ACM SIGACT News Distributed Computing Column 5

Sergio Rajsbaum*

November 1, 2001

Abstract

The Distributed Computing Column covers the theory of systems that are composed of a number of interacting computing elements. These include problems of communication and networking, databases, distributed shared memory, multiprocessor architectures, operating systems, verification, internet, and the web.

This issue consists of four parts:

- a survey of SIROCCO'01 by Pierre Fraigniaud,
- a survey of POMC'01 by Rui Fan,
- a survey of PODC'01 by myself,
- the paper "Paxos Made Simple" by Leslie Lamport.

This year was the 20-th anniversary of PODC, and included a special celebration honoring the work of Leslie Lamport, on the occasion of his 60th birthday, and was collocated with the Workshop on Principles of Mobile Computing (POMC). I briefly describe the PODC conference, and reproduce the toast by Fred Schneider in the honor of Leslie. The Business Meeting report is by Gil Neiger. Finally, there is Leslie's paper on his own algorithm which was discussed in a few of the PODC lectures. I am very grateful to the many people contributing to this issue.

Request for Collaborations: Please send me any suggestions for material I should be including in this column, including news and communications, open problems, and authors willing to write a guest column or to review an event related to theory of distributed computing.

Structural Information and Communication Complexity: A Survey of SIROCCO '01

Pierre Fraigniaud[1].
CNRS, Laboratoire de Recherche en Informatique
Université Paris-Sud
91405 Orsay cedex, France.

*Compaq Cambridge Research Laboratory, One Cambridge Center, Cambridge, MA 02142-1612. `Sergio.Rajsbaum@compaq.com`, `rajsbaum@math.unam.mx`. On leave from Instituto de Matemáticas, UNAM.

[1] *http://www.lri.fr/~pierre*. Pierre Fraigniaud co-chaired the Program Committee of SIROCCO '01, in collaboration with Francesc Comellas and Josep Fabrega, from Departament de Matemàtica Aplicada IV, Univ. Politècnica de Catalunya (UPC), Barcelona, Spain.

Abstract

SIROCCO '01 is the 8th edition of the series of Colloquia on Structural Information and Communication Complexity. It held at Vall de Núria, Catalonia, Spain, 27-29 June, 2001. This article is a summary of the scientific contributions of the participants to this colloquium, as well as a description of the general atmosphere of the colloquium.

1 A brief history of SIROCCO[2]

The series of colloquia SIROCCO (for Structural Information and Communication Complexity) was motivated by the need for building a community of researchers interested in the study of the factors which are significant for the computability and the communication complexity of problems. More precisely, the main stream of SIROCCO is the study of the interplay between "knowledge" given to entities (e.g., processing elements of a distributed systems, software agents, etc.) and the global computation and communication "power" of these entities. For instance, the first edition of SIROCCO was, to a large extend, devoted to *compact routing* (i.e., the ability to route in a system with nodes of limited storage) and to *sense of direction* (i.e., the formal generalization to arbitrary distributed networks of the informal notion of geographical directions). To be more specific, these two topics are concerned with questions such as: For a given function f, can we route in order-n networks whose nodes are given $O(f(n))$ bits of memory? What are the functions which can be computed by an anonymous networks whose nodes are given sense of direction? Etc.

Since then, the list of interests of the colloquium became much longer, and the recent editions of SIROCCO included presentations on *routing*, *broadcasting*, *distributed computing*, *communication complexity*, *network topologies*, *communication and computation models*, *graph theory*, etc., with applications to *parallel and distributed systems*, *high-speed networks*, *mobile computing*, *wireless networks*, *systems of software agents*, etc. The kernel of the colloquium remains however the study of the interplay between structure and knowledge on one hand, and communication- and computation-complexity on the other hand.

2 The 8th edition of SIROCCO

The URL of this 8th edition is *http://www-mat.upc.es/sirocco2001/*.

2.1 Structure of the colloquium

Conference chair: David Peleg (Weizmann Inst., Israel) – Formerly Nicola Santoro (Carleton U., Canada);

Program Committee co-chairs: Francesc Comellas (UPC, Spain), Josep Fabrega (UPC, Spain) and Pierre Fraigniaud (CNRS, U. Paris-Sud, France);

Program Committee: Francesc Comellas (UPC, Barcelona, Spain), Stefan Dobrev (Slovak Acad. of Sc., Slovak Rep.), Danny Dolev (Hebrew U., Jerusalem, Israel), Josep Fabrega (UPC, Barcelona, Spain), Paola Flocchini (U. Ottawa, Canada), Pierre Fraigniaud (CNRS, U. Paris-Sud, France), Satoshi Fujita (U. Hiroshima, Japan), Cyril Gavoille (U. Bordeaux, France), Lefteris Kirousis (CTI, Patras, Greece), Evangelos Kranakis (U. Carleton, Ottawa, Canada),

[2]This history reflects the point of view of the author, and not necessarily the point of view of the conference chairs.

Art Liestman (SFU, Burnaby, Canada), Fabrizio Luccio (U. Pisa, Italy), Jaroslav Opatrny (Concordia U., Montreal, Canada), David Peleg (Weizmann Inst., Israel), Andrzej Proskurowski (Univ. of Oregon, Eugene, USA), Sergio Rajsbaum (UNAM, Mexico, and Compaq), Jop F. Sibeyn (Max-Planck Inst., Germany), Ramesh K. Sitaraman (UMass, Amherst, USA), Sunil M. Shende (U. Rutgers, Candem, USA), Richard Tan (U. Utrecht, Netherlands), Walter Unger (U. Aachen, Germany), Eli Upfal (Brown University, USA), Ugo Vaccaro (U. Salerno, Italy), Sebastiano Vigna (U. Milano, Italy), and Imrich Vrto (Inst. Math, Bratislava, Slovak Rep.).

Local organization: Francesc Comellas (UPC) (Chair), Josep Fabrega (UPC), Margarida Mitjana (UPC), Xavier Muñoz (UPC), and Javier Ozón (UPC).

2.2 A tradition of great locations

SIROCCO 2001 held at Vall de Núria, Catalonia, Spain, 27-29 June, 2001. Vall de Núria is a small remote valley located 2000 meters high in the Pyrennees, with a long history from at least the 7th century. It is currently a popular ski area in winter, and a famous starting point for hikes in summer (see *http://www.valldenuria.com*). There stands a comfortable hotel with conference facilities, reachable only by rack-railway (beside a 3-hour hike along a steep narrow trail). This location continues the tradition of great locations for SIROCCO: SIROCCO '98 held on the Amalfi coast (Italy), SIROCCO '99 on the Atlantic coast near Bordeaux (France), and SIROCCO '00 in the ancient city of L'Aquila in the Abrusio (Italy). In addition to the choice of the location, and again perpetuating the tradition of SIROCCO, the local organizing committee from the Universitat Politècnica de Catalunya (UPC) in Barcelona made a great job, providing abundant fine cuisine and drinks, and healthy hikes. And one should not blame the organizing committee if some participants did not balance these types of activities in the appropriate way.

Actually, and as usual for SIROCCO, the environment strongly and positively influenced the colloquium. The number of attendees is usually around 60, and this relatively small format allows the participants to interact and to work together, creating a strong emulation between them. Just as an example, everyone who chaired an open problem session knows how hard it could be to motivate the audience. The open problem session of SIROCCO 2001 hold from 12:00 to 1:30 in... the night, after the banquet, and everyone wanted to present a problem!

2.3 Invited talks

SIROCCO 2001 was preceded by the 6th edition of the International Workshop on Interconnection Networks (IWIN) which holds every odd year in Europe since 1991[3], and focuses on the algorithmic and structural aspects of interconnection networks. SIROCCO and IWIN decided to share their budgets for the invited speakers. During the whole week at Vall de Núria, there were hence four invited lectures:

- J. Diaz (UPC, Barcelona): "Adversarial queueing theory: recent results".

  This talk explained how one can use adversarial queueing theory for the analysis of routing policies in the dynamic setting (i.e., packets are continuously injected in the system). In this theory, there is no probabilistic assumption on the arrival of the packets. It is only assumed that bursts in the traffic do not occur too frequently.

