RESEARCH PROBLEMS OF DECENTRALIZED SYSTEMS WITH LARGELY AUTONOMOUS NODES

by

Jerome H. Saltzer
Massachusetts Institute of Technology

A currently popular systems research project is to explore the possibilities and problems for computer system organization that arise from the rapidly falling cost of computing hardware. Interconnecting fleets of mini- or micro-computers and putting intelligence in terminals and concentrators to produce so-called "distributed systems" has recently been a booming development activity. While these efforts range from ingenious to misguided, many seem to miss a most important aspect of the revolution in hardware costs: that more than any other factor, the entry cost of acquiring and operating a free-standing, complete computer system has dropped and continues to drop rapidly. Where a decade ago the capital outlay required to install a computer system ranged from $150,000 up into the millions, today the low end of that range is below $15,000 and dropping.

The consequence of this particular observation for system structure comes from the next level of analysis. In most organizations, decisions to make capital acquisitions tend to be more centralized for larger capital amounts, and less centralized for smaller capital amounts. On this basis we may conjecture that lower entry costs for computer systems will lead naturally to computer acquisition decisions being made at points lower in a management hierarchy. Further, because a lower-level organization usually has a smaller mission, those smaller-priced computers will tend to span a smaller range of applications, and in the limit of the argument will be dedicated to a single application. Finally, the organizational units that acquire these computers will by nature tend to operate somewhat independently and autonomously from one another, each following its own mission. From another viewpoint, administrative autonomy is really the driving force that leads to acquisition of a computer system that spans a smaller application range. According to this view, the large multiuser computer center is really an artifact of high entry cost, and does not represent the "natural" way for an organization to do its computing.

A trouble with this somewhat oversimplified analysis is that these conjectured autonomous, decentralized computer systems will need to communicate with one another. For example: the production department's output will be the inventory control department's input, and computer-generated reports of both departments must be submitted to higher management for computer analysis and exception display. Thus we can anticipate that the autonomous computer systems must be at least loosely

---

© 1977 by J. H. Saltzer. All rights reserved.

coupled into a cooperating confederacy that represents the corporate information system. This scenario describes the corporate computing environment, but a similar scenario can be conjectured for the academic, government, military, or any other computing environment. The conjecture described here has been explored for validity in an undergraduate thesis [d'Oliveira, 1977].

The key consequence of this line of reasoning for computer system structure, then, is a technical problem: to provide coherence in communication among what will inevitably be administratively autonomous nodes of a computer network. Technically, autonomy appears as a force producing incoherence: one must assume that operating schedules, loading policy, level of concern for security, availability, and reliability, update level of hardware and software, and even choice of hardware and software systems will tend to vary from node to node with a minimum of central control. Further, individual nodes may for various reasons occasionally completely disconnect themselves from the confederacy, and operate in isolation for a while before reconnecting. Yet to the extent that agreement and cooperation are beneficial, there will be need for communication of signals, exchange of data, mutual assistance agreements, and a wide variety of other internode interaction. One-at-a-time ad hoc arrangements will probably be inadequate, because of their potential large number and the programming cost in dealing with each node on a different basis.

Coherence can be sought in many forms. At one extreme, one might set a company-wide standard for the electrical levels used to drive point-to-point communication lines that interconnect nodes or that attach any node to a local communication network. At the opposite extreme, one might develop a data management protocol that allows any user of any node to believe that there is a central, unified database management system with no identifiable boundaries. The first extreme might be described as a very low-level protocol, the second extreme as a very high-level protocol, and there seem to be many levels in between, not all strictly ordered.

By now, considerable experience has been gained in devising and using relatively low-level protocols, up to the point that one has an uninterpreted stream of bits flowing from one node of a network to another [Cerf, 1974]. The ARPANET and TELENET are perhaps the best-developed examples of protocols at this level, and local networks such as the ETHERNET [Metcalfe, 1975] provide a similar level of protocol on a smaller scale geographically. In each of those networks, standard protocols allow any two autonomous nodes (of possibly different design) to set up a data stream from one to the other; each node need implement only one protocol, no matter how many other differently designed nodes are attached to the network. However, standardized coherence stops there; generally each pair of communicating nodes must make some (typically ad hoc) arrangement as to the interpretation of the stream of bits: it may represent a stream of data, a set of instructions, a message to one individual, or something else. For several special cases, such as exchange of mail or remotely submitting batch jobs, there have been developed higher-level protocols; there tends to be a distinct ad hoc higher-level protocol invented for each application [Feinler, 1976]. Some workers have explored the problems of protocols that interpret and translate data across machines of different origin [Levine, 1977]. Others have tried to develop a network-wide file system without user-noticeable boundaries [Thomas, 1973; Millstein, 1976].

