IE2050
Operating Systems

2nd Year, 2nd Semester

Assignment

Analysis Report on NileOS

OS_17

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

04.05.2024

Declaration

We certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Student ID	Name	Signature
IT22564986	K. B. O. V. Bandara	[illegible]
IT22365132	C. B. Gunasena	[illegible]
IT22361172	D. G. N. Mayadunne	[illegible]
IT22571298	P. K. A. Silva	[illegible]

Table of Contents

1. Introduction

The exponential growth of data has made big data processing one of the most challenging problems in the computing domain. However, traditional general-purpose operating systems are not well-suited to address the fundamental processing, data distribution, and communication needs of big data applications, in terms of scalability and performance. These operating systems, designed with prioritizing interactivity, cannot accurately identify the workload profile of running processes and often introduce unnecessary overheads that significantly impact performance.

In this context, the paper presents NileOS, a new distributed, asymmetric core-based micro-kernel, designed specifically for big data processing. NileOS adopts a core-based Asymmetric Multiprocessing (AMP) approach, optimizing interrupt management and I/O operations to suit the Map-Reduce model. The micro-kernel's design is based on Inter-processor Interrupts over Ethernet (IPIoE) frames and a Bare Metal Operating System Markup Language (BOSML), enabling transparent deployment, and separating the application implementation from its deployment perspective.

Paper Title: "NileOS: A Distributed Asymmetric Core-Based Micro-Kernel for Big Data Processing"

Conference: Future Technologies Conference (FTC) 2020

Received: December 16, 2020

Accepted: December 25, 2020

Date of Publication: December 30, 2020

Date of Current Version: January 7, 2021

Digital Object Identifier: 10.1109/ACCESS.2020.3048082

Authors:

- Ahmad El-Rouby
- Andrew Khalaf
- Arig Mostafa
- Fady Mohamed
- Nour Ghaly
- Amr El-Kadi (Senior Member, IEEE)
- Karim Sobh (Corresponding Author, kmsobh@aucegypt.edu)

Affiliation: Department of Computer Science and Engineering, The American University in Cairo, New Cairo 11835, Egypt

Funding: This paper was supported in part by the American University in Cairo (AUC).

2. Workload Matrix

Student ID	Name	Workload
IT22564986	K. B. O. V. Bandara	Introduction, Overview and Objectives of the NileOS covering Introduction, Design Goals and Decisions sections of the paper, Glossary
IT22365132	C. B. Gunasena	Approaches and Proposed solution covering Proposed Solution: NileOS section of the paper
IT22361172	D. G. N. Mayadunne	Evaluation and Outcomes covering Experiments section of the paper
IT22571298	P. K. A. Silva	Interpretation and Final Remarks covering Discussion and Conclusion and Future work sections of the paper

3. Analysis

Overview and Objectives

1) Overview

The paper considers the issues faced by the exponential growth of data, highlighting the growing complexity of big data processing in the computer domain. It highlights the limits of general-purpose operating systems in addressing the basic processing, data distribution, and communication requirements of big data applications, regarding scalability and performance. This highlights the massive cost developed by stopping large batch operations, particularly in distributed processing environments.

To solve these problems, the research offers a distributed core-based asymmetric multiprocessing microkernel designed exclusively for large data processing. This micro-kernel is intended to be installed on a cluster of service hardware, allowing communication across cores via message passing and service call mechanisms. The communication techniques facilitate Inter-Processor Interrupts (IPI) for local cores and a specialized network protocol, IPI over Ethernet (IPIoE), for remote cores. This also presents the BareMetal Operating System Markup Language (BOSML), which is a framework for simplifying message flow across various kernel images operating on different cores. The primary objective of the proposed micro-kernel is to extend the MapReduce model for big data processing, solving the requirement of an extensible representation framework to streamline message exchange.

Artificial Intelligence (AI) technologies, such as machine learning and deep learning, are essential for deriving meaningful insights from large datasets in the data-driven world of today. AI algorithms enable organizations to forecast future events, optimize processes, and make well-informed decisions. Modern computing relies heavily on the combination of AI with big data processing, which helps businesses manage, analyze, and extract meaningful insights from massive amounts of data from a variety of sources. Businesses may unlock the full potential of their data resources, promoting innovation, improving productivity by utilizing AI in big data processing. To achieve these targets computational power for big data processing should improve.

This study makes a significant addition to the field of big data processing by presenting a specialized micro-kernel that meets the difficulties of scalability, performance, and resource management in distributed computing systems. The paper's design decisions and aims pave the way for more efficient and effective large data processing solutions, directing to the right path for future research and development in this area.

