IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

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

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

AHMAD EL-ROUBY, ANDREW KHALAF, ARIG MOSTAFA, FADY MOHAMED, NOUR GHALY, AMR EL-KADI, (Senior Member, IEEE), AND KARIM SOBH
Department of Computer Science and Engineering, The American University in Cairo, New Cairo 11835, Egypt
Corresponding author: Karim Sobh (kmsobh@aucegypt.edu)

This work was supported in part by the American University in Cairo (AUC).

ABSTRACT Big Data applications have demanding expectations on computational resources front. Thus, general purpose operating systems are not a good fit. In this paper, we present a new special purpose distributed micro-kernel designed with big data applications' needs in mind. The new micro-kernel adopts a core-based Asymmetric Multiprocessing (AMP) approach. It optimizes interrupt management and I/O to suit the Map-Reduce model. The proposed micro-kernel design is based on Inter-processor Interrupts over Ethernet (IPIoE) frames and a BareMetal Operating System Markup Language (BOSML). A transparent deployment mechanism is presented to completely shield the developer of the micro-kernel service from the underlying distribution infrastructure and decouple the application implementation from its deployment perspective. Based on the initial prototype and the experiments presented, a considerable gain in performance of average 2.34 folds was achieved using the distributed TeraSort benchmark over Linux/Hadoop.

INDEX TERMS Asymmetric multiprocessing, big data, distributed micro-kernels, core-based, general-purpose operating systems, inter-processor interrupts, inter-processor interrupts over Ethernet, network protocols, markup languages, IPIOE, BOSML, micro-kernel, operating systems, tickless.

I. INTRODUCTION

Given the exponential growth of data, big data processing is increasingly becoming one of the most challenging problems in the computing domain [1]. General-purpose operating systems are not designed to cater for big data fundamental processing, data distribution and communication needs, namely scalability and performance [2]. They cannot accurately identify the workload profile of running processes through heuristics. Interrupting long batch processes unnecessarily introduces an overhead that significantly affects performance. Moreover, distributed processing over general-purpose operating systems entails a lot of network overhead due to the deep library stack upon which network protocols are implemented for standardization. In addition, the widely adopted SMP approach is not the best solution for less-interactive workload profiles, such as Big Data. This is due to the address space switching and poor cache management resulting from inaccurate affinity predictions and unnecessary interrupts.

Highly scalable Big Data processing frameworks like Map-Reduce depend on the concept of ''move-code-to-data''; so, data is initially distributed and processed by different cores based on locality and affinity [3]. Operating systems used in such environments are Linux, FreeBSD as well as other UNIX- based systems. Mainly designed with interactivity in mind, these systems do not provide the required resource allocation needs for Big Data processing [4].

When multi-core hardware became pervasive, moving to Symmetric Multiprocessing (SMP) was the fastest and easiest way to be able to support multi- core environments. Such approach allowed utilizing existing kernel designs with the least amendments possible. The SMP model is indeed of great benefit in the general use case but causes further performance degradation in the big data processing domain.

In this paper, we propose a distributed core-based asymmetric multiprocessing micro-kernel designed specifically for big data processing. The micro-kernel was essentially designed to be deployed on a cluster of commodity hardware. Different cores communicate through message passing and service invocation techniques. Local cores exchange messages using Inter-Processor Interrupts (IPI), which is a special type of interrupt used in multi-core environments allowing cores to signal each other. Unlike local cores which utilize shared memory to exchange messages, remote cores utilize a special network protocol, IPI over Ethernet (IPIoE). IPIoE

The associate editor coordinating the review of this manuscript and approving it for publication was Yinliang Xu.

allows cores to exchange remote messages over net- work in a transparent way. It was essential to have an extensible representation framework in place for different kernel images, running on different cores, to exchange messages. A BareMetal Operating System Markup Language (BOSML), was introduced as an extensible framework backbone for ex- changing messages. The proposed micro-kernel is a special- purpose one designed to extend the MapReduce [5] model for big data processing.

In section 2 we present the proposed solution design goals and decisions. Section 3 presents our proposed kernel architecture, different components, and detailed design. Previously conducted experiments and results are presented in section 4 followed by analysis and discussion in section 5. Finally, we conclude and present future work in section 6.

II. DESIGN GOALS AND DECISIONS

Our design goals and decisions target big data applications that adopt an execution profile with dependency on the distribution of data and avoiding moving huge amounts of data across the network. Essentially, we are targeting a MapReduce-like application profile that depends on the concept of ''move- code to-data''. The data is distributed and accessible by different cores and each core processes the data subject to its locality and affinity. [3]

A. DESIGN GOALS

- Design Goal 1: *Reducing overheads but not at the expense of extendibility*. Our design should set reducing overheads as a general objective. However, this should not translate to a monolithic design of the kernel. Monolithic designs do not emphasize extendibility, and the overhead of message passing is negligible compared to big data processing. Instead, a micro-kernel approach is adopted.
- Design Goal 2: *Minimizing interrupts on data processing tasks or eliminating them when possible*. Since our micro-kernel targets optimizing big data processing, it is designed to minimize interrupts on tasks performing long processing of big data. Eliminating all interrupts might not be feasible to allow synchronization between tasks. However, as a general design goal, unnecessary interrupts should be eliminated. This will reduce memory overhead through adopting a single address space approach per core which will also enhance the cache and the TLB effect. In a time-sharing operating system, preemptive scheduling allows user programs to share computer resources with minimized response-time. On the contrary, batch processing jobs would greatly benefit by being assigned to uninterruptible CPUs unless deemed necessary [6]. The timer introduces multiple challenges for long batch processing. While the process is running an interrupt is fired periodically hence delaying the CPU and introducing overheads. The overheads can be categorized into two; power and performance. The power overhead is represented by the dynamic power waste entailed by the use of the ''always switching'' device to implement the periodic timer for the scheduler. The power overhead is considered a major concern in hardware [7]. Performance overhead is introduced due to the frequent unnecessary interrupts of long running batch processes. When a process is interrupted a switch in the address space occurs causing the Table Lookaside Buffer cache (TLB) to flush which significantly slows down the processing. Such symptoms are referred to as system noise which can cause up to two-thirds of the slowdown of an application [8]. It has also been proven that system noise is a major hurdle when it comes to scalability [9]. The overhead incurred directly from the timer is relatively small, however, the indirect overhead is significantly large. An equation to quantify such noise was introduced by Tsafrir *et al.* in [8] and called it the ''noise law''. It was concluded that despite limiting the ticks by the timer improving the situation it does not help significantly especially in large clusters. It is established that the optimal solution is eliminating the ticks altogether.
- Design Goal 3: *A CPU Core is the fundamental building block of the system*. Overcoming the bottlenecks of symmetric multiprocessing's schedulers, our system should be designed with the core being the building block of the system. An asymmetrical multiprocessing model allows the kernel to carry out different dedicated tasks on specific cores. There are two types of kernel architectures targeting multiprocessing: Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP). In the SMP architecture, all cores running the same kernel image, are treated equally, and can be used for scheduling any of the ready processes. On the other hand, in the AMP architecture, each core gets a predefined task and may even get a different kernel image. The difference between SMP and AMP stems from either differences in the hardware core or the software image that the core is running [10].
- Design Goal 4: *A single system image of a distributed system*. Dealing with separate cores in a distributed environment can be challenging for the user. It also requires the target big data application to be heavily tailored to make use of the distribution. This problem exposes the need to have resource location transparency and resource access transparency mechanisms within the system. The application business logic should not be aware of the distribution of the workload across remote nodes. Distributed operating systems are responsible for making distributed resources appear as if they are local to the users and their applications [11]. This gives the system the property of having a Single System Image (SSI) as it hides the distributed nature of the available resources and achieves high levels of transparency [12].
- Design Goal 5: *Ability to extend horizontally by adding more nodes with minimum effort*. The distributed

micro-kernel should run on many nodes that host several resources, especially the cores. Since the micro- kernel is targeting big data that could require additional resources, the design should provide seamless mechanisms to scale through adding more nodes and discover their resources transparently and efficiently.

- Design Goal 6: *Ability to assign a specific task to a specific core transparently, whether remote or local*. Since the core is the building block of the system, each core should be addressable and be assigned a specific task to carry out. This core could reside locally to the application or could be remote. The system should be able to provide the semantics of location transparency.
- Design Goal 7: *A programming model should be in place to add services to the kernel without the need to understand the underlying architecture by the micro-kernel service developer*. A micro-kernel by nature provides a set of services that are required for resource management through sending messages. Most services must run in user mode. Also, services should be easily added to the micro-kernel. Since some functionality of the service might require privileged access to kernel resources and data structures, the functions within the service must be divided into privileged functions and user-mode functions. Since transparency is emphasized as a design goal, there exists another consequent goal to have a programming model that allows embedding a service into the kernel transparently. The target should always be to avoid switching to kernel mode except when needed. The service developer should have minimum insight about the underlying micro-kernel architecture.

B. DESIGN DECISIONS

In order to achieve the previously mentioned design goals, firm primary design decisions need to be taken at an early stage; other secondary decisions are still under investigation.

