Multics and Plan 9: The Big Bangs in the Distributed Computing System Universe

Seyedeh Leili Mirtaheri and Ehsan Mousavi Khaneghah | University of Calabria
Amir Saman Memaripour | University of California, San Diego
Lucio Grandinetti | University of Calabria
Mohsen Sharifi | Iran University of Science and Technology
Zarrintaj Bornaee | Iranian Research Organization for Science and Technology

Investigating the development of pioneer systems, Multiplexed Information and Computing Service (Multics) and Plan 9, helps illustrate how these approaches have influenced today's computing systems. Studying previous systems' objectives and current statuses can facilitate new solutions and help point to possible failures.

An increasing number of computing systems are based on distributed computing models, and great efforts are in progress to overcome the present issues in this field. For example, a great deal of scientific research has gone into virtualization, load balancing, and resource sharing in the past decades, and numerous algorithms have been proposed to address these issues. A glance at the first appearance of these systems reveals Multiplexed Information and Computing Service (Multics)[1] and Plan 9[2] as the pioneers of the distributed computing era, with development starting at Bell Labs. Furthermore, many breakthroughs in these systems, such as the Multics security mechanisms and Plan 9 namespaces, haven't been properly implemented in systems. Consequently, investigating the evolution, novelty, and present state of these systems, along with considering the scientific and technological trends leading to their development will guide us in improving the performance of computing systems, such as cluster,[3] grid,[4] and cloud[5] services.

Each new computing system, with the addition of new services, has some advantages compared to previous ones. However, this doesn't mean that the new computing systems are better and more efficient than previous ones. Each computing system focuses on certain issues in distributed computing models,[6] and hence supports specific scientific programs better than others. Several studies and much research have been conducted to compare new computing systems[7,8] and to specify their use in executing scientific programs.[9]

Here, we investigate the development of Multics and Plan 9 and illustrate how they've influenced today's computing systems.

History of Development

The development of Multics dates back to 1965, when a group of engineers and scientists from Bell Telephone Laboratories, the General Electric Computer Department, and the Massachusetts Institute of Technology Project on Mathematics and Computation (MAC) collaborated to develop one of the first commercial time-sharing operating systems. Fernando J. Corbató from MIT, who had been the supervisor of the Compatible Time-Sharing System (CTSS) project—one of the first time-sharing operating systems—accepted the same responsibility in this project. The Information Processing Techniques Office of the US Department of Defense supplied a portion of the financial requirements for this project for eight years and provided the Project MAC group with a total of 16 million dollars. Beside this financial support, the GE Computer Department (which was later sold to Honeywell) and Bell Labs designated considerable resources to this project.[10]

In the first days, Multics was defined as a research project, and a joint conference between all involved groups in 1965 published six papers outlining the project's fundamental ideas. Later, Honeywell joined the project and turned Multics into a commercial project, making it a candidate for educational, industrial, and governmental applications. In the first release, Multics was a general-purpose operating system that could execute various applications on GE-645 computers. However, the system was dependent on expensive processors with special features, including virtual segmented and paged memory, to grant user requests. Moreover, special-purpose hardware provided Multics with a synchronous multiprocessor and shared virtual and physical memory system to meet its runtime requirements.[11]

Employing High-Level Languages

A major part of the Multics source code is in Programming Language One (PL/I)—a high-level language introduced by IBM. This was the first time that a high-level language was used to implement an operating system. Although a large number of system programmers believed that using a high-level language would result in reduced performance, PL/I was chosen to facilitate maintenance and development of Multics.[10] Besides refining the PL/I language, the development team made several changes in the Multics hardware infrastructure to address its new demands. Memory addressing logic, process control mechanisms, and memory segmentation and paging algorithms are examples of these changes. Furthermore, the generalized input/output controller (GIOC) module was designed and implemented to reduce process execution delay while performing I/O requests.[12]

Despite some unexpected recurrent phases in its development process, Multics became mature enough in 1968 that the development team began to use it as their operational system. In October 1969, Multics became available to the public through MIT, and users started to pay for their share in a CRC manner, just as they did with other MIT computing services. In these years, Multics was the foremost time-sharing operating system at MIT, with more than 500 registered users and availability of 22 hours on all weekdays.[10]

Moving to UNIX and Plan 9

In 1969, Bell Labs ceased its cooperation in the Multics project and GE relinquished its computer department to Honeywell. The system's complexity and immensity were among the reasons that Bell Labs abandoned this project. After that, the remaining researchers in the project, including Ken Thompson, Dennis Ritchie, M.D. McIlroy, and J.F. Ossanna, decided to design an operating system similar to Multics, but of a smaller size. Consequently, they designed and implemented UNIX, the new time-sharing operating system, in B language. Since 1973—after Dennis Ritchie proposed the C programming language—C has been used for implementing UNIX and its utilities.[13]

In the mid-1990s, difficulties in using time-sharing systems led to the appearance of a new generation of computer systems with lower computational power and smaller size.[2] Furthermore, advancements in computer networks and the increasing number of computer systems drove the development of new computing system concepts, such as data and resource sharing over a network.[14] As a result, new computing systems were expected to deal with the drawbacks of time-sharing operating systems such as UNIX. In the late 1990s, a group of system programmers—including Ken Thompson, Rob Pike, Dave Presotto, and Phil Winterbottom—introduced Plan 9, a computing system with centralized administration and public access. The Plan 9 approach proposed a distributed computing environment that would provide users with remote access to centralized, shared resources, such as computing and storage, using an inexpensive hardware configuration.[2,15] Plan 9's design concepts and architecture were similar to those of the Cambridge distributed system in many ways. In Plan 9, central computers are responsible for serving user requests, making use of computing systems, such as multiprocessors and shared memory. Figure 1 illustrates the basic hardware organization in Plan 9.

Plan 9 inherited many basic concepts from UNIX, including using file systems for naming and accessing resources, and adapting these resources to the new computing paradigm. This approach led to the genesis of new protocols such as 9P, a network protocol providing remote access to

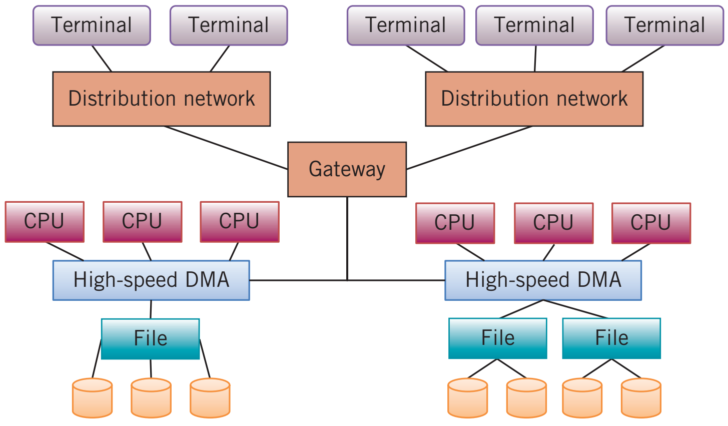

Figure 1. Basic architecture of Plan 9.

files. In addition, new naming mechanisms were designed to accommodate every user with an exclusive view of the shared resources in the computing network.[16] As a result, every user can configure a custom computing environment and move it to any other computer; this mobility allows users to perform computation on several machines. All aspects of Plan 9, such as processes, graphics, and network management, are based on these fundamental concepts, namespaces, and file-mapped resources, which makes it intrinsically different from UNIX.[2,15]

Plan 9 as a Commercial Product

In 1989, Plan 9 reached such a level of maturity that its developers started to use it as their main operating system. As a result, researchers developed various compilers, programming languages, libraries, and applications for Plan 9 with regard to its remote-execution capabilities. Furthermore, many UNIX applications were revised and ported to Plan 9.[2] The first distribution of this operating system, which was introduced in 1992 for academic use, relied on personal computers and NeXT stations as user terminals, and on SGI Magnums, SGI Powers, and Sparcstations as computing servers. The second distribution of Plan 9, called Brazil, including a collection of small utilities, was published to the public market in 1995. In June 2000, the third distribution became available on the Internet for free. This distribution introduced a new color-graphics operator and a program-connection mechanism, named draw and plumbing, respectively. Moreover, Plan 9 users could update the third distribution using wrap, an update utility that downloads and installs update packages.[15]

A new edition of the 9P protocol, called 9P2000, was published with modifications in the authentication mechanism, longer file name support, and a new attribute in directory metadata to store the last modifier. In 2002, the fourth distribution of Plan 9 was released based on the 9P2000 protocol. This release provided a block storage server called venti, a user space file system for authenticating users called factotum, and a key storage mechanism for storing factotum keys called secstore.[17] Moreover, the fourth release accompanies a new update method that provides users with the latest file versions using a public fossil file server. This distribution of Plan 9 is still operational, and is being updated in a daily manner using a public-file-server source repository, sources.cs.bell-labs.com.[18]

Innovation and Creativity

Multics possesses many novel features and has introduced many creative ideas to the computing systems field. It was designed with a segmented-, virtual-, and shared-memory multiprocessor architecture, and was developed using high-level languages and software engineering techniques. Thanks to its design considerations, Multics supports multiple programming languages and online reconfiguration and provides powerful security mechanisms.[11] It was on the Multics system, moreover, that the first commercial relational database management system was designed.[10] As no existing computer could suit its hardware requirements, GE-645 systems were designed and produced, based on the GE-635, which was the most similar candidate to its perquisite specifications. In the design process of Multics, several fundamental ideas were proposed, and the most important ones are as follows:[10,11]

