ARID ZONE JOURNAL OF ENGINEERING, TECHNOLOGY & ENVIRONMENT
AZOJETE September 2022. Vol. 18(3):413-426
Published by the Faculty of Engineering, University of Maiduguri, Maiduguri, Nigeria.
Print ISSN: 1596-2644, Electronic ISSN: 2545-5818
www.azojete.com.ng

ORIGINAL RESEARCH ARTICLE

A RESOURCE PERFORMANCE MODEL FOR EFFICIENT DISTRIBUTION OF RESOURCES TO OPERATING SYSTEM SERVICES IN FACTORED OPERATING SYSTEM

H. Mikailu[1], I. Agaji[2]*, B. V. Nachamada[3] and I. Gambo[4]
[1]*Deparment of computer Science, Nigerian Army University, Biu, Nigeria*
[2]*Department of Computer Science, Federal University of Agriculture Makurdi, Nigeria*
[3]*Department of Computer Science, University of Jos, Jos, Nigeria*
[4]*Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria*
**Corresponding author's email address: ior.agaji@uam.edu.ng*

ARTICLE INFORMATION

Submitted 23 Nov., 2021
Revised 7 May, 2022
Accepted 15 May, 2022

Keywords:
Factored Operating System
Multicore
Multi-Agent
fleets of servers
performance function

ABSTRACT

Central processing will be composed of thousands of heterogeneous cores in the near future. The existing systems are difficult to scale, adapt or tackle the heterogeneous nature of the future multicore technology. This study developed a resource performance model for efficient distribution of resources to Operating System (OS) services. A Multi-Agent based method was used to design the architecture of the model while the Unified Modeling Language and flowchart were used in the detailed design of the proposed model. The model was simulated using Java 2 Enterprise Edition. In simulating the model, four (4) variables were used to determine the processor core capacity. The result of the simulation shows an efficient distribution of 5000 cores to four (4) OS services (servers) with each having 1250 fleet of servers. The percentage differences in the four of servers from the minimum are 0.11%, 0.09% and 0.04% respectively. The result shows that the distribution of processor cores to OS services is efficient since the differences in total performance function of the fleets of servers were very little. Therefore, to maximize the profit that comes with multicore systems, this efficient model is needed for processor cores distribution to OS service in a Factored Operating System.

1.0 Introduction

Processor performance is increasing rapidly as the number of cores on the chip increases at each release of a new generation of central processing unit (CPU) and with multicore, CPU is becoming not only faster but more powerful and efficient. The emergence of two families of computing technology (multicore and cloud computing) has the potential to improve the computational system capacity to the average computer user (Nafaa, 2015). Air Force Research Laboratory (2012) reported that in the future, computer chips with 1,000 to 10,000 cores on one slice of silicon will be produced with the recent technological advancement. The growth in hardware manufacturing technology and the progressing evolution which agrees to Moore's law, could afford computer microprocessor chip with thousands of processor cores on a single microprocessor in the next decade (Wentzlaff and Agarwal, 2016). The evolution of multicore processors suggest that single-stream performance may not get louder in the future. Microprocessor manufacturers have resorted to integrating multiple processors onto a single chip (Wentzlaff and Agarwal, 2016).

In respect to the anticipation of exponential increase in a number of cores, the genre has fundamentally changed. The question and the problem now is how to efficiently utilize the abundant resources available, and not how to contend with scarce system resources. There exists a need for the software developers to step up its development technology to produce systems that are scalable and adaptable to the ever-changing hardware environment. Furthermore, the future central processors will be composed of thousands of heterogeneous cores, while the design of the existing systems are such that they are not scalable. The benefits that the multicore technology might not be harvested, hence the necessity of this research (Nafaa, 2015).

Factored Operating System (FOS) is, therefore, designed with this motive, taking scalability and adaptability as the primary design constraints. FOS is a new operating system designed for multicore processors and cloud computers. In FOS, OS services are implemented as independent distributed systems running on different cores from user applications where each service is divided into a fleet or parallel set of collaborating processes that commune using messages. FOS aims to design system services that scale from a few to thousands of cores (Air Force Research Laboratory, 2012).

This study is therefore aimed at developing resource performance model for efficient distribution of resources to OS services for a FOS to address the inefficiency of the conventional OS in utilizing multicore resources. This is allows the resource performance model to compute resource capacity, and efficiently distribute these resources to various OS services that will form fleets of independent distributed servers. For a better scalability to be attained, system resources would be employed as many processor cores running simultaneously or as an individual, rapid and potent core for executing a single-threaded and serial part of the application processes.

2. Review of Relevant Literature

The International Technology Roadmap for Semiconductors, ITRS, (2015) reported that in each new generation of technology, the producers of microprocessors often increment the relative frequency of the processors, which was possible on account of Moore's Law. In the early twenty-first century, a physical cap in the form of thermal limits was encountered. By increasing both the number of transistors on a chip and rising the clock speed was not achievable. Attempting increased number of transistors on a chip and rising the clock speed induced severe trouble associated with heat dissipation. Therefore, microchip producers resolved to go on with Moore's Law by yet manufacturing microchips with an increased number of transistors having insignificant frequency increments. Hence, the main reason behind the change of viewpoint, from a single core technology to that of multicore systems.

