Editorial: Distributed Systems

Distributed Systems as a subject of academic study dates from the later 1970s. Over the past decade it has become a very broad field as illustrated by the range of topics embraced in this special issue. The theme common to all the papers is the design of systems which include explicit communications between separate computers – in some cases across networks, in others between closely connected processors in a single machine or closely coupled configuration of machines.

Distributed systems are now firmly part of the commercial scene. This has come about because of the widespread availability of high performance commodity processors and networking interfaces. The availability of powerful processor chips has moved the ownership of instruction sets away from the computer vendors to the silicon companies. The computer vendors are increasingly competing on the basis of system architecture: distributed systems provide a significant degree of flexibility in cost-performance trade-offs. The local networking revolution has made the cost of networking machines in the same office trivial; developments in the field of telecommunications are now putting in place the equivalent wide-area capabilities.

The challenge of distributed systems comes from the fact that they have some additional properties to the sequential processor model which has been the foundation for application design since the dawn of the digital age. Distributed systems have shown that some rather grand simplifying assumptions are no longer true:

- *everything is in one place*: many components of a distributed system will be remote; latency in access time may be variable, and transient failures may prevent access; while the details of differences in communication protocols can be hidden, the potential for delay and failure cannot
- *data can be accessed directly*: data may be in a remote computer and held in a different format; this requires a procedural view of all interfaces rather than a data-centred view
- *everything is sequential*: in a distributed system, there are many computers and overlapped execution is inevitable – this concurrency can be exploited to increase performance, but has also to be controlled if a system is to operate consistently
- *everything is synchronous*: achieving ordering in distributed systems is one of the challenging research topics; it cannot be assumed that something happening in one component excludes something else happening elsewhere in the system
- *everything is homogeneous*: in the general case a distributed system will include examples of different processors, different data formats, different programming languages and different network protocols; abstractions are needed to mask these from the applications programmer; mechanisms are needed to bridge between systems that show different technology choices
- *there is one of everything*: replication of critical components can be exploited for increased availability and fault tolerance because of the inherent redundancy of distributed systems
- *everything stays in one place*: components can migrate in a distributed system, to balance load, to reduce access times or to reflect changes in configuration to meet new operational needs
- *memory is a common resource*: in a distributed system memory is disjoint because of the introduction of communications; while there are schemes for giving the illusion of shared memory in a distributed system, they are not general enough to scale up to worldwide systems of heterogeneous machines
- *there is a global name space*: as soon as two computers are connected, their name spaces have to be linked; the same is true for distributed systems. Where systems belong to different organizations there will be a tension between the desire to cooperate and the need to preserve autonomy of control; this predicates against global management and requires a more federal approach.

The papers selected for this issue look at a number of these changing assumptions:

Daszuk and Son & Paker explore closely coupled distributed systems. Daszuk proposes a structured approach to the organization of operating systems for distributed systems. Son and Paker look at the problem of routing messages across networks of Transputer systems that avoid deadlock problems arising from one message stuck in the network blocking the progress of others.

Mancini & Shrivastava, and Dollimore & Miranda look at issues arising in object-oriented systems based on local or wide area networks. Mancini and Shrivastava explore how to deal with the problem of recovering storage assigned to objects which are no longer referenced by any computer in a network in a way which is tolerant of communication failures and processor crashes. Dollimore and Miranda are concerned with how distributed applications can share common objects in office automation applications.

Verrall looks at the problems of heterogeneity in distributed systems: how can one overcome differences in data representation, differences in access procedures, particularly in a situation where the focus is on re-using existing components with minimal change.

Agrawal & Malpani look at a topic of importance in synchronization and replication; they present efficient techniques for the propagation of events and their logical timing amongst interested sites.

Geesink and Boderik & Riordon explore two very practical problems. Geesink describes procedures for passing active messages (ones that cause other messages to be sent and data to be updated) in a coordinated way that is resilient to failures of nodes in the network. Boderik & Riordon look at strategies for organizing queries to minimize latency in a distributed database environment.

From all of these papers emerges the message that changed assumptions pose a challenge for the designers of distributed systems – how much should they expose the application programmer to the new assumptions?

Some people have proposed *transparency* as a panacea, requiring that operating systems preserve the old assumptions and provide mechanisms to make this true. Others say expose the distribution to the application programmer, but in a way in which the costs and benefits are emphasized, and low-level details are simplified.

It will be interesting to see how this debate is resolved in the 1990's as distributed systems technology becomes more widespread.

ANDREW HERBERT