- provide users with easy access to remote terminals in the same way as to local access;
- afford a continuous view of system performance to users, resembling electric and telephone utilities;
- support scalability and system resize without user and system reorganization;
- maintain a reliable internal file system that hosts the exclusive version of user data and programs;
- give an expedient access control mechanism to manage shared data and regulate how permitted users can access the shared data;
- provide hierarchical representation and organization of logical memory and system management;

- serve users with various computing demands such that those with low computing requirements don't notice the presence of users with high computing needs, and vice versa;
- support multiple programming environments and user interfaces at the same time; and
- provide a flexible general organization that easily adapts to hardware and software evolutions and ever-increasing user computing requests.

An important aim in the Multics design was to share hardware infrastructure and resources among users, which is a particular type of multiprogramming in which execution control is switched between processes of different users. Besides that, Multics functions in a nonstop manner, and all of its hardware devices, such as processors and memory modules, can be changed without interrupting system activity. Consequently, a mechanism had to be devised to isolate user programs from hardware infrastructure.[1,11] Great variety of resources, especially I/O, led to design of a unique resource interface that reduced programs dependency on hardware resources.[12] Moreover, the existence of shared spaces between users introduced the concept of interprocess security, and the ring model was used to protect user privacy and provide a secure execution environment.[1,11] The Multics security mechanism, however, used many rings, in comparison with today's systems, which have only two rings; the high number of rings degraded performance considerably.

Resource Sharing

According to the point of view of Multics developers, any resource in a computing system can be classified as a virtual memory, a processor, a memory module, or an I/O device.[10,11] Main and secondary memory, which store programs and user data, play the fundamental role in the Multics architecture, and all memories and processors are directly connected by means of exclusive buses. Every memory module is exclusively connected to all processors and communicates with them through message-passing mechanisms. The I/O controllers are linked to and interact with memory modules in the same manner.[1] The traffic controller and general I/O controller modules are responsible for managing processors and I/O devices, respectively. The traffic controller module maintains related information and structures of every process and performs process scheduling and intercommunication. Moreover, it provides parallel-processing features, avoids the overhead of idle processes, and supervises available processors. It also serves requests of other modules concerning processes, such as a process suspension. Furthermore, it accommodates users with a simple virtualization mechanism that makes them capable of using different supervisors and executing different application types, such as real time and batch, simultaneously.[1,11]

The GIOC module supplies a number of system functions that enable user applications to transfer data regardless of underlying I/O devices. Moreover, these functions create an intermediate layer between user applications and hardware with minimum computing overhead, and conceal the complexity and variety of I/O devices. To perform device-dependent operations, several system functions exist that receive a set of input parameters and trigger the requested action, using more computing cost than general-purpose functions. For example, differences between writing on magnetic tapes and disks aren't conspicuous to Multics users, and hardware changes, such as replacing a floppy drive with a magnetic tape, don't affect user applications. But head relocation of a tape device and a magnetic disk are different, and special functions are used in such cases.[12]

I/O Streams and Data Sharing

In Multics, a collection of streams is used to access and operate any I/O device. A read or a write request is followed by reading from or writing to a stream, and every I/O device must be attached to a stream before being used. In this manner, users can access I/O devices that aren't physically attached to the system or provided with any interface module, resembling user-level virtualization. In this architecture, all applications directly access pseudo-devices instead of physical I/O devices. Moreover, there are other types of device interface modules in Multics, such as object stream, which translates one I/O request to another and receives its inputs from streams instead of pseudo-devices.[12] Synonym is one of the most important examples of this module and results in a sequence of control switches. The synonym module implements many fundamental concepts in Multics, such as broadcasters and virtual devices.[1]

Through resource sharing between users, data security emerged as a serious issue when the Multics file system was being designed. In a multiuser system, users can access system resources from remote and unidentified terminals. Consequently, providing users with security features and protecting their privacy are matters of great importance. Thus, every user's data and stored files must be organized in such a way that only the owner has access to them.

