Inspiring Distribution in Distributed Computation

by

Robert E. Filman

Daniel P. Friedman

Computer Science Department

Indiana University

Bloomington, Indiana 47405

TECHNICAL REPORT NO. 99

INSPIRING DISTRIBUTION IN DISTRIBUTED COMPUTATION

ROBERT E. FILMAN

DANIEL P. FRIEDMAN

DECEMBER, 1980

This material is based upon work supported by the National Science Foundation under Grants MCS77-22325 and MCS-790-4183.

Inspiring Distribution in Distributed Computation

1] Introduction

The marvels of miniaturized silicon are leading to a world of cheap microprocessors. These microcomputers bring with them the hope of faster and cheaper versions of the conventional, main frame computer - an army of small automata, eager to increment and loop, ready to go out and solve our computing problems. However, as any manager of a large software project can assure you, a large collection of dumb computing agents does not add up to a working program. Microprocessors need to be told not only what to do, but how to do it. They need to cooperate and communicate in their processing task - but their cooperation must not turn into a bureaucracy, expending more energy on communication than production.

One of our interests is that of organizing the solution of computing problems in a distributed environment. Towards this end, we have been studying models of distributed processing, economic systems, and heuristic programming algorithms. While our work is still in a preliminary stage, we have acquired a broad knowledge of programming models for distribution and heuristic models for problem solution.

A distributed processing system is characterized by several attributes:

1) There must be multiple processing elements, all capable of concurrent, asynchronous program execution.

2) The processing elements must communicate (transfer information) between themselves, though there is a cost (time delay) associated with this transfer. This criterion contrasts purely concurrent, shared memory systems from "true distributed computation".

3) The processing elements should all be used as full computers - not merely i/o channels - though some of them may be distinguished by possession of particular peripheral devices.

4) No processor should be able to perceive the global state or global time of the system.

5) The communication mechanism ought to be directed - communications should have both a specific origin and a specific destination. This implies that (psuedo) broadcast communication mechanisms are somewhat suspect. If a system wishes to broadcast as a syntactic shorthand for a series of directed communications, then the cost of that broadcast should be proportional to the number of destinations.

6) One should be able to arbitrarily increase the processing capacity of the system by the addition of new processors. If one can add or delete processors dynamically, without stopping and resetting the system, so much the better.

Of course, these are criteria for distribution. Just as a penguin is a bird (though it might not match the Platonic ideal of birdhood), one can speak of a distributed system that does not have all of the characteristics of this ideal. The mental image of the ideal distributed system is a graph; the vertices of the graph are processing elements; the edges, communication channels. The ideal distributed system will be adaptable to any connected graph. However, architectures that require a particular graph (complete, or star, for example) may still be interesting.

The above describes a distributed processing system in physical terms. Just as the most effective programming languages have not been those that provide the programmer with the most transparent view of the machine hardware (far from it!), the most successful languages for programming distributed systems probably ought to shield the programmer from the communication primitives and processor allocations of the actual machines.

2] Models and Languages

There have been many proposals for models and languages for distributed processing. We have been examining some of these (particularly Actors [Hew], ADA [Weg], Concurrent Processes [MiM], CSP [Hoa], Dataflow [Den, ArG], Distributed Processes [BrH], Exchange Functions [Zav], Frons [FrW], Networks of Parallel Processes [KaM], PLITS [Fel], and Shared Variables [LyF]). We are particularly interested in the ability of each programming model to easily express problem solving organizations.

These are a diverse set of models; the authors have had different goals in their creation. Some, like the Milne-Milner and Lynch-Fischer models, spring from purely mathematical concerns (correctness, complexity). Others, such as Distributed Processes, are especially interested in the kind of resource allocation issues inherent in operating systems. We shall identify six major dimensions of a distributed problem solving system.

A) Does the model provide for a fixed number of processes (Fixed), or does it allow for process creation and destruction (Var)? Can new process types be dynamically defined (DyDef)?

B) Does the model provide a "send and forget" communication facility (Mess : messages), or does it require communicants to coordinate their information exchange (Coor)? Does the receiver of a message have primitive control of the incoming message queue (QC)?

C) Are the communication channels between processes predetermined (Pre), or or can processes dynamically establish connections (Dyn)? Is there an implicit complete graph (CG) (any process has (or can acquire) communication links to any other))? Here we refer to the "software" communication channels between processes.

D) Is the relationship between communicants symmetric (Sym), or asymmetric (Asym)? Here we are referring to the symmetry involved in creating the information transfer, not to the symmetry of information flow. Message sending and procedure calling are thus asymmetric operations; exchange functions are not.

E) Is the system "call be need" (Need), computing only when a result is required, "call by value" (Value), processing and preparing results without regard to the ultimate use?

F) Does the model provide abstract data types (ADT)?

The preliminary state of this comparision is presented below.