[3] A companion workshop is organized every even year in North America.

- C. Gavoille (Univ. Bordeaux): "Compact and distributed data structures".

  It is explained how one can distribute information among nodes of a system so that functions regarding a small set of entities (e.g., routing, distance, adjacency checking, etc.) can be computed based solely on the (expected to be small) information held by these entities.

- C. Kaklamanis (CTI, Patras): "Wavelength routing in WDM trees".

  WDM is a switching technique for optical networks that allows the simultaneous use of different wavelengths in one optical fiber. This talk surveyed the algorithmic solutions and challenges related to the establishment of end-to-end paths between entities of a system, using a minimum number of wavelengths.

- J. Urrutia (Univ. Nat. de Mexico): "Distributed computational geometry".

  It is explained how one can take advantage of structural information (e.g., planar geometric networks) for the design of distributed algorithms.

These presentations are good illustration of SIROCCO's main stream. The analysis of routing protocols, of the impact of spreading out information, and of the potential benefits of structural knowledge, are among the most popular problems investigated by the SIROCCO community.

2.4 SIROCCO 2001 contributions[4]

Papers for presentations at the colloquium were refereed and selected according to scientific criteria that are standard in the community of parallel and distributed computing as well as in the community of theoretical computer science. Note that the spirit of SIROCCO (i.e., a workshop format with intensive collaborations between the participants) leads to solicit position papers outlining open problems and research directions, as well as research papers. This year SIROCCO selected 25 presentations out of the 35 papers submitted. As for the previous editions, the proceedings of SIROCCO 2001 were published by Carleton Scientific, and can be requested at *www.carleton-scientific.com*.

Instead of classifying SIROCCO 2001 contributions in the usual way, making distinction between distributed computing, routing, scheduling, etc., I will do my best to describe the scientific content of these contributions in regard with the factors which are significant for the computability and the communication complexity of problems.

In their paper J. Bose and P. Morin study the packet routing problem in planar graphs whose nodes are given Cartesian coordinates. I.e., given the coordinates of the destination, and knowing the coordinates of its neighbors, a current node must decide to which neighbor the packet must be forwarded. The ability to take such a decision, and the efficiency of the result are discussed, depending on additional hypotheses on the structure of the graphs. In all cases, nodes know very little of their environment. L. Barrière and S. Dobrev give to nodes the initial knowledge "I am in an Abelian Cayley graph of degree 4", and show, using the underlying 2-dimensional lattice of such graphs, how leader election can be solved. Their general upper bound matches the lower bound for many specific networks (e.g., rings, toruses). The generalization to Abelian Cayley graphs of larger degree looks doable but difficult. The generalization to arbitrary Cayley graphs is a real challenge.

Three papers consider problems in which a restricted knowledge is given to the nodes, whilst complex tasks are still achievable by these nodes. In their paper, P. Flocchini, G. Prencipe, N. Santoro and P. Widmayer consider a set of robots in the plane. Each robot has a compass. The ability

[4]Again, this summary reflects the point of view of the author of this survey, and not necessarily the point of view of the authors of the cited papers.

of the robots to form patterns (e.g., a line, a square, etc.) is discussed as a function of the degree of agreement of the compasses. An interesting case seems to be when compasses agree on one dimension but not on the other, i.e., they agree on where is North, but there is no agreement on East and West. G. Malewicz, A. Russel and A. Shvartsman continue the work initiated by S. Dolev, R. Segala, and A. Shvartsman in SIROCCO '99 on scheduling independent tasks on processors which ignore *when* communications will take place, and *who* will be involved. The objective is to minimize the overlapping of work between the processors. For instance, in the case of two processors, a strategy consists for one processor to perform the tasks in the order $t_1, t_2, \dots, t_k$ while the other performs the tasks in the order $t_k, t_{k-1}, \dots, t_1$. The problem becomes obviously more complex in larger systems, and, somewhat surprisingly, efficient schedules avoiding the communication-time unawareness can be designed. Finally, P. Damaschke revisits the broadcast problem in an environment where nodes only know their neighbors (blind broadcasting), and improves the previously known bounds for networks of small degree.

E. Godard and Y. Métivier build a theory to capture the notion of "knowledge" given initially to nodes of a network G. Nodes are involved in a distributed computation aiming to check whether G satisfies some specified properties (e.g., be a tree, be planar, etc.). The computation proceeds by a sequence of local relabeling. For instance, let $\{\epsilon, \ell, T\}$ be a set of three labels. Initially, every node is given label ϵ. The nodes proceed as follows: (1) a node which has exactly one neighbor labeled ϵ changes its own label to ℓ; (2) a node which has all its neighbors labeled ℓ changes its own label to T. One can check that this process allows to recognize trees, in the sense that the graph is a tree if and only if one node eventually becomes labeled T. The initial knowledge given to the nodes is expressed in term of labeling as well. (In the previous example, there is no initial knowledge.) The authors develop an algebraic theory which allows to characterize recognizable classes of graphs. An interesting application is the surprising result, already noticed by M. Yamashita and T. Kameda (IEEE TPDS 7(1), 1996) that the class of graphs recognizable given a bound on the number of nodes is the same as the class of graphs recognizable without *a priori* knowledge.

Three groups of papers were typical examples of topics which have a long tradition in SIROCCO. One is dealing with the evolution of dynamic system, another with the design of compact routing technique, and the third with the design of topology-dependent communication protocol (e.g., routing, broadcasting, etc.).

More precisely, the first group is dealing with a coloring game on graphs. In its simplest setting, we are given a graph whose node are colored black or white. The coloring evolves synchronously: at each round, every node takes the color which has the majority among its neighbors. The importance of this game follows from its obvious relation with error propagation in networks. A set M is a dynamic monopoly (dynamo) if starting with only the nodes of M colored black, the whole network converges to an all-black configuration. R. Královič and P. Ružička study the relation between the *immunity index* (i.e., the smallest integer k such that if the initial configuration contains at least k white nodes, then it will not converge to the all-black configuration), and the *catastrophic index* (i.e., the smallest integer k such that if the initial configuration contains at least k black nodes, then it will converge to the all-black configuration). T. Nakata and M. Yamashita consider the case where the coloring evolves at each round by choosing k nodes uniformly at random and updating their colors to white with probability $p = \#\text{white neighbors}/\#\text{neighbors}$, and to black with probability $1-p$. Their main results are a collection of upper bounds on the expected number of steps required to reach a monochromatic configuration. Finally, N. Zeh and N. Santoro revisit the problem in weighted directed graphs, and consider the complexity of finding monopolies when the colors evolve according to a weighted vote among neighbors.

Regarding the second group, i.e., the design compact routing protocols, two papers revisit the well known *interval routing* (IR) technique. IR consists to (1) label the nodes from 1 to n, and (2) label the edges by intervals (more than one interval can be given to an edge). Routing is then performed as follows: A node x holding a message destinated to a node y, labeled ℓ, forwards that message along the edge whose one interval contains ℓ. The constraint is that such process must route messages along shortest paths. The objective is to minimize the maximum number of intervals labeling any edge. V. Chepoi and A. Rollin show that 7 intervals per edge are sufficient for IR in plane quadrangulation whose internal nodes have degree at least 4. (A typical example of a member of such class of graphs is a partial tessellation of the plane with square boxes, but the class is much larger.) Note that it is not known whether every partial tessellation of the plane with square boxes admits IR with one interval per edge. The graphs supporting IR with one interval form a large class of graphs which is actually not well characterized. Any effort in this direction is welcome as it is known that these graphs are efficient networks for basic distributed computations such as leader election, broadcasting, spanning tree, etc. Y. Ganjali characterizes the class of graphs supporting multi-dimensional linear IR where the condition on the routes to be minimal is relaxed.