2) Objectives with design goals

The paper outlines 7 key design goals for the NileOS micro-kernel to minimize the amount of data transferred across the network with the concept of ‘move-code-to-data’ by distributing and processing data among distinct cores.

- **Design goal 1**: Reducing overheads without impacting extendibility. The design implements a micro-kernel approach rather than a monolithic design, as the overhead of message passing is small compared to big data processing.
- **Design goal 2:** Minimize or eliminate interruptions on data processing tasks. This is to reduce memory overhead and enhance cache and TLB (translation lookaside buffer) performance. Unnecessary interrupts can introduce significant performance degradation for long-running batch processes.
- **Design goal 3:** Treate the CPU core as the fundamental building block, using an asymmetric multiprocessing (AMP) model rather than symmetric multiprocessing (SMP). This allows dedicate specific cores to different tasks.
- **Design goal 4:** Providing a single system image of the distributed environment, with resource location transparency and access transparency, so the application is unaware of the underlying hardware distribution.
- **Design goal 5:** Enable horizontal scalability by allowing easy addition of more nodes to the distributed system, with transparent discovery of resources.
- **Design goal 6:** Enable assignment of tasks to specific cores, whether local or remote, to control the core-based architecture.
- **Design goal 7:** Provide a programming model that allows adding services to the micro-kernel without requiring a deep understanding of the underlying architecture by the service developer.

These design goals aim to address the limitations of general-purpose operating systems in handling the requirements of big data processing.

3) Design Decisions

To achieve the previously mentioned design goals, these 6 design decisions need to be taken at the early stage of the system development while some decisions are still under research.

- **Design Decision 1**: Each core within a node has its own kernel image and address space to reduce overheads and unnecessary disruptions during processing tasks. This approach enables cores to handle specific events efficiently without interrupting other tasks.
- **Design Decision 2**: Cores within a node are assigned specific roles - 'Worker' cores for processing big data tasks, 'Management' cores for resource management, and 'I/O' cores for handling I/O devices. This role-based division optimizes core functionalities.
- **Design Decision 3**: Timer interrupts are disabled on worker cores to minimize interruptions, with timer interrupts routed to management cores when necessary. This decision aims to reduce interruptions, especially on worker cores, enhancing processing efficiency.
- **Design Decision 4**: Implementing basic functionalities for Network Management, Disk I/O Management, Memory Management, and Worker roles is essential for the micro-kernel's core functionality. These roles are crucial for supporting the micro-kernel's operations.
- **Design Decision 5**: Cores synchronize and communicate using Inter-processor Interrupts (IPIs) with location transparency, enabling remote communication and synchronization using a specialized Inter Processor Interrupts over Ethernet protocol (IPIoE). This mechanism supports efficient communication and service allocation across cores.

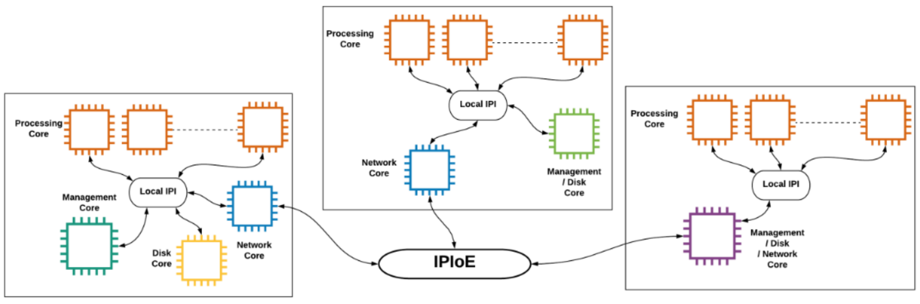


Figure 1 : Overall NileOS Diagram

- **Design Decision 6**: Sharing the Virtual Page Table's level 4 (PTEs) among all cores to optimize memory usage. Each worker core has a single process without tick interrupts, allowing single virtual table per core and efficient memory utilization through a design like an inverted page table.

These design decisions are important in designing the architecture and functionality of the NileOS micro-kernel, aligning with the main design goals of optimizing big data processing, reducing overheads, and enhancing system efficiency.

Approaches and Proposed Solution

1) System Overview

NileOS is made up of numerous nodes and operates as a distributed system. Every node is composed of versatile x86_64 hardware design. It is believed that nodes are linked through a fast network. From a single core point of view, remote resources can be accessed seamlessly without the user needing to know their physical location, making them appear as if they were local. Applications should not have knowledge of how the hardware is distributed; they should treat the system as a single, solid computing resource. Researchers proposed design inherently supports location transparency, allowing cores to transparently invoke micro-kernel service methods by name.