To enable users to manage and share their files with others, Multics developed linking and locking mechanisms. Using these mechanisms, users can specify sharing policies. Moreover, a consistency control mechanism was employed to avoid inconsonant data in case of parallel access to a single file. A general backup mechanism autonomously replicates user files and improves system security and reliability. As a result, Multics file security is delegated to the system rather than the users.[19]

Dynamic Reconfiguration

Service continuity and reliability were two major goals in the Multics design. To achieve these goals, system operation had to be independent of any physical device, which could potentially require repair and replacement. Hence, a general model was devised to dynamically bind and unbind computing and storage resources to and from a multiprocessor and multimemory infrastructure. Thus, processors and memory modules could be added and removed without interrupting the system's operation.[20] Moreover, this model empowered the system to be dynamically reconfigured without users' information. This dynamic reconfiguration improved system reliability, and allowed the system to be developed and operated at the same time.[11]

Hardware Independence

Despite all the innovations, many problems with the Multics design, such as depending on a specific hardware infrastructure, led to its failure. Conversely, the Plan 9 operating system could run on various hardware architectures; it was a hardware-independent system.[15] To support different architectures, data and program transfer issues were addressed, and embedded mechanisms were added to Plan 9 to deal with them. The 9P protocol, which represented data in text format for transmission, was used to exchange data between incompatible architectures.[14,15] In this manner, data are accessed on both the kernel and application layers using file system interfaces, and the byte order is preserved through the use of the text format. For example, an application can get the current system time by a simple file-read operation, `/dev/time`, and without performing any system call that might be dependent on underlying hardware. Additionally, an intermediate step included in the source code compilation made code transfer feasible. In other words, every source code is translated to an intermediate language that's hardware-independent. Next, a proper compiler compiles the source code to the host computer's machine language. Consequently, compiling a single source code on different hardware architectures will result in a unique intermediate file, which will later be converted to an executable file for the destination machine using a loader program conforming to its architecture.[15] Thus, programs can also be exchanged between various hardware architectures. Furthermore, users can perform the same operations with the same commands on various machines using Plan 9's namespaces, which also provide users with transparent access to different compilers and programs.[15]

Plan 9 is composed of a series of service functions running on a centralized hardware infrastructure consisting of large multiprocessor servers without the need for storage capacity and large storage servers. To access computing resources, Plan 9 users employ a terminal that provides a personal computer with a mouse and bitmap display to run the window system. From the sharing of centralized processing and storage resources, maintenance costs decline significantly, and resource management and supervision is easier. Both processing servers and terminals in Plan 9 share the same kernel, enabling users to choose between local and remote program execution. This feature benefits all Plan 9 users, whether they're administrators or regular users, and they can switch between centralized and distributed execution models according to their needs.[15]

Namespaces and File-Mapped Resources

All the resources that a Plan 9 process can access, except for its memory, are encapsulated in a namespace and can be reached in a same manner via a file system. Besides traditional file systems—which shared storage servers use to denote stored data on the disk—Plan 9 uses another file system type to represent physical devices, such as terminals, and abstract concepts, such as processes and environmental variables. These file systems are implemented through kernel-resident drivers, user-level processes, or remote servers. To point to a file in a kernel, Plan 9 uses a data structure called a channel. Thus, a file descriptor at the user level is just a handle to a channel in a kernel, and every I/O system call will be translated to a combination of nine defined primitive operations, such as read and write, on the corresponding channel.[16]

The file system interface is implemented using the 9P protocol, which differs from Network File System (NFS) in many ways. In contrast to the other file system protocol, 9P addresses and transfers files instead of blocks. Moreover, the relative path of every file to the root directory of the file server is used to address it. To respond to a 9P request, the file manager traverses the file system hierarchy to find, open, and perform required I/O operations on the corresponding file. Furthermore, a kernel-resident file system, called mount, translates local 9P requests to RPC messages to access remote file systems. In this way, Plan 9 provides transparent access to shared and remote resources.[14,16]

One of the substantial applications of Plan 9 namespaces is the window system, which is implemented as a fileserver, serving such files as `/dev/cons` and `/dev/bitblt` and providing clients with an exclusive copy of these files in their local namespaces.[15] In other words, the $8^{1/2}$ system serves window requests using namespaces and I/O operations on a file collection, named in a conventional manner. To use the window system, clients need to link their standard input, output, and error files to `/dev/cons` and perform ordinary file operations on them. In UNIX, `/dev/tty` has been employed in a similar way to `/dev/cons`. In contrast to `/dev/cons`, `/dev/tty` is implemented as several code segments in a kernel requiring special supervision and control mechanisms. The $8^{1/2}$ window system monitors the file corresponding to each window and performs the requested actions in view of the current namespace.[21]

Network interfaces are other examples of devices provided as file systems in Plan 9. Thus, a gateway in Plan 9 is a fileserver that affords its interfaces to users. To access a remote network, a process contacts the gateway and mounts the file system of that remote network in its local namespace. The mount driver receives further process requests, translating them to proper requests that will be sent to the gateway. Consequently, the gateway can monitor all network communications of a process, making required protocol conversions feasible.[15]

Besides the mentioned points, Multics and Plan 9 have introduced other novel ideas that are applicable in today's distributed systems. Remote process execution and remote application debugging are examples of these innovations.[15,18]

Scientific and Industrial Trends

The development of Multics coincided with the emergence of space exploration, commercial data processing services, and other applications that required high-performance computing and reliable systems. There's no doubt that the computing problems and requirements in 1964 influenced the creation of Multics and played a considerable role in its design and development. Artificial intelligence, which had been introduced in 1956, attracted a great deal of human and financial resources in 1960. Moreover, the US Department of Defense funded many research projects in this field, and the US Air Force considered a colossal budget for developing computing infrastructures. At the same time, vehicle manufactures were experiencing new demands to simulate and model their automobiles. Furthermore, other computing system clients, including public and private data processing service providers, research institutes, and universities, were interested in using shared computing environments. This inclination was more conspicuous in universities and colleges with a large number of potential users and meager financial resources. Thus, sharing rarely used computing resources could be a great advancement.