The image of a loose confederacy of cooperating autonomous nodes requires at a minimum the level of coherence provided by these networks; it is not yet clear how much more is appropriate, only that the opposite extreme in which the physically separate nodes effectively lose their separate identity is excluded by the earlier arguments for autonomy. Between lies a broad range of possibilities that need to be explored.

Coherence and the object model

The first problem is to develop a framework for discussion that allows one to pose much more specific questions. As a way to put some structure on the range of possibilities, it is appropriate to think first in terms of familiar semantic models of computation, and then to inquire how the semantic model of the behavior of a single node might be usefully extended to account for interaction with other, autonomous nodes. To get a concrete starting point that is as developed as possible, let us give initial consideration to the object model [Liskov, 1975; Liskov, 1977; Wulf, 1976]*. Under that view, each node is a self-contained system with storage, a program interpreter that is programmed in a high-level object-oriented language such as CLU or Alphard, and an attachment to a data communication network of the kind previously discussed.

We immediately observe that several interesting problems are posed by the interaction between the object model and the hypothesis of autonomy. There are two basic alternative premises that one can start with in thinking about how to compute with an object that is represented at another node: send instructions about what to do with the object to the place it is stored; or send a copy of the representation of the object to the place that wants to compute with it. (In-between combinations are also possible, but conceptually it is simpler to think about the extreme cases first.) An initial reaction might be to begin by considering the number of bits that must be moved from one node to another to carry out the two alternatives, but that approach misses the most interesting issues: reliability, integrity, responsibility for protection of the object, and naming problems. Suppose the object stays in its original home. Semantics for requesting operations, and reporting results and failures are needed. For some kinds of objects, there may be operations that return references to other, related objects. Semantics to properly interpret these references are required. Checking of authorization to request operations is required. Some way must be found for the (autonomous) node to gracefully defer, queue, or refuse requests, if it is overloaded or not in operation at the moment.

Suppose on the other hand, that a copy of the object is moved to the node that wants to do the computation. Privacy, protection of the contents, integrity of the representation, and proper interpretation of names embedded

---

* Two other obvious candidates for starting points are the data flow model [Dennis, 1975; Arvind, 1976] and the actor model [Hewitt, 1977], both of which already contain the notion of communications; since neither is developed quite as far as the object model we leave them for future examination.

in the object representation all are problems. Yet, making copies of data seems an essential part of achieving autonomy from nodes that contain needed information but aren't always accessible. Considering these two premises as alternatives seems to raise simultaneously so many issues of performance, integrity of the object representation, privacy of its content, what name is used for the object, and responsibility for the object, that the question is probably not posed properly. However, it begins to illustrate the range of considerations that should be thought about. It also suggests the following, more specific, problems that require solutions:

1. To arrange systematically that an object have multiple representations at one point in time but stored at different places. One would expect to achieve reliability and response speed this way [Alsberg, 1976]. An example of non-systematic multiple representation occurs whenever one user of a time-sharing system confronts another with the complaint, "I thought you said you fixed that bug", and receives the response, "I did. You must have gotten an old copy of the program. What you have to do is type..." Semantics are needed to express the notion that for some purposes any of several representations are equally good, but for other purposes they aren't.

2. An object at one node needs to "contain" (for example, use as part of its representation) objects from other nodes. This idea focuses on the semantics of naming remote objects. It is not clear whether the names involved should be relatively high-level (e.g., character-string file names) or low-level (e.g., segment numbers). Ideas involving the interaction among semantics and mechanics of naming in very large address spaces may turn out to have application to the decentralized case [Bishop, 1977].

