DISTRIBUTED OPERATING SYSTEM- AN OVERVIEW

Rashmi Dewan, Nikita Pahuja, Shivangi Kukreja
Student, Computer Science & Engineering, Maharshi Dayanand University
Gurgaon, Haryana, India
rashmidewan9@gmail.com
nikpahuja28@gmail.com
shivangikukreja@gmail.com

ABSTRACT

The intention of this paper is to provide an overview on the subject of distributed operating system. The overview includes previous and existing concepts, current technologies. This paper also covers definition, overview, challenges in building dos, distributing computing models, design consideration, advantages of distributed operating systems, disadvantages of distributed operating system . Through this paper we are creating awareness among the people about this rising field of operating system. This paper also offers a comprehensive number of references for each concept of operating system.

KEYWORDS

- **DISTRIBUTED** – give a share or a unit of (something) to each of a number of recipients.
- **SYSTEM** – a set of things working together as parts of a mechanism or an interconnecting network; a complex whole.
- **PERFORMANCE** – an act of presenting a play, concert, or other form of entertainment.
- **PROTOCOL -** the official procedure or system of rules governing affairs of state or diplomatic occasions.

1) INTRODUCTION

A distributed operating system is an operating system that runs on several machines whose purpose is to provide a useful set of services, generally to make the collection of machines behave more like a single machine. The distributed operating system plays the same role in making the collective resources of the machines more usable that a typical single-machine operating system plays in making that machine's resources more usable. Usually, the machines controlled by a distributed operating system are connected by a relatively high quality network, such as a high speed local area network. Most commonly, the participating nodes of the system are in a relatively small geographical area, something between an office and a campus.
Distributed operating systems typically run cooperatively on all machines whose resources they control. These machines might be capable of independent operation, or they might be usable merely as resources in the distributed system. In some architectures, each machine is an equally powerful peer as all the others. In other architectures, some machines are permanently designated as master or are given control of particular resources. In yet others, elections or other selection mechanisms are used to designate

some machines as having special roles, often controlling roles.

Sometimes distinctions are made between parallel operating systems, distributed operating systems, and network operating systems, though the latter term is now a bit archaic. The distinctions are perhaps arbitrary, though they do point out differences in the design space for making operating systems control operations across multiple processing engines.

- A parallel operating system is usually defined as running on specially designed parallel processing hardware. It usually works on the assumption that elements of the hardware (such as the memory) are tightly coupled. Often, the machine is expected to be devoted to running a single task at very high speed.
- A distributed operating system is usually defined as runing on more loosely coupled hardware. Unlike parallel operating systems, distributed operating systems are intended to make a collection of resources on multiple machines usable by a set of loosely cooperating users running independent tasks.
- Network operating systems are sometimes regarded as systems that attempt merely to make the network connecting the machines more usable, without regard for some of the larger problems of building effective distributed systems.

Although many interesting research distributed operating systems have been built since the 1970s, and some systems have been in use for many years, they have not displaced traditional operating systems designed primarily to support single machines; however, some of the components originally built for distributed operating systems have become commonplace in today's systems, notably services to access files stored on remote machines. The failure of distributed operating systems to capture a large share of the marketplace may be primarily due to our lack of understanding on how to build them, or perhaps their lack of popularity stems from users not really needing many distributed services not already provided.

Distributed operating systems are also an important field for study because they have helped drive general research in distributed systems. Replicated data systems, authentication services such as Kerberos, agreement protocols, methods of providing causal ordering in communications, voting and consensus protocols, and many other distributed services have been developed to support distributed operating systems, and have found varying degrees of success outside of that field. Popular distributed component services like CORBA owe some of their success to applying hard lessons learned by researchers in distributed operating systems. increasingly, cooperative applications and services run across the Internet, and they face similar problems to those seen and frequently solved in the realm of distributed operating systems.