Borkar (2007) affirmed that Moore's Law remains in effect whenever the increment in the number of transistors on integrated circuits is maintained. It has, therefore, become reasonable to anticipate computer systems with thousands of heterogeneous processor cores in the future. To this end, it becomes critical for operating systems to scale with the new hardware technology. The resources of the upcoming multicore systems should be managed with efficiency other than expected, else it will be improbable for application processes executing on such a platform to take full benefits of the newly available technology.

According to Laplante and Milojicic (2016), the deceleration of processor scaling due to Moore's law resulted in the resurgence research in new types of computing structures, the need arises for rethinking operating systems paradigms. It is envisioned that by 2030 operating systems will be created using a new technology paradigm that would be nearly unrecognizable today.

Wickizer *et al*. (2008) and Wentzlaff and Agarwal (2009) demonstrated that locks may not proffer the wanted scalability sought after by the new technological developments. Thus, a study of the factored operating system was incited. Their performance differences were made visible depending on core count. Two observations were made in this research, they were; (i) with an increase in the number of cores, the lock contention cost factor became the largest, and the synchronization overhead was seen to have consumed most resources. (ii) the total execution time was heightened with more than eight active cores in the benchmark. The research indicated that synchronizing the operating cost induced by locks was the reason for performance declined. Thus, if software design remains as it is at present, lock contention will continue to be a big problem to multicore processor systems.

Schart (2016) identified three fundamental design problems of OS for multicore structure: they were locks, sharing of processor cores and cache-coherent shared memory. In this study, the impact of these challenges on scalability and proposed locks avoidance to counter scalability threat was highlighted. This would be achieved via splitting up operating system services onto dedicated cores where processes will request for operating system functionality in a concurrent manner. The work also proposed the splitting up of application and OS, so as to avoid sharing of cores with each other, rather, each core will be dedicated to every thread on the system. Lastly, instead of cache coherent shared memory, the work proposed message passing to be used for operating system or application communication. The work offered some improvements, but it is not clear, how to structure an OS with locks that proffer satisfactory design scalability and load scalability as well. Hence, it is sensible to name lock as a challenge because of the technological advancement of multicores.

Reuther *et al*. (2017) presented a detailed feature analysis of 15 supercomputing and big data schedulers. For big data workloads, the scheduler latency was the most critical performance characteristic of the scheduler. A theoretical model of the latency of these schedulers was developed and used to design experiments targeted at measuring scheduler latency. Detailed benchmarking of four of the most popular schedulers Slurm, Son of Grid Engine, Mesos, and Hadoop YARN were conducted. The model results indicated that scheduler performance could be characterized by two key parameters: the marginal latency of the scheduler t_s and a nonlinear exponent α_s. For all four schedulers, the utilization of the computing system decreased to less than 10% for computations lasting only a few seconds, while multi-level schedulers such as LLMapReduce that transparently aggregated short computations could improve utilization for these short computations to >90% for the four schedulers that were tested. Thus, conventional supercomputing schedules currently filled the role of managing heterogeneous resources but have inherent scalability limitations.

In another study, Asmussen *et al.* (2016) integrated cores and memories into a packet-switched network-on-chip (NoC) and equipped each core with a Data Transfer Unit (DTU) as the common hardware component. The only means for the core to communicate with other cores or memories was through the DTU, offering message passing and memory access. Controlling the DTU allowed the control of the core and therefore, the software running on the core. OS services like file systems and network stacks were provided based on a core-neutral communication protocol between DTUs. The work introduced network-on-chip-level isolation, presented the design of microkernel-based OS, M3, and the familiar hardware interface, and evaluated the performance of the prototype in comparison to Linux. The result showed that without using accelerators, M3 outmatches Linux in some application-level benchmarks by more than a factor of five. However, this design decreases *system* utilization as the processing element (PE) was idle for a specific time, waiting for an incoming message or the completion of a memory transfer.

Furthermore, Clements *et al.* (2013) introduced a scalable commutativity rule that provides a new approach for software developers to understand and exploit multicore scalability right at the software interface. The work introduced COMMUTER to aid programmers to analyze interface commutativity and test to ensure implementation scales in commutative situations. COMMUTER was applied to 18 POSIX calls and the results were used to guide the implementation of a new research operating system kernel called sv6. Linux scales for 68% of the 13,664 tests generated by COMMUTER for these calls and COMMUTER finds many problems that have been observed to limit application scalability, while on the hand, the sv6 scales for 99% of the tests. Thus, using sv6, it was observed that it was practical to achieve a broadly scalable implementation of POSIX by applying the rule, and that commutativity was also essential to achieving scalability and performance on real hardware. However, it became hard to identify bottlenecks and impractical to fix problems since product testing was done late in the production.

Rakhee and Garg (2014) provided an overview to multicore processors, multicore processor parallelism and performance measurement for multicore Central Processing Units (CPUs).and addressed the techniques used to evaluate multicore CPU performance, metrics, factors, benchmark tools used to measure multicore CPU performance. The works were anticipated new approaches in multicore CPU performance analysis as multicore CPU production increases to new levels.