Regarding the third group, i.e., the conception of topology-dependent communication protocols, P. Salinger and P. Tvrdík designed a broadcast protocol for CCCs under the all-port line model (i.e., a communication step between two nodes is a path joining these two nodes, and it is required that two paths corresponding to two simultaneous communication be edge-disjoint). The protocol of P. Salinger and P. Tvrdík is optimal up to dimension 123 (CCC_{123} has roughly 10^{39} nodes!). Still, it is not optimal, and, even for simple regular topologies as hypercube, the design of optimal broadcast protocols remains a challenge. A. Osterloh revisits the famous permutation routing problem on meshes, and generalizes the results known for 2-dimensional meshes to d-dimensional meshes, $d \geq 2$. Finally, G. Fertin and A. Raspaud consider the variant of the broadcast problem in which the destination set is composed of nodes at distance at most k from the source. They solve the problem for standard topologies (e.g., ring, meshes, trees, etc.).

Two papers focus on the transfer of data between a client and a server. J. Watkinson, M. Adler, and F. Fich revisit the famous Shannon's theorem stating that if a node holds a word w of n bits drawn from a probability distribution D, then the expected number of bits required to transmit w to another node is at least the entropy of D, i.e., $\sum_w D(w)\log_2(1/D(w))$. Motivated by asymmetric transmission systems (cable modems, ADSL, satellite, etc.), J. Watkinson M. Adler, and F. Fich consider the case of an asymmetric channel, and the problem of sending data in the slow direction, i.e., from the client to the server. They improve some earlier works by Adler and Maggs in FOCS '98. A. Kesselman and Y. Mansour propose on-line protocols for the transfer of a video stream along a channel. More precisely, congestion forces some frames to be discarded, and the authors hence study several types of discarding policy aiming to maximize the QoS "perceived by the client", this latter notion been captured by several cost functions on the frames.

S. Akl and S. Bruda tackle the difficult task of building a theory for parallel real-time systems. Rather than awkwardly summarize their result, let me just mention that they asked the following question: *Can we find any problem that is solvable by an algorithm that uses $n > 1$ processors, and is not solvable by a sequential algorithm, even if this sequential algorithm runs on a machine whose processor is n times faster than each of the n processors used by the parallel implementation?* And they answered yes to this question!

A collection of papers consider different aspects of network design and network modeling, including low-earth-orbit satellite constellation by P. Gvozdjak and J. Peters; wavelength division multiplexed (WDM) optical networks by J.-C. Bermond, L. Chacon, D. Coudert, and F. Tillerot; and path layout in ATM (Asynchronous Transfer Mode) networks by M. Makúch and P. Ružička. Also inspired by network design problems, M. Elkin and D. Peleg presented solutions for a variant of the graph spanner problem, and I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos improved the bounds on the size of minimum feedback vertex sets in meshes and butterflies (a feedback set is a set of nodes whose removal produces an acyclic graph). Last but not least, P. Hafner investigated the structure of the McKay-Miller-Širáň family of graphs, whose some members are among the best known graphs with regards to the Moore bound (i.e., maximizing the order for a given degree and diameter).

To conclude this survey of SIROCCO 2001 papers, it is worth to note that "orthodox" distributed computation was not absent from the colloquium. A. Mostéfaoui, S. Rajsbaum, M. Reynal, and M. Roy carry on the development of their recent theory on consensus in presence of faults, aiming to capture the structure of the inputs allowing consensus to be solved despite process crashes. Although characterization seems to be beyond the current state of the art, generic forms of the inputs allowing (rapid) solvability of consensus are presented in the paper. S. Cantarell, A. Datta, F. Petit, and V. Villain address the mutual exclusion problem in the message passing model. Previous algorithms solving the problem used large messages. This paper solves the problem using messages of bounded size.

3 Future SIROCCO's

The 9th SIROCCO should hold in Greece in June 2002, remaining in a Mediterranean country for one more year. There will be a big change though since one will move from the mountains to the sea, as the local organization chair, Y. Stamatiu from CTI and Univ. Patras, is considering several places in the Cyclades. The program committee of SIROCCO 2002 will be co-chaired by C. Kaklamanis and L. Kiroussis, both from CTI and Univ. Patras.

Year 2003 will be an important date for SIROCCO: its 10th anniversary. The colloquium may then move to the Rockies, back to Canada where the first edition held. An alternative is the very north of Europe in a Scandinavian country, probably Sweden.

The current evolution of distributed systems, i.e., a decreasing amount of centralized control and an increasing amount of mobility, should increase the importance of the topics investigated by SIROCCO, not only from a distributed computing point of view, but also in relation to problems arising in parallel computing, networking, and, more generally, in the understanding of complex intelligent systems. The understanding of the impact of the local power of system-entities on the global power of the system forms the kernel of SIROCCO. We do hope that whoever is interested in this topic will join SIROCCO's community and attend the next edition of the colloquium.

So, looking forward to seeing you next June in the Cyclades!

Workshop on Principles of Mobile Computing: A Survey of POMC '01

Rui Fan[5].
MIT Laboratory for Computer Science
Cambridge, MA 02139

1 Introduction

Ad hoc mobile computing is an important emerging branch of distributed computing possessing unique and challenging research issues. An ad hoc mobile system consists of a number of mobile hosts interacting via multi-hop communication. Ad hoc systems should provide the ability to form spontaneous networks, and should be resilient to host failures and host mobility. These characteristics underlie the difficulty in designing effective ad hoc systems and algorithms. Researchers have traditionally envisioned ad hoc computing for military or disaster-relief scenarios. However, there clearly exist scientific and commercial applications which can exploit mobility as well. Previous work on ad hoc networks has primarily focused on scheduling, routing and multicasting algorithms. But it is important to consider these algorithms not in isolation, but in the context of end-user applications that utilize them.

The POMC workshop on the Principles of Mobile Computing was held for the first time in Newport, RI on August 29-30, 2001, collocated with PODC. POMC joins other conferences and workshops on mobile computing such as MobiCom, MobiHOC, and DIALM. An intended purpose of POMC is "to bring together researchers in mobile computing and distributed algorithms." It is hoped that the breadth of expertise will allow researchers to view mobile systems in larger perspective. In the following article, I will summarize the works presented at POMC. As will be seen, the organization of this paper loosely follows the organization of the Internet's protocol stack. The papers referenced below may all be found in the 2001 POMC and 2001 PODC conference proceedings.

2 The Physical Layer

Most mobile networks use radio transmission, so a basic problem when broadcasting is collision between concurrent transmissions. The joint PODC / POMC paper by Clementi, Monti and Silvestri, "Distributed multi-broadcast in unknown radio networks", studies the efficiency of performing a series of independent broadcasts in such a network. They give a lower bound which is proportional to the logarithm of the network size times the number of broadcasts, for both deterministic and randomized algorithms. A consequence of this result is that a "perfect pipeline" protocol, i.e. a protocol requiring $O(SB(n) + r)$ time, where $SB(n)$ is the time required for a single broadcast in a size n network, and r is the number of broadcasts, is not achievable. The authors also gave a nearly optimal deterministic broadcast algorithm for low degree networks.

3 The Data Link Layer

A frequent and difficult problem arising in mobile networks is partitioning of the network caused when hosts move out of radio range of each other. The second joint PODC / POMC paper by Li,

[5] rfan@theory.lcs.mit.edu

Li, Halpern, Bahl, Wang and Wattenhofer, "Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks", gives an algorithm to determine the minimum power hosts need to broadcast with in order to maintain a strongly connected network. The broadcast power should be minimized because higher power usage decreases the lifetime of the network, and increases the likelihood of collision during transmission. This work relies on the use of existing directional antennas, which detect the direction a transmission originates from but not its distance. The algorithm is just for every host to broadcast with the minimum power such that every 150 degree sector around the host contains some other host. This algorithm is adaptive to host failures and joins. Under realistic simulation parameters, it decreased the broadcast range needed to maintain connectivity by a factor of 3.