Model	A	B	C	D	E	F
Actors	DyDef	Mess	Dyn-CG	Asym	Value**	ADT
ADA	Fixed	Coor-QC	Pre	Asym	Value	ADT
CP/MiM	DyDef	Coor	Pre	Sym		
CSP/Hoa	Var	Coor	Pre	Sym***	Value	
DFlow	Fixed	Mess	Pre	Asym	Value	
DP/BH	Fixed	Mess	Pre	Asym	Value	ADT
ExF/Zav	Fixed	Mess	Pre	Sym	Value	
Frons	DyDef*	Mess	Dyn-CG	Sym	Need	
NPP/KaM	DyDef*	Coor	Dyn	Asym	Need	
PLITS	Var	Mess-QC	Dyn-CG	Asym	Value	ADT
SV/LyF	Fixed	Mess	Pre	Sym	Value	

Notes:

* Not essentially a "processes" system.

** Has a "delay" operator to simulate call-by-need.

*** In the CSP model, connection is made "almost" symmetrically. There are input guards, but not output guards. Kieburtz and Silberschatz [KiS] have discussed the difficulties in removing this asymmetry. Bernstein [Ber] has proposed a possible implementation of output guards.

A distributed problem solver, of the form envisioned earlier, will need to be able to acquire new processes dynamically (perhaps even define new process types), have flexible communication channels, and some sort of "send and forget" primitive. Data abstraction is a useful tool; it will likely make the task of building dynamic distributed systems easier. Queue control is of less obvious value, and can be simulated with "advising functions" [Tei].

Dynamic communication channelling is desirable, but difficult. Obviously, there is not a small factory attached to our distributed system, taking sand and sunshine in one end, and adding components to the network out the other. An alternate mechanism might be literal broadcasting - it should be possible to build microcomputers with radio receivers and transmitters, capable of tuning to specific channels for particular communications. (This model might work especially well with both a centralized organization (radio is an effective tool of an autocracy) and with a production system formalism (get the latest updates to the working memory on channel 102), see below).

3] Sociological and Heuristic Strategies

There is a distinction among models of distributed computing between what we will call the "diffusing control" and "interacting agents" approaches. Is the computation initiated by activating one process, which then creates and initiates its sub-processes, activation (and problem context) diffusing outward? Or is the system a number of independent agents, always vigilant, entertaining requests from their neighbors, but otherwise tending to their own affairs? The perspective is important. At the hardware level, agents clearly interact; there is no magic for creating a new physical processor. However, at a certain level of programming abstraction, ignorance of the actual processor allocation becomes desirable. The language at that level can still choose to be hierarchical (diffusing) or heterarchical (interacting). There are merits to each approach. If control diffuses, then the sender of a control message must be prepared for the failure of the receiver. Independent, interacting agents do not start with this conceptual difficulty. Rather, greater effort must be made in such systems to get the agents working together on a problem. If agents do nothing except make progress on useful problems, the loss of a processor may be only a minor inconvenience.

The idea of "useful progress" may be a foreign notion. Most conventional programming languages are a-step-at-a-time, imperative formulations. The validity of the successive steps is entirely dependent upon the successful completion of the previous steps.

There are other formulations for expressing computable functions. Production systems (such as [New]) are examples of such a formalism. A production system consists of two parts: a working memory and a set of productions. Each production has two pieces, a "pattern" and an "action". When some part of the working memory matches the pattern of a particular production, that production "fires", executing its action. Actions are programs; they typically add elements to the working memory. Elements are never removed from the working memory. Thus, the firing of a production never makes another production's firing cease to be valid.

Formal logic shares this property: the proof of a theorem does not invalidate the truth of any other theorem. Programming languages predicated on this idea of "incremental discovery" could be more easily distributed.

The Sting operator [FrW] is similiar in effect. Sting is an interlock-free test-and-set primitive. The intent is to "sting" an object with a value. If the particular object to be stung has already been stung (by someone else) the operation becomes a "no-op". Thus, if a swarm of processors are working on a problem, the first "sting" of the answer is permitted to succeed. The others, if they find the problem already solved, can proceed to other tasks; if not, their change has no global effect.

4] Economic Models

Distributed computing networks are not the only organizations that require internal cooperation and communication. Human economic activity shows both some of the same requirements, and some of the same goals as a distributed computing network. There are some interesting parallels between human economic systems, and potential organizational models for distributed systems.

How are economies organized? One important dimension is centralization. In a centralized system, there is a master directorate (node) that sets the goals of the system, and divides the task into sub-pieces, with each sub-task specified for a particular worker. When the task is modular and well-defined it is possible to organize a distributed system in this fashion. Efficiency can be achieved in such a structure if the task is well understood, and the initial allocation of sub-goals and resources can be made to reflect this understanding. However, central planning does not lend itself well to ill-defined problems. Additionally, there may be a communications overload from the planning node to the workers, while most of the communication ability of the system, between the working nodes, goes underutilized. A centralized system has the capacity to respond coherently to the occurrence of a dramatic change in the problem space.

It is only a small step from a fully centralized economy to a partially centralized (hierarchical) model. The central authority defines the major tasks. These are parcelled out to regional sub-authorities, each of whom is allotted a resource of workers. This structure can be iterated. At the limit, it resembles a corporate hierarchy tree. Hierarchical organization can respond well to local aberrations. Its response to dramatic global changes is somewhat slower, however, as command must filter through several command layers. As any distributed system must have its elements physically distributed in the three dimensional world, a hierarchical system has the potential of reflecting this physical reality.