Baumann *et al.* (2009) experimented with an Advanced Micro Devices (AMD) system with four CPUs and each with four processors, making sixteen processors used for the benchmarks. At first processes were devoted to a processor updating the space of a little part of memory, and cache coherence techniques on hardware responded by releasing these modifications to other processor cores. As core counts grew, it was observed that the performance declined, demonstrating inadequate load adaptation. Efficiency was noticed to drop by a factor of 40 when achievements between one core and sixteen cores were compared. Subsequently, inter-core communication (messaging) was tested and was observed to have performed better. No degradation whatsoever was noticed by the increase in number of cores using a message passing method. Where delay was seen is when the focus was not on only on passing a message, but on executing it as well. However, in this circumstance, a linear increase in time was detected, which is attributed to a queuing delay.

Figure 1 illustrates the core layout in FOS. It depicts three applications and operating system server cores. Two applications denoted as A_0 and A_2 enjoy the benefit of user-space cache-coherent shared memory. The operating system servers denoted as S do not utilize cache-coherent shared memory, but interact via messaging. Application A_1 does not require cache-coherent shared memory. This feature can either be enabled or excluded, depending on the specific requirements of an application.

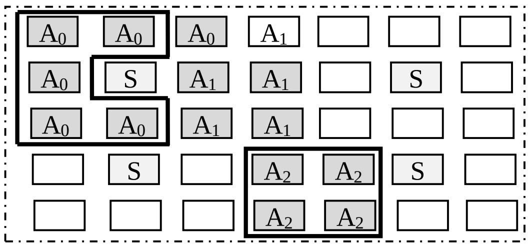

Figure 1: Cores Layout in a FOS Environment (Source: Schartl, 2016)

Wentzlaff *et al*. (2011) evaluated fleets within FOS and designed and implemented three critical fleets (network stack, page allocation, and file system) and compared with Linux. The comparisons showed that FOS achieved superior performance and had better scalability than Linux for large multicores. With 32 cores, FOS page allocator performed 4.5 times better than Linux, and FOS network stack performed 2.5 times better. Additionally, the work demonstrated how fleets could adapt to changing resource demand and the importance of spatial scheduling for excellent performance in multicores. Their experiments have demonstrated that this design is appropriate for the multicore design architectures of the future while proffering improved scalability than the existing solutions. The uniqueness of factored operating system design architecture grants operating system the privilege to scale to high core counts without meddling with the application.

Wentzlaff *et al*. (2011) experiment concerning the impact of proper spatial scheduling on performance in multicore processor systems used the read-only file system fleet with 'good' and 'bad' layouts and was executed on a 16-core Intel Xeon E7340. This machine had a minimal intra-socket communication cost applying user-space messaging, which revealed significant communication heterogeneity between intra-socket and inter-socket communication. The file system fleet consisted of four servers, and the clients' number increases from one to ten. Single file system server resides on each socket in the good layout experiment. The file system component of libfos was self-aware and selected the local file system server for all requests. In the bad layout experiment, all servers resided on a single socket and clients were distributed among remaining sockets. The result of the experiment showed that good layout is uniformly better than the lousy layout in performance. For large multicores with hundreds of cores, it was anticipated that communication heterogeneity will increase further. The authors asserted that global structures that achieved symmetric performance, e.g., buses, will not scale to large core counts and new on-chip will expose heterogeneous communication costs. The work maintained that future multicores would feature much more significant heterogeneity and commensurately higher end-to-end performance inequality from spatial scheduling.

To improve the system performance through utilizing the exponential increase in transistor hardware, microprocessor industries have reversed to integrating multiple processors onto a single die. Some present examples include Intel and AMD's Quad-core offerings, Tilera's 64-core processor, and an 80-core Intel prototype processor. According to major microprocessor manufacturers' suggestion, the trend of integrating more cores onto a single microprocessor will go on. By inferring the doubling of transistor resources every one and half year, in a short period, over 6000 processor cores on a single microprocessor will be integrated (Beckmann, 2010; Wentzlaff and Agarwal, 2016).

Efforts have been applied to combat heterogeneity problems of hardware as well as scalability and adaptability, however, conventional supercomputing schedules currently filled the role of managing heterogeneous resources but have inherent scalability limitations. How to structure an OS with locks that can proffer satisfactory design scalability and load scalability as well remains a challenge. Moreover, some of the proposals design, decreases *system* utilization as the processing element was idle for a specific time, waiting for an incoming message or the completion of a memory transfer. Also, a linear increase in time was detected, which is attributed to a queuing delay (Salto and Alba, 2015).

Thus, since caches could be employed to their fullest potency, load scalability can be increased. The new method is to maintain operating system services and application code on different cores, as free cores becomes a commodity. It is, therefore, beneficial that a set of cores be devoted to operating system services only, since the number of cores is in the rise, having observed that processor core sharing inflicts extra costs in terms of cache misses. It is highly believed that this will result in more efficient use of the available hardware when the proposed resource performance model for efficient distribution of resources to OS services for a FOS is implemented (Salto and Alba, 2015).