Each core needs to offer a specific set of services for the system to work properly. An asymmetric multiprocessing model is achieved by assigning roles through specialized services. At present, system consists of 4 core roles which may be expanded in the future: management, network, disk, and application.

2) The Micro-Kernel Backbone

The design of NileOS architecture prioritized extensibility. Kernel developers have the ability to create and integrate their own services into the micro-kernel with little difficulty. Ensuring that any service can be accessed from any node at any time is a top priority.

The central component of the NileOS micro-kernel, known as the Service Transport Layer (STL), serves as the foundation for adding new services and distributing incoming requests to registered services. STL is split into two main sections: the kernel STL (STL_K) for kernel mode services, and the user-space STL (STL_U) for user-mode services and applications interaction.

NileOS includes a necessary group of services essential for the operation of the micro-kernel, such as Physical Memory Service, Virtual Memory Service, Network Service, Disk I/O Service, and Gossip Service.

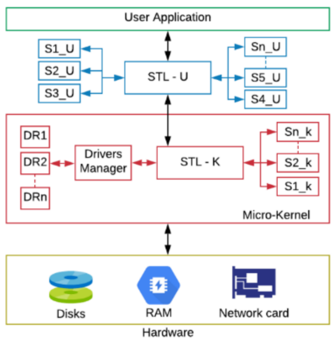


Figure 2: NileOS Kernel Architecture

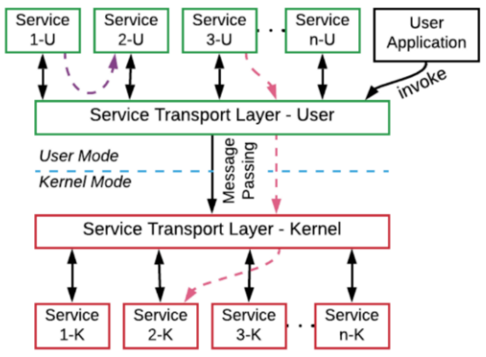


Figure 3: Service transport layer communication

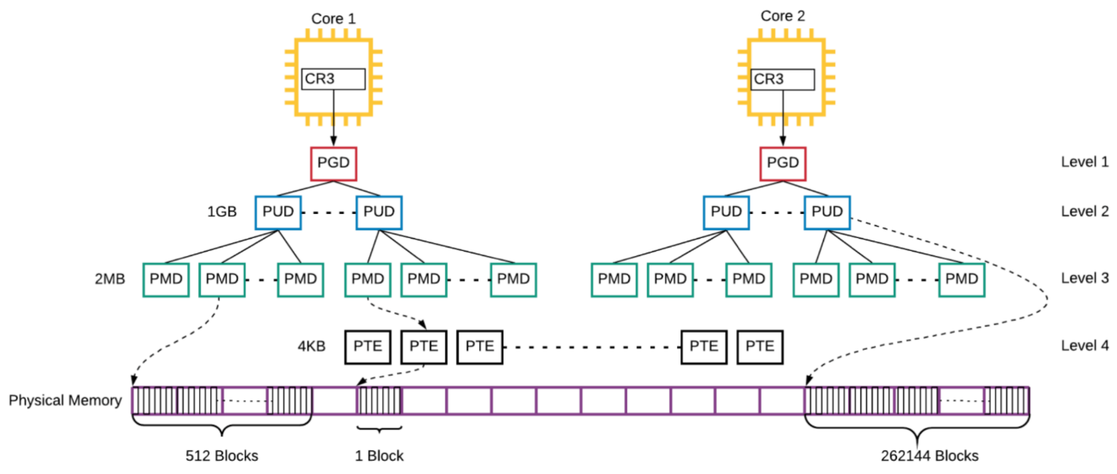


Figure 4: Virtual Memory Diagram

3) Service Deployment Mechanism

A service deployment mechanism has been implemented to facilitate the addition of more services to the NileOS micro-kernel. Service developers use special preprocessor tags in their C code to identify service methods that need kernel privileges or should be executed in user mode. The service deployment preprocessor automatically creates code to register these services with the necessary STL component.