An alternate approach to processor organization is a *laissez faire* economy. Each task has certain goals and an allocation of currency. Currency can be used to purchase processor power and to generate new tasks. When a task has exhausted its currency, it can appeal to its own source (banker) for more. Its banker can then decide, on the basis of the results that the task presents, whether to grant that task more resources. The scheme can be applied recursively, to the banker's banker, and so forth, back to the resources of the human who originated the request. Such a scheme lends itself to ill-defined tasks, ones where a promising line can be recognized, but not necessarily generated, and to "useful progress" programming models. While such systems are not fragile, there are difficulties in both focusing the organization in the presence of a rapidly changing environment, and in terminating the activity of tasks that have ceased to be useful. This kind of mechanism is essentially what was employed in the agenda priority schemes of Lenat [Len]; it parallels some remarks of Hewitt.

One could also imagine distributed systems organized as mixed economies (partially centralized, and partially free market) or as indicative planned systems (with centralized goals and directives shaping a free market economy.)

Smith's Contract Nets [Smi] is another proposed system for distributed organization. Processes that have sub-problems broadcast their request to the other processes. A free process, or one that has particular knowledge about that task, "bids" to obtain the contract. Contract Nets are a protocol; Smith does not go into how an arbitrary system might be organized to perform a contract approach.

5] Conclusions

Distribution promises inexpensive and efficient computation. To realize that promise, much work needs to be done both to define the right models for distribution, and to select the appropriate algorithms for apportioning computations among processes.

Acknowledements

We thank John Barnden, Jim Burns, Mitch Wand and Dave Wise for comments and conversations that helped us during the development of these ideas. Research reported herein was supported (in part) by the National Science Foundation under grants numbered MCS79-04183 and MCS77-22325.

References

[ArG] Arvind, Gostelow K. and Plouffe, K. The (Preliminary) Id Report: An Asynchronous Programming Language and Computing Machine. Technical Report 114, Dept. of Information and Computer Science, University of California, Irvine (May, 1978).

[Ber] Bernstein, A. J. Output Guards and Nondeterminism in "Communicating Sequential Processes". ACM Trans. Programming Languages and Systems 2, 2 (April 1980) 234-238.

[BrH] Brinch Hansen, P. Distributed Processes: A Concurrent Programming Concept. CACM 21, 11 (November 1978) 934-941.

[Den] Dennis, J. B. First Version of a Data Flow Procedure Language. In B. Robinet (ed.) Programming Symposium Springer, Berlin, (1974) 362-376.

[Fel] Feldman, J. A. High Level Programmming for Distributed Computation. CACM 22, 6 (June 1979) 353-367.

[FrW] Friedman, D. P. and Wise, D. S. An approach to fair applicative multiprogramming. In G. Kahn (ed.), Proceedings of International Symposium on Semantics of Concurrent Computation, Springer, Berlin (1979) 203-225.

[Hew] Hewitt, C., Attardi, G. and Lieberman, H. Security and modularity in message passing. Proceedings of the First International Conference on Distributed Systems, IEEE, (1979) 347-358.

[Hoa] Hoare, C. A. R. Communicating sequential processes. CACM 21, 8 (August 1978), 666-677.

[KaM] Kahn, G. and MacQueen, D. Coroutines and networks of parallel processes. In B. Gilchrist (ed.), Information Processing 77, North-Holland, Amsterdam (1977), 993-998.

[KiS] Kieburtz, R. B. and Silberschatz, A., Comments on "Communicating Sequential Processes". ACM Trans. Programming Languages and Systems 1, 2 (October 1979) 218-225.

[Len] Lenat, D. B. Automated Theory Formation in Mathematics. 5th International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, (1977), 833-842.

[LyF] Lynch, N. and Fischer, M. On describing the behavior and implementation of distributed systems. In G. Kahn (ed.), Proceedings of International Symposium on Semantics of Concurrent Computation, Springer, Berlin (1979) 147-171.

[MiM] Milne, G. and Milner, R. Concurrent Processes and their syntax. JACM 26, 2 (April 1979), 302-321.

[New] Newell, A. Production systems: models of control structures. In W. Chase (ed.) Visual Information Processing, Academic Press, New York, 1972, pp. 463-526.

[Smi] Smith, R. The contract net protocol: high-level communication and control in a distributed problem solver. Proceedings of the First International Conference on Distributed Systems, IEEE, (1979) 185-192.

[Tei] Teitelman, W. Interlisp reference manual. Xerox Palo Alto Research Center Technical Report. Palo Alto, California (October 1979).

[Weg] Wegner, P. Programming with ADA: an Introduction by Means of Graduated Examples, Prentice-Hall, Englewood Cliffs, New Jersey (1980).

[Zav] Zave, P. and Fitzwater, D. R. Specification of Asynchronous Interactions Using Primitive Functions. Technical Report 598, Dept. of Computer Science, University of Maryland, College Park, Maryland (1977).