Cooperative Chaos*

Geoffrey H. Kuenning David H. Ratner Peter Reiher Gerald J. Popek
Richard G. Guy†

Technical Report UCLA-CSD-970041

November 14, 1997

Abstract

Future computers will be smaller, more targeted to special applications, more ubiquitous, and more willing to communicate than existing machines. This will lead to a dynamic environment where thousands or even millions of computers will be able to share information with nearby devices, and significant new applications will become possible by doing so. Such a situation is potentially chaotic, yet these machines will benefit from cooperation. We propose a new paradigm, called *Cooperative Chaos*, that recognizes the inevitability of this development and provides mechanisms to deal with it.

1 Introduction

Traditionally, much of computer science has concentrated on the monolithic computer, which performs most tasks locally with relatively little communication. The notable exception is the massively parallel system, in which communication dominates in the attempt to solve a single problem rapidly. Both of these paradigms share the characteristic of *organization*, in which the structure of the system has been carefully arranged to address the problem at hand.

In recent years, a new trend has emerged. Embedded systems, always important to industry, have become ubiquitous while gaining significant processing power. For example, the modern automobile contains more aggregate processing power than yesterday's mainframes. At the same time, communication capabilities are becoming common and even expected. Most portable computers are able to exchange data with larger static machines and networks, and there is even a wristwatch that can download calendar data from another computer. Various researchers [10, 11] have built systems to allow laptop and palmtop machines to share data dynamically and opportunistically. The average computer user today depends on communication, and becomes frustrated or crippled when it is unavailable.

These trends are likely to accelerate, and we believe that multitudes of tiny, communicating machines will become a normal computing environment. Although the general-purpose computer will not disappear, future computing will be dominated (at least numerically) