REACHING AGREEMENT

A Fundamental Task— Even in Distributed Computer Systems

by Fred B. Schneider, Özalp Babaoğlu, Kenneth P. Birman, and Sam Toueg

Coordinating computers can be as difficult as coordinating the actions of people.

The problem arises in working with a distributed computing system, which consists of a collection of computers interconnected by communication channels. The computers are usually physically separated, and therefore if they fail, they do so independently. This makes it possible for such a system to be fault-tolerant, for data and tasks can be replicated so that if one computer fails, the others can assume its work. Unfortunately, coordinating the actions of a collection of computers when failures *must* be tolerated can be difficult, if not (provably) impossible.

A provably impossible form of coordination is illustrated in the metaphor recounted on the facing page. In the Coordinated Attack Problem, each of the generals can be thought of as a computer and the unreliable messenger as an imperfect communications channel. In terms of the metaphor, we prove that if the communications channel connecting two fault-tolerant computers can lose messages, it is impossible for the computers to agree on whether or not to perform an action suggested by one of them.

Is the Coordinated Attack Problem of practical interest? Unfortunately, it is. If information is to be replicated at computers so that it can remain available despite possible failure of some of those computers, then some arrangement must be made for the replicated data to be kept consistent. This mandates that when one copy of the data is updated, all available copies be updated. Performing the update is an instance of the Coordinated Attack Problem: either all or none of the available copies must be updated.

ATTACKING THE BYZANTINE GENERALS PROBLEM

A second example, the Byzantine Generals Problem, demonstrates another practical problem in distributed computing systems. This problem differs from the Coordinated Attack Problem in two ways. First, Byzantine generals can be traitors and exhibit arbitrary and malicious behavior; in the Coordinated

THE BYZANTINE GENERALS PROBLEM

The generals of the Byzantine army are preparing a final campaign. There are N generals, of whom t, at most, are traitors. The traitorous generals are not known to the others, but may collude and attempt to foil a coordinated attack by the rest. As long as the armies controlled by the nontraitorous generals all attack or all retreat, the campaign will be successful.

The Commanding General, known to all the others, decides whether the army should attack. Generals communicate by means of a reliable messenger service. A protocol is needed that will achieve *Byzantine Agreement:*

Agreement. All nontraitorous generals execute the same action.

Validity. If the Commanding General is not a traitor, then all nontraitorous generals execute her command.

"The generals correspond to processors, traitors...to faulty processors, and the Commanding General's order...to the value of the sensor being read."

Attack Problem, generals always exhibited correct behavior. Second, Byzantine generals have access to reliable communication; in the Coordinated Attack Problem, communication was not reliable.

The Byzantine Generals Problem must be solved whenever processing is replicated in order to counteract the effects of failures in a computing system. This is the basis for triple-modular-redundancy (TMR), which is used by the computing system on board the space shuttle, as well as in other applications in which fault-tolerance is required. The idea is simple. If all processors read the same input from sensors and process it using the same program, then all non-faulty processors will produce identical results. Thus, as long as a majority of the processors are non-faulty, simply voting on the outputs produced by the processors will produce the correct action. A key premise, however, is that all non-faulty processors read the same input. Simply reading from sensors does not ensure that all correct processors will agree on these inputs—a faulty sensor might furnish different values to different processors.

A protocol is required that will permit the non-faulty processors to agree on the value of the sensor. A trivial solution to this problem would be for processors always to use the same value, say 0. This is clearly unacceptable. An agreement protocol must also ensure that if the sensor is non-faulty, processors actually use its value in their computation.

A solution to the Byzantine Generals Problem is exactly the needed protocol. The generals correspond to processors, traitors correspond to faulty processors, and the Commanding General's order corresponds to the value of the sensor being read.