3. Related to the previous problem are issues of object motion: suppose object A, which contains as a component object B, is either copied or moved from one node to another, either temporarily or permanently. Can object B be left behind or be in yet another node? The answer may depend on the exact combination of copy or new, temporary or permanent. Autonomy is deeply involved here, since one cannot rely on availability of the original node to resolve the name of B. The Distributed Computing System (DCS) at the University of California, Irvine, provided a first cut trial of this idea by arranging for processes to move from one node to another without having to change their names [Farber, 1972].

4. More generally, semantics are needed for gracefully coping with objects that aren't there when they are requested. (Information stored in autonomous nodes will often fall in this category.) This idea seems closely related to the one of coping with objects that have multiple versions and the most recent version is inaccessible*.

---

* Semantics for dealing systematically with errors and other surprises have not really been devised for monolithic, centralized systems either. However, it appears that in the decentralized case, the problem cannot so easily be avoided by ad hoc tricks or finesse as it was in the past.

5. Algorithms are needed that allow atomic update of two (or more) objects stored at different nodes, in the face of errors in communication and failures of individual nodes*. There are several forms of atomic update: there may be consistency constraints across two or more different objects (e.g., the sum of all the balances in a bank should always be zero) or there may be a requirement that several copies of an object be kept identical. Process coordination semantics that were adequate for single-node systems do not necessarily stand up under the pressures of robustness and delay of the decentralized system. Reed and Kanodia have developed a promising semantics together with an implementation model in terms of messages [Reed, 1977]. The semantic view that objects are immutable may provide a more hospitable base for extension to interaction among autonomous nodes than the view that objects ultimately are implemented by cells that can contain different values at different times. (The more interesting algorithms for making coordinated changes in the face of errors seem to implement something resembling immutable objects [Lampson, 1976; Thomas, 1976]).

Constraining the range of errors that must be tolerated seems to be a promising way to look at these last two problems. Not all failures are equally likely, and more important, some kinds of failures can perhaps be guarded against by specific remedies, rather than tolerated. For example, a common protocol problem in a network is that some node both crashes and restores service again before anyone notices; outstanding connections through the network sometimes continue without realizing that the node's state has been reset. Careful choice in the semantics of the host-net interface can locally eliminate this kind of failure instead of leaving it as a problem for higher level protocols.

The following oversimplified world view, to be taken by each node may offer a systematic way to think about multiply represented objects and atomic operations: there are two kinds of objects, mine and everyone else's. My node acts as a cache memory for objects belonging to others that I use, and everyone else acts as a backing store. These roles are simply reversed for my own objects. (One can quickly invent situations where this view breaks down, causing deadlocks or wrong answers, but the question is whether or not there are real world problems for which this view is adequate.)

Finally, it is apparent that one can get carried away with generalized algorithms that handle all possible cases. An area requiring substantial investigation is real world applications. It may turn out that only a few of these issues arise often enough in practice to require systematic solutions. It may be possible, in many cases, to cope with distant objects quite successfully as special cases to be programmed one at a time. For example,

---

* Most published work on making atomic updates to several objects or several sites has concentrated on algorithms that perform well despite communication delay or that can be proven correct [Lamport, 1976 Acta Inf.; Stearns, 1976; Eswaran, 1976; Ellis, 1976; Rothnie, 1977]. Unfortunately, algorithms constructed without consideration of reliability and failure are not easily extended to cope with those additional considerations, so there seems to be no way to build on that work.

recent measurements on the Multics system suggest that even though that system is designed for maximum ease in sharing data, actual use of the facilities for shared writable objects is mostly quite stylized: the full generality is exploited quite rarely [Montgomery, 1977].

Other problems in the semantics of coherence

Usual models of computation permit only "correct" results, with no provision for tolerating "acceptably close" answers. Sometimes provision is made to report that no result can be returned. In a loose confederacy of autonomous nodes, exactly correct results may be unattainable, but no answer at all is too restricting. For example, one might want a count of the current number of employees, and each department has that number stored in its computer. At the moment the question is asked, one department's computer is down, and its count is inaccessible. But a copy of last month's count for that department is available elsewhere. An "almost right" answer utilizing last month's count for one department may well be close enough for the purpose the question was asked, but we have no semantics available for requesting or returning such answers. A more extreme example would be if the Federal Reserve tried to determine the money supply by interrogating every bank's computer so as to obtain the sum of all checking account balances in the United States. Obtaining an exact result seems unrealistic as well as unnecessary.