In addition to the already mentioned influences, technological enhancements affected Multics in both design and implementation. Because of the potential failure of hardware devices, Multics was designed in a hot-swappable manner to resist hardware breakdowns and provide continuous service. In every computer system, failure of fundamental hardware devices, such as the processor, are likely and can interrupt user request processing. To deal with these failures, Multics was designed as an online-reconfigurable system that allowed hardware replacement without interruption in system operations.[10] Furthermore, the system can be partitioned into two operational subsystems, enabling development in one subsystem while the other is serving users' requests.[11]

As a result of ever-increasing variety in hardware architectures, Plan 9 was designed to support a wide range of hardware infrastructures, resembling today's heterogeneous clusters. Compatibility with various hardware architectures had barely been heeded before Plan 9, and was one of the major reasons that Multics lost its share of the public market. This approach is still a considerable challenge and plays a significant role in today's distributed computing systems.[21]

In the mid-1990s, users showed great interest in using minicomputers, which led to a move from traditional, centralized systems to distributed computing. Hefty time-sharing systems such as Multics were being replaced by a network of personal computers, mostly comprised of UNIX workstations. Despite the feeble performance of early workstations, users were inclined to acquire them to avoid the time-consuming paperwork and infinitesimal time quotas of time-sharing systems. The widespread application of these systems led to a demand for operating systems with proper graphic, network, and sharing support, which UNIX didn't provide at that time. Moreover, UNIX was inherently designed for personal computers and lacks a viable mechanism, like 9P in Plan 9, for managing shared resources.[2]

The Plan 9 developer team believed in the possibility of producing a midget operating system—capable of managing extraordinary systems and supporting a wide variety of hardware and network architectures—by implementing several abstract concepts.[15] As the team saw it, combining high-level abstractions with a minor programming effort, in contrast with a larger system, could yield a substantial subset of the larger system's features. This approach, called minimalism, made UNIX much tinier and lighter than other operating systems of that era, Multics included. Plan 9 employed this idea and other verified techniques of UNIX besides introducing novel methods to facilitate resource sharing and management in distributed environments.[2]

Current State

The most glorious moments in the Multics lifetime date back to the 1980s, when it was a lucrative product in European markets. In August 1985, Multics received the B2 certificate, a security grade for computer systems issued by NCSC, and for many years remained the only system that had reached this level of security.[19] Consequently, Multics has been used in many critical applications and environments, including the US Air Force, General Motors, Ford, Industrial Nucleonic, the University of Southwestern Louisiana, and the French university system. As mentioned previously, many of these applications were among the original motives for designing Multics and influenced its development in many ways.[10]

In the mid-1980s, Multics lost a great deal of its market share, as Honeywell couldn't provide convenient hardware for its users. In short, Multics lost its appeal for a great number of users who wanted to use faster processors. To revive its appeal, in April 1986 a group of engineers—including Vince Scarafino (Ford), Bob Kan (EDS), Paul Amaranth (Oakland University), Norm Poworz (Canadian Department of National Defense), and Norm Barnecut (Calgary University)—began to move Multics from its original hardware infrastructure to DPS 6 Plus. Later in 1987, Honeywell stopped supporting parts of Multics that were fundamental to this new infrastructure. Earlier in 1985, when it was the only profitable product in the system software market, Honeywell had stopped developing Multics.[11]

On 31 October 2000 the last running Multics system stopped performing at 17:08 in one of the facilities of the Canadian Department of National Defense in Halifax. Earlier, this system had been modified to support the Y2K standard, and had served as the Canadian Department of National Defense's main system until September 2000.[10] Although Multics was hardly successful in the public market, it has notably affected application and software development techniques, and its perspective is becoming more popular every day. Moreover, its descendants, such as UNIX, are still in progress and preserve its basic ideas in many aspects.[11]