3. Methodology

The research methodology adopted in this work is the Multi-Agent Oriented Software Development Method. To develop a resource performance model for efficient distribution of resources to Operating System (OS) services for a Factored Operating System, a Multi-Agent based method is used to design the architecture of the model. A Unified Modeling Language (UML) and flowchart were used to design the proposed model. The model is simulated using Java 2 Enterprise Edition. In simulating the model, four (4) variables are were to determine the processor core capacity, they are: processor type (pt), processor speed (sp), register size (rs) and cache size (cs). Random integer numbers were generated between 1 to 40 to represent the values of the variables, each with an associated weight of 0.4, 0.3, 0.2 and 0.1 respectively. These were used in order of priority to compute and determine the rank of processor cores capabilities. These processor cores were then logically and efficiently distributed to the four (4) OS services using the algorithm proposed for this model to form fleets of independent distributed servers. One advantage of agent-based architectures is that it is relatively easy to extend and expand, as new conditions are discovered or prioritized. The method is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements.

3.1 Proposed Design of the Resource Performance Model

The resource performance model computes the capacity resources and distributes it to the OS services. This is intended to compute the capacities of processor cores and distributes them to OS services (PC security management, file system naming, scheduler and file management). This is to enable the schedulers to allocate resources optimally to the requesting process jobs, having estimated their resource requirements, to attain efficient system throughput.

Design principles applied in this research are to replaces the traditional OS use of time multiplexing in processing with application code via a system call interface with space multiplexing. Also, OS services are factored into function-specific services and implemented as a parallel independent distributed service. This is because operating system services are divided from user application code and executed on separate cores. Moreover, each service is divided into a fleet or parallel set of collaborating servers that interact using messages. The aim is to develop systems that scale from a few to thousands of cores.

3.2 The Proposed Architecture of Resource Performance Model

The architecture of the proposed framework is shown in Figure 2. In this framework, a multi-agent method is modeled in the form of a distributed system designed to represents resource performance model as a group of autonomous agents. The resource agent extracts resource core information, computes capacities of the resource cores and distributes these resources to the various operating services to form fleets of servers.

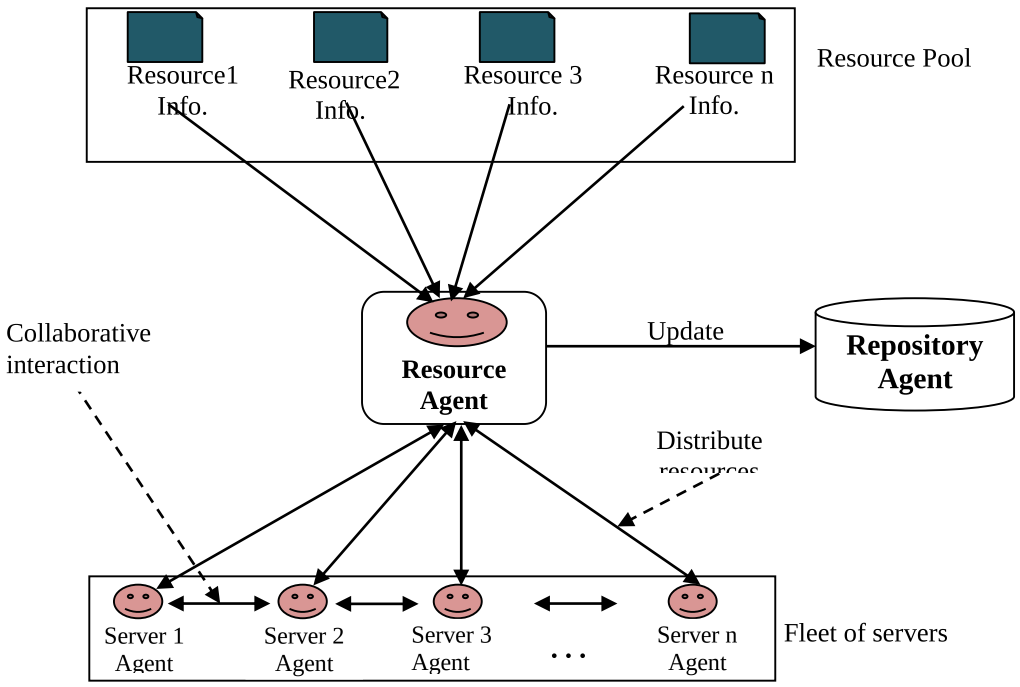

Figure 2: The Architecture of the Resource Performance Model

3.3 The Resource Performance Model

The proposed resource performance model is depicted in Figure 3, which shows the activities performed, such as: computing resource capacity and distributing resources to operating system services. The parameters of both the resources R are input into the model. The system begins by extracting the parameters of and R_j; where R_j represents 1 to n number of resources each with four parameters as well. The computed resources are then distributed to the various operating system services to form fleets of servers. The resource performance sub-model shows that for each 1 to n resource (processor cores) performance function (PF) were computed and the results generated. Each of the resource was associated with its performance capacity. The algorithm described the factors that picture the quality of a server processing capability, such as processor type, speed, register and cache size. The server performance was calculated to gauge the processing capability of each server. Thus, this was used as performance metrics for optimal servers' distributions.