A general solution to the problem of providing acceptably close answers seems to require a perspective from Artificial Intelligence, but particular solutions may be programmable if there were available semantics for detecting that one object is an out-of-date version of another, or that a requested but unavailable object has an out-of-date copy. It is not clear at what level these associations should be made.

Semantics are also needed to express constraints or partial contraints of time sequence. (e.g., "reservations are to be made in the order they are requested, except that two reservation requests arriving at different nodes within one minute may be processed out of order.") Lamport has suggested one approach to thinking about this problem [Lamport, March, 1976]. Note that the possibility of unreliable nodes or communications severely complicates this problem.

The semantics of protection of information, just beginning to be understood in the single system case, are a bewildering maze when one considers the decentralized system. The capability model seems to offer little help when the capabilities must be sent from one node to other, potentially hostile ones, since one can't be sure that the capability won't be tampered with. Nevertheless, the capability model may be useful for cases where the nodes are known to be friendly [Donnelley, 1976]. Cryptographic techniques seem to offer some aid in authentication and in protecting control signals in addition to their traditional use in protecting data in transit [Branstad, 1975; Diffie, 1976; Kent, 1976]. Application of information flow models to decentralized systems is a promising idea [Karger, 1977].

The semantics of autonomy are not clear. When can I disconnect my node from the network without disrupting my (or other) operations? How do I refuse to report information that I have in my node in a way that is not disruptive? If my node is overloaded, which requests coming from other nodes can be deferred without causing deadlock? Early work in this area on DCS points the way, but needs to be extended to more cases [Rowe, 1973].

Heterogeneous and Homogeneous Systems

A question that is immediately encountered is whether or not one should assume that the various autonomous nodes of a loosely coupled confederacy of systems are identical either in hardware or in lower level software support. The assumption of autonomy and observations of the way the real world behaves both lead to a strong conclusion that one must be able to interconnect heterogeneous (that is, different) systems. Yet, to be systematic, some level of homogeneity is essential, and in addition the clarity that homogeneity provides in allowing one to see a single research problem at a time is very appealing.

It may be that the proper approach to this issue lies in careful definition of node boundaries. Suppose that we insist that every node present to every other node a common, homogeneous interface, whose definition we hope to specify. That interface may be a native interface, directly implemented by the node, or it may be simulated by interpretation, using the (presumably different) native facilities of the node. This approach allows one to work on the semantics of decentralized systems without the confusion of hetrogeneity, yet it permits at least some non-conforming systems to participate in a confederacy. There is, of course, no guarantee that an arbitrary previously existing computer system will be able to simulate the required interface easily or efficiently.

Conclusion

The various problems suggested here are by no means independent of one another, although each seems to have a flavor of its own. In addition, they probably do not span the complete range of issues that should be explored in establishing an appropriate semantics for expressing computations in a confederacy of loosely coupled, autonomous computer systems. Further, some are recognizable as problems of semantics of centralized systems that were never solved very well. But they do seem to represent a starting point that can to lead to more carefully framed questions and eventually some new conceptual insight.

Acknowledgement

Many of the ideas discussed here were suggested by David D. Clark, David P. Reed, Liba Svobodova, and students in an M.I.T. graduate seminar held during the Spring Semester, 1976-77.

References

Alsberg, P.A., Belford, G.G., Day, J.D., and Grapa, E., "Multi-Copy Resiliency Techniques," University of Illinois Center for Advanced Computation Document #202, May, 1976.

Arvind, et al., "Programming in a viable data flow language," Univ. of Calif. (Irvine) Department of Information and Computer Science, Technical Report 89.

Bishop, P.B., "Computer Systems with a Very Large Address Space and Garbage Collection," Ph.D. thesis, M.I.T. Department of Electrical Engineering and Computer Science, May, 1977, also Laboratory for Computer Science Technical Report TR-178.

Branstad, D.K., "Encryption Protection in Computer Data Communications," Proc. Fourth Data Communications Symposium, Quebec, October, 1975, pp. 8.1-8.7.

Cerf, V.G., and Kahn, R.E., "A Protocol for Packet Network Interconnection," IEEE Trans. on Communications 22, 5 (May, 1974) pp. 637-648.

d'Oliveira, C., "A Conjecture About Computer Decentralization," B.S. thesis, M.I.T. Department of Electrical Engineering and Computer Science, August, 1977.

Dennis, J.B., "First Version of a Data Flow Procedure Language," M.I.T. Laboratory for Computer Science Technical Memo TM-61, May, 1975.

Diffie, W., and Hellman, M.E., "New Directions in Cryptography," IEEE Trans. on Info. Theory 22, 6 (November, 1976) pp. 644-654.

Donnelley, J.E., "A Distributed Capability Computing System (DCCS)," ARPANET Request for Comments #712, Network Information Center, Stanford Research Institute, Menlo Park, California, February, 1976.

Ellis, C.A., "Consistency and Correctness of Duplicate Database Systems," Sixth ACM Symposium on Operating System Principles, November, 1977, pp. 67-84.

Eswaran, K.P., et al., "The Notions of Consistency and Predicate Locks in a Database System," Comm. of ACM 19, 11 (November, 1976) pp. 624-633.

Farber, D.J., and Heinrich, F.R., "The Structure of a Distributed Computer System: The Distributed File System," Proc. First Int. Conf. on Computer Comm., 1972, pp. 364-370.

Feinler, E., and Postel, J., ARPANET Protocol Handbook, NIC 7104, Network Information Center, Stanford Research Institute, Menlo Park, California, April, 1976.

Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages," to be published in A.I. Journal.

Karger, P., "Non-Discretionary Access Control for Decentralized Computing Systems," M.S. thesis, M.I.T. Department of Electrical Engineering and Computer Science, May, 1977, also Laboratory for Computer Science Technical Report TR-179.

Kent, S.T., "Encryption-Based Protection Protocols for Interactive User-Computer Communication," S.M. thesis, M.I.T. Department of Electrical Engineering and Computer Science, May, 1976, also Laboratory for Computer Science Technical Report TR-162.

Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Mass. Computer Associates Technical Report CA-7603-2911, March, 1976.

Lamport, L., "The Synchronization of Independent Processes," Acta Informatica, 7, 1976, pp. 15-34.

Lampson, B., and Sturgis, H., "Crash Recovery in a Distributed Data Storage System," to be published in the Comm. of ACM.

Levine, P.H., "Facilitating Interprocess Communication in a Heterogeneous Network Environment," S.M. thesis, M.I.T. Department of Electrical Engineering and Computer Science, June, 1977.

Liskov, B.H., and Zilles, S., "Specification Techniques for Data Abstraction," IEEE Trans. Software Engineering SE-1, 1, (1975) pp. 7-19.

Liskov, B.H., et al., "Abstraction Mechanisms in CLU," Comm. of ACM 20, 8 (August, 1977), pp. 564-576.

Metcalfe, R.M., and Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks," Comm. of ACM 19, 7 (July, 1976) pp. 395-404.

Millstein, R.E., "Second Semi-Annual Report," Massachusetts Computer Associates Report CADD-7608-1611, August, 1976.

Montgomery, W., "Measurements of Sharing in Multics," Sixth ACM Symposium on Operating Systems Principles, November, 1977, pp. 85-90.

Reed, D.P., and Kanodia, R.J., "Synchronization with Eventcounts and Sequencers," to appear in the Comm. of ACM.

Rothnie, J.B., et al., "The Redundant Update Methodology of SDD-1: A System for Distributed Databases," Computer Corporation of America Report CCA-77-02, February, 1977.

Rowe, L.A., Hopwood, M.D., and Farber, D.J., "Software Methods for Achieving Fail-Soft Behavior in the Distributed Computing System," Proc. IEEE Symposium on Computer Software Reliability, 1973, pp. 7-11.

Stearns, R.E., et al., "Concurrency Control for Database Systems," extended abstract, IEEE Symposium on Foundations of Computer Science, CH1133-8 C, October, 1976, pp. 19-32.

Thomas, R.H., "A Resource Sharing Executive for the ARPANET," Proc. AFIPS Nat. Comp. Conf., 1973, pp. 155-163.

Thomas, R.H., "A Solution to the Update Problem for Multiple Copy Data Bases Which Use Distributed Control," Bolt Beranek and Newman Report #3340, July, 1976.

Wulf, W.A., et al., "An Introduction to the Construction and Verification of Alphard Programs," IEEE Trans. on Software Engineering SE-2, 4 (December, 1976) pp. 253-265.