4 The Network Layer

In sparse mobile networks, even the previous algorithm cannot ensure network connectivity. The invited paper by Chatzigiannakis, Nikoletseas and Spirakis, "On the average and worst-case efficiency of some new distributed communication and control algorithms for ad-hoc mobile networks", addresses message delivery in such networks by creating a mobile equivalent of cellular network backbones. In this algorithm, a group of hosts are enlisted to act as couriers for the network. The couriers, called the support, move in a coordinated fashion which ensures they will traverse the entire geographic area in which mobile hosts are present. Messages between hosts are sent to the support, and delivered when the support eventually passes by the destination. Two models for the movement of the support were given, one in which the support moves as a chain, and the other in which the support moves randomly. Using Markov analysis, the authors demonstrated that the support will deliver messages in time proportional to the area of movement divided by the broadcast area of the support. This algorithm is particularly applicable in military scenarios, where it is plausible to have special units follow a protocol for movement to enable communication between all units.
Chen and Murphy, in "Enabling disconnected transitive communication in mobile ad hoc networks", study message delivery by optimistically propagating messages toward their destinations. They exploit the idea that while network partitions are common in ad hoc networks due to host movement, reconnections between partitions are also common due to the same movement. For hosts across different partitions to communicate, their algorithm propagates messages in a direction where a reconnection is likely to occur. At the moment of reconnection, the message is passed to the destination partition. Hosts perform local computations to determine their utility in relaying messages between different partitions, and messages are passed between neighbors with the highest utilities. The effectiveness of the algorithm depends on the accuracy of the utility calculations. The utility should be tailored to the movement patterns of the hosts.

5 The Transport Layer

Gupta, in "A distributed and adaptive core selection and migration method for bandwidth-efficient multicast routing in mobile ad hoc networks", studies the core selection problem for multicast trees. The core, or root, of a multicast tree should be selected to have small average distance to the hosts in the multicast group. In the mobile setting, it is important for the core to be stable in the face of multicast group membership changes. In addition, selecting the core should not require global topology information, which is expensive to acquire and unstable. Gupta selects the centroid of the

tree as the core. The centroid is the ideal root in terms of average distance to the multicast group members, and it may be computed using local information using a convergecast type operation. In addition, it is easy to adjust the centroid after group membership changes.
Mohammed and Gupta, in " Secure multicasting in mobile ad hoc networks: A case study in developing adaptive protocols", consider key management for secure multicast. The main issue is whether to use a shared key or different keys between a parent and its children in the tree. Shared keys require less overhead, but are problematic because they need to be changed each time the multicast group's membership changes. An algorithm was given which uses shared keys with stable children, and different keys with unstable children.

6 Clustering

Richa, Obraczka and Sen, in "Application-oriented self-organizing hierarchical clustering in dynamic networks", study clustering algorithms for ad hoc networks. Clustering and hierarchies are important techniques for mobile networks because they localize effects of changes in network topology. Self-organizing clusters are also important because they automatically adapt themselves when the network changes. Richa defines application classes as groups of applications for ad hoc networks with common constraints such as minimizing communication time, minimizing memory usage, or conserving energy. They study self-organizing clustering algorithms for power-aware applications. A summary of their work can be found in the technical report [6].

7 The Application Layer

Walter, Cao and Mohanty, in "A k-mutual exclusion algorithm for wireless ad hoc networks", give a resilient algorithm for k-mutual exclusion in an ad hoc network. Interestingly, the algorithm is similar to the ad hoc network routing protocol TORA. In some sense, this represents a collapse of the application layer of the Internet protocol stack down to the network layer. The mutual exclusion is based on token ownership, which stems from the desire to minimize the amount of communication in the algorithm. The token based algorithm uses only local topology information and local communication. A permissions based mutual exclusion algorithm might require global communication, and would be less reliable and efficient. As in TORA, the algorithm maintains a height for each host, and forms a directed acyclic graph on the network graph with edges pointing from higher to lower hosts, and with sinks at the token holders. Request for tokens are passed down a path, and tokens are passed up the same path. The edges on the path are then reversed to form a new DAG with the new token holder as a sink. The algorithm can cope with host failures and also partitions.
Killijian, Cunningham, Meier, Mazare and Cahill, in "Towards group communication for mobile participants", study group communication techniques in ad hoc networks. One natural grouping in an ad hoc setting is of hosts which are near each other. Killijian defines a proximity group as a set of hosts lying within a shape anchored at a reference point. He shows various applications of proximal groups. Group membership is managed by beaconing and exchange of local connectivity information. Group membership is also determined using failure detectors, and by tracking the host movements and anticipating when they move out of range.
The invited paper by Nesargi and Prakash, "Issues pertaining to service discovery in mobile ad hoc

[6] A. W. Richa, K. Obraczka, and A. Sen, Power-aware Self-Organizing Hierarchical Clustering in Dynamic Network, Technical Report ASU, Arizona State University, August 2001.

networks", gives group communication even greater prominence by constructing a group membership service as an intermediate layer between applications and the network layers. The membership service is broken into an application interface layer (GAI) which interacts with applications, and a virtual synchrony and membership layer (VSM) which delivers messages in order and informs the GAI of changes in membership. The group membership architecture is applied to a DHCP-like application for assigning IP addresses. The group is defined as the set of all existing hosts. When new hosts join the group and request an IP, the application performs a 2-phase commit on the group to ensure an unused IP is assigned.

8 The Future

Défago, in "Distributed computing on the move: From mobile computing to cooperative robots and nanorobotics", considers future application of ad hoc computing to cooperative robotics and nanorobotics. Défago advocates a three phase process of identifying applications for this domain, discovering generic problems and models for the applications, and devising and verifying algorithms for the problems. Due to the autonomy and complexity of robots, mobile systems of robots may require sophistication beyond what is available today in mobile research. It may be helpful to draw inspiration from diverse sources such as biological or economic systems.

A Survey of PODC '01

Sergio Rajsbaum

This year marks the 20th anniversary of the ACM Principles of Distributed Computing conference, and indeed it was a very special event, coinciding with Leslie Lamport's sixtieth birthday. In addition, PODC was run in concert with a special Workshop on Principles of Mobile Computing (POMC), surveyed earlier in this column. The second annual PODC Influential Paper Award was presented. This year marks the beginning of a collaboration between PODC and the Springer-Verlag journal Distributed Computing. See the PODC'01 web site http://www.podc.org/podc2001/ for additional information.

1 Leslie Lamport's 60th birthday Celebration

Leslie's celebration brought to PODC a very warm atmosphere, and gave us the chance to reflect on the history of our field through his many contributions in mutual exclusion, logical clocks, Byzantine agreement, one-time signatures, atomic registers, sequential consistency, theory of specification, replicated state machines, LaTeX. Last year his paper, *Time, Clocks, and the Ordering of Events in a Distributed System*, received the first PODC Influential Paper Award. His work on the Paxos consensus algorithm was discussed in various talks. Many of the papers people write today have their origins in questions asked first by Lamport.

Attendees, included several old timers that do not attend PODC regularly anymore, and others that do not work in PODC areas, were attracted by the celebration, enriching the conference with their presence. A small sample of the nice atmosphere is given by the following toast, beautifully read by the author the night of the banquet.

Toast for Leslie Lamport
by Fred B. Schneider

I've been asked to toast Leslie Lamport. There is little doubt what Leslie would do in this situation:

We start with a specification of a toast. The liveness property is easy: Starvation is avoided provided the toast terminates. The safety property? Establish, by words and actions: "Lamport is toast."

To discharge the safety part, we turn to the question of specifying Leslie Lamport. "A scout is trustworthy, loyal helpful friendly, courteous kind, obedient, cheerful, thrifty, brave, clean, and reverent." Clearly, Les is not a boy scout.

But there can be no question the Leslie is brave. For him, discretion is not the better part of valor. Most of us put-up with lousy typesetting systems, lousy formal notations (proofs and the like), and lousy logics. Lamport wouldn't–and didn't. This led to TLA, Latex, and structured proofs. And now, most of us put-up with HIS lousy typesetting system, lousy formal notations, and lousy logic.

Because Leslie Lamport is brave, he asks those embarrassing hard questions...

- *How about building mutual exclusion without first assuming it?*
- *How about writing fully formal proofs?*
- *How about failures that don't get detected?*

And we have seen how these questions have come to define the scope of PODC.