Distributed operating systems are hard to design because they face inherently hard problems, such as distributed consensus and synchronization. Further, they must properly trade off issues of performance, user interfaces, reliability, and simplicity. The relative scarcity of such systems, and the fact that most commercial operating systems' design still focuses on single-machine systems, suggests that no distributed operating system yet developed has found the proper trade-off among these issues.

Research continues in distributed operating systems, particularly in certain critical elements of them that have obvious value, especially file systems and other forms of data sharing. Other continuing research in distributed operating systems focuses on their use in important special cases, such as high-performance clustered servers and grid computing. Cloud computing is a recent development closely related to distributed operating systems. The increasing popularity of smart phones and tablets points out further need, if not for distributed operating systems, than at least for better methods to allow mobile devices to share their resources and work cooperatively. The emerging field of ubiquitous computing offers different hardware, networking, and application characteristics likely to spur further research on distributed operating systems. Peer systems, currently used primarily to share data, are also likely to spur further research in distributed operating systems issues. Sensor networks are another form of highly specialized distributed system that has benefited from the lessons of distributed operating systems.

2) OVERVIEW

At each locale (typically a node), the kernel provides a minimally complete set of node-level utilities necessary for operating a node’s underlying hardware and resources. These mechanisms include allocation, management, and disposition of a node’s resources, processes, communication, and input/output management support functions. Within the kernel, the communications sub-system is of foremost importance for a distributed OS.

In a distributed OS, the kernel often supports a minimal set of functions, including low-level address space management, thread management, and inter-process communication (IPC). A kernel of this design is referred to as a *microkernel*. Its modular nature enhances reliability and security, essential features for a distributed OS. It is common for a kernel to be identically replicated over all nodes in a system and therefore that the nodes in a system use similar hardware. The combination of minimal design and ubiquitous node coverage enhances the global system's extensibility, and the ability to dynamically introduce new nodes or services.

2.1) SYSTEM MANAGEMENT COMPONENTS

System management components are software processes that define the node's *policies*. These components are the part of the OS outside the kernel. These components provide higher-level communication, process and resource management, reliability, performance and security. The components match the functions of a single-entity system, adding the transparency required in a distributed environment.

The distributed nature of the OS requires additional services to support a node's responsibilities to the global system. In addition, the system management components accept the "defensive" responsibilities of reliability, availability, and persistence. These responsibilities can conflict with each other. A consistent approach, balanced perspective, and a deep understanding of the overall system can assist in identifying diminishing returns. Separation of policy and mechanism mitigates such conflicts.

2.2) WORKING TOGETHER AS AN OPERATING SYSTEM

The architecture and design of a distributed operating system must realize both individual node and global system goals. Architecture and design must be approached in a manner consistent with separating policy and mechanism. In doing so, a distributed operating system attempts to provide an efficient and reliable distributed computing framework allowing for an absolute minimal user awareness of the underlying command and control efforts.

The multi-level collaboration between a kernel and the system management components, and in turn between the distinct nodes in a distributed operating system is the functional challenge of the distributed operating system. This is the point in the system that must maintain a perfect harmony of purpose, and simultaneously maintain a complete disconnect of intent from implementation. This challenge is the distributed operating system's opportunity to produce the foundation and framework for a reliable, efficient, available, robust, extensible, and scalable system. However, this opportunity comes at a very high cost in complexity.

2.3) THE PRICE OF COMPLEXITY

In a distributed operating system, the exceptional degree of inherent complexity could easily render the entire system an anathema to any user. As such, the logical price of realizing a distributed operation system must be calculated in terms of overcoming vast amounts of complexity in many areas, and on many levels. This calculation includes the depth, breadth, and range of design investment and architectural planning required in achieving even the most modest implementation.

These design and development considerations are critical and unforgiving. For instance, a deep understanding of a distributed operating system's overall architectural and design detail is required at an exceptionally early point. An exhausting array of design considerations are inherent in the development of a distributed operating system. Each of these design considerations can potentially affect many of the others to a significant degree. This leads to a massive effort in balanced approach, in terms of the individual design considerations, and many of their permutations. As an aid in this effort, most rely on documented experience and research in distributed computing.