On the other hand, Plan 9 is in a completely different situation, and its development has never stopped. As mentioned previously, Plan 9 is designed with regard to two fundamental concepts.[15] Using isolated namespaces for every group of processes, one of these concepts was a breakthrough in operating systems whose effects we can still perceive in today's systems.[16] This idea can significantly increase system scalability, which is defined as one of the basic features of distributed systems. This idea has been employed in many software systems thus far, and has potential and latent applications in today's systems, not least in the Hadoop Distributed File System. To get rid of the centralized metadata server in the HDFS architecture, called NameNode, Hadoop developers are trying to adapt the concept of Plan 9's namespaces in their storage clusters and improve its scalability, besides increasing its maximum number of storage nodes.[22,23] There have been other applications of Plan 9's namespaces, such as Helios, an operating system that provides its services as a collection of namespaces in a similar manner to Plan 9.[24]

The other principle in Plan 9's design is to provide uniform access to every computing system resource using a file-system-like interface. Today's operating systems, including both Linux and Mac OS, use this principle.[14] Additionally, the idea of presenting an interface to system resources using the 9P protocol and file servers has been employed in modern resource-management techniques. For example, Xcpu is a process management system for grid and cluster computing environments and has employed the general idea of the 9P protocol. In this system, users can access the process execution service provided by Xcpu by contacting a 9P server. Consequently, users can make their processes migrate to, and execute on, remote servers using file system interfaces.[25]

There are other dormant applications of Plan 9's techniques in operating and distributed computing systems. In Ashwin Ganti's work,[26] he introduces a way to employ Plan 9's concept of authentication servers to avoid some security vulnerabilities in Linux. These clues show that the old techniques of Plan 9 and Multics are still applicable and useful in the design of new systems.

Old Souls in New Bodies

Today's computing systems have inherited many ideas from Multics and Plan 9 and applied their novel and inventive techniques in various aspects of system design. Grid and cloud computing are among these environments that have attained considerable human and financial resources in the past decade. In early 2002, grid computing was introduced and became applicable after its fundamental ideas and protocols were implemented in Globus.[4] It chiefly provides an infrastructure to share computing resources on extra-large scales and solve computing problems cooperatively and in a distributed manner.[27] Multics introduced and employed distributed resource sharing, a basic object of grid design, for many years before the introduction of grid computing. C.R. Licklider and Robert Taylor introduced this concept to provide a similar service to public utilities, such as electric-light systems, and implemented it as a part of the Multics operating system in 1968.[27] Furthermore, grid developers claim that they have designed a self-configuring, self-tuning, and self-healing system and completely covered the ideas Corbato used to design Multics.

In addition, grid computing and Plan 9 have followed corresponding aims, including security, data management, and resource discovery, and are analogous in this way.[28] In the latter, 9P provides and facilitates distributed resource sharing, which is considered a basic feature in grid computing.[15,27] Moreover, grid computing employs grid services to provide naming mechanisms, remote process execution, communication protocol support, and execution environment virtualization, which Plan 9 implemented with namespaces and file-system-like interfaces.[16] In Andrey Mirtchovski and his colleagues' work, they compare Plan 9 with Globus and infer that, despite its having been designed for a single administration domain, Plan 9's simple and general design enables it to be extended and support new requirements of grid computing environments.[28]

Cloud computing is another modern computing environment, defined as a large-scale, distributed-computing paradigm providing remote users with a pool of abstract, virtualized, dynamically extendible, and managed resources based on their demands and available via the Internet. In this model, computing and storage power, besides platforms and services, are called computing resources.[7] This definition is focused on certain aspects of distributed systems, such as scalability and access transparency, and demonstrates that cloud configuration and service directly follows user demands.[6,7] Cloud is analogous to Plan 9 and Multics in various respects, and several research papers have been published to investigate similarities and differences in these systems. Chen, Paxson, and Katz discussed how cloud computing employs the concept of utility computing, which originally was proposed by Multics developers.[29] Moreover, these authors have demonstrated similarities between the security mechanisms of these two methods as well as the way in which Multics has influenced cloud computing. For example, many of the cloud's access control and authorization mechanisms have their origin in Multics security mechanisms, which administer user access with regards to their privileges. Furthermore, Multics security perspectives, such as diminishing reliance on the system administrator, have been followed by cloud developers and led to more autonomous security mechanisms, such as the Amazon EC2 Blacklist.[29]

As mentioned previously, the Multics payment methods of utility computing was adopted by cloud developers, and has led to the introduction of the pay-as-you-go model where cloud computing charges users for the amount of service they have received. In both payment methods, users are freed of maintenance and upgrade costs and simply pay their share. Furthermore, proper interfaces are embedded in cloud computing to provide users with transparent access to resources in the shortest time possible.[30]