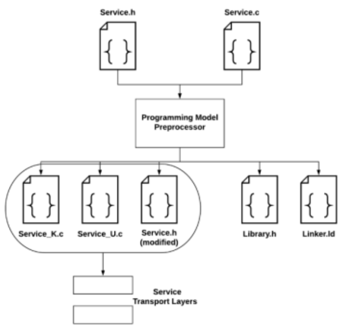


Figure 5: Service Deployment mechanism

4) Inter-Processor Interrupts over Ethernet (IPIoE)

NileOS uses a custom-built IPIoE (Inter-Processor Interrupts over Ethernet) protocol to enable cores on different nodes to communicate with each other. The protocol transports expandable BOSML messages as payloads and offers visibility by targeting particular cores instead of just nodes.

Information regarding the overall structure of the IPIoE header, as well as the headers unique to request, reply, broadcast, and gossip packets, is presented.

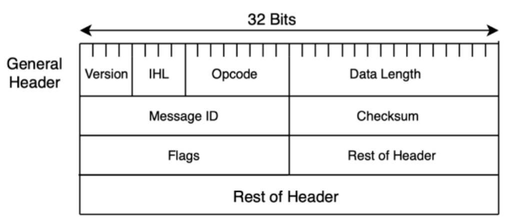


Figure 6: General IPIoE header

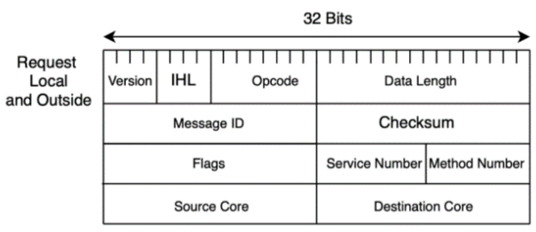


Figure 7: General IPIoE request header

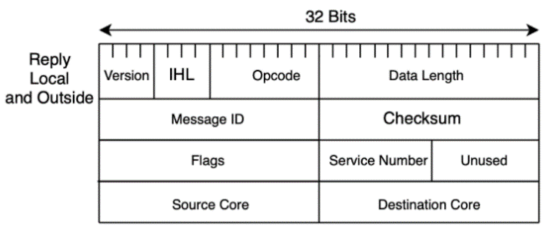


Figure 8: General IPIoE reply header

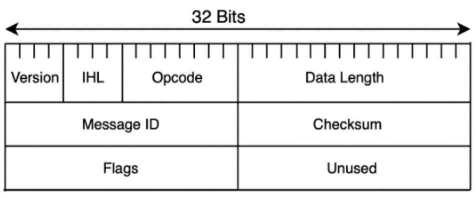


Figure 9: IPIoE broadcast header

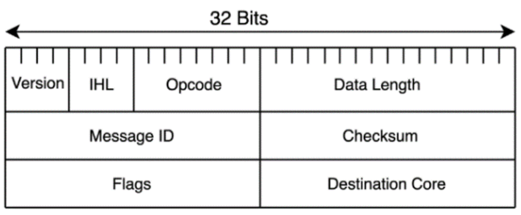


Figure 10: IPIoE Gossip header local and outside

5) Message Passing and BareMetal Operating System Markup Language (BOSML)

NileOS utilizes a communication method between services that involves a markup language named BOSML (Bare Metal Operating System Markup Language). Local cores communicate by sending BOSML messages through shared memory using local IPIs, whereas remote cores send BOSML data within the payload of IPIoE network packets. BOSML enables service methods to be called seamlessly, hiding actual physical core addresses. Services offer "Discovery Methods" which give schema details that allow other services to correctly refer to their methods.

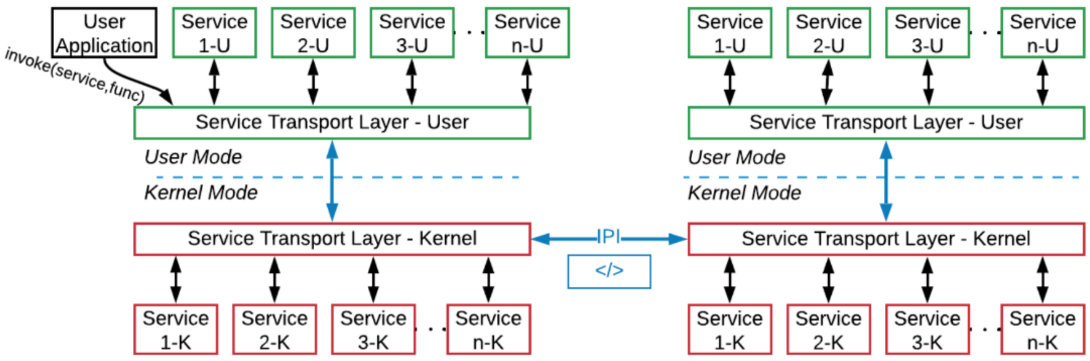


Figure 11: Local IPI communication

6) System communication flow

The overall communication flow for service invocation within a node, across nodes on the same LAN, and across different LANs is explained, showcasing the interaction between the different NileOS components like the STL, IPIs, IPIoE, and BOSML messaging.