- Design Decision 1: *Each core within a node has its own kernel image and its own address space*. This is a design approach that could enable us to achieve design goals 2 and 3. Looking closely at one node, each core within that node has some assigned functionality that it should perform. For example, the bootstrap's kernel image might build up the memory virtual page table for itself and the remaining of the cores before even starting them up. This also addresses the point of reducing overheads in the sense that a core's kernel image might be responsible for handling a certain event while another core is processing a task. Thus, an overhead of the unnecessary disruption on the processing task is eliminated. Moreover, single address space per core will eliminate the overheads entailed during interrupt handling.
- Design Decision 2: *Within a node, cores are assigned roles*. Cores are divided into 3 groups: ''Worker'' cores, ''Management'' cores, and I/O cores. A role of a ''Worker'' entitles the core to process big data intensive tasks. A ''Management'' role entitles the core to manage the resources of the node, such as scheduling and virtual memory. A role of ''I/O'' entitles the core to manage I/O devices such as mass storage and network.
- Design Decision 3: *Disable timer interrupts on worker cores, except when needed and route the timer interrupts to the management core*. At least one core per node needs to have the timer enabled. In support of design goals 1 and 2, our design aims to emphasize reducing the interruptions throughout the system, especially on the worker cores. One major source of interruptions is the periodic timer interrupt which is routed to all cores in general-purpose operating systems. Such timer interrupts are important for scheduling but for the case of worker cores they pose a source of unnecessary interruptions. Therefore, the timer interrupts will be disabled on the worker cores and will only be routed to some of the management cores by enabling the timer, through configuring their local APIC timer.
- Design Decision 4: *Implement the functionality of the Network Management role, Disk I/O Management role, Memory Management role, and Worker role as the bare basic functionality needed for the micro-kernel to be able to perform and be of use*. As we decided to adopt a micro-kernel architecture, there are some roles that must be implemented to support the basic functionality of the micro-kernel. The worker role functionality must be implemented. Another functionality is managing the network component of the system which is crucial to achieve a distributed environment. Moreover, Disk I/O is essential to be able to read the big data files subject to processing from disk.
- Design Decision 5: *Cores synchronize and communicate using Inter-processor Interrupts (IPIs) with location transparency*. Remote communication and synchronization are done using a special IPI over Ethernet protocol (IPIoE). In order to support synchronization and communication between the cores, the micro-kernel must implement mechanisms to utilize Inter-processor Interrupts (IPIs). IPIs could happen either locally or remotely; a core should be able to send an IPI to another core on a remote node. For the case of local IPIs, the core will save any exchanged data in a shared memory region and send an interrupt to the designated core signaling the core to perform a task. This should be the main infrastructure for micro-kernel service invocation across cores. Knowing that the core is the building block of the system and that IPI could happen across remote nodes, the remote IPI signal should include the service invocation parameters. This provokes the need to have a new protocol that transmits the data packets as an extensible message as well as the signal. The protocol is referred to as IPI over Ethernet (IPIoE). The remote IPI should be transparent to the issuing core and the receiving core.

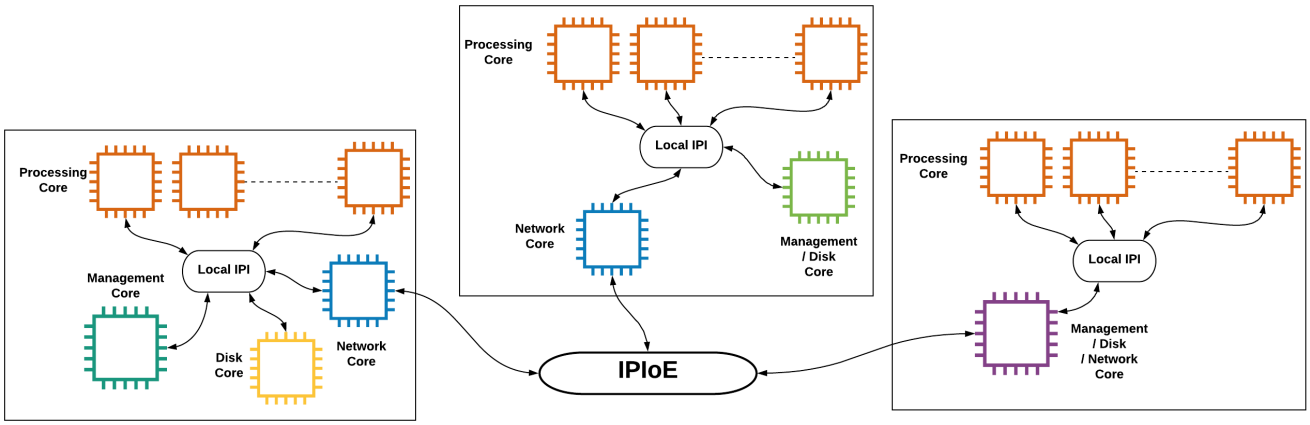


FIGURE 1. Overall NileOS view.

- *D*esign Decision 6: *The Virtual Page Table's level 4 (PTEs) is shared between all cores*. Knowing that a single node might have many cores, having a completely separate virtual table per process needs a gigantic amount of memory to store. However, design decision 3 gave us room for optimization. Each worker core is bound to have a single process without tick interrupts, thus no context switching happens and no swapping of virtual tables takes place. This allows us to have a single virtual table per core. Yet, the memory problem associated with having a huge number of cores on a single node is still not solved. Thus, we utilize a virtual table design similar to that of an inverted page table for the purpose of saving memory [13]. This is achieved by allocating (at startup) page frames necessary for page table level 4 based on the available physical memory size.

III. PROPOSED SOLUTION: NileOS

Having discussed the problem at stake in addition to some of the partial solutions to some aspects of the problem, and the design goals and decisions needed, we would like to present our proposed solution. A new micro-kernel is proposed to target long batch processing big data applications specifically and still provide other means of interactivity on demand. It will trade off the generality that bottleneck heavy batch processing have in favor of optimizing the processing of big data. Our solution will be implemented on general-purpose hardware present in most computers; Intel-based x86_64 architecture. In this section, we will start by presenting a high-level overview of the overall architecture of the proposed micro-kernel and its different components. A detailed explanation of each component will be presented as well as its interaction with other components in the architecture.

A. SYSTEM OVERVIEW

Our system is a distributed system that consists of multiple nodes. Each node is built up of general purpose x86_64 hardware architecture. Nodes are assumed to be connected via a high-speed network. From a single core perspective, the remote resources are made available through location transparency, and thus, remote resources are perceived as local. Applications are not meant to be aware of the underlying hardware distribution and instead use the system as a single unified computing power. Cores can invoke micro-kernel service methods transparently by name utilizing location transparency that is inherently supported by our proposed design.

For the system to function, each core provides a set of services which defines its role. By having roles through dedicated services, an asymmetric multiprocessing model is achieved. The exact roles, services, and resources to be utilized by a node are determined by a configuration file. Currently, our system has 4 different core roles that can be extended in the future: management, network, disk and application. The management cores handle all the system management non-processing functionality, such as scheduling and housekeeping tasks. Network cores handle all the network operations such as sending and receiving packets. Disk cores handle all I/O operations to and from the disk. The application cores are meant to handle the big data processing tasks. The micro-kernel tries to reduce the overhead on such application cores by minimizing interrupts. For example, the recurring timer interrupt is disabled on the application core.

Figure 1 demonstrates the overall architecture of the proposed system. Each node has multiple cores. Cores within each node communicate over local IPIs. Cores of different nodes communicate over IPIoE through each node's network core. A gossip protocol is in place utilizing the IPIoE control protocol to disseminate information across the whole core body of the environment. IPIoE broadcast is avoided and only used in special cases when needed. Communication among all cores is realized via extensible messages. Local cores can exchange extensible messages using local IPIs and Bare Metal Operating System Markup Language (BOSML) over shared memory. Each node is responsible for managing its own resources, but it needs to be aware of the available resources in the whole system through some directory

of services. For remote communication, such as invoking a remote functionality, BOSML messages are encapsulated in IPIoE packets. Our system allows more services to be added to the cores easily using a transparent service deployment mechanism such that service developers do not need to know about the underlying architecture of the micro-kernel.

B. THE MICRO-KERNEL BACKBONE

The architecture of our distributed operating system was designed with extensibility in mind. Kernel developers can write their own services that could easily be plugged into the micro-kernel with minimal effort. Location transparency of any running service in any given node at any time is a priority. To achieve this goal, as described in Figure 2, we decided to implement the core of our kernel, which shall be referred to as Service Transport Layer (STL), to act as the backbone of the micro-kernel to register new services as well as propagate incoming requests to registered services. Each node should have an STL. The STL is responsible for dispatching a specific function from within a service when needed. Both the service and the function are invoked by name. The STL is divided into 2 main sub layers. The lower layer runs in the kernel space and is able to execute privileged functions, while the upper layer runs in the user space and acts as a dispatcher for user mode service invocations.

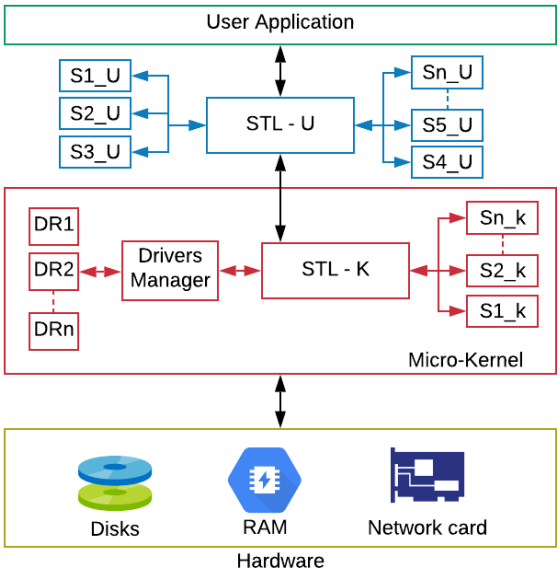


FIGURE 2. NileOS Kernel Architecture.