PROCEEDING BEYOND BYZANTIUM

Research aimed at developing and understanding the Coordinated Attack Problem and the Byzantine Generals Problem is actively being pursued at Cornell, M.I.T., and IBM's research laboratory at San Jose. By identifying those aspects of the environment that influence the cost of achieving agreement, researchers learn how to construct agreement protocols that are well suited for particular applications. For example, suppose that the communication channels link only pairs of processors and that up to t processors might be faulty; in this case, achieving agreement can take as many as $t+1$ rounds of message exchange. This is because a faulty processor can send conflicting information along disjoint routes to other processors and can generate spurious messages and then pretend to be forwarding them on behalf of other processors.

One of us (Babaoğlu) has recently shown that as few as two rounds suffice to establish Byzantine Agreement if all processors share broadcast channels. In fact, there exists a continuum of Byzantine Agreement protocols that require between 2 and $t+1$ rounds, depending on the number of broadcast channels available and the interconnection topology of processors. Faster agreement protocols are possible when more processors share more broadcast channels. Clearly, there are trade-offs

involving delay, fault-tolerance, and hardware cost. Broadcast channels help to speed up Byzantine Agreement because when a faulty processor broadcasts a message on such a channel, it has witnesses to its action. Subsequent attempts to send conflicting information can then be detected and ignored by other processors. The Xerox Ethernet and most other local-area communication networks support the requisite broadcast property for these protocols, making them quite practical.

Byzantine Agreement can be made practical even when broadcast channels are not available. It turns out that $t+1$ rounds are necessary to reach agreement only if t failures *actually* occur during execution of the agreement protocol. If f failures occur, where $f<t$, agreement can be achieved in fewer rounds. Thus, the cost of execution is proportional to the amount of fault-tolerance actually needed. Another of us (Toueg) recently was involved in the development of such an early-stopping protocol. It can tolerate as many as $N/3$ faulty processors and terminates in $2f+3$ rounds. Since most executions of the protocol are failure-free, agreement is typically achieved in three rounds. And when t failures occur, the protocol is as efficient as the best previously known protocols—even those that do not support early stopping.

INEXACT AGREEMENTS AND FAULT TOLERANCE

One can devise fast agreement protocols that do not achieve agreement and validity, but come close. For example, an Inexact Agreement (formulated by Schneider) allows processors, each of which has its own value, to converge on some value. While this would not be very useful when the Byzantine generals must decide whether to attack or retreat, it is perfectly appropriate for applications such as clock synchronization. Clocks on computers typically run at different rates and are difficult to synchronize because delays in message delivery are variable—receipt of the message "It is 9:00", for example, tells the receiver very little about what time it is now; it tells only at what time the message was sent. On the other hand, having synchronized clocks can be quite useful, especially in a system intended to be fault-tolerant, since it is easy to detect that a processor has halted by noting that it has taken too long for an expected reply to arrive. An Inexact Agreement can form the basis for a clock synchronization protocol: periodically, processors use Inexact Agreement to agree on a clock value and then reset their local clocks accordingly.

Byzantine Agreement and other protocols that underlie the construction of fault-tolerant systems are complex and subtle. The ISIS Project (under Birman's direction) is concerned with packaging such protocols and building tools that a programmer can use to convert a fault-intolerant program into a fault-tolerant one without worrying about the details of agreement and replication. ISIS programmers have access to a broadcast (agreement) routine that can be used to disseminate information to copies of program modules. The system also provides failure-monitoring facilities so that an ISIS program can reconfigure itself in response to failures. A prototype ISIS is now running on the computer science department's network of DEC VAX

SELECTED READING

Babaoğlu, Ö., and R. Drummond. 1985. Streets of Byzantium: Network Architectures for fast reliable broadcasts. *IEEE Transactions on Software Engineering* SE-11(6): 546–54.

Birman, K. P., T. A. Joseph, T. Raeuchle, and A. El Abbadi. 1985. Implementing fault-tolerant distributed objects. *IEEE Transactions on Software Engineering* SE-11(6): 502–08.