3) CHALLENGES IN BUILDING DISTRIBUTED OPERATING SYSTEM

One core problem for distributed operating system designers is concurrency and synchronization. These issues arise in single-machine operating systems, but they are easier to solve there. Typical single-machine systems run a single thread of control simultaneously, simplifying many synchronization problems. The advent of multicore machines is complicating this issue, but most multicore machines have relatively few cores, lessoning the problem. Further, they typically have shared access to memory, registers, or other useful physical resources that are directly accessible by all processes that they must synchronize. These shared resources allow use of simple and fast synchronization primitives, such as semaphores. Even modern machines that have multiple processors typically include hardware that makes it easier to synchronize their operations.

Distributed operating systems lack these advantages. Typically, they must control a collection of processors connected by a network, most often a local area network (LAN), but occasionally a network with even more difficult characteristics. The access time across this network is orders of magnitude larger than the access time required to reach local main memory and even more orders of magnitude larger than that required to reach information in a local processor cache or register. Further, such networks are not as reliable as a typical bus, so messages are more likely to be lost or corrupted. At best, this unreliability increases the average access time.

This imbalance means that running blocking primitives across the network is often infeasible. The performance implications for the individual component systems and the system as a whole do not permit widespread use of such primitives. Designers must choose between looser synchronization (leading to odd user-visible behaviors and possibly fatal system inconsistencies) and sluggish performance. The increasing gap between processor and network speeds suggests that this effect will only get worse.

Theoretical results in distributed systems are discouraging. Research on various forms of the Byzantine General problem and other formulations of the problems of reaching decisions in distributed systems has provided surprising results with bad implications for the possibility of providing perfect synchronization of such systems. Briefly, these results suggest that reaching a distributed decision is not always possible in common circumstances. Even when it is possible, doing so in unfavorable conditions is very expensive and tricky. Although most distributed systems can be designed to operate in more favorable circumstances than these gloomy theoretical results describe (typically by assuming less drastic failure modes or less absolute need for complete consistency), experience has shown that even pragmatic algorithm design for this environment is difficult.

A further core problem is providing transparency. Transparency has various definitions and aspects, but at a high level it simply refers to the degree to which the operating system disguises the distributed nature of the system. Providing a high degree of transparency is good because it shields the user from the complexities of distribution. On the other hand, it sometimes hides more than it should, it can be expensive and tricky to provide, and ultimately it is not always possible. A key decision in designing a distributed operating system is how much transparency to provide, and where and when to provide it.

A related problem is that the hardware, which the distributed operating system must virtualize, is more varied. A distributed operating system must not only make a file on disk appear to be in the main memory, as a typical operating system does, but must make a file on a different machine appear to be on the local machine, even if it is simultaneously being accessed on yet a third machine. The system should not just make a multi-machine computation appear to run on a single machine, but should provide observers on all machines with the illusion that it is running only on their machine.

Distributed operating systems also face challenging problems because they are typically intended to continue correct operation despite failure of some of their components. Most single-machine operating systems provide very limited abilities to continue operation if key components fail. They are certainly not expected to provide useful service if their processor crashes. A single processor crash in a distributed operating system should allow the remainder of the system to continue operations largely unharmed. Achieving this ideal can be extremely challenging. If the topology of the network connecting the system's component nodes allows the network to split into disjoint pieces, the system might also need to continue operation in a partitioned mode and would be expected to rapidly reintegrate when the partitions merge.

The security problems of a distributed operating system are also harder. First, data typically moves over a network, sometimes over a network that the distributed operating system itself does not directly control. This network may be subject to eavesdropping or malicious insertion and alteration of messages. Even if protected by cryptography, denial of service attacks may cause disconnections or loss of critical messages. Second, access control and resource management mechanisms on single machines typically take advantage of hardware that helps keep processes separate, such as page tables. Distributed operating systems cannot rely on this advantage. Third, distributed operating systems are typically expected to provide some degree of local control to users on their individual machines, while still enforcing general access control mechanisms. When an individual user is legitimately able to access any bytes stored anywhere on his own machine, preventing him from accessing data that belongs to others is a much harder problem, particularly if the system strives to provide controlled high-performance access to that data.