For the purpose of this study, four processor variables were used, they include; processor type (pt), processor speed (ps), register size (rs) and cache size (cs). Therefore, each available server (S) was ranked in accordance with its respective computation capability. The server performance function (PF) is expressed as:

$$PF(S) = (\beta 1 \text{ x pt}) + (\beta 2 \text{ x ps}) + (\beta 3 \text{ x rs}) + (\beta 4 \text{ x cs}) \qquad (1)$$

where β1, β2, β3 and β4 are the weights of the first to fourth terms respectively, for instance, β1 = 0.4, β2 = 0.3, β3 = 0.2 and β4 = 0.1, this is in descending order of priority. Higher PF value for a server means having better performance possibility as opposed to other servers. Hence, by applying the ranking method, servers were ranked according to their PF values. If the PF value of a server was high, then the server rank was high as well.

The resource distribution flowchart module in Figure 3 depicts resource distribution into various operating system services. It showed 1 to n number of operating system services (e.g. system file naming, file management, scheduling, PC security management and error detection), and 1 to n number of processor cores (resources) to be distributed to the services. Resource R_j is assigned to service S_i, where j = 1 to n and i = 1 to m respectively. This algorithm fully distributed resource cores to the operating system services forming fleets of servers. The allocation of resources to operating system services was in sequential order until all the operating system services were allocated a processor core in the first round and also the next. This algorithm addressed the allocation problem pointed out by Beckmann (2010) as the current scheduling problem in factored operating system. Furthermore, when an additional resource was added to the system at any time, the resource agent followed the same procedure to redistribute the processor cores.

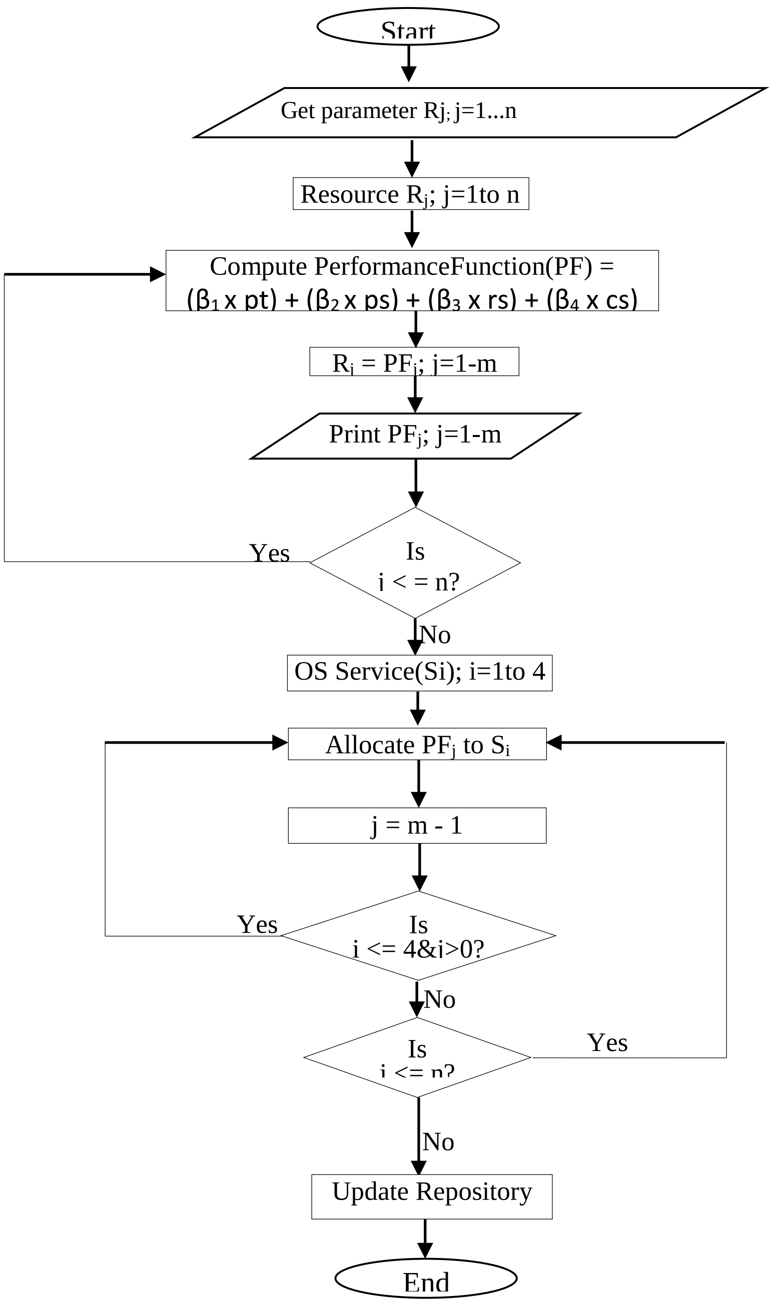

Figure 3: The Flowchart of the Proposed Model

3.4 The Sequence Diagram of the Proposed Model

Figure 4 depicts a unified modelling language (UML) sequence diagram that described agents' interaction in a manner in which they interacted with each other. The resource agent computes the capacity or performance function of each resource and distributes the resources to the various operating system services.