To further elaborate, the STL is divided into two main parts, as described in Figure 3. The first part is the kernel STL or STL_K which is responsible for dispatching methods related to services that need to be run in the kernel mode. On the other hand, we have the STL_U which also allows services that don't need kernel privileges and user applications to communicate together. The STL is designed as an interface that has the capability of dispatching service methods on behalf of a service invocation from a user application. For example, if Service X needs to dispatch method y that belongs to Service Y where both are running in kernel mode, then X dispatches the method through the STL_K which takes both the method and the service identifiers as arguments and calls the corresponding function. After the successful execution of this method, the result is then returned to the calling service through the STL_K. Another example would be if a user application requires extra memory from the heap manager which would be a service plugged into the STL_U, the request is delivered to the heap manager through the STL_U after which the heap manager decides if it needs kernel privileges or not. If the service decides that there is no need for such privileges, then the result is returned to the user application. However, if there is a need to use these privileges, to map physical frames into the page table for instance, there should be a mirrored service that has the capability to do just that. Accordingly, we decided to offer the kernel developers the opportunity to create both versions of the service; one that works in user mode and another that works in kernel mode.

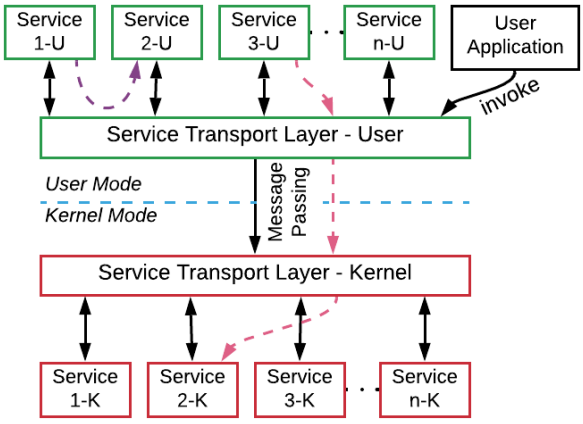


FIGURE 3. Service transport layer communication.

STL runs on each core and therefore allows services running on different cores to communicate easily through IPIs which offer location transparency mentioned earlier. Also, in the case of communication between services running on different nodes, STL has the ability to identify the location of the remote services then send an IPIoE. The IPIoE packet contains an extensible message that could help identify the origin of the packet and information about which method needs to run together with the passed parameters. Moreover, new user services can be added to the micro-kernel and registered with the STL using an extensible service deployment mechanism. The proposed micro-kernel comes with a set of required services that are needed for the micro-kernel to be able to function and be of use. Such services are registered with the service transport layer. Extra services can be added by developers upon the need. The default essential services are:

- *P*hysical Memory Service: The physical memory service is deployed on a memory management core. It abstracts any interactions related to the physical memory. Different cores might request from the physical memory

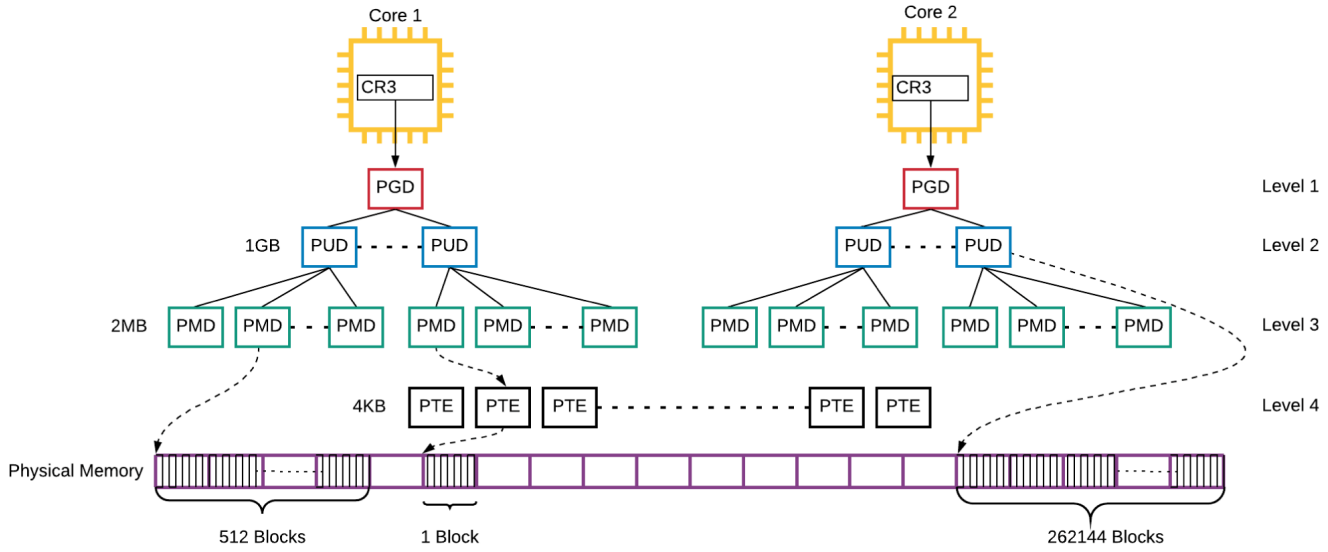


FIGURE 4. Virtual memory diagram.

service to allocate or deallocate physical frames. The service allows other services to get more information about the physical memory in a given node including all of the memory regions present as well as the total usable memory. Moreover, the physical memory service monitors the usage of memory which allows other services to easily request or release memory frames.

- *V*irtual Memory Service: The virtual memory service is responsible for building the virtual page table for each core. Figure 4 illustrates the x86_64 PML4 page table architecture created by the service for each core and highlights the overlapping shared level-4 page tables [13]. This service first creates Level-4 virtual page tables according to the usable physical memory available in a given node provided by the physical memory service. These tables are statically allocated at the initialization time of the kernel to save up future time and space. It is faster to have the tables created in advance without any need to allocate them during run time. On the other hand, statically allocating space for each of the virtual tables belonging to each core in a given node would require a huge amount of memory. Therefore, we decided to have a pool of level-4 virtual memory tables that would be shared among all cores. Since each of these tables can map the same amount of physical memory it was apparent that we wouldn't need more tables than that. Therefore, the virtual memory service is responsible for creating the first 3 levels of the virtual memory page table for each core, linking them together, and just leaving the last level unmapped.
  A target core can issue a request to map a memory frame and consequently the virtual memory service will perform a page walk to identify the level-3 entry corresponding to the requested virtual address. If the level-3 entry is already connected to a level-4 table then the entry corresponding to the virtual address at level-4 will be identified and used, else a free table from the shared level-4 will be selected and connected to the level-3 entry to be used. A free physical frame will be requested from the physical memory service and linked to the level-4 entry corresponding to the target virtual address.
  Our design closely borrows some design aspects from inverted page tables [13] and the memory needed for storing the virtual tables is efficiently reduced. Our design also has the privilege of allowing to map 2MB pages from level-3 directly and bypassing pre-allocated level-4 tables at run time. This allows for better physical memory utilization and less fragmentation based on the target application running on the core which can be controlled flexibly through the service parameters. The design can be extended to map 1GB pages directly following the same technique with some minor modifications.
- *N*etwork Service: This service is responsible for handling all the network-related tasks. It should be able to communicate with remote nodes. Network protocols can be selectively assigned to specific network cores. For example, inter-Processor Interrupts over Ethernet (IPIoE).
- *D*isk I/O Service: One other service that our micro-kernel should provide is Disk management. This service will be able to discover the available disks, configure them and be able to read and write to them using the ATA transfer mode with Direct Memory Access DMA support.
- *G*ossip Service: This service is meant to allow a local node to discover what resources other nodes are exporting to the distributed environment. The gossip service is utilized by the management core, it is responsible

for sending gossip messages, receiving gossip messages and forwarding them to the registry service as well as relaying them to other nodes through gossiping further. To eliminate network contention and overhead, not everything is being broadcasted, rather only important events that inform registry services about the availability of resources. The gossip service is used mainly to minimize network traffic. A gossip message is sent based on some predefined criteria. One such criteria could be when two cores communicate; when a core invokes a service in another core the gossip service sends a gossip message updating it with some information about other cores and services that it knows about. Initially, each node broadcasts its cores capabilities once upon start and incremental updates are performed over the gossip protocol. The gossip service utilizes the IPIoE protocol, introduced in subsection III-D, and the payload of the packet that encapsulates the gossip message is an extensible one encoded using BOSML [14].

C. SERVICE DEPLOYMENT MECHANISM

A micro-kernel by definition provides a minimal set of services that are necessary for process and memory management. More services need to be added easily. Additionally, services need to be linked with the service transport layer. For that reason, the service deployment mechanism illustrated in Figure 5 has been introduced. Service developers write a header file alongside a C file and place them in a special directory. Special preprocessor tags are being used to annotate service methods. Developers annotate methods that require kernel privilege using the "STL_K" tag and those that should run in user mode with "STL_U". The service deployment preprocessor runs before compiling the micro-kernel sources. It searches for the special tags and creates 2 new C files, one containing the methods that should run in kernel mode and the other contains those which should run in user mode. It also creates routines for registering the services with the service transport layer. This allows services and methods to be called by name using a unified API. This can be extended in the future to be performed at run time through decoupling the compilation of the services from that of the micro-kernel and having the objects ready to be loaded at run time.