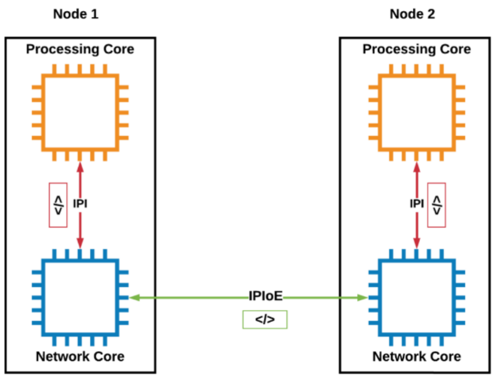


Figure 12: Remote communication

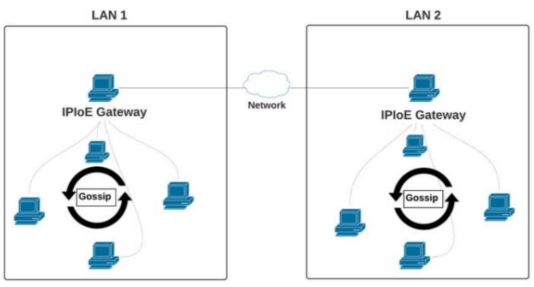
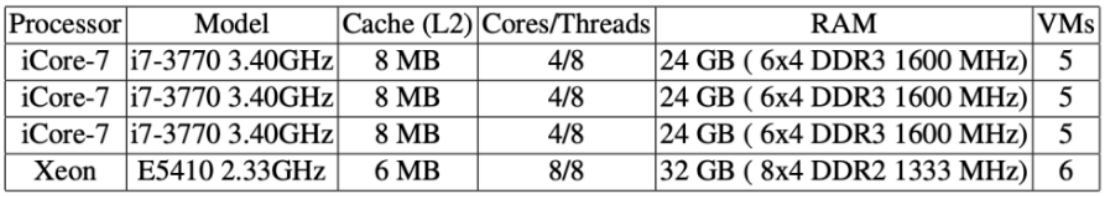


Figure 13: LAN to LAN communication

Evaluation and Outcomes

The researchers conducted experiments to evaluate the performance of their proposed AMP by comparing it’s TeraSort benchmark against traditional Hadoop big Data platform. The experiments were conducted on 4 hardware nodes with a total of 24 physical cores and 104GB of RAM. They created 21 virtual machines, each with 4 cores and 4GB RAM, totaling 84 virtual cores.



Processor	Model	Cache (L2)	Cores/Threads	RAM	VMs
iCore-7	i7-3770 3.40GHz	8 MB	4/8	24 GB (6x4 DDR3 1600 MHz)	5
iCore-7	i7-3770 3.40GHz	8 MB	4/8	24 GB (6x4 DDR3 1600 MHz)	5
iCore-7	i7-3770 3.40GHz	8 MB	4/8	24 GB (6x4 DDR3 1600 MHz)	5
Xeon	E5410 2.33GHz	6 MB	8/8	32 GB (8x4 DDR2 1333 MHz)	6

Figure 14: TeraSort environment configurations.

There are two identical Terasort environments with the same configurations. One environment ran Linux Debian Distribution with Hadoop and HDFS which represent the traditional big data processing environment using Hadoop while the other ran their proposed AMP micro-kernel. Each virtual machine in this environment has a dedicated management core with the timer enabled, a dedicated core for networking and disk I/O, and 2 worker cores, which sums up to a total of 40 workers. A C++ implementation of Terasort that integrates an implementation of the quick sort algorithm.

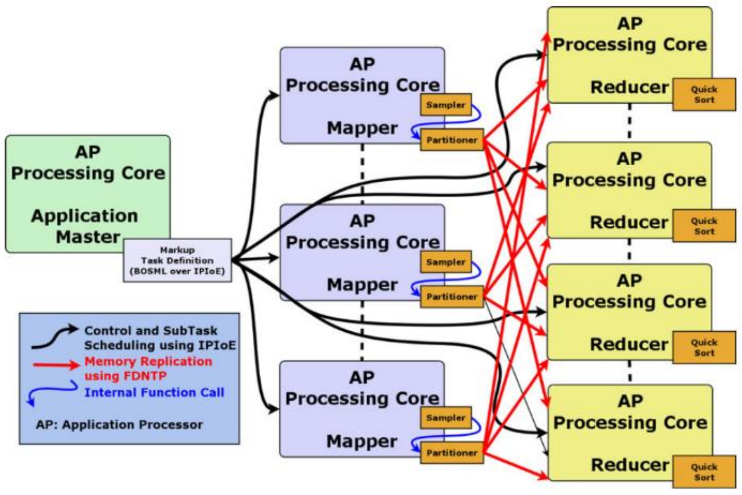


Figure 15: AMP Microkernel TeraSort environment.