So while many here have toasted Leslie Lamport for this with their invited talks, I'll not do that. Tonight I would like us – in this first toast – to toast Leslie Lamport the scientist. For that is what sets him apart, in my opinion. He might not dress the part of the gentleman scientist of yesteryear, but:

- *This is a guy who always returns his referee reports on time,*
- *who never publishes the same result in multiple places,*
- *who writes the paper before he writes the extended abstract,*
- *and who answers his mail – even from random no-names asking stupid questions.*

Leslie, with sneakers and tee shirt, who has always been a child of the '60's, I propose a toast to you, ever the scientist, a valued friend and collaborator, now a child in your 60's.

Cheers!

The celebration, chaired by Cythia Dwork, featured a series of lectures illustrating and celebrating the impact of the work of Leslie Lamport, whose abstracts are included in the proceedings:

- Martín Abadi "Leslie Lamport's Properties and Actions"
- Jim Anderson "Lamport on Mutual Exclusion: 27 Years of Planting Seeds"
- Maurice Herlihy "On Beyond Registers: Wait-free Readable Objects"

- Butler Lampson "The ABCD's of Paxos"
- Nancy Lynch "Implementing Atomic Objects in a Dynamic Environment"
- Amir Pnueli "Sticks and Stones: A Coding Scheme for Parameterized Verification"
- Chris Rowley "The Latex Legacy 2.09 And All That"

2 Best Student Paper Award

The paper "An Efficient Distributed Algorithm for Constructing Small Dominating Sets" by Lujun Jia, Rajmohan Rajaraman, and Torsten Suel was the winner of this award. Lujun is a PhD student of Rajmohan at Northeastern University.

The dominating set problem asks for a small subset D of nodes in a graph such that every node is either in D or adjacent to a node in D. They present randomized distributed algorithms for this problem that run in polylogarithmic time, independent of the diameter of the network, and that return a dominating set of size within a logarithmic factor from optimal, with high probability. Finding a dominating set of minimum size is NPcomplete, and the best known approximation is logarithmic in the maximum degree of the graph.

3 2001 PODC Influential Paper Award

The PODC Influential Paper Award was created to acknowledge an outstanding paper on the principles of distributed computing, whose significance and impact on the theory and/or practice of distributed computing has been evident for at least a decade. This year is the second time the award is presented at PODC. The paper that received it is

> "Impossibility of Distributed Consensus with One Faulty Process," Michael Fischer, Nancy Lynch and Michael Paterson *Journal of the ACM,* vol. 32, no. 2, April 1985.

Members of the selection committee were Richard Ladner, Yoram Moses, Michael Saks, and Nir Shavit (Chair). The following is a description of the winning paper's contributions, as described by Jennifer Welch and Vassos Hadzilacos (with minor editing), reproduced from the conference proceedings.

Jennifer Welch writes the following. The result of this paper (commonly known as FLP) is that, surprisingly, it is impossible for a set of processors in an asynchronous distributed system to agree on a binary value, even if only a single processor is subject to an unannounced crash. Although the result was motivated by the problem of committing transactions in distributed database systems, the proof is sufficiently general that it directly implies the impossibility of a number of related problems, including consensus.

This result has had a monumental impact in distributed computing, both theory and practice. Systems designers were motivated to clarify their claims concerning under what circumstances the systems work.

On the theory side, people have attempted to get around the impossibility result by changing the system assumptions or the problem statement. Work on changing the system assumptions includes the study of partially synchronous models and of various kinds of failure detectors. Modified problem statements include randomized algorithms, approximate agreement, k-set agreement, and condition-based approaches.

The proof technique used in FLP, valency arguments, has been used and adapted to show many other impossibility and lower bound results in distributed computing. These include impossibility results for consensus, k-set consensus, and renaming in various models, and lower bounds on contention and on the number of rounds for synchronous consensus.

The FLP result forms the basis of work on the wait-free hierarchy, in which data types are classified and compared according to the maximum number of processes for which they can solve wait-free consensus. The calculation of consensus numbers relies on valency arguments.

Finally, work on applying ideas from topology to fault-tolerant distributed computing were inspired by the posing of the k-set consensus problem, which in turn was inspired by the FLP result.

Vassos Hadzilacos writes the following. This paper proved to be the most fundamental result in fault-tolerant distributed computing. The result asserts that a particularly simple and natural problem, consensus, is not solvable in the very attractive asynchronous model of computation, even under strong assumptions about the number and nature of failures that may occur — specifically, even if communication is perfectly reliable, if at most one process can fail, and if it can do so only by crashing.

This result has been extremely influential both because of its implications and because of the proof technique it pioneered. Its importance stems from the fact that consensus lies at the heart of many practical problems, including atomic commit, leader election, atomic broadcast, and the maintenance of consistent replicated data. The unsolvability result has motivated researchers to explore models that retain as many of the asynchronous model's attractive features as possible, while making consensus (and related practical problems) solvable. These explorations include randomization, partially synchronous models, and unreliable failure detectors. The proof technique introduced in the nominated paper, now know as "the bivalence argument", is notable for its originality, its elegance, and its wide use to obtain other impossibility results in distributed computing. The paper is written masterfully, and although it contains an unusually deep result, it is accessible even to advanced undergraduate computer science students.

* * *

The following is a brief historical perspective of the paper written by the authors themselves.

Nancy Lynch and Mike Fischer began working on the distributed consensus problem in early 1980 and proved a lower bound on the number of rounds needed to reach agreement in a synchronous distributed setting.

During 1981-82, Lynch worked with Danny Dolev and others on an approximate version of the consensus problem. For this problem, it turned out that the algorithms for the synchronous setting extended easily to the asynchronous setting, simply by adding enough extra processes to accommodate the differences in process views due to asynchrony. This observation led her to wonder whether synchronous algorithms for exact agreement could similarly be adapted to the asynchronous setting. She made no progress on such an adaptation, however.

Independently, Fischer was introduced to the problem of asynchronous consensus by Butler Lampson during a visit to Xerox PARC in early summer 1982. Lampson suggested an algorithm to solve the problem, and Fischer was anxious to understand it and to prove it correct. Although he made some progress on the former, he failed utterly on the latter.

Later that summer, Fischer, Lynch, and Mike Paterson tried to resolve this problem during a visit by Fischer to MIT, a subsequent visit by Paterson to Yale, and some phone calls. We worked simultaneously on trying to produce an algorithm and trying to prove that no such algorithm exists. Only gradually did we come to realize that the problem is fundamentally unsolvable.

The intuition behind the impossibility proof is pretty simple: Initially, either decision, 0 or 1, is possible. Assume a correct algorithm. At some point in time the system as a whole must commit to one value or the other. That commitment must result from some action of a single process. Suppose that process fails. Then there is no way for the other processes to know the commitment value; hence, they will sometimes make the wrong decision. Contradiction!

This simple argument turned out to be surprisingly difficult to make precise. The subtlety of the argument became apparent to us when we tried to explain the result to others: many people expressed skepticism and disbelief. It took a lot of time, and careful polishing of the proof on our part, before the correctness of the result became generally accepted.

Little did we imagine how influential the paper would turn out to be! We assumed that the main value of our impossibility result was to close off unproductive lines of research on trying to find fault-tolerant consensus algorithms. But much to our surprise, it opened up entirely new lines of research. There has been analysis of exactly what assumptions about the distributed system model are needed for the impossibility proof. Many related distributed problems to which the proof also applies have been found, together with seemingly similar problems which do have solutions. Eventually a long line of research developed in which primitives were classified based on their ability to implement wait-free fault-tolerant consensus.

4 PODC 2001 Business Meeting

Thanks to Gil Neiger, who chaired the Business Meeting, for preparing this information.

The meeting proceeded as follows:

- Report from 2001 Program Chair (Nir Shavit)
- Report from 2000 Treasurer (Ajay Kshemkalyani)
- Report from 2001 Treasurer (Aleta Ricciardi)
- Report on 2001 Local Arrangements (Alex Shvartsman)
- Report on 2002 Local Arrangements (Gil Neiger for Rajeev Joshi)
- Presentation by 2002 Program Chair (Keith Marzullo)
- PODC 2003 planning
- Status of endowment of PODC Influential Paper Award
- New relationship between PODC and *Distributed Computing*
- Special election for Steering Committee Member At-Large

PODC 2000

Ajay Kshemkalyani, the Treasurer for PODC 2000, gave a report on the final budget from PODC 2000. Revenues were $29,456 and expenses $27,542 for a surplus of $1,914. The surplus has been transferred to the endowment for the PODC Influential-Paper Award (see below).