Distributed operating systems must often address the issue of local autonomy. In many (but not all) architectures, the distributed system is composed of workstations whose primary job is to support one particular user. The distributed system must balance the needs of the entire collection of supported users against the natural expectation that one's machine should be under one's own control. The local autonomy question has clear security implications, but also relates to how resources are allocated, how scheduling is done, and other issues.

In many cases, distributed operating systems are expected to run on heterogeneous hardware. Although commercial convergence on a small set of popular processors has reduced this problem to some extent, the wide variety of peripheral devices and customizations of system settings provided by today's operating systems often makes supposedly identical hardware behave radically differently. If a distributed operating system cannot determine whether running the same operation on two different component nodes produces the same result, it will face difficulties in providing transparency and consistency.

All the previously mentioned problems are exacerbated if the system scale becomes sufficiently large. Many useful distributed algorithms scale poorly, because the number of messages they require faces combinatorial explosion, or because the delays required to include large numbers of nodes in computations become unreasonable, or because data structures grow in proportion to the number of participants. High scale ensures that partial failures will become more common, and that low probability events will begin to pop up every so often. High scale might also imply that the distributed operating system must operate away from the relatively friendly world of the LAN, leading to greater heterogeneity and uncertainty in communications.

An entirely different paradigm of building system software for distributed systems can avoid some of these difficulties. Sensor networks, rather than performing general purpose computing, are designed only to gather information from sensors and send it to places that need it. The nodes in a sensor network are typically very simple and have low power in many dimensions, from CPU speed to battery. As a result, while inherently distributed systems, sensor network nodes must run relatively simple code. Operating systems designed for sensor networks, like TinyOS, are thus themselves extremely simple. By proper design of the operating system and algorithms that perform the limited applications, a sensor network achieves a cooperative distributed goal without worrying about many of the classic issues of distributed operating systems, such as tight synchronization, data consistency, and partial failure. This approach does not seem to offer an alternative when one is designing a distributed operating system for typical desktop or server machines, but may prove to be a powerful tool for other circumstances in which the nodes in the distributed systems need only do very particular and limited tasks.

Other limited versions of distributed operating system also avoid many of the worst difficulties faced in the general case. In cloud computing, for example, the provider of the cloud does not himself have to worry about maintaining transparency or consistency among the vast number of nodes he supports. His distributed systems problems are more limited, relating to management of large numbers of nodes, providing strong security between the users of portions of his system, ensuring fair and fast use of the network, and, at most, providing some basic distributed system primitives to his users. By expecting the users or middleware to customize the basic node operations he provides to suit their individual distributed system needs, the cloud provider offloads many of the most troublesome problems in distributed operating systems. Since the majority of users of cloud computing don't need those problems solved, anyway, this approach suits both the requirements of the cloud provider and the desires of the typical customer.

4) DISTRIBUTED COMPUTING MODELS

THREE BASIC DISTRIBUTIONS

To better illustrate this point, examine three system architectures; centralized, decentralized, and distributed. In this examination, consider three structural aspects: organization, connection, and control. Organization describes a system's physical arrangement characteristics. Connection covers the communication pathways among nodes. Control manages the operation of the earlier two considerations.

4.1) ORGANIZATION

A centralized system has one level of structure, where all constituent elements directly depend upon a single control element. A decentralized system is hierarchical. The bottom level unites subsets of a system’s entities. These entity subsets in turn combine at higher levels, ultimately culminating at a central master element. A distributed system is a collection of autonomous elements with no concept of levels.