Remote process execution is another principle in modern computing environments, such as cloud and grid computing, originally derived from Plan 9's execution model.[15] But the way in which Plan 9 remotely executes processes differs from today's systems, especially in cloud computing, in a few aspects. In this regard, the *cpu* command delivers a Plan 9 process to a specified remote computer for execution of a user's demand.[16] In contrast, cloud and grid computing can't rely on a single machine to perform required process computations, as that would entail distributed process execution. The main aim of employing this technique is to overcome the performance limitations of computer systems and provide high-performance computing facilities. Therefore, the *cpu* command must be modified to fit the new environment. In this manner, XCPU is an enhanced version of the *cpu* command that's tailored to today's applications, dividing a process into many threads to be distributed and executed on several machines.[25]

Virtualization techniques are among other prerequisites for cloud computing to automate management and allocation of resources to provide applications that vary in computing, storage, and communication models with the illusion of an infinite pool of resources. Virtualization is used, moreover, to improve various aspects of security in the cloud.[30] Similarly, Multics employed virtualization techniques to prevent hardware failure from interrupting system operations. Consequently, it can be inferred that what we call virtualization today is derived from the innovative ideas of Multics developers, such as pseudodevices, which are being used in today's virtual machines.[1]

In addition to the already mentioned points about how Multics and Plan 9 have influenced distributed systems, they have played an important role in forming developers' perspectives and roadmaps when designing the new generation of computing systems. File-mapped resources and hardware virtualization are among the innovations of these systems that system developers are still using, and show the importance of studying previous techniques and algorithms to design superior systems.

Studying the development history of these systems can help us apply their creative ideas—for example, Plan 9's namespaces in HDFS metadata servers—and avoid making similar mistakes—such as relying on a specific hardware architecture as did Multics. In short, instead of duplicating the same work over and over again, we can enhance the old approaches and make something that fulfills our needs while saving a great deal of time and money. We took a critical view in this article to magnify the advantages and drawbacks of the proposed ideas of these systems and the way in which they have affected the new generation of distributed systems. ■

References

1. J.H. Saltzer, *Traffic Control in a Multiplexed Computer System*, tech. report, Dept. of Electrical Eng., Massachusetts Inst. of Technology, 1966.
2. R. Pike et al., "Plan 9 from Bell Labs," *Proc. United Kingdom UNIX User Group Summer Proc.*, 1990, pp. 1–9.
3. G.K. Thiruvathukal, "Guest Editors' Introduction: Cluster Computing," *IEEE Computing in Science & Eng.*, vol. 7, no. 2, 2005, pp. 11–13; doi:10.1109/MCSE.2005.33.
4. I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," *Global Grid Forum*, vol. 22, 2002; http://toolkit.globus.org/alliance/publications/papers/ogsa.pdf.
5. M. Parashar et al., "Cloud Paradigms and Practices for Computational and Data-Enabled Science and Engineering," *IEEE Computing in Science & Eng.*, vol. 15, no. 4, 2013, pp. 10–18.
6. A.S. Tanenbaum and M.V. Steen, *Distributed Systems, Principles and Paradigms*, 2nd ed., Prentice Hall, 2006.
7. I. Foster et al., "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. Grid Computing Environments Workshop*, 2008, pp. 1–10.
8. C.J. Reynolds et al., "Scientific Workflow Makespan Reduction through Cloud Augmented Desktop Grids," *Proc. IEEE Int'l Conf. Cloud Computing Technology and Science* (CloudCom), 2011.
9. G. Juve et al., "Comparing FutureGrid, Amazon EC2, and Open Science Grid for Scientific Workflows," *IEEE Computing in Science & Eng.*, vol. 15, no. 4, 2013, pp. 20–29.
10. F.J. Corbato, J.H. Saltzer, and C.T. Clingen, "Multics—The First Seven Years," *Proc. Spring Joint Computer Conf.*, 1972.
11. F.J. Corbato and V.A. Vyssotsky, "Introduction and Overview of the Multics System," *Proc. Fall Joint Computer Conf.*, pt. I, 1965, pp. 185–196; doi:10.1145/1463891.1463912.
12. R.J. Feiertag and E.I. Organick, "The Multics Input/Output System," *ACM Sigops Operating Systems Rev.*, vol. 6, nos. 1/2, 1972, pp. 35–41.
13. D.M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Comm. ACM*, vol. 17, no. 7, 1974, pp. 365–375.
14. D. Presotto and P. Winterbottom, "The Organization of Networks in Plan 9," *Proc. Usenix Conf.*, 1993, pp. 271–280.
15. D. Presotto et al., "Plan 9, A Distributed System," *Proc. Spring 1991 EurOpen Conf.*, 1991, pp. 43–50.
16. R. Pike et al., "The Use of Name Spaces in Plan 9," *Proc. 5th ACM Sigops European Workshop*, 1993, pp. 72–76.
17. S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage," *Proc. FAST Conf. File and Storage*, 2002.
18. Bell Labs, *Plan 9 from Bell Labs*, 2009; http://plan9.bell-labs.com/plan9/about.html.
19. P.A. Karger and R.R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation,"

Proc. 18th Ann. Computer Security Applications Conf., 2002, pp. 119–126.