PODC 2001

Ajay Kshemkalyani, the Treasurer for PODC 2000 and General Chair for PODC 2001, thanked the following donors for their generous donations to PODC 2001: Addison-Wesley; AT&T Labs; Compaq Computer Corporation; IBM Research; Intel Corporation; Lucent Technologies; Microsoft Research; Motorola Labs; Sun Microsystems; University of Connecticut and its Booth Research Center; and Valaran Corporation.

Alex Shvartsman, the Chair of Local Arrangements for PODC 2001, thanked the rest of the local-arrangements staff: Alex Russell (vice chair), Alaa Amin, Chryssis Georgiou, Kishori Konwar, Greg Malewicz, Mariam Momenzadeh, and Peter Musial. He also thanked the University of Connecticut and its Booth Research Center for financial contributions and the loan of audio-visual equipment; and Robin Industries of Bristol, Rhode Island, for the donation of New England braided chair pads.

PODC 2002

PODC 2002 will be held July 20th–24th, 2002 at the DoubleTree Hotel Monterey in Monterey, California. The conference will begin with an opening reception the evening of *Saturday*, July 20th (tutorials may be scheduled earlier that day). There will be three full days of technical sessions Sunday–Tuesday, July 21st–23rd. The conference will end at noon on Wednesday, July 24th.

The DoubleTree Hotel Monterey is located in downtown Monterey in the Fisherman's Wharf area, adjacent to the Monterey Conference Center. The following is the conference hotel's URL: `http://www.doubletreemonterey.com/`

Keith Marzullo, the Program Chair for PODC 2002, discussed his philosophy for broadening the scope of PODC in 2002:

> PODC was created to provide a forum for the discussion of principles of distributed computing. Principles can also be about specification, the influence of hardware on the software architecture, performance and applications as well as about PODC's traditional interests in theory and algorithms. We have broadened the interests of the program committee to better represent the kind of papers we are trying to attract. We are especially interested in high-quality papers that articulate a principle of distributed computing and its impact.

The conference will be actively seeking papers reporting on the implementation, analysis, evaluation, and deployment of real systems. The Program Committee has been expanded to comprise 21 members, many of whom pursue research in these categories. The Program Committe will favor work that explores new territory or reflects on experience with practical applications of the community's knowledge.

The PODC 2002 call for papers is now available at the following URLs:

`http://www.podc.org/podc2002/cfp.ps`
`http://www.podc.org/podc2002/cfp.pdf`

The PODC 2002 Conference Committee comprises Elizabeth Borowsky (Treasurer), Rajeev Joshi (Local Arrangements), Hyunyoung Lee (Publicity), Keith Marzullo (Program Chair), Gil Neiger (Webmaster), and Aleta Ricciardi (General Chair).

PODC 2003

The meeting considered two principal possibilities for PODC 2003.

The first option was to join the fourth ACM Federated Computing Research Conference (FCRC). PODC had earlier joined FCRC in 1996 (Philadelphia) and 1999 (Atlanta). FCRC 2003 is scheduled for June 7–14, 2003 will be held in a suburb of San Diego, California.

The second option was presented by Mark Moir of Sun Labs and would be to hold PODC in the Boston area (for the first time). Under this option, PODC would be run as a stand-alone conference at a time chosen by the PODC community (likely mid-July to mid-August).

There was considerable discussion of the pros and cons of the two sites. It focused more on the relative hotel costs of the different options and less on the distinction between joining FCRC and running alone. A vote was taken, and it was decided to hold PODC 2003 in Boston. Mark Moir, who will chair local arrangements, was requested to seek out affordable options there. Paul Attie will serve as vice chair for local arrangements.

PODC and *Distributed Computing*

Two special collaborations were announced between PODC and the Springer-Verlag journal *Distributed Computing* whose Managing Editor is Vassos Hadzilacos.

The first is that there will be special annual issues of *Distributed Computing* to contain a selection of papers from each PODC. PODC Program Chairs will serve as guest editors for these issues, with James Anderson already serving in this capacity for an issue of papers from PODC 2000.

The second collaboration will be a special issue of *Distributed Computing* to celebrate PODC's 20th anniversary. This issue will contain surveys, retrospectives, and personal perspectives. It will be edited by Hagit Attiya and Sergio Rajsbaum. More information about this issue can be found at the following URL: `http://www.cs.technion.ac.il/~hagit/podc20/`

Special Election

PODC 2000 added a new member to the PODC Steering Committee, the Member At-Large. This is a 3-year position whose term coincides with that of the Chair of the committee. In 2000, Keith Marzullo was elected to this position for a term to run through the end of PODC 2003. Because he will be the Program Chair for PODC 2002, Dr. Marzullo will be *ex officio* on the Steering Committee through the end of PODC 2004. For this reason, a special election was held to select a new At-Large Member to serve through PODC 2003.

There were three nominees for this position: Rachid Guerraoui, of École Polytechnique Fédérale de Lausanne (EPFL); Yoram Moses, of the Technion; and André Schiper, also of EPFL. Yoram Moses was elected.

Paxos Made Simple

Leslie Lamport

Abstract

The Paxos algorithm, when presented in plain English, is very simple.

1 Introduction

The Paxos algorithm for implementing a fault-tolerant distributed system has been regarded as difficult to understand, perhaps because the original presentation was Greek to many readers [5]. In fact, it is among the simplest and most obvious of distributed algorithms. At its heart is a consensus algorithm—the "synod" algorithm of [5]. The next section shows that this consensus algorithm follows almost unavoidably from the properties we want it to satisfy. The last section explains the complete Paxos algorithm, which is obtained by the straightforward application of consensus to the state machine approach for building a distributed system—an approach that should be well-known, since it is the subject of what is probably the most often-cited article on the theory of distributed systems [4].

2 The Consensus Algorithm

2.1 The Problem

Assume a collection of processes that can propose values. A consensus algorithm ensures that a single one among the proposed values is chosen. If no value is proposed, then no value should be chosen. If a value has been chosen, then processes should be able to learn the chosen value. The safety requirements for consensus are:

- Only a value that has been proposed may be chosen,
- Only a single value is chosen, and
- A process never learns that a value has been chosen unless it actually has been.

We won't try to specify precise liveness requirements. However, the goal is to ensure that some proposed value is eventually chosen and, if a value has been chosen, then a process can eventually learn the value.

We let the three roles in the consensus algorithm be performed by three classes of agents: *proposers*, *acceptors*, and *learners*. In an implementation, a single process may act as more than one agent, but the mapping from agents to processes does not concern us here.

Assume that agents can communicate with one another by sending messages. We use the customary asynchronous, non-Byzantine model, in which:

- Agents operate at arbitrary speed, may fail by stopping, and may restart. Since all agents may fail after a value is chosen and then restart, a solution is impossible unless some information can be remembered by an agent that has failed and restarted.
- Messages can take arbitrarily long to be delivered, can be duplicated, and can be lost, but they are not corrupted.

2.2 Choosing a Value

The easiest way to choose a value is to have a single acceptor agent. A proposer sends a proposal to the acceptor, who chooses the first proposed value that it receives. Although simple, this solution is unsatisfactory because the failure of the acceptor makes any further progress impossible.

So, let's try another way of choosing a value. Instead of a single acceptor, let's use multiple acceptor agents. A proposer sends a proposed value to a set of acceptors. An acceptor may *accept* the proposed value. The value is chosen when a large enough set of acceptors have accepted it. How large is large enough? To ensure that only a single value is chosen, we can let a large enough set consist of any majority of the agents. Because any two majorities have at least one acceptor in common, this works if an acceptor can accept at most one value. (There is an obvious generalization of a majority that has been observed in numerous papers, apparently starting with [3].)

In the absence of failure or message loss, we want a value to be chosen even if only one value is proposed by a single proposer. This suggests the requirement:

P1. An acceptor must accept the first proposal that it receives.