D. IPIoE PROTOCOL

Considering that what slows down the performance at the network level is the generalized deep network stack and application-level file transfer, we have decided to have a specially designed control protocol. While designing the protocol we have kept in mind that the building block of our distributed system is the core and we wanted to push the distribution semantics to the kernel level. This way we can reduce the overheads of having the distribution semantics at the application level and avoid the deep network stack. Hence, the need to have a protocol to address a specific core not a node is apparent. This protocol is not meant for large data transfer. However, it carries extensible messages as a mean

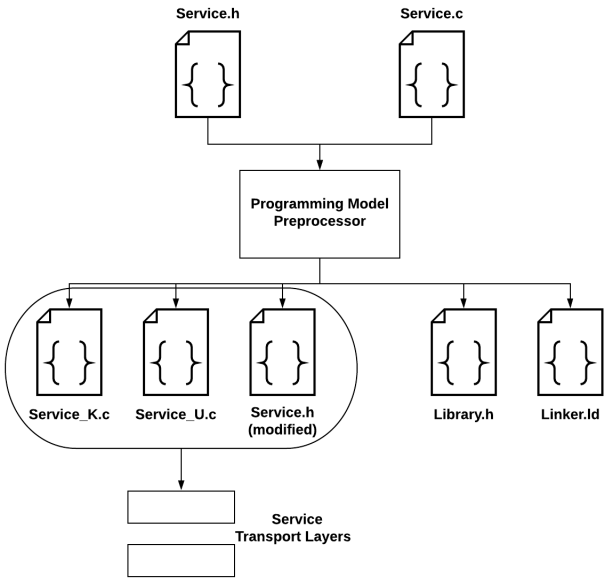


FIGURE 5. Service deployment mechanism.

for message passing between services. The IPIoE protocol is considered as an extension to local IPI. It gives the benefit of transmitting IPIs over the network transparently.

The headers of the IPIoE can differ depending on the operation requested by the packet sender. However, there are common attributes among all packet types which were placed at a fixed position for standardization purposes. As per Figure 6, the common attributes include the version number which is very important in the future to avoid any sort of erroneous communication resulting from protocol version incompatibilities. To support having dynamic headers for the network packets, the header length is included and depends solely on the header type. The opcode attribute refers to the requested operation and currently includes Request, Reply, Broadcast Locally, and Gossip operations. This field has the potential to add up to 256 different operations, including the ones stated above, in upcoming updates for any future usages. The header also contains the data length to avoid having fixed size packets and wasting storage resources. To ensure the reliability of the protocol, Message ID and Checksum fields were included to avoid having corrupted packets. Finally, the flags field, which is in the form of a bitmap, currently includes encryption and encoding flags and can support up to 16 flags in total; a single flag is represented by a bit.

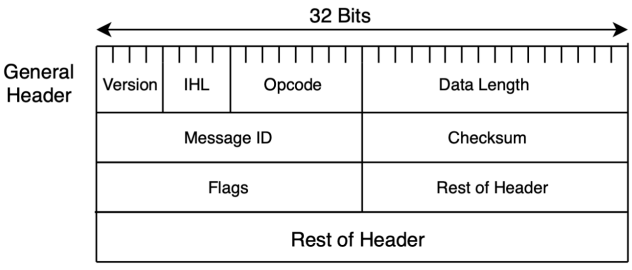


FIGURE 6. General IPIoE header.

One of the headers supported by the IPIoE protocol is the request header presented in Figure 7. The request header contains the same information as the general header in addition to the service and method numbers the requester wishes to invoke remotely. Moreover, the source and destination core identifiers are appended to the header since the main communicating entities targeted by this protocol are the cores.

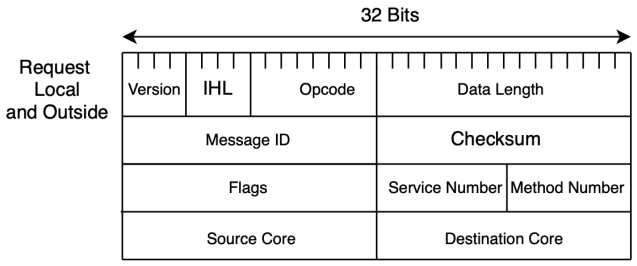


FIGURE 7. General IPIoE request header.

On the other hand, the reply header presented in Figure 8 is exactly similar to the request header except for the method number since it is merely a reply and does not need to be invoked.

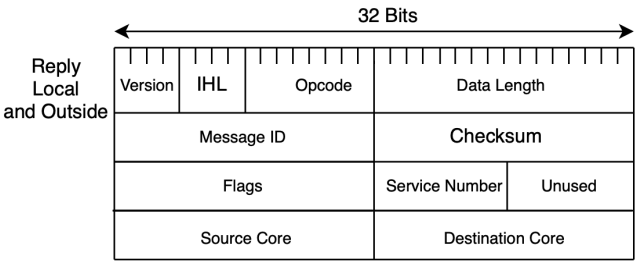


FIGURE 8. General IPIoE reply header.

The broadcast header presented in Figure 9 is the same as the general header since it is only sent when the node boots up. It only transmits the packet to all nodes in the network so the only field needed is the opcode to instruct other nodes to add the sender to their tables. The packet may contain some extra information to be used for polling further updates about the sender node and its cores. After the first broadcast message is sent, further changes are sent via gossip packets that are directed to certain nodes and destination cores.

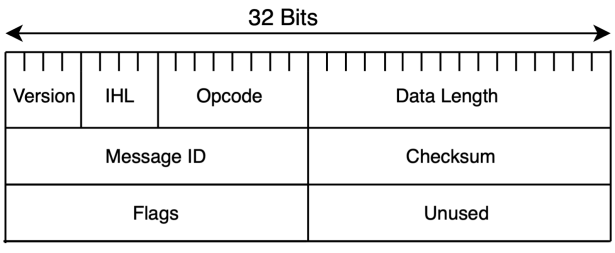


FIGURE 9. IPIoE broadcast header.

The gossip header type presented in Figure 10 is used by gossip packets which transmit information to some destination core. The gossip header includes only the destination core, and there is no need for the source core and the service fields as there is no service to invoke. Any extra service specific information that needs to be transmitted by any packet will be encapsulated in the payload encoded in the form of BOSML.

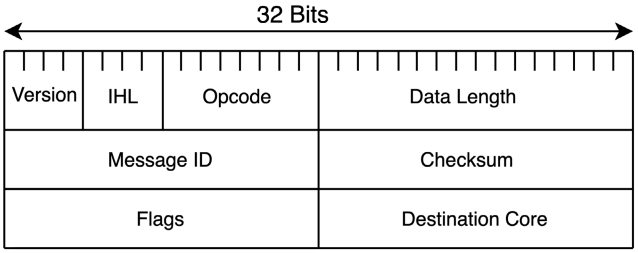


FIGURE 10. IPIoE Gossip header local and outside.

E. MESSAGE PASSING AND BARE METAL OPERATING SYSTEM MARKUP LANGUAGE (BOSML)

One of our important design goals is to allow service methods to be invoked in an extensible and transparent manner abstracting all information about the physical location of the cores hosting the target service. To acheive that we have considered a message passing mechanism between all services and their methods. This mechanism allows for integrating services seamlessly. In case the requested service is on the local node, a shared memory region is allocated in the virtual tables of the communicating cores and the message is stored in that shared memory. However, if the requested service resides on a remote node, the message is encapsulated in the payload of an IPIoE packet and sent to the target core.

BOSML is a markup language that allows for the extensibility and interoperability of our services. All messages exchanged between services are in BOSML, giving service developers the freedom to invoke the methods with minimal restrictions. XML messages are used regardless of the physical location of services, whether local or remote. Additionally, a service needs to define a schema that other services will use to invoke its methods. Any deployed service needs to expose a method called the Discovery Method which allows for other services to know its schema. In Figure 11, an example of the output of the Discovery Method is shown where the root tag is the service name followed by its methods' parameters and information about each parameter, whether it should be encoded, encrypted, its type and size, etc. allowing for maximum extensibility and portability.

```
< Ackermann's Function Service >
  < Call Ackhermann's Function >
    < Parameters >
      < FirstParameter >
        < Type > uint_64 </ Type > < Encrypted > uint_64 </ Encrypted >
        < Size >    8    </ Size > < Encoded >   uint_64 </ Encoded >
      </ FirstParameter >
    </ Parameters >
  </ Call Ackhermann's Function >
</ Ackermann's Function Service >
```

FIGURE 11. Discovery function output.

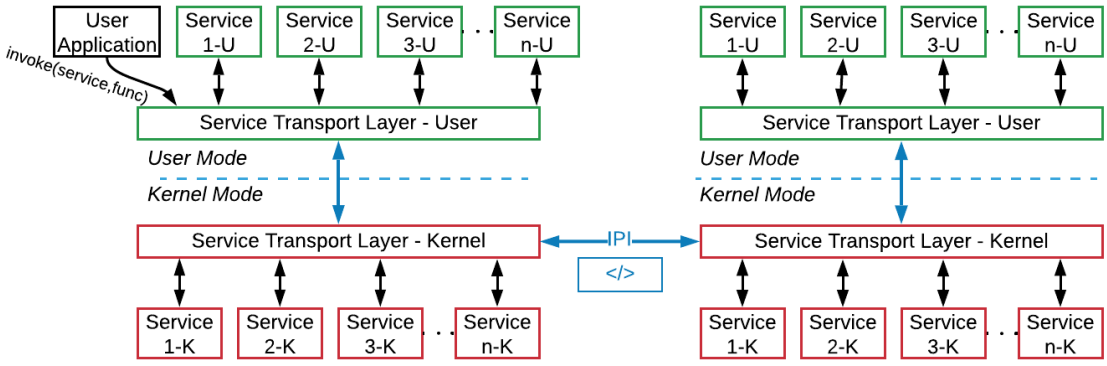


FIGURE 12. Local IPI communication.