Figure 15 depicts the implementation of the TeraSort algorithm on the proposed NileOS AMP. There are three main components,

1.Application Master: There is only a single instance of the Application Master running on a dedicated worker core. It is responsible for coordinating and managing the overall TeraSort execution.

2.Mappers: Multiple mapper instances can be launched, each running on a dedicated worker core. The number of mappers is configurable based on task configuration. In this experiment, the number of mappers was fixed to 20.

3.Reducers: Like mappers, multiple reducers can be launched, each on a dedicated worker core, based on the sort of task configuration. Number of reducers changes in each experiment run. (10 to 20)

The Application Master starts with issuing IPIoEs to startup mappers. It provides the tasks to the mappers and the number of reducers. Each mapper loads data from the disk storage and applies the sampling techniques and notifies the application master using IPIoE to start up the reducers. The shuffling phase begins, and the data partitions are transferred over the network using a FDNTP. Each reducer receives its subset data and runs a quicksort engine to sort the data and store the output on a disk.

They used TeraGen to generate 9 different input sizes: 2, 4, 8, 12, 16, 24, 28 and 32 GB and different number of reducers: 10, 12, 16, 18 and 20.

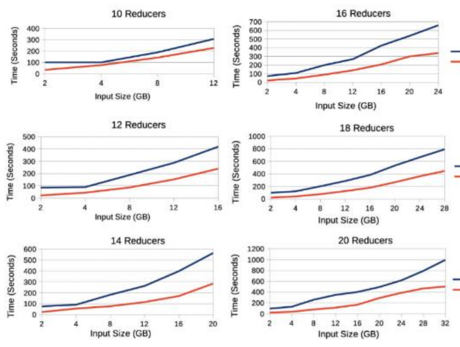


Figure 16: AMP Microkernel vs Linux Runtime (Reducers/Input size)

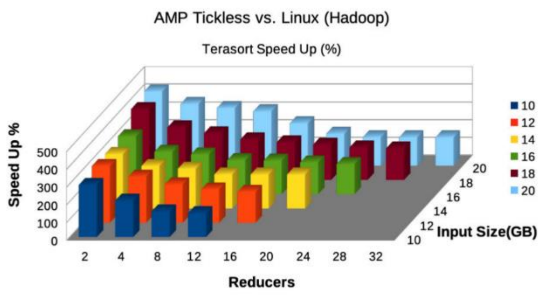


Figure 17: AMP Microkernel speed up

The speedup increased with more reducers but decreased with larger input sizes on the fixed hardware. The average speedup of the AMP micro-kernel over Linux/Hadoop is 234%; 2.34 folds.

They were able to isolate and measure the shuffling phase but could not isolate the network transfer time and data partitioning. The speed increased proportionally with the number of reducers. However, the speed started to decrease due to communication between a larger number of reducers, resulting in a higher packet loss rate.

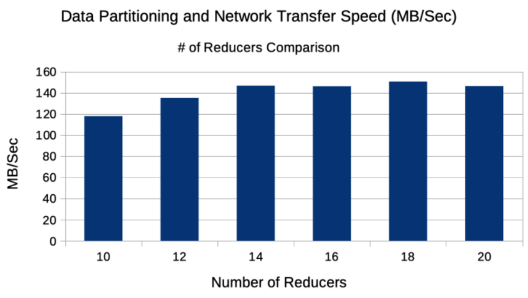


Figure 18: Shuffling speed

Researchers employ ANOVA approach, considering factors such as input data size, number of reducers, and execution environment. Each run in the experiment has fixed factor values and is executed 3 times to collect 3 readings. The ANOVA results table shows that all the target factors are significant as well as the interaction between the environment and the reducers, and the interaction between the input size and the reducers.

Interpretation and Final Remarks

This section explores the findings, implications and insights derived from the research conducted on the proposed micro-kernel system.

- **Memory Management Efficiency**: The discussion focuses on the importance of managing memory efficiently in big data applications. It highlights the drawbacks of using small 4 KB page frames in general-purpose operating systems and emphasizes the significance of using larger page frames like 1 GB and 2 MB in the OS to enhance performance and reduce memory fragmentation.