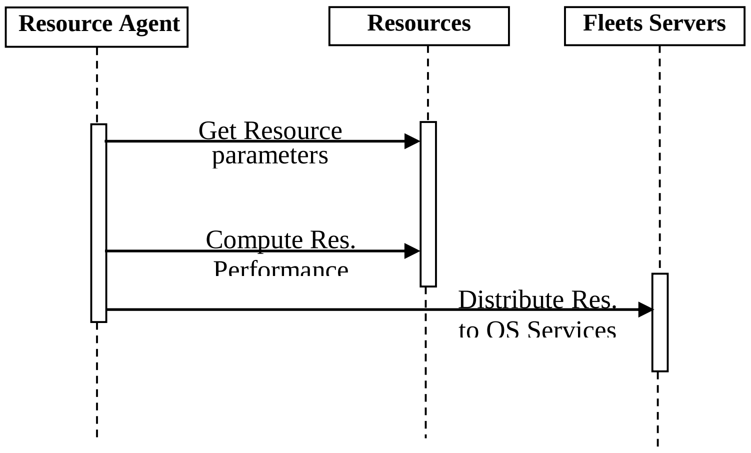

Figure 4: The Sequence Diagram of the Proposed Model

The model was simulated using java-2 Enterprise Edition. The properties of an array were used as signals that formed the initial data generation to train the model. The array length and the sum values of the array elements were used as the input parameters of the process jobs. A randomly generated data between -1 and 1 were used as input weights for the network.

4 The Experiment

In the experiment, the processor cores performance capacity was computed and distributed to the OS services. This experiment computes the capacities of processor cores and distributes them to the four OS services identified in the simulation of the model. In this experiment, 5000 processor cores were used for the simulation purpose. The data input for this experiment include the Processor Type, Processor Speed, Register Size, and Cache Size which are used as processor core parameters. These parameters were used to determine the capacity of each processor in the simulation. In Figure 5, the input page for the experiment is depicted.

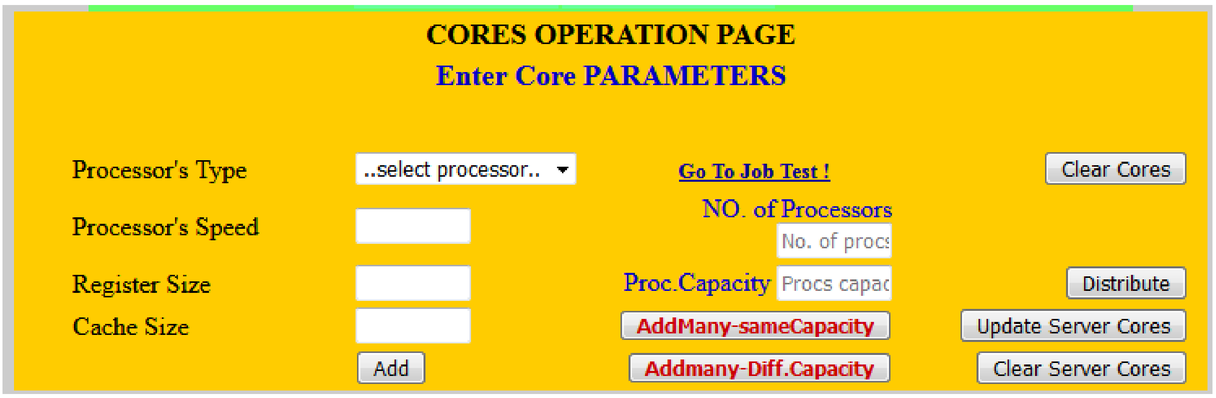

Figure 5: Processor Computation and Distribution Input Page

5 The Result and Discussion

The sample results of the computed and distributed processor cores are shown in Tables 1, 2, 3 and 4, along with the total number of 1250 processors for each server. Where FMA and SCH represent the File Manager and Scheduler respectively.