4.2) CONNECTION

Centralized systems connect constituents directly to a central master entity in a hub and spoke fashion. A decentralized system (aka network system) incorporates direct and indirect paths between constituent elements and the central entity. Typically this is configured as a hierarchy with only one shortest path between any two elements. Finally, the distributed operating system requires no pattern; direct and indirect connections are possible between any two elements. Consider the 1970s phenomena of "string art" or a spirograph drawing as a fully connected system, and the spider's web or the Interstate Highway System between U.S. cities as examples of a partially connected system.

4.3) CONTROL

Centralized and decentralized systems have directed flows of connection to and from the central entity, while distributed systems communicate along arbitrary paths. This is the pivotal notion of the third consideration. Control involves allocating tasks and data to system elements balancing efficiency, responsiveness and complexity.
Centralized and decentralized systems offer more control, potentially easing administration by limiting options. Distributed systems are more difficult to explicitly control, but scale better horizontally and offer fewer points of system-wide failure. The associations conform to the needs imposed by its design but not by organizational limitations.

5) ADVANTAGES OF DISTRIBUTED OPERATNG SYSTEM

There are three important advantages in the design of distributed operating system:

5.1) MAJOR BREAKTHROUGH IN MICROPROCESSOR TECHNOLOGY

Micro- processors have become very much powerful and cheap, compared with mainframes and minicomputers, so it has become attractive to think about designing large systems consisting of small processors. These distributed systems clearly have a price/performance advantages over more traditional systems.

5.2) INCREMENTAL GROWTH

The second advantage is that if there is a need of 10 per cent more computing power, one should just add 10 per cent more processors. System architecture is crucial to the type of system growth, however, since it is hard to give each user of a personal computer another 10 per cent.

5.3) RELIABILITY

Reliability and availability can also be a big advantage; a few parts of the system can be down without disturbing people using the other parts; On the minus side, unless one is very careful, it is easy for the communication protocol overhead to become a major source of inefficiency.

6) DISADVANTAGES OF DISTRIBUTED OPERATING SYSTEM

Although distributed systems have their strengths, they also have their weaknesses. In this section, we will point out a few of them.

We have already hinted at the worst problem: software. With the current state-of-the-art, we do not have much experience in designing, implementing, and using distributed software. What kinds of operating systems, programming languages, and applications are appropriate for these systems? How much should the users know about the distribution? How much should the system do and how much should the users do? The experts differ (not that this is unusual with experts, but when it comes to distributed systems, they are barely on speaking terms). As more research is done, this problem will diminish, but for the moment it should not be underestimated.

A second potential problem is due to the communication network. It can lose messages, which requires special software to be able to recover, and it can become overloaded. When the network saturates, it must either be replaced or a second one must be added. In both cases, some portion of one or more buildings may have to be rewired at great expense, or network interface boards may have to be replaced (e.g., by fiber optics). Once the system comes to depend on the network, its loss or saturation can negate most of the advantages the distributed system was built to achieve.

Finally, the easy sharing of data, which we described above as an advantage, may turn out to be a two-edged sword. If people can conveniently access data all over the system, they may equally be able to conveniently access data that they have no business looking at. In other words, security is often a problem. For data that must be kept secret at all costs, it is often preferable to have a dedicated, isolated personal computer that has no network connections to any other machines, and is kept in a locked room with a secure safe in which all the floppy disks are stored. The disadvantages of distributed systems are summarized in Fig. 1-3.

Despite these potential problems, many people feel that the advantages outweigh the disadvantages, and it is expected that distributed systems will become increasingly important in the coming years. In fact, it is likely that within a few years, most organizations will connect most of their computers into large distributed systems to provide better, cheaper, and more convenient service for the users. An isolated computer in a medium-sized or large business or other organization will probably not even exist in ten years.

SUMMARY

A **distributed operating system** is a software over a collection of independent, networked, communicating, and physically separate computational nodes. Each individual node holds a specific software subset of the global aggregate operating system. Each subset is a composite of two distinct service provisioners. The first is a ubiquitous minimal kernel, or microkernel, that directly controls that node’s hardware. Second is a higher-level collection of system management components that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications.