- **Page Frame Options**: NileOS offers the flexibility of choosing between different memory page frame sizes (1 GB, 2 MB, and 4 KB) based on the application's requirements. This approach aims to optimize memory utilization and avoid fragmentation by allocating memory in larger chunks, especially beneficial for big data applications that handle large datasets.

- **Abstraction Model**: Unlike traditional operating systems that maintain multiple address spaces for each process, This OS adopts an abstraction model based on cores with a single process. This design decision aims to smoothen the memory management and improve performance by simplifying the address space management within the system.

- **Scalability Considerations**: The discussion delves into the impact of input size on system scalability, highlighting the challenges posed by fixed physical resources when scaling virtual resources aggressively. It emphasizes the need to balance virtual and physical resources to maintain scalability without overburdening the underlying hardware.

- **ANOVA Analysis**: The discussion interprets the results of the ANOVA analysis, emphasizing the significant influence of input size on execution time. It clarifies that while the input size has a substantial impact on performance, the fixed physical resources used in the experiment may inflate this impact, affecting the perception of scalability.

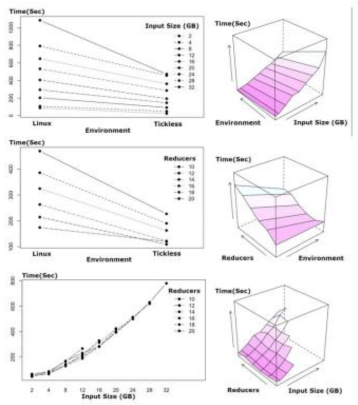
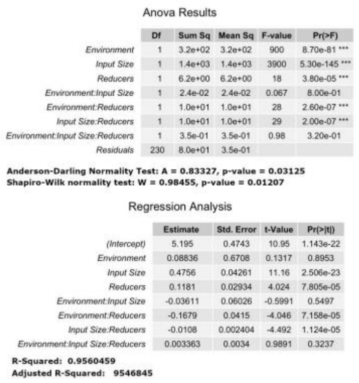


Figure 19: ANOVA factors interaction models



Anova Results

	Df	Sum Sq	Mean Sq	F-value	Pr(>F)
Environment	1	3.2e+02	3.2e+02	900	8.70e-81 ***
Input Size	1	1.4e+03	1.4e+03	3900	5.30e-145 ***
Reducers	1	6.2e+00	6.2e+00	18	3.80e-05 ***
Environment:Input Size	1	2.4e-02	2.4e-02	0.067	8.00e-01
Environment:Reducers	1	1.0e+01	1.0e+01	28	2.60e-07 ***
Input Size:Reducers	1	1.0e+01	1.0e+01	29	2.00e-07 ***
Environment:Input Size:Reducers	1	3.5e-01	3.5e-01	0.98	3.20e-01
Residuals	230	8.0e+01	3.5e-01		

Anderson-Darling Normality Test: A = 0.83327, p-value = 0.03125
Shapiro-Wilk normality test: W = 0.98455, p-value = 0.01207

Regression Analysis

	Estimate	Std. Error	t-Value	Pr(>|t|)
(Intercept)	5.195	0.4743	10.95	1.143e-22
Environment	0.08836	0.6708	0.1317	0.8953
Input Size	0.4756	0.04261	11.16	2.506e-23
Reducers	0.1181	0.02934	4.024	7.805e-05
Environment:Input Size	-0.03611	0.06026	-0.5991	0.5497
Environment:Reducers	-0.1679	0.0415	-4.046	7.158e-05
Input Size:Reducers	-0.0108	0.002404	-4.492	1.124e-05
Environment:Input Size:Reducers	0.003363	0.0034	0.9891	0.3237

R-Squared: 0.9560459
Adjusted R-Squared: 9546845

Figure 20: ANOVA results and regression analysis

- **Experimental Verification**: The discussion underscores the importance of experimental verification to validate the theoretical gains in memory management efficiency and scalability proposed by NileOS. It suggests that increasing physical hardware resources in proportion to the workload could enhance system performance and scalability.

Overall, the discussion section provides a comprehensive analysis of the research findings, highlighting the key aspects of memory management, scalability challenges, and the implications of the proposed micro-kernel design for big data processing.

Conclusion and Extensions

The research presented in this paper has demonstrated the potential of the NileOS micro-kernel in addressing the challenges of big data processing. The proposed design, which adopts an asymmetric core-based architecture, has shown promising results in terms of performance and scalability. The key design decisions, such as assigning dedicated roles to cores, minimizing interrupts on worker cores, and implementing efficient memory management strategies, have contributed to the overall effectiveness of NileOS.