F. SYSTEM COMMUNICATION FLOW

In this section the overall system communication flow will be presented and how different components of the architecture communicate and interact together. When a user level application aims to invoke a method within a service, it initiates a request to the invoke API of the STL segment running in the user mode (STL_U). The user passes the service name and the method desired. If the needed service runs in the user mode and there is no need for kernel privileges, the STL will dispatch the service method in the user mode. If a return value is expected, then the service will return it to the STL which will relay it to the user application. If the needed service, or part of it, requires running in privileged mode and needs the kernel, the STL in the user mode will relay the request to the STL in the kernel mode (STL_K). The STL_K then acts similar to the STL_U as explained earlier. It is worth mentioning that services can call each other. Having a part of the service running in user mode prevents unnecessary context switching to the kernel mode when the needed service can fulfil its target while in user mode.

Service invocation can take place across different cores within the same node through utilizing local IPI between cores within the same node as illustrated in Figure 12. The STL is running on each core and therefore allows services running on different cores to communicate easily and realize location transparency. It is essential in that case to switch to kernel mode to be able to send an IPI to the target core which will entail triggering the STL dispatch service routines on that target core. Prior to sending the IPI all the parameters need to be encapsulated within a BOSML shared buffer that both cores can access.

In the case of service method invocation between cores located in two different nodes, the STL has the ability to identify the location of the target service. It then sends an IPIoE with information about the origin of the packet and the target service and method that needs to be invoked. All the service parameters will need to be compiled into BOSML and encapsulated within the payload of the IPIoE packet. The network core will essentially need to be consulted for sending the packet, as illustrated in Figure 13, to carry out the network communication. The target service will eventually execute on

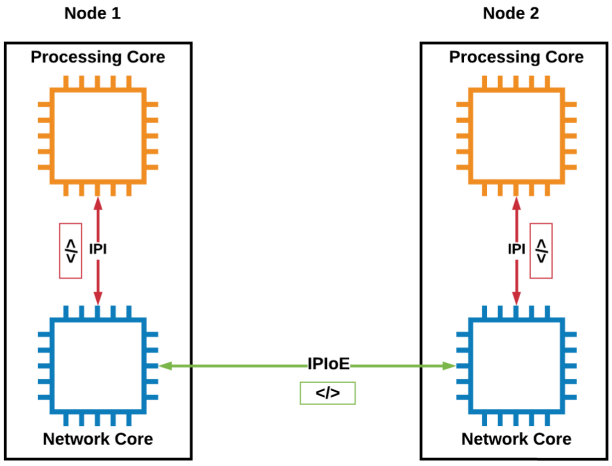


FIGURE 13. Remote communication.

the remote node/core and the reply to the calling core/node will take place over another IPIoE packet.

So far, we have covered the communication flow within the same LAN. One important case that ensures maximum scalability is service method invocation across LANs; source and destination cores are in two different nodes in two different LANs. The most famous and widely used standard network addressing scheme is the IP addressing built on top of the IP routing protocol. IPIoE is a custom control protocol that cannot be used for routing purposes across different networks. To be able to extend our design across different LANs an IPIoE to IP translator needs to exist within each LAN as illustrated in Figure 14; an IPIoE gateway for masquerading packets from IPIoE to IP and vice versa. When a node/core in a LAN attempts to communicate with another node/core in a different LAN, it sends an IPIoE to the LAN's gateway. The gateway then encapsulates the IPIoE packet into an IP packet. The gateway's IP address is used as the source IP in the IP packet while the destination network gateway is added as the destination IP. When the IP packet reaches the destination gateway, the gateway extracts the IPIoE packet from its IP payload. The IPIoE packet is then sent to the designated node/core over the destination local LAN. A gateway needs to be able to address the other gateways for

remote communication outside the same LAN via IP. This requires some sort of a registry service or a service directory that is well known by all gateways. Once the gateway of a specific LAN is up, it contacts that service directory and updates it with information about its status, its resources, and the services it exposes. In exchange, the gateway gets to know similar information about the other gateways available. The gateway directory service should be conceptually a centralized service deployed over redundant scalable underlying infrastructure for maximum scalability, availability, and fault-tolerance.

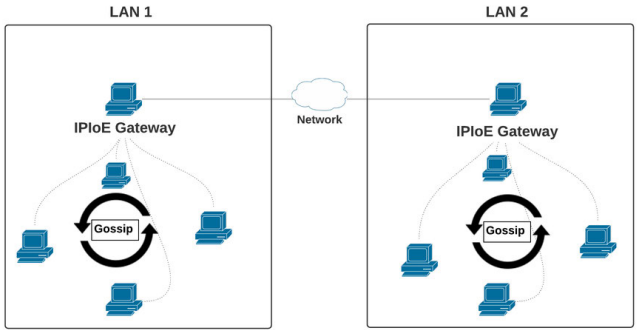


FIGURE 14. LAN to LAN communication.

IV. EXPERIMENTS

In this section, for completion purposes, we are presenting several experiments to assess the performance of a primitive prototype implementation that were conducted and presented earlier in [2]. The promising results were the main drive behind the most recent extensions of the micro-kernel design.

The experiments were designed to test the end-to-end run time of a real big data standard distributed application, including disk I/O and network transfer. They aimed at comparing the Terasort benchmark [15], [16] run time of the traditional Hadoop Big Data platform and our proposed AMP micro-kernel deployed over a distributed environment. We used a deployment environment of 4 hardware nodes as per the table in figure 15.

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 15. TeraSort environment configuration.

The whole environment has a total of 24 physical cores, 16 of which supports hyper-threading resulting in 32 virtual core threads, and a total memory of 104 GB of RAM. We used the VirtualBox hypervisor to instantiate 21 virtual machines, each of which has 4 cores and 4 GB of RAM. The total number of virtual cores perceived by the virtual machines is 84 cores, which is much more than the number of physical CPUs/Cores.

We have set up 2 different environments with the same configuration as above. The first environment runs the Linux Debian Distribution with Hadoop and HDFS [17]–[20] installed and configured to run Hadoop MapReduce [5] implementation of TeraSort. The Second environment runs our proposed AMP distributed micro-kernel. Each virtual machine 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 is used to run the TeraSort benchmark on the proposed micro-kernel environment.

Figure 16 demonstrates our implementation of the TeraSort algorithm which is built up of three components: Application Master, Mapper, and Reducer. There is only one instance of the application master and it runs on a dedicated worker core. A number of mappers and reducers can be launched, each on a dedicated worker core, based on the sort task configuration. All application components communicate and synchronize over IPIoE. In Hadoop there is no control over the number of mappers as it is dependent on where a target input file distributed blocks are stored over HDFS, but we can configure a TeraSort task to use a specific number of reducers which greatly affects the execution time. In our proposed micro-kernel environment, we fixed the number of mappers to 20 which will reserve 20 cores for that, and we will change the number of reducers in different experiment runs.

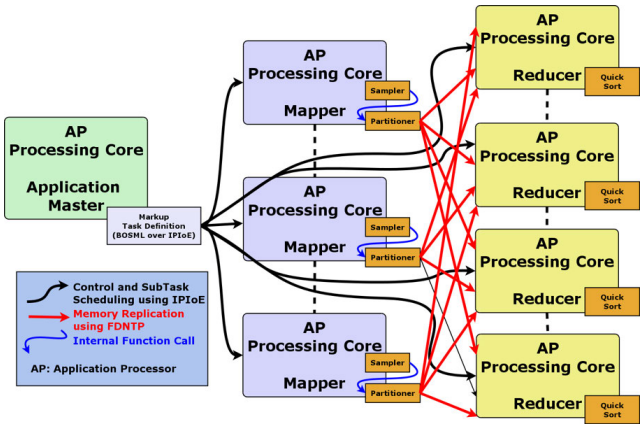


FIGURE 16. AMP Microkernel TeraSort environment.

In our proposed micro-kernel implementation of TeraSort the application master starts first, issues IPIoEs to start up mappers, assigns them their jobs, and provides them with the intended number of reducers so they can split the data accordingly. Each mapper loads its data from the disk storage and applies sampling techniques to identify data partitions. When the mappers finish their tasks, they notify the application master using IPIoE messages to start up the reducers and hence the shuffling phase. Data partitioning and network transfer over FDNTP [2], Fast Data Transfer Network Protocol, is started to assign each reducer a subset of the data. Each

reducer launches its quick sort engine to sort its subset of the data and store its output on disk. The sampling phase carried out by the mappers ensures that the partitions generated by different reducers are organizationally sorted. It is important to mention that all parameters passed to reducers are being encapsulated in BOSML messages within the payload of the IPIoE packets.

We have used TeraGen [15], [16] to generate 9 data sets with different sizes: 2, 4, 8, 12, 16, 24, 28 and 32 GB. We conducted different runs with different data sizes and different number of reducers: 10, 12, 14, 16, 18, and 20 reducers. Some limitations restricted running experiments with all combinations of sizes and reducers; for example, due to memory limitations we could not run the sort on data larger than 12 GB with 10 reducers, and we could not sort 32 GB except with 20 reducers. Figure 17 shows the gain in the execution time by the proposed kernel over Linux/Hadoop using different reducers and input sizes.

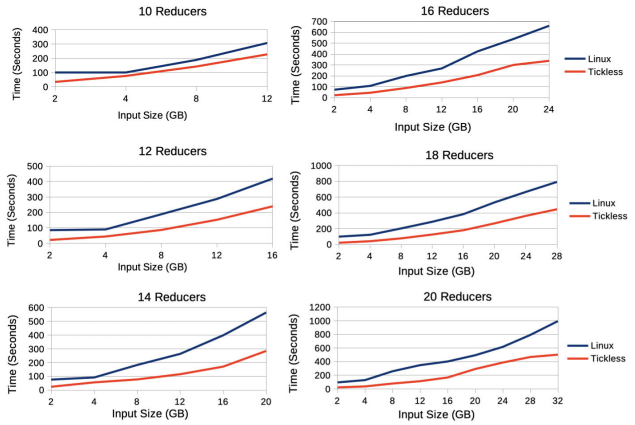


FIGURE 17. AMP Microkernel vs Linux Runtime (Reducers/Input Size).