But this requirement raises a problem. Several values could be proposed by different proposers at about the same time, leading to a situation in which every acceptor has accepted a value, but no single value is accepted by a majority of them. Even with just two proposed values, if each is accepted by about half the acceptors, failure of a single acceptor could make it impossible to learn which of the values was chosen.

P1 and the requirement that a value is chosen only when it is accepted by a majority of acceptors imply that an acceptor must be allowed to accept more than one proposal. We keep track of the different proposals that an acceptor may accept by assigning a (natural) number to each proposal, so a proposal consists of a proposal number and a value. To prevent confusion, we require that different proposals have different numbers. How this is achieved depends on the implementation, so for now we just assume it. A value is chosen when a single proposal with that value has been accepted by a majority of the acceptors. In that case, we say that the proposal (as well as its value) has been chosen.

We can allow multiple proposals to be chosen, but we must guarantee that all chosen proposals have the same value. By induction on the proposal number, it suffices to guarantee:

P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

Since numbers are totally ordered, condition P2 guarantees the crucial safety property that only a single value is chosen.

To be chosen, a proposal must be accepted by at least one acceptor. So, we can satisfy P2 by satisfying:

P2^a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v.

We still maintain P1 to ensure that some proposal is chosen. Because communication is asynchronous, a proposal could be chosen with some particular acceptor c never having received any proposal. Suppose a new proposer "wakes up" and issues a higher-numbered proposal with a different value. P1 requires c to accept this proposal, violating P2^a. Maintaining both P1 and P2^a requires strengthening P2^a to:

P2^b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

Since a proposal must be issued by a proposer before it can be accepted by an acceptor, P2^b implies P2^a, which in turn implies $P2$.

To discover how to satisfy P2^b, let's consider how we would prove that it holds. We would assume that some proposal with number m and value v is chosen and show that any proposal issued with number $n > m$ also has value v. We would make the proof easier by using induction on n, so we can prove that proposal number n has value v under the additional assumption that every proposal issued with a number in $m \,..\, (n-1)$ has value v, where $i \,..\, j$ denotes the set of numbers from i through j. For the proposal numbered m to be chosen, there must be some set C consisting of a majority of acceptors such that every acceptor in C accepted it. Combining this with the induction assumption, the hypothesis that m is chosen implies:

> Every acceptor in C has accepted a proposal with number in $m \,..\, (n-1)$, and every proposal with number in $m \,..\, (n-1)$ accepted by any acceptor has value v.

Since any set S consisting of a majority of acceptors contains at least one member of C, we can conclude that a proposal numbered n has value v by ensuring that the following invariant is maintained:

P2^c. For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either (a) no acceptor in S has accepted any proposal numbered less than n, or (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S.

We can therefore satisfy P2^b by maintaining the invariance of P2^c.

To maintain the invariance of P2^c, a proposer that wants to issue a proposal numbered n must learn the highest-numbered proposal with number less than n, if any, that has been or will be accepted by each acceptor in some majority of acceptors. Learning about proposals already accepted is easy enough; predicting future acceptances is hard. Instead of trying to predict the future, the proposer controls it by extracting a promise that there won't be any such acceptances. In other words, the proposer requests that the acceptors not accept any more proposals numbered less than n. This leads to the following algorithm for issuing proposals.

1. A proposer chooses a new proposal number n and sends a request to each member of some set of acceptors, asking it to respond with:

   (a) A promise never again to accept a proposal numbered less than n, and

   (b) The proposal with the highest number less than n that it has accepted, if any.

   I will call such a request a *prepare* request with number n.

2. If the proposer receives the requested responses from a majority of the acceptors, then it can issue a proposal with number n and value v, where v is the value of the highest-numbered proposal among the responses, or is any value selected by the proposer if the responders reported no proposals.

A proposer issues a proposal by sending, to some set of acceptors, a request that the proposal be accepted. (This need not be the same set of acceptors that responded to the initial requests.) Let's call this an *accept* request.

This describes a proposer's algorithm. What about an acceptor? It can receive two kinds of requests from proposers: *prepare* requests and *accept* requests. An acceptor can ignore any request without compromising safety. So, we need to say only when it is allowed to respond to a request. It can always respond to a *prepare* request. It can respond to an *accept* request, accepting the proposal, iff it has not promised not to. In other words:

P1^a. An acceptor can accept a proposal numbered n iff it has not responded to a *prepare* request having a number greater than n.

Observe that P1^a subsumes P1.

We now have a complete algorithm for choosing a value that satisfies the required safety properties—assuming unique proposal numbers. The final algorithm is obtained by making one small optimization.

Suppose an acceptor receives a *prepare* request numbered n, but it has already responded to a *prepare* request numbered greater than n, thereby promising not to accept any new proposal numbered n. There is then no reason for the acceptor to respond to the new *prepare* request, since it will not accept the proposal numbered n that the proposer wants to issue. So we have the acceptor ignore such a *prepare* request. We also have it ignore a *prepare* request for a proposal it has already accepted.

With this optimization, an acceptor needs to remember only the highest-numbered proposal that it has ever accepted and the number of the highest-numbered *prepare* request to which it has responded. Because P2^c must be kept invariant regardless of failures, an acceptor must remember this information even if it fails and then restarts. Note that the proposer can always abandon a proposal and forget all about it—as long as it never tries to issue another proposal with the same number.

Putting the actions of the proposer and acceptor together, we see that the algorithm operates in the following two phases.

Phase 1. (a) A proposer selects a proposal number n and sends a *prepare* request with number n to a majority of acceptors.

(b) If an acceptor receives a *prepare* request with number n greater than that of any *prepare* request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2. (a) If the proposer receives a response to its *prepare* requests (numbered n) from a majority of acceptors, then it sends an *accept* request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

(b) If an acceptor receives an *accept* request for a proposal numbered n, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than n.

A proposer can make multiple proposals, so long as it follows the algorithm for each one. It can abandon a proposal in the middle of the protocol at any time. (Correctness is maintained, even though requests and/or responses for the proposal may arrive at their destinations long after the proposal was abandoned.) It is probably a good idea to abandon a proposal if some proposer has begun trying to issue a higher-numbered one. Therefore, if an acceptor ignores a *prepare* or *accept* request because it has already received a *prepare* request with a higher number, then it should probably inform the proposer, who should then abandon its proposal. This is a performance optimization that does not affect correctness.

2.3 Learning a Chosen Value

To learn that a value has been chosen, a learner must find out that a proposal has been accepted by a majority of acceptors. The obvious algorithm is to have each acceptor, whenever it accepts a proposal, respond to all learners, sending them the proposal. This allows learners to find out about a chosen value as soon as possible, but it requires each acceptor to respond to each learner—a number of responses equal to the product of the number of acceptors and the number of learners.

The assumption of non-Byzantine failures makes it easy for one learner to find out from another learner that a value has been accepted. We can have the acceptors respond with their acceptances to a distinguished learner, which in turn informs the other learners when a value has been chosen. This approach requires an extra round for all the learners to discover the chosen value. It is also less reliable, since the distinguished learner could fail. But it requires a number of responses equal only to the sum of the number of acceptors and the number of learners.

More generally, the acceptors could respond with their acceptances to some set of distinguished learners, each of which can then inform all the learners when a value has been chosen. Using a larger set of distinguished learners provides greater reliability at the cost of greater communication complexity.

Because of message loss, a value could be chosen with no learner ever finding out. The learner could ask the acceptors what proposals they have accepted, but failure of an acceptor could make it impossible to know whether or not a majority had accepted a particular proposal. In that case, learners will find out what value is chosen only when a new proposal is chosen. If a learner needs to know whether a value has been chosen, it can have a proposer issue a proposal, using the algorithm described above.

2.4 Progress

It's easy to construct a scenario in which two proposers each keep issuing a sequence of proposals with increasing numbers, none of which are ever chosen. Proposer p completes phase 1 for a proposal number n_1. Another proposer q then completes phase 1 for a proposal number $n_2 > n_1$. Proposer p's phase 2 *accept* requests for a proposal numbered n_1 are ignored because the acceptors have all promised not to accept any new proposal numbered less than n_2. So, proposer p then begins and completes phase 1 for a new proposal number $n_3 > n_2$, causing the second phase 2 *accept* requests of proposer q to be ignored. And so on.

