Distributed Operating Systems

SAPE J. MULLENDER

University of Twente, Enschede, The Netherlands ⟨mullender@cs.utwente.ne⟩

Distributed systems came hand in hand with workstations, personal computers, and local networks. The possibility of interconnecting computers in a local network created new opportunities for unrestricted data sharing while giving each user the dedicated computing cycles needed for interactive graphical user interfaces. It provided an opportunity to make systems that were more reliable than their hardware components. It allowed incremental system growth by adding or replacing individual components [Schroeder 1993].

Exploiting these new opportunities became the new field of distributed and network operating systems research. The first projects were started in the middle seventies, but the bulk of activity in the field took place in the eighties. Now, halfway through the nineties, activity in distributed systems research appears to be declining.

A distinction between *distributed* operating systems and *network* operating systems has sometimes been made. A network operating system is essentially a centralized operating system whose components have been distributed over multiple nodes, while a distributed system is one in which this distribution is combined with replication to achieve fault tolerance as well as performance.

Another distinction is necessary between *distributed* systems and *parallel* systems. In parallel-systems research, the focus is on completing large computations in minimal time, by exploiting the presence of multiple processing nodes. Distributed systems also exploit parallelism, but the main focus is on fault tolerance.

Most computers are connected to networks now so that all systems are becoming, to a greater or lesser extent, distributed. Most system builders, therefore, need some knowledge of distributed systems, and distributed-systems research is becoming mixed with other research areas.

A major—probably *the* major—motivation for distributed systems research used to be the quest for dependable systems, systems that would tolerate failures in order to become more reliable than their parts. This quest has largely succeeded in that we now have a wide range of techniques and algorithms that work.

However, the subsequent integration of such techniques and algorithms in everyday systems has largely failed. I find two important causes for this. One is the reliability of current computer hardware, the other is the difficulty of changing systems that have become accepted as standards.

Computer hardware is now very reliable. Disk manufacturers claim mean times between failure of a million hours and more, so that very few disks ever fail during their operational lifetime. Because of this, in most situations there is little need for replicated data storage. The extra complexity of the software might actually reduce the reliability of a system due to the presence of more bugs.

Highly distributed services, such as electronic mail and the Domain Name Service, have their specialized fault-tolerance mechanisms. In the World-Wide Web no fault tolerance exists at the moment, but some replication will be

introduced in the next few years. It appears that only a small set of specialized applications and application domains needs mechanisms that provide reliability beyond what networked but non-distributed systems can give today.

The other reason is that the world is currently burdened with a few operating-system standards that cannot easily be extended with fault-tolerance mechanisms without major change. There is such an investment in existing software that any short-term changes are unlikely. In any case, the world's most widely used operating systems have many more urgent problems that need solving before increased tolerance of hardware failures will become noticeable.

Although distributed-system techniques have not caught on very much in local-network environments, they have become essential in far-flung applications such as electronic mail and naming systems. Another area in which fault-tolerance techniques have caught on is databases. Many large database systems are now distributed and use techniques such as logging and two-phase commit to make distributed transactions atomic and fault-tolerant [Lynch et al. 1993].

However, no matter how fault-tolerant an algorithm, it will fail if sufficiently many sufficiently severe failures occur. When designing a distributed system, it is therefore essential to use a failure model that describes the kind and number of failures that the system must be able to mask. An assumption in the most commonly used failure models in distributed systems is that the system is asynchronous; that is, that messages can take an arbitrarily long time to reach their destination and that processors, due to their load, can take arbitrarily long to complete a task. In an asynchronous system, one cannot rely on timing. Although the hardware out of which distributed systems are built is quite synchronous, the assumption of asynchrony is a realistic one in most environments, because the operating-system software uses time sharing, thus virtualizing time.

This creates an important contrast between most distributed systems and distributed real-time systems. In real-time systems, guarantees must be provided that deadlines are met, and this requires a synchronous system base: message delays and processing times must be bounded. Since this difference is fundamental, research on dependable real-time systems has become an almost separate branch of distributed-systems research.

Name servers use a naming database to map names to services and objects in a distributed system. Since objects and services are almost always referred to by name, unavailability of the name service renders everything else in the system unavailable as well.

Name servers must combine a number of properties, each of which is difficult to achieve: they must be highly available, which implies replication and fault tolerance; they must be scalable to a very large size; and they must be correct and up-to-date.

Unfortunately, real networks exhibit failures in which these properties cannot all be met simultaneously. Networks sometimes become partitioned—some machines can no longer communicate with others. As a result, updates in one part of the system cannot reach another, so that, at least in parts of the network, the naming database can no longer be correct and up-to-date. This can be prevented from happening only by forbidding updates to the naming database while partitions are present, but this reduces the availability of the name service.

The most widely used name service today is the Domain Name Service [RFC-1035]. It sacrifices correctness and up-to-dateness for availability. As a result, it is possible that a name will be incorrectly mapped. However, as Needham [1993] points out, "Naming data doesn't change very fast, so inconsistencies will be rare; you will find that something doesn't work if you try to use

obsolete naming data, so you can attend to it; and, even if you don't find out, and use obsolete data, it doesn't matter much *sub specie æternitatis*."

The advent of multimedia may form a new source of inspiration to distributed-systems research [Mullender et al. 1994]. The real-time nature of processing audio and video requires a synchronous systems platform in which, with high probability, processing and message-passing deadlines can be met. The mechanisms that support multimedia will support fault tolerance as well: a synchronous system can mask more failures than an asynchronous one.

REFERENCES

LYNCH, N. A., MERRITT, M., WEIHL, W. E., AND FEKETE, A. 1993. *Atomic Transactions*. Morgan-Kaufmann, San Mateo, CA.

MULLENDER, S. J., LESLIE, I. M., AND MCAULEY, D. 1994. Operating-system support for distributed multimedia. In *Proceedings of the Summer Usenix Conference*, Boston, MA, 209–220.

NEEDHAM, R. M. 1993. Names. In *Distributed Systems*, 2nd ed. S. J. Mullender, Ed. ACM Press, New York, 315–327.

RFC-1035. *Domain Names—Implementation and Specification*.

SCHROEDER, M. D. 1993. A state-of-the-art distributed system: Computing with BOB. In *Distributed Systems*, 2nd ed. S. J. Mullender, Ed. ACM Press, New York, 1–16.