20. E. Mousavi Khaneghah and M. Sharifi, “AMRC: An Algebraic Model for Reconfiguration of High Performance Cluster Computing Systems at Runtime,” *The J. Supercomputing*, vol. 67, no. 1, 2013; doi:10.1007/s11227-013-0982-z.
21. R. Pike, “$8^{1/2}$, The Plan 9 Window System,” *Proc. Usenix Summer Conf.*, 1991; http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/operating/plan9-2.pdf.
22. K. Shvachko et al., “The Hadoop Distributed File System,” *Proc. 26th Symp. Mass Storage Systems and Technologies* (MSST), 2010, pp. 1–10.
23. E. Mousavi Khaneghah et al., “A Dynamic Replication Mechanism to Reduce Response-Time of I/O Operations in High Performance Computing Clusters,” *Proc. ASE/IEEE Int'l Conf. Big Data*, 2013, pp. 738–743.
24. E.B. Nightingale et al., “Helios: Heterogeneous Multiprocessing with Satellite Kernels,” *Proc. ACM Sigops 22nd Symp. Operating Systems Principles*, 2009, pp. 221–234.
25. R. Minnich and A. Mirtchovski, “XCPU: A New, 9p-Based, Process Management System for Clusters and Grids,” *Proc. IEEE Int'l Conf. Cluster Computing*, 2006, pp. 1–10.
26. A. Ganti, “Plan 9 Authentication in Linux,” *ACM Sigops Operating Systems*, vol. 42, no. 5, 2008, pp. 27–33.
27. I. Foster, “The Grid: A New Infrastructure for 21st Century Science,” *Physics Today*, vol. 55, no. 2, 2002, pp. 27–42.
28. A. Mirtchovski et al., “Plan 9—An Integrated Approach to Grid Computing,” *Proc. Parallel and Distributed Processing Symp.*, 2004; doi:10.1109/IPDPS.2004.1303349.
29. Y. Chen, V. Paxson, and R.H. Katz, “What's New About Cloud Computing Security?” tech. report UCB/EECS-2010-5, Electrical Eng. and Computer Science Dept., Univ. of California, Berkeley, 2010; www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.pdf.
30. M. Armbrust et al., “A View of Cloud Computing,” *Comm. ACM*, vol. 53, no. 4, 2010, pp. 50–58.

Seyedeh Leili Mirtaheri is a researcher at the Center of High-Performance Computing for Parallel and Distributed Processing, University of Calabria, Italy. Her research interests include distributed and parallel systems, peer-to-peer computing, cluster computing, mathematics, and scientific computing. Mirtaheri has a PhD in computer science engineering from the Iran University of Science and Technology (IUST). She received the First Award of Inventions at the 2011 National Science Foundation (NSF) Invention Festival and the 2009 IUST Award for Excellence in Researching. Contact her at mirtaheri@unical.it.

Ehsan Mousavi Khaneghah is a researcher at the Center of High-Performance Computing for Parallel and Distributed Processing, University of Calabria, Italy. His research interests include runtime management system software, distributed systems, cluster computing, and mathematics. Khaneghah has a PhD in computer science engineering from the IUST. He received the 2011 First Award of Inventions at the NSF Invention Festival and the 2009 IUST Excellent Researchers Award. Contact him at mousavi@unical.it.

Amir Saman Memaripour is a PhD student at the Department of Computer Science and Engineering at the University of California, San Diego. His research interests include distributed systems, high-performance computing, storage systems, network security, and Web technologies. Memaripour has an MS in computer engineering from IUST. Contact him at amemarip@eng.ucsd.edu.

Lucio Grandinetti is a full professor at the Faculty of Engineering, vice rector, PhD coordinator in management science and engineering, director of the Center of Excellence on High-Performance Computing, and co-managing director of the Supercomputing Center, at the University of Calabria, Italy. His research focuses on high-performance computing and grid computing. Grandinetti has a PhD in electronic engineering and systems science from the University of Pisa, Italy. Contact him at lucio@unical.it.

Mohsen Sharifi is a professor of software engineering at the School of Computer Engineering, Iran University of Science and Technology, where he also directs a distributed system software research group and laboratory. His research interests include the development of distributed systems, solutions, and applications, particularly for use in various fields of science. Sharifi has a PhD in computer science from the Victoria University of Manchester, UK. Contact him at msharifi@iust.ac.ir.

Zarrintaj Bornaee is an assistant professor at the Iranian Research Organization for Science and Technology. Her research interests include distributed computing systems, scalability, resource management, data mining, and information technology. Bornaee has a PhD in information technology from Jawaharlal Nehru University. Contact her at bornae@irost.ir.