To guarantee progress, a distinguished proposer must be selected as the only one to try issuing proposals. If the distinguished proposer can communicate successfully with a majority of acceptors, and if it uses a proposal with number greater than any already used, then it will succeed in issuing a proposal that is accepted. By abandoning a proposal and trying again if it learns about some request with a higher proposal number, the distinguished proposer will eventually choose a high enough proposal number.

If enough of the system (proposer, acceptors, and communication network) is working properly, liveness can therefore be achieved by electing a single distinguished proposer. The famous result of Fischer, Lynch, and Patterson [1] implies that a reliable algorithm for electing a proposer must use either randomness or real time—for example, by using timeouts. However, safety is ensured regardless of the success or failure of the election.

2.5 The Implementation

The Paxos algorithm [5] assumes a network of processes. In its consensus algorithm, each process plays the role of proposer, acceptor, and learner. The algorithm chooses a leader, which plays the roles of the distinguished proposer and the distinguished learner. The Paxos consensus algorithm is precisely the one described above, where requests and responses are sent as ordinary messages. (Response messages are tagged with the corresponding proposal number to prevent confusion.) Stable storage, preserved during failures, is used to maintain the information that the acceptor must remember. An acceptor records its intended response in stable storage before actually sending the response.

All that remains is to describe the mechanism for guaranteeing that no two proposals are ever issued with the same number. Different proposers choose their numbers from disjoint sets of numbers, so two different proposers never issue a proposal with the same number. Each proposer remembers (in stable storage) the highest-numbered proposal it has tried to issue, and begins phase 1 with a higher proposal number than any it has already used.

3 Implementing a State Machine

A simple way to implement a distributed system is as a collection of clients that issue commands to a central server. The server can be described as a deterministic state machine that performs client commands in some sequence. The state machine has a current state; it performs a step by taking as input a command and producing an output and a new state. For example, the clients of a distributed banking system might be tellers, and the state-machine state might consist of the account balances of all users. A withdrawal would be performed by executing a state machine command that decreases an account's balance if and only if the balance is greater than the amount withdrawn, producing as output the old and new balances.

An implementation that uses a single central server fails if that server fails. We therefore instead use a collection of servers, each one independently implementing the state machine. Because the state machine is deterministic, all the servers will produce the same sequences of states and outputs if they all execute the same sequence of commands. A client issuing a command can then use the output generated for it by any server.

To guarantee that all servers execute the same sequence of state machine commands, we implement a sequence of separate instances of the Paxos consensus algorithm, the value chosen by the i^{th} instance being the i^{th} state machine command in the sequence. Each server plays all the roles (proposer, acceptor, and learner) in each instance of the algorithm. For now, I assume that the set of servers is fixed, so all instances of the consensus algorithm use the same sets of agents.

In normal operation, a single server is elected to be the leader, which acts as the distinguished proposer (the only one that tries to issue proposals) in all instances of the consensus algorithm. Clients send commands to the leader, who decides where in the sequence each command should appear. If the leader decides that a certain client command should be the 135^{th} command, it tries to have that command chosen as the value of the 135^{th} instance of the consensus algorithm. It will usually succeed. It might fail because of failures, or because another server also believes itself to be the leader and has a different idea of what the 135^{th} command should be. But the consensus algorithm ensures that at most one command can be chosen as the 135^{th} one.

Key to the efficiency of this approach is that, in the Paxos consensus algorithm, the value to be proposed is not chosen until phase 2. Recall that, after completing phase 1 of the proposer's algorithm, either the value to be proposed is determined or else the proposer is free to propose any value.

I will now describe how the Paxos state machine implementation works during normal operation. Later, I will discuss what can go wrong. I consider what happens when the previous leader has just failed and a new leader has been selected. (System startup is a special case in which no commands have yet been proposed.)

The new leader, being a learner in all instances of the consensus algorithm, should know most of the commands that have already been chosen. Suppose it knows commands 1–134, 138, and 139—that is, the values chosen in instances 1–134, 138, and 139 of the consensus algorithm. (We will see later how such a gap in the command sequence could arise.) It then executes phase 1 of instances 135–137 and of all instances greater than 139. (I describe below how this is done.) Suppose that the outcome of these executions determine the value to be proposed in instances 135 and 140, but leaves the proposed value unconstrained in all other instances. The leader then executes phase 2 for instances 135 and 140, thereby choosing commands 135 and 140.

The leader, as well as any other server that learns all the commands the leader knows, can now execute commands 1–135. However, it can't execute commands 138–140, which it also knows, because commands 136 and 137 have yet to be chosen. The leader could take the next two commands requested by clients to be commands 136 and 137. Instead, we let it fill the gap immediately by proposing, as commands 136 and 137, a special "no-op" command that leaves the state unchanged. (It does this by executing phase 2 of instances 136 and 137 of the consensus algorithm.) Once these no-op commands have been chosen, commands 138–140 can be executed.

Commands 1–140 have now been chosen. The leader has also completed phase 1 for all instances greater than 140 of the consensus algorithm, and it is free to propose any value in phase 2 of those instances. It assigns command number 141 to the next command requested by a client, proposing it as the value in phase 2 of instance 141 of the consensus algorithm. It proposes the next client command it receives as command 142, and so on.

The leader can propose command 142 before it learns that its proposed command 141 has been chosen. It's possible for all the messages it sent in proposing command 141 to be lost, and for command 142 to be chosen before any other server has learned what the leader proposed as command 141. When the leader fails to receive the expected response to its phase 2 messages in instance 141, it will retransmit those messages. If all goes well, its proposed command will be chosen. However, it could fail first, leaving a gap in the sequence of chosen commands. In general, suppose a leader can get α commands ahead—that is, it can propose commands $i+1$ through $i+\alpha$ after commands 1 through i are chosen. A gap of up to $\alpha - 1$ commands could then arise.

A newly chosen leader executes phase 1 for infinitely many instances of the consensus algorithm—in the scenario above, for instances 135–137 and all instances greater than 139. Using the same proposal number for all instances, it can do this by sending a single reasonably short message to the other servers. In phase 1, an acceptor responds with more than a simple OK only if it has already received a phase 2 message from some proposer. (In the scenario, this was the case only for instances 135 and 140.) Thus, a server (acting as acceptor) can respond for all instances with a single reasonably short message. Executing these infinitely many instances of phase 1 therefore poses no problem.

Since failure of the leader and election of a new one should be rare events, the effective cost of executing a state machine command—that is, of achieving consensus on the command/value—is the cost of executing only phase 2 of the consensus algorithm. It can be shown that phase 2 of the Paxos consensus algorithm has the minimum possible cost of any algorithm for reaching agreement in the presence of faults [2]. Hence, the Paxos algorithm is essentially optimal.

This discussion of the normal operation of the system assumes that there is always a single leader, except for a brief period between the failure of the current leader and the election of a new one. In abnormal circumstances, the leader election might fail. If no server is acting as leader, then no new commands will be proposed. If multiple servers think they are leaders, then they can all propose values in the same instance of the consensus algorithm, which could prevent any value from being chosen. However, safety is preserved—two different servers will never disagree on the value chosen as the i^{th} state machine command. Election of a single leader is needed only to ensure progress.

If the set of servers can change, then there must be some way of determining what servers implement what instances of the consensus algorithm. The easiest way to do this is through the state machine itself. The current set of servers can be made part of the state and can be changed with ordinary state-machine commands. We can allow a leader to get α commands ahead by letting the set of servers that execute instance $i + \alpha$ of the consensus algorithm be specified by the state after execution of the i^{th} state machine command. This permits a simple implementation of an arbitrarily sophisticated reconfiguration algorithm.

References

[1] Michael J. Fischer, Nancy Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[2] Idit Keidar and Sergio Rajsbaum. On the cost of fault-tolerant consensus when there are no faults—a tutorial. TechnicalReport MIT-LCS-TR-821, Laboratory for Computer Science, Massachusetts Institute Technology, Cambridge, MA, 02139, May 2001. Also published in SIGACT News 32(2) (June 2001).

[3] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.

[4] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[5] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.