The microkernel and the management components collection work together. They support the system’s goal of integrating multiple resources and processing functionality into an efficient and stable system. This seamless integration of individual nodes into a global system is referred to as transparency, or single system image; describing the illusion provided to users of the global system’s appearance as a single computational entity.

ACKNOWLEDGMENTS

We thank our guide for his timely help, giving outstanding ideas and encouragement to finish this research work successfully.

SIDE BAR

Comparison: it is an act of assessment or evaluation of things side by side in order to see to what extent they are similar or different. It is used to bring out similarities or differences between two things of same type mostly to discover essential features or meaning either scientifically or otherwise.

Content: The amount of things contained in something. Things written or spoken in a book, an article, a programme, a speech, etc.

REFERENCES

1) Tanenbaum, Andrew S (September 1993). "Distributed operating systems anno 1992. What have we learned so far?". *Distributed Systems Engineering* **1** (1). pp. 3–10. doi:10.1088/0967-1846/1/1/001.
2) Nutt, Gary J. (1992). *Centralized and Distributed Operating Systems*. Prentice Hall. ISBN 978-0-13-122326-4.
3) Gościński, Andrzej (1991). *Distributed Operating Systems: The Logical Design*. Addison-Wesley Pub. Co. ISBN 978-0-201-41704-3.
4) Fortier, Paul J. (1986). *Design of Distributed Operating Systems: Concepts and Technolog*. Intertext Publications.
5) Hansen, Per Brinch, ed. (2001). *Classic Operating Systems: From Batch Processing to Distributed Systems*. Springer. ISBN 978-0-387-95113-3.
6) Using LOTOS for specifying the CHORUS distributed operating system kernel Pecheur, C. 1992. Using LOTOS for specifying the CHORUS distributed operating system kernel. Comput. Commun. 15, 2 (Mar. 1992), 93-102.
7) COOL: kernel support for object-oriented environments Habert, S. and Mosseri, L. 1990. COOL: kernel support for object-oriented environments. In Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications (Ottawa, Canada). OOPSLA/ECOOP '90. ACM, New York, NY, 269-275.

RELATED REFERNCES

1) Li, K. and Hudak, P. 1989. Memory coherence in shared virtual memory systems. ACM Trans. Comput. Syst. 7, 4 (Nov. 1989), 321-359.
2) Garcia-Molina, H. and Salem, K. 1987. Sagas. In Proceedings of the 1987 ACM SIGMOD international Conference on Management of Data (San Francisco, California, United States, May 27–29, 1987). U. Dayal, Ed. SIGMOD '87. ACM, New York, NY, 249-259.
3) Harris, T., Marlow, S., Peyton-Jones, S., and Herlihy, M. 2005. Composable memory transactions. In Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Chicago, IL, USA, June 15–17, 2005). PPoPP '05. ACM, New York, NY, 48-60.
4) Herlihy, M. and Moss, J. E. 1993. Transactional memory: architectural support for lock-free data structures. In Proceedings of the 20th Annual international Symposium on Computer Architecture (San Diego, California, United States, May 16–19, 1993). ISCA '93. ACM, New York, NY, 289-300.
5) Herlihy, M., Luchangco, V., Moir, M., and Scherer, W. N. 2003. Software transactional memory for dynamic-sized data structures. In Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing (Boston, Massachusetts, July 13–16, 2003). PODC '03. ACM, New York, NY, 92-101.
6) Shavit, N. and Touitou, D. 1995. Software transactional memory. In Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing (Ottawa, Ontario, Canada, August 20–23, 1995). PODC '95. ACM, New York, NY, 204-213.
7) Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., and Zhao, B. 2000. OceanStore: an architecture for global-scale persistent storage. In Proceedings of the Ninth international Conference on Architectural Support For Programming Languages and Operating Systems

(Cambridge, Massachusetts, United States). ASPLOS-IX. ACM, New York, NY, 190-201.