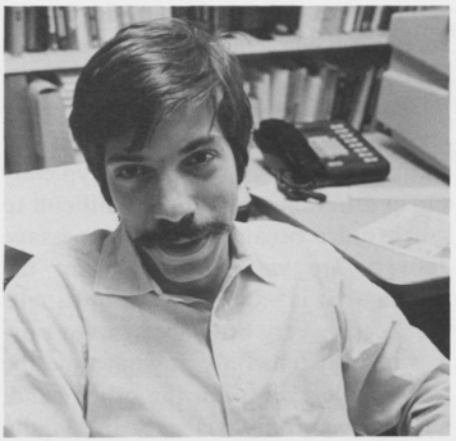
Mahaney, S., and F. B. Schneider. 1985. Inexact agreement: Accuracy, precision, and graceful degradation. In *Proceedings of 4th annual SIGACT-SIGOPS symposium on principles of distributed computing. In press.*

Toueg, S., K. Perry, and T. K. Srikanth. 1985. Fast distributed agreement. In *Proceedings of 4th annual SIGACT-SIGOPS symposium on principles of distributed computing. In press.*

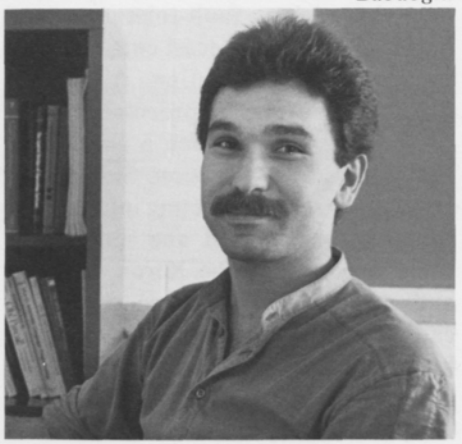
11/780's and SUN Workstations. It has been used to build a number of fault-tolerant application systems and has performed surprisingly well.

Reaching agreements and tolerating faults are modes of conduct for computer-to-computer as well as person-to-person communication. At least in the world of computers, we are finding that logic and imagination are the keys to implementation.

Schneider



Babaoğlu



Birman

Toueg

The four authors of this article are all faculty members of Cornell's Department of Computer Science.

Fred B. Schneider, an associate professor, studied at Cornell for his undergraduate degree, awarded in 1975, and received his doctorate from the State University of New York at Stonybrook in 1978. His main interests are in programming methodology, concurrency, and distributed systems; he is currently writing a text on concurrent programming.

Özalp Babaoğlu, an assistant professor, came to Cornell in 1981 from the Lawrence Berkeley Laboratories, where he was a staff scientist. He holds the B.Sc. degree from George Washington University and the Ph.D., granted in 1981, from the University of California at Berkeley. While in graduate school, he spent a summer at the IBM Research Laboratory in San Jose, and a semester as a foreign scholar at the National Research Council in Pavia, Italy. In 1982 he was a co-recipient of the Sakrison Memorial Award for his contributions to the Berkeley UNIX operating system. His specialties are operating systems, performance evaluation and modeling, and distributed systems.

Kenneth P. Birman received his undergraduate degree from Columbia University in 1978 and his doctorate from the University of California at Berkeley in 1981. At Columbia he helped develop computer systems for analyzing 24-hour electrocardiogram recordings, and after earning his doctorate he spent six months in the cardiology department of the Vienna state hospital in Austria, developing a database system for clinical use. At Berkeley he developed a network version of the UNIX operating system. He joined the Cornell faculty as an assistant professor in 1982.

Sam Toueg, an assistant professor, studied at the Israel Institute of Technology for his B.S. degree, awarded in 1977, and went to Princeton University for graduate work. After receiving the Ph.D. in 1979, he was a postdoctoral fellow at IBM's T. J. Watson Research Center and then came to Cornell in 1981. A specialist in computer networks and distributed computing, he has been a referee for several professional journals.