Z. Lv, D. Chen, and A. K. Singh, “Big data processing on volunteer computing,” ACM Transactions on Internet Technology, vol. 21, no. 4, pp. 1–20, 2021.

L. M. Haji, S. R. M. Zeebaree, O. M. Ahmed, M. A. M. Sadeeq, H. M. Shukur, and A. Alkhavvat, “Performance Monitoring for Processes and Threads Execution-Controlling,” in 2021 International Conference on Communication & Information Technology (ICICT), 2021, pp. 161–166.