The experimental evaluation using the TeraSort benchmark has revealed a significant performance gain of an average 2.34 folds over the Linux/Hadoop environment. The ANOVA analysis conducted in the study highlighted the substantial impact of input size on execution time, suggesting that the fixed physical resources used in the experiment may have inflated this effect. The authors suggest that increasing the physical hardware resources in proportion to the workload could help maintain the proportional gain in performance, as the system scales to handle larger datasets.

While the current implementation of NileOS has demonstrated its potential, the authors acknowledge the need for further research and development. Potential areas for future work include optimizing the IPIoE network protocol to enhance communication between nodes within the same LAN and across different LANs, as well as exploring the addition of new services to the micro-kernel to expand its capabilities and support evolving big data processing requirements. By addressing these challenges, the NileOS micro-kernel can continue to evolve and provide a more efficient and scalable solution for big data processing in distributed computing environments.

4. Appendices

Glossary

Asymmetric Multiprocessing (AMP): An architecture where each core is assigned a specific role, such as "Worker", "Management", or "I/O", rather than treating all cores equally as in Symmetric Multiprocessing (SMP).

BareMetal Operating System Markup Language (BOSML): An extensible markup language used for message passing and service invocation between different kernel images running on various cores, both locally and remotely.

Inter-Processor Interrupt (IPI): A special type of interrupt used in multi-core environments to allow cores to signal each other, enabling local communication and synchronization.

IPI over Ethernet (IPIoE): A specialized network protocol developed for NileOS to enable transparent communication and service invocation between remote cores over the network.

Service Transport Layer (STL): The core of the NileOS micro-kernel, responsible for registering new services, dispatching service method invocations, and facilitating communication between services.

Single System Image (SSI): A property of distributed systems where the distributed nature of resources is hidden from the user, making the system appear as a single, unified system.

Symmetric Multiprocessing (SMP): A traditional architecture where all cores are treated equally and can be used for scheduling any ready processes.

TeraSort: A distributed benchmark used to evaluate the performance of big data processing systems, involving sorting a large dataset.

Virtual Page Table: The data structure used to manage the virtual memory mapping between the virtual addresses used by processes and the physical memory addresses.

Gossip Protocol: A communication protocol utilized by the management core in NileOS to disseminate information across the core body of the environment, facilitating efficient information sharing and resource management.

Extensible Messages: Messages exchanged between cores in NileOS that can be extended and customized to carry various types of information, enhancing flexibility and interoperability within the system.

IPIoE Control Protocol: A specialized protocol used in NileOS for remote communication and synchronization between cores over Ethernet, ensuring efficient and transparent communication across nodes.

Service Transport Layer (STL): The core component of NileOS responsible for managing services, dispatching service method invocations, and facilitating communication between services within the micro-kernel.

Virtual Memory Service: A critical service in the OS responsible for building and managing the virtual page tables for each core, optimizing memory utilization, and enhancing performance in memory management.

Discovery Method: A method within services in the OS that allows other services to access and understand the schema of a particular service, enabling seamless integration and invocation of service methods.

Jumbo Packets: Larger network packets that can be used in NileOS to increase the payload size for exchanging BOSML messages, potentially enhancing communication efficiency within the system.

Inverted Page Tables: A memory management technique facilitated by NileOS to optimize memory allocation and reduce memory footprint by pre-allocating page tables and enhancing memory access speed.

Virtual Page Table (VPT): Data structure in virtual memory systems that maps virtual addresses to physical addresses. Each process has its own virtual address space, and the VPT maintains the mapping for each process.

Level 4 Page Table Entries (PTEs): In the context of x86-64 architecture the level 4-page table entries (PTEs) are the highest level of page table entries. They map virtual addresses to physical addresses.

5. References

[1] A. El-Rouby,A. Khalaf, A. Mostafa, F. Mohamed, N. Ghaly, A. El-Kadi, K. Sobh, "NileOS: A Distributed Asymmetric Core-Based Micro-Kernel for Big Data Processing," IEEE Access, vol. 9, pp. 3696-3711, 2021. Accessed: April 2024

[2] A. Silberschatz, P. B. Galvin, and G. Gagne, “Operating System Concepts, 9th ed.” Hoboken, NJ, USA: Wiley, 2012. Accessed: May 2024

[3] IsideBIGDATA. “The Exponential Growth of Data”. Accessed: May 2024. [Online]. Available: https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/