Figure 18 shows the speed up percentage of the AMP micro-kernel over Linux/Hadoop for different reducers and input sizes. The performance gain increased as we increased the number of reducers within the same input size. Increasing the input size using the same number of reducers has a negative effect on the performance gain as the physical hardware resources are fixed across all experiment runs. The average speedup of the AMP micro-kernel over Linux/Hadoop is 234%; 2.34 folds.

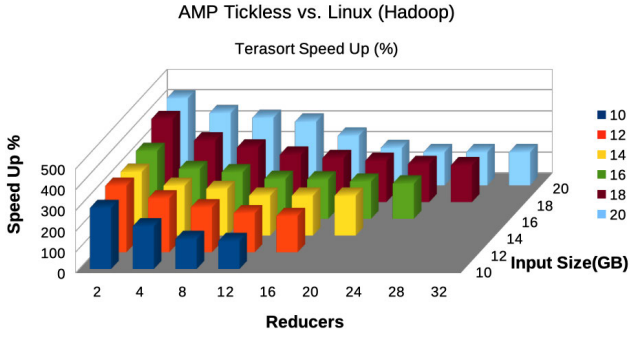


FIGURE 18. AMP Microkernel speed up.

We were able to isolate and measure the shuffling phase in our TeraSort implementation, yet within the shuffling phase we could not isolate the network transfer time and data partitioning. Figure 19 shows the shuffling speed with respect to the number of reducers used. Up to 18 reducers, the speed increased proportionally with the number of reducers. Speed starts to decrease after that because of the network congestion resulting from communication between larger number of reducers resulting in higher packet loss rate. We also attribute this to the target underlying infrastructure used, where 5 virtual machines are running per physical node on average, and bridged over and sharing the same physical network interface.

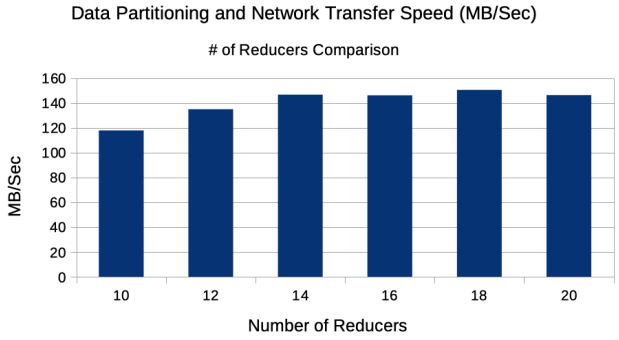


FIGURE 19. Shuffling speed.

Finally, we have conducted a factorial ANOVA [21] and regression analysis. Our investigated factors are the input size, number of reducers, and the environment; AMP micro-kernel and Linux/Hadoop. The yield is basically the end-to-end execution time of the overall TeraSort run. Each run in our experiment has fixed factor values and is executed 3 times to collect 3 readings. Figure 20 presents the ANOVA and the regression analysis results. The Normal Q-Q graph demonstrates the normality of the data which is further verified using Anderson-Darling and Shapio-Wilk [21] normality tests comparing the P-Values with significance level of 0.01. 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. More importantly, the mean squares pie plot shows the magnitude of the influence of each factor. Finally, a regression analysis is presented with a very high R-Squared and adjusted R-Squared values which emphasizes that the model can be used to predict results for factor values outside the experiment range.

Figure 21 demonstrates the factors interaction using 2-D and 3-D graphs. It is evident from the graphs that the environment has a significant effect on the execution time which emphasizes the gain in performance by the AMP micro-kernel over Linux/Hadoop.

V. DISCUSSION

NileOS was designed targeting Big Data applications requiring huge computational power and dedicated hardware.

Anova Results

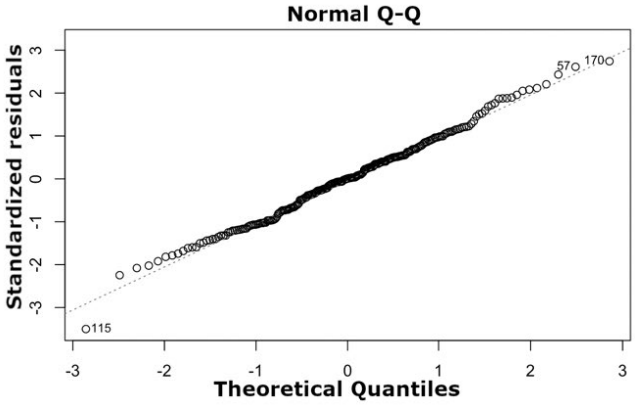
	Df	Sum Sq	Mean Sq	F-value	Pr(>F)
Environment	1	3.2e+02	3.2e+02	900	8.70e-81 ***
Input Size	1	1.4e+03	1.4e+03	3900	5.30e-145 ***
Reducers	1	6.2e+00	6.2e+00	18	3.80e-05 ***
Environment:Input Size	1	2.4e-02	2.4e-02	0.067	8.00e-01
Environment:Reducers	1	1.0e+01	1.0e+01	28	2.60e-07 ***
Input Size:Reducers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



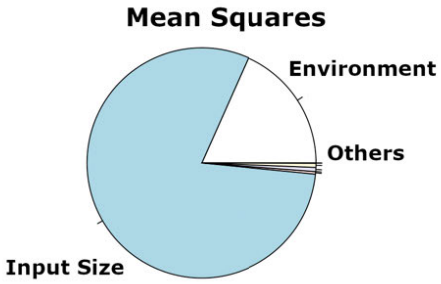


FIGURE 20. ANOVA results and regression analysis.

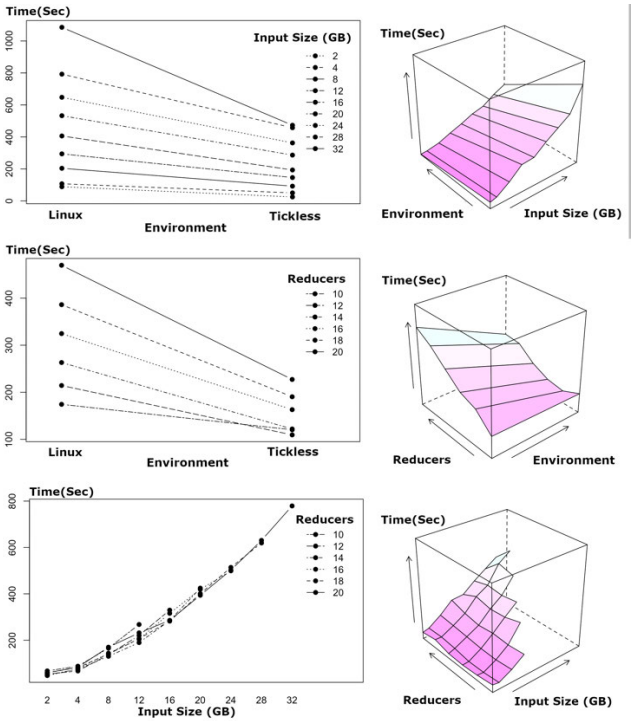


FIGURE 21. ANOVA factors interaction models.

There was a need to assign each core a specific role that could change during run time but needs to be defined at any given point of time. This division of labor scheme allows the operating system to have dedicated cores for running user applications without any interruption to achieve maximum efficiency. On the other hand, cores that are not dedicated to executing user applications are assigned to other roles such as management or I/O. This could only be achieved by following an asymmetric multiprocessing design that allows different cores to assume different roles, which stems mainly from design decisions 1, 2, and 3. UNIX-based operating systems follow a symmetric design which unfortunately does not allow for assigning individual roles to cores. In order to achieve this goal, one would need to actually redesign the kernel architecture to meet the desired specifications [10], [22]–[29]. Consequently, it is inherently challenging to impose roles on individual cores to carry out specific tasks.

GenerOS, an AMP kernel extension built on top of Linux, allow assigning different roles to different cores, and is able to run legacy applications. However, GenerOS is designed as a multi-core operating system that runs only on a single node and does not support distribution except through the traditional means at the library level; essentially the distribution semantics are not designed and implemented at the kernel level [30]. NileOS on the other hand supports assigning individual roles to different cores on different nodes using the IPI over network packets that were designed as part of the kernel network service.

In NileOS, there was a dire need to manage the memory as efficiently as possible. Generally, general-purpose operating systems use 4 KB page frames for their memory model which have the drawback of being slow and having a massive memory footprint for storing the corresponding virtual page table. Such virtual page tables are also being built incrementally based on the application needs. Although, one virtue

about such small page frames is the reduced fragmentation, but this is insignificant in the case of big data applications that are characterized by loading large amounts of data in memory. Big Data applications usually have a fixed pattern of execution that starts by reading the data from the disk into the memory, process it, then generating results. It is very evident that using large page frames and allocating most of the page table upfront is very essential to achieve performance. NileOS allows for choosing between 1 GB, 2 MB, and 4 KB memory page frames, and avoids memory fragmentation as much as possible. This is accomplished by dividing the physical memory into collections of 1 GBs which consist of 512×512 (262144) blocks of physical memory frames of size 4 KB and the other option is a collection of 2 MBs which consist of consecutive 512 memory pages of 4 KB size. If the core requests a 1 GB or 2 MB frame, then they are provided with one of the free collections. On the other hand, if a 4 KB frame is requested, the system searches for the first collection that has the least number of frames available within.