Table 1: File Manager Server

FILE MANAGER FLEET OF SERVERS INFORMATION!	
Processor Id	Performance (GHz)
FMA-1	40.0
FMA-56	39.0
FMA-57	38.0
FMA-120	37.0
FMA-121	37.0
FMA-122	36.0
FMA-249	33.0
FMA-250	32.0
FMA-395	27.0
FMA-1080	6.0
FMA-1085	5.0
FMA-1142	4.0
FMA-1143	4.0
FMA-1147	3.0
FMA-1148	3.0
FMA-1211	2.0
FMA-1212	2.0
FMA-1213	2.0
FMA-1249	1.0
FMA-1250	1.0
Total Capacity	25541.0

Table 2: Scheduler Server

SCHEDULER FLEET OF SERVERS INFORMATION!	
Processor Id	Performance (GHz)
SCH-1	40.0
SCH-2	40.0
SCH-56	39.0
SCH-57	38.0
SCH -120	37.0
SCH -250	32.0
SCH -305	31.0
SCH -306	30.0
SCH -908	12.0
SCH -909	12.0
SCH -910	11.0
SCH -969	10.0
SCH -972	9.0
SCH -1023	8.0
SCH -1029	7.0
SCH -1079	6.0
SCH -1080	6.0
SCH -1085	5.0
SCH -1142	4.0
SCH -1250	1.0
Total Capacity	25523.0

Table 3: System File Naming Server

SYSTEM FILE NAMING FLEET OF SERVERS INFORMATION!	
Processor Id	Performance (GHz)
SFN-1	40.0
SFN -121	37.0
SFN -122	36.0
SFN -123	36.0
SFN -184	35.0
SFN -185	34.0
SFN -249	33.0
SFN -785	15.0
SFN -849	14.0
SFN -850	13.0
SFN -851	13.0
SFN -908	12.0
SFN -909	12.0
SFN -910	11.0
SFN -1079	6.0
SFN -1080	6.0
SFN -1085	5.0
SFN -1142	4.0
SFN -1143	4.0
SFN -1147	3.0
SFN -1249	1.0
SFN -1250	1.0
Total Capacity	25513.0

Table 4: PC Security Manager Server

PC SECURITY MANAGEMENT FLEET OF SERVERS INFORMATION!	
Processor Id	Performance (GHz)
PSM -1	40.0
PSM -2	40.0
PSM -56	39.0
PSM -57	38.0
PSM -120	37.0
PSM -122	36.0
PSM -123	36.0
PSM -783	16.0
PSM -784	16.0
PSM -785	15.0
PSM -849	14.0
PSM -969	10.0
PSM -970	9.0
PSM -971	9.0
PSM -972	9.0
PSM -1023	8.0
PSM -1029	7.0
PSM -1079	6.0
PSM -1080	6.0
PSM -1085	5.0
PSM -1142	4.0
PSM-1250	1.0
Total Capacity	25533.0

4.3 Discussions

Tables 1, 2, 3 and 4 constitute the result of processor cores computed and distributed to OS services that included file manager, scheduler, system file naming and PC security manager services of the OS. The sever capacities were observed to range from 1GHz to 40GHz as performance capacity in the tables. Table 1 shows that Processor Id FMA-1, Processor Id FMA-395 and Processor Id FMA-1080 with Performance Capacity of 40.0GHz, 27.0GHz and 6.0GHz respectively were allocated to the File Manager Server. The result in Table 1 is the File Manager Server of the OS service that consists of 1250 sets of processors allocated to it. This is also observed with the other OS services in Table 2, the Processor Id SCH-1, Processor Id SCH-972 and Processor Id SCH-1142 with Performance Capacity of 40.0GHz, 9.0GHz and 4.0GHz respectively were allocated to the Scheduler Server. The result in Table 2 is the File Manager Server of the OS service that consists of 1250 sets of processors allocated to it. In Table 3, the Processor Id SFN-1, Processor Id SFN-850 and Processor Id SFN-1250 with Performance Capacity of 40.0GHz, 13.0GHz and 1.0GHz respectively were allocated to the System File Naming Server. The result in Table 3 is the File Manager Server of the OS service that consists of 1250 sets of processors allocated to it. In Table 4, the Processor Id PSM-1, Processor Id PSM-970 and Processor Id PSM-1085 with Performance Capacity of 40.0GHz, 9.0GHz and 5.0GHz respectively were allocated to the PC Security Management Server. The result in Table 2 is the File Manager Server of the OS service that consists of 1250 sets of processors allocated to it. The performance capacities of these OS services were summed, giving a total of 25541.0GHz for file manager, 25513.0GHz for scheduler, 25523.0GHz system file naming and 25533.0GHz for the PC security manager. Thus, the framework used these results to aid optimal scheduling of process jobs concerning their resource requirements. It is observed that the resource computation and distribution to various OS services were adequately accurate and nearly equal in capacities that can proffer solution to the allocation problem (how many cores should be allocated to each OS service) as pointed by (Beckmann, 2010).

Moreover, the result of the simulation shows an efficient distribution of 5000 cores to four (4) OS services (servers) with each having 1250 fleet of servers. The percentage differences in the four of servers from the minimum are 0.11%, 0.09% and 0.04% respectively. The result shows that the distribution of processor cores to OS services is efficient since the differences in total performance function of the fleets of servers were very little. Therefore, to maximize the profit that comes with multicore systems, this efficient model is needed for processor cores distribution to OS service in a Factored Operating System.