Current general-purpose operating systems adopt an abstraction model that is based on the concept of a process, and hence the need to maintain multiple address spaces; one for each process. Alternatively, NileOS adopts an abstraction model that is based on a core with a single process assigned to it and hence a single address space per core. As an alternative, user-level multi-threads are being used within a core whenever needed. This eliminates overheads resulting from process context switching and using multi-threading within a core whenever needed as an alternative. All security issues resulting from adopting a single address space, such as malicious processes possibly accessing privileged information of another process [31]–[33], will not be valid in the new proposed architecture; basically, one process per core.

IPI and IPIoE use BOSML to exchange messages locally and remotely. NileOS assumes that the data is initially distributed prior to processing it and this allows the adoption of ''move-code-to-data'' mechanism considering that the data is always larger than the code. Loading the right code to the proper core is done using synchronization constructs built on top of IPI/IPIoE and BOSML. This also mitigates any issues with BOSML performance, as big data applications usually fetch the data at the beginning and then process it for a long time. There are some limitations on the packet size of IPIoE. For transparency purposes maximum BOSML message that can be exchanged between cores, either local or remote, is fixed and need to fit within a network packet. The net payload size that can store a BOSML message should be 1486 bytes after subtracting the Ethernet and the IPIoE headers. Nevertheless, since IPIoE messages can be exchanged between different networks over the IP protocol as described in subsection III-D, then we need to take into account the IP header size which is another 20 bytes. This will make the maximum payload space available to store a BOSML message 1466 bytes. This allows NileOS to have an overall extendable architecture that scales as needed without sacrificing performance nor generality. Of course, using larger network packets, such as jumbo packets, can increase the BOSML message size, but this will entail either restricting transmission to be within the same LAN, or make sure to adjust routers MTUs to accommodate larger sizes.

NileOS follows a micro-kernel architecture approach; hence, everything that is not needed in the kernel, is abstracted in the form of services to reduce the overhead of context switching especially when run in the user-mode. Therefore, a service deployment mechanism that deploys these services in a distributed environment transparently is needed. The service that needs to be deployed can then be locally or remotely deployed without the need for the service developers to know the underlying architecture, they only need to know the programming model. Therefore, the programming model was simplified using simple special preprocessor tags over the service methods to identify whether it requires kernel privileges or not. This will allow the service developer to optimize the overall system through minimizing the need to switch to the kernel. The deployment service will make sure to deploy the service methods accordingly at the right levels; kernel-level or user-level. This provides transparency for the developer and adds to the extendibility of NileOS.

Experimentally, it is evident from the experiment results presented in section IV that building a special purpose kernel for Big Data processing achieves considerable performance gain. Taking into account the network transfer and optimizing it to suit distributed Big Data processing has a considerable contribution to the performance gain we observed. It is worth mentioning that we were forced to oversubscribe the CPUs/Cores in our test environment because of the limited availability of HW resources. A resource contention is realized over the physical CPUs/Cores assigned to different virtual machines simultaneously. We anticipate that if the experiments are repeated with dedicated CPU/Cores for each virtual machine a better performance gain can be achieved, which we believe needs to be verified experimentally.

The ANOVA analysis shows that the input size has a large and significant magnitude over the execution time. This might give the illusion of poor scalability, yet the fixed physical resources used explains the inflated impact of the input size. Increasing the virtual resources pressures the underlying fixed physical resources through the aggressive multiplexing of different virtual machines over the limited CPUs/Cores available. We believe that increasing the physical hardware resources in proportion to the workload will keep the proportional gain in performance constant. A larger hardware environment than the one we used will be needed to verify this aspect experimentally. In any case, the underlying physical environment is common in both cases, AMP Tickless and Linux/Hadoop.

It is evident that the added tickless feature to the worker cores contributed largely to the gain in performance. The major gain of the tickless approach, other than dedicating CPUs/Cores for processing tasks, is the realization of a single address space per CPU/Core which saves up a lot of TLB flushing. Moreover, soft irqs are being handled by the management cores which relieves the processing cores from having to check and handle them.

One major aspect of the success of the proposed AMP micro-kernel is the transparent interaction of the different CPUs/Cores irrespective of their location through the transparent IPIoE messaging mechanism. The TeraSort application was written in a way that the user level logic does not perceive the distributed nature of the underlying environment which realizes a Single System Image (SSI) allowing easy and reliable scalability without any changes to the application logic.

One important feature that needs to be in place in the future is the dynamic run-time assignment of roles to cores based on the run-time workload. A number of efficient optimization algorithms have shown a great improvement in task scheduling especially in cloud computing environments, such as moth search, moth-flame optimization, monarch butterfly optimization, and elephant herding optimization. We do believe that the extensible service-oriented design we adopt for our proposed kernel enables the seamless integration of such algorithms.

VI. CONCLUSION AND FUTURE WORK

Big data applications will soon be one of the computing mainstream areas. Existing general-purpose operating systems cannot cater for their special resource utilization requirements and thus could not provide optimum performance. Our approach differs from general purpose operating systems by trading the generality for maximizing performance with a set of goals and design decisions that were inspired by other approaches. Multiple approaches were combined to formulate an efficient scalable platform for big data processing characterized by being similar to Map-Reduce execution profile. We rely on the concept of move-code-to-data as it depends on the distribution of the data in the first place and avoids moving huge amounts of data across the network. An asymmetric multiprocessing architecture model was presented in which a core is the building block of the system and roles can be assigned to specific cores. To minimize the interrupts on the big data processing cores, a specific role of being a ''Worker'' core is now possible and hence we could disable the timer from cores assigned that role. Memory management is being taken seriously at the core of the design and a single address space is adopted to reduce performance overhead resulting from cache and TLB flushing. Moreover, some ideas were borrowed from inverted page tables to enhance both the page table footprint and speed of allocating memory by an application. A distributed execution model was established through utilizing extensible messaging framework over BOSML. Local cores exchange BOSML messages using local IPIs and shared memory, and remote cores exchange messages using the special control protocol IPIoE. All distribution semantics and mechanisms are being pushed down into the kernel with maximum transparency at the application level. A transparent deployment mechanism is presented to completely shield the developer of the micro-kernel service from the underlying distribution infrastructure and decouple the application implementation from its deployment perspective. Based on the initial prototype and the experiments presented, a considerable gain in performance of average 2.34 folds was achieved using the distributed TeraSort benchmark over Linux/Hadoop.

In the future, there needs to be further iterations over the IPIoE network protocol to enhance the communication between the nodes within the same LAN and across different LANs. The dynamic nature of the protocol header will allow for such revisions. One of the most important protocols needed as well would be a data transfer protocol since IPIoE is only designed for control purposes. The service transport layer has also enabled future development on other services that would be vital for the operation of the micro-kernel. Services such as the directory service, distributed file system and heap manager need to be designed and implemented. Furthermore, the standard libraries need to be imported to NileOS micro-kernel. Features like distributed exception handling support as well as extending the ELF format to support loading distributed applications need to be in place. This will allow for porting applications previously developed based on these libraries as well as giving this platform the capability to do further tests, experiments, and benchmarks to compare the overall performance between NileOS and different competitors.

REFERENCES

[1] IsideBIGDATA. *The Exponential Growth of Data*. Accessed: May 2020. [Online]. Available: https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/

[2] K. Sobh and A. El-Kadi, ''A tickless amp distributed core-based micro-kernel for big data,'' in *Proc. Future Technol. Conf. (FTC)*, K. Arai, R. Bhatia, and S. Kapoor, Eds. Cham, Switzerland: Springer, 2020, pp. 556–577.

[3] R. Suganya and A. Abdullah, *Big Data in Medical Image Processing*. Boca Raton, FL, USA: CRC Press, 2018, doi: 10.1201/b22456.

[4] J. Giceva, G. Zellweger, G. Alonso, and T. Rosco, ''Customized OS support for data-processing,'' in *Proc. 12th Int. Workshop Data Manage. New Hardw. (DaMoN)*. New York, NY, USA: Association Computing Machinery, 2016, doi: 10.1145/2933349.2933351.

[5] J. Dean and S. Ghemawat, ''Mapreduce: Simplified data processing on large clusters,'' in *Proc. 6th Symp. Operating Syst. Design Implement.*, San Francisco, CA, USA, 2004, pp. 137–150.

[6] D. Thakur. (Apr. 2013). *Time Sharing Operating System*. [Online]. Available: https://ecomputernotes.com/fundamental/disk-operating-system/time-sharing-operating-system

[7] M. Travers, ''Cpu power consumption experiments and results analysis of Intel i7-4820k,'' School Elect. Electron. Eng., Newcastle Univ., Tyne, U.K., Tech. Rep. NCL-EEE-MICRO-TR-2015-197, 2015. [Online]. Available: http://async.org.uk/tech-reports/NCL-EEE-MICRO-TR-2015-197.pdf

[8] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, ''System noise, OS clock ticks, and fine-grained parallel applications,'' in *Proc. 19th Annu. Int. Conf. Supercomput. (ICS)*. New York, NY, USA: Association Computing Machinery, 2005, pp. 303–312, doi: 10.1145/1088149.1088190.

[9] T. Jones, P. Tomlinson, M. Roberts, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, and B. Maskell, "Improving the scalability of parallel jobs by adding parallel awareness to the operating system," in *Proc. ACM/IEEE Conf. Supercomput.* Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2003, p. 10, doi: 10.1145/1048935.1050161.
[10] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: Wiley, 2012.
[11] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*, 2nd ed. Upper Saddle River, NJ, USA: Pearson Prentice-Hall, 2007.
[12] R. Buyya, T. Cortes, and H. Jin, "Single system image," *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 2, pp. 124–135, May 2001, doi: 10.1177/109434200101500205.
[13] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice-Hall Press, 2014.
[14] P. Eugster, F. L. Fessant, and P. Felber, "The 'art' of programming gossip-based systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 41, no. 5, pp. 37–42, 2007.
[15] S. Li, S. Supittayapornpong, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded terasort," in *Proc. 6th Int. Workshop Parallel Distrib. Comput. Large Scale Mach. Learn. Big Data Anal.*, May/Jun. 2017, pp. 389–398.
[16] MAPR. *Terasort Benchmark Comparison for Yarn*. Accessed: Jan. 31, 2019. [Online]. Available: https://mapr.com/resources/terasort-benchmark-comparison-yarn/
[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Fast and interactive analytics over Hadoop data with spark," *USENIX Login*, vol. 37, no. 4, pp. 45–51, 2012.
[18] O. O'Malley, "Terabyte sort on Apache Hadoop," Yahoo, Sunnyvale, CA, USA, Tech. Rep., 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.1187&rep=rep1&type=pdf
[19] A. Jlassi and P. Martineau, "Benchmarking Hadoop performance in the cloud—An in depth study of resource management and energy consumption," in *Proc. 6th Int. Conf. Cloud Comput. Services Sci.*, Rome, Italy, Apr. 2016, pp. 1–11. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01325027
[20] D. Borthakur. (Jan. 2008). *HDFS Architecture Guide*. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.pdf
[21] D. Montgomery, *Design and Analysis of Experiments*, 8th ed. Hoboken, NJ, USA: Wiley, 2012. [Online]. Available: https://books.google.com.eg/books?id=XQAcAAAAQBAJ
[22] D. Knoll, B. Sehorz, and S. Sigl, "Understanding the freeBSD operating system. Review of the freeBSD 5.4 kernel," Dept. Comput. Sci., Univ. Salzburg, Salzburg, Austria, Feb. 2006. [Online]. Available: https://scholar.googleusercontent.com/scholar?q=cache:HZa2K-39BLMJ:scholar.google.com/+Understanding+the+freeBSD+operating&hl=en&as_sdt=0,5&as_vis=1
[23] M. Santosa, "Choosing the right timer interrupt frequency on Linux," Skripsi Program Studi Sistem Informasi, 2010.
[24] S. Siddha, V. Pallipadi, and A. Ven, "Getting maximum mileage out of tickless," in *Proc. Linux Symp.*, vol. 2, 2007, pp. 201–207.
[25] L. Parziale, *Sap on DB2 9 for Z/Os: Implementing Application Servers on Linux for System Z* (International Technical Support Organization). IBM, 2005. [Online]. Available: link http://www.redbooks.ibm.com/abstracts/sg246847.html?Open
[26] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proc. 35th SIGMOD Int. Conf. Manage. Data (SIGMOD)*. New York, NY, USA: ACM, 2009, pp. 165–178, doi: 10.1145/1559845.1559865.
[27] T. Rabl, H.-A. Jacobsen, and S. Mankovskii, "Big data challenges in application performance management," in *Proc. 5th Extremely Large Database Conf.*, Oct. 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1016.8896&rep=rep1&type=pdf
[28] J. Klein and I. Gorton, "Runtime performance challenges in big data systems," in *Proc. Workshop Challenges Perform. Methods Softw. Develop. (WOSP)*. New York, NY, USA: ACM, 2015, pp. 17–22, doi: 10.1145/2693561.2693563.
[29] N. Khan, I. Yaqoob, I. A. T. Hashem, Z. Inayat, W. K. M. Ali, M. A. Alam, M. Shiraz, and A. Gani, "Big data: Survey, technologies, opportunities, and challenges," *Sci. World J.*, vol. 2014, Jul. 2014, Jul. 712826.
[30] Q. Yuan, J. Zhao, M. Chen, and N. Sun, "Generos: An asymmetric operating system kernel for multi-core systems," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Apr. 2010, pp. 1–10.
[31] M. Brinkmann, "Porting the GNU Hurd to the L4 microkernel," Free Softw. Found., Inc., Boston, MA, USA, Aug. 2003.
[32] J. Corbet. (May 2013). *(Nearly) Full Tickless Operation in 3.10*. Accessed: Feb. 3, 2019. [Online]. Available: https://lwn.net/Articles/549580/
[33] M. Guarnieri, "The unreasonable accuracy of Moore's law [historical]," *IEEE Ind. Electron. Mag.*, vol. 10, no. 1, pp. 40–43, Mar. 2016.

AHMAD EL-ROUBY has been working as a Software Engineer at DELL EMC, since June 2019. He is currently a Computer Engineering Graduating Senior with The American University in Cairo (AUC). He worked on many projects during his studies at AUC: web and app development, machine deep learning, database design, and digital design. His thesis project is on bare metal layer for an asymmetric distributed operating system. He worked as an Undergraduate Teaching Assistant for CS introductory-level courses for three years. He did a web development internship at Sypron Solution for a month in 2018. He also did another six-week internship at Mentor Graphics in Quality Assurance. He was the President of the Computer Science and Engineering Association, from 2016 to 2017.

ANDREW KHALAF received the B.Sc. degree in computer engineering with a focus systems engineering and automation. He is an Alumni of The American University in Cairo (AUC). He has always been a technology enthusiast and continuously pursues becoming an early adaptor of emerging technologies. He is a strong believer in the power of effective education for inducing a change in the society. He worked closely with faculty in incorporating relevant educational technologies in the classroom for three years through a work-study position at the Center for Learning and Teaching (CLT), AUC. Throughout his studies, he realized the importance of leadership skills and communication skills in addition to technical skills. He interned at Credit Agricole's Main office in which he worked closely in a group to demonstrate a proof of concept in a carpooling project. He interned at the Orange Egypt's Main Office, where he led a group in developing an early prototype for a data gathering system to be used with the network coverage bases. His special research interests include operating systems, system design, and software engineering.

ARIG MOSTAFA received the bachelor's degree in computer engineering from the American University of Kuwait (AUK), in fall 2017. She is currently pursuing the degree with The American University in Cairo (AUC). She completed her internship at Gulf Business Machine (GBM), Kuwait, in which she was trained on Hyperledger with a focus on Hyperledger Fabric as well as the basic information on IBM Mainframes (Z13). She is completing the Virtual Data Science internship at GBM Digital Business Solutions. She participated in many extracurricular activities as a president of a culture student organization and the operations head of multiple clubs, such as the Astronomy Club and Animal Rights Association, in which her proudest achievement is being able to invite Dr. Farouk El-Baz and finally taking a treasured pictorial with him. She also served as the Operations Director of the newly opened IEEE branch at AUC. She is a believer in the importance of culture diversity and the sociology behind it. She is passionate about learning the mechanics of the mainframe as well as embedded systems, chip design, and SOCs. Her current research interests include AI and data analytics.

FADY MOHAMED is currently pursuing the degree in computer engineering with The American University in Cairo. He is currently a Software Engineer with the Dell Technologies Center of Excellence, Egypt, where he is also responsible for implementing efficient solutions for the LiveOptics product. He has published a paper at the Workshop on Open-Source EDA Technology, which is co-located with the prestigious ICCAD. He participated in the Egyptian Collegiate Programming Contest twice after qualifying from The American University in Cairo Programming Contest. His research interests include operating systems, big data, and performance tuning domains. He received the fifth place in Egypt in the Google HashCode Contest. He and his team received the second place in Dell Technologies IOT Hackathon out of 150 participating teams, Egypt.

NOUR GHALY is currently pursuing the degree in computer engineering and minoring in business administration at The American University in Cairo. She has great experience in her field as she has participated in several internships, some of which are networks team at Qatar National Bank, as well as a Software Engineer at Qatari startup called C-Wallet. These internships developed both her technical and interpersonal skills. Above all that, she enjoys reading and working out and is keen on staying active every day, both mentally and physically. She has worked on many projects in university in various topics, which include mobile application development (front end and back end), networking, embedded systems, and machine learning. She is working on a research-based thesis project on asymmetric distributed tickless microkernel for big data processing.

AMR EL-KADI (Senior Member, IEEE) received the D.Sc. degree in electrical engineering and computer science from the George Washington University. He was a Consulting Engineer with the Information, Technology and Facilities Department, World Bank, Washington, DC, USA. He is currently a Professor and the former Chair of the Computer Science and Engineering Department, The American University in Cairo. He was a member of the IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices (SEEPP) that developed the Software Engineering Code of Ethics and Professional Practices. He is serving as the Middle East Representative for the IEEE Technical Committee on Operating Systems and Applications Environments and a member for ACM and Eta Kappa Nu (the U.S. National Electrical and Computer Engineering Honor Society).

KARIM SOBH received the B.Sc., M.Sc., and Ph.D. degrees in computer science from The American University in Cairo. He worked with Nile University (NU), Cairo, as an Assistant Professor and the University of California at Santa Cruz (UCSC) as a Visiting Lecturer. He is currently an Assistant Professor with the Department of Computer Science and Engineering, The American University in Cairo (AUC). He is also the Founder of Code-Corner, a software development firm providing software development, subcontracted services, cloud deployment services, consultation services, and turn-key solutions using open-source technologies. His specialization is in operating systems, networks, distributed systems, and cloud computing, and his Ph.D. topic is cloud environments metering. As a Systems Architecture Consultant at IBM Egypt, his role was to provide system architecture consultations and implementation services for large projects.