5. Conclusion

This study was embarked upon because the conventional OS with a single scheduler is inefficient in multicore resource utilization, particularly with the advancement of multicore hardware technology. The framework has computed resource capacities and efficiently distributed processor cores to the various OS services in FOS with a negligible difference in capacities. The work enables multiple processes to access scheduling services in parallel, enable simultaneous allocation of processor cores to processes, thereby, utilizing the abundant system's processor cores. The model surmounts the scalability and adaptability issue that comes with ever growing core counts presented by the advancement in hardware technology. A performance processor core resource was developed to compute resource capacity and then allocated these processor cores to the various FOS services. This enables the framework to function in a distributed fashion and also proffered solution of cores allocation to OS services which is a major scheduling problem of FOS identified by Beckmann (2010). To improve on the overall performance throughput of the model, a process job transfers between the processor cores will be implemented in the future.

References

Air Force Research Laboratory 2012. FOS: A Factored Operating System for High Assurance and Scalability on Multicores. Massachusetts Institute of Technology Technical Report, Rome, NY, USA, 27pp. https://pdfs.semanticscholar.org/8257/e7 21b495476132ba49d515fce417e2 bdb991.pdf

Asmussen, N., Volp, M., Nothen, B., Hartig, H. and Fettweis, G. 2016. Operating-Systems Chair and Vodafone Chair Mobile Communications Systems. M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores. Retrieved on 3rd December 2017, from https://os.inf.tudresden.de/papers_ps/asmussen-m3-asplos16. pdf

Baumann, A., Barham, P., Dagand, P., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schiipbach, A. and Singhania, A. 2009. The multi-kernel: A New OS Architecture for Scalable Multicore Systems. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. 11-14 October, Big Sky, Montana, USA., pp. 1-15. https://www.sigops.org/s/conferences/sosp/2009/papers/baumann-sosp09.pdf

Beckmann, N. 2010. Distributed Naming in a Factored Operating System. MSc. Thesis. Department of Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A. https://www.researchgate.net/publication/267718821

Borkar, S. 2007. Thousand Core Chips – A Technology Perspective. Proceedings of DAC. 4-8 June, San Diego, California, USA., pp. 746-749. http://impact.asu.edu/cse520fa08/ThousandCore.pdf

Clements, AT., Kaashoek, MF., Zeldovich, N., Morris, RT. and Kohler, E. 2013. The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors. Retrieved on 23rd July 2017, from https://people.csail.mit.edu/nickolai/ papers/clements-sc.pdf

Laplante, P. and Milojicic, D. 2016. Rethinking Operating Systems for Rebooted Computing. Proceedings of The IEEE International Conference on Rebooting Computing (ICRC). 19-21 October, San Diego, California, USA., pp. 1-8., https://ieeexplore.ieee.org/abstract/document/7738695/authors

Nafaa, H. 2015. FOS (Factored Operating System): An Operating System for Multicore and Clouds. Retrieved on 3rd April, 2018, from http://www.slideshare.net/mobile/naffa1/factore-operating-system-an-operating-system-for-multicore-and clouds

Rakhee, C. and Garg, RB. 2014. Multicore Processor, Parallelism and their Performance Analysis. International Journal of Advanced Research in Computer Science & Technology, 2(3): 31-37.

Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A. and Kepner, J. 2017. Scalable System Scheduling for HPC and Big Data. Preprint Submitted to Journal of Parrallel and Distributed Computing, 1-34. Retrieved on 17th December 2018 from, https://arxiv.org/pdf/1705.03102.pdf

Salto, C. and Alba, E. 2015. Adapting distributed Evolutionary Algorithms to Heterogeneous Hardware. Transactions on Computational Collective Intelligence XIX, pp.103-125. DOI:10.1007/978-3-662-49017-4_7

Schartl, A. 2016. Design Challenges of Scalable Operating Systems for Manycore Architectures. Retrieved on 7th April 2017, from http://www4.cs.fau.de/Lehre/WS16/PSKVBK/slides/slides-schaertl.pdf

The International Technology Roadmap for Semiconductors, 2015. The Executive Report 2.0. Retrieved on 18th February 2017 from https://www.semiconductors.org/wpcontent/uploads/2018/06/0_2015-ITRS-2.0-Executive-Report-1.pdf

Wentzlaff, D. and Agarwal, A. 2009. Factored Operating System (fos): The Case for Scalable Operating System for Multicores. Newsletter ACM SIGOPS Operating Systems Review Archive, 43(2): 76-85. DOI: 10.1144/1531793.1531805

Wentzlaff, D., Gruenwald, C., Beckmann, N., Modzelewski, K., Belay, A., Kasture, H., Youseff, L., Miller, J. and Agarwal, A. 2011. Fleets: Scalable services in a factored operation system. Computer Science and Artificial Intelligence Laboratory Technical Report, Massachusetts Institute of Technology, Cambridge, Ma 01239, USA.

Wentzlaff, D. and Agarwal, A. 2016. Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores. Retrieved on 6th April 2018, from http://www.slideshare.net/mkindika/factored-operating-systems

Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, F., Morris, R., Pesterev, A., Stein, L., Ming Wu, M., Dai, Y., Zhang, Y. and Zhang, Z. 2008. Corey: An Operating System for Many Cores. Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08) , San Diego, California, (December 2008), 43-57 http://www.usenix.org/events/osdi/tech/full_papers/boyd-wickizer/boyd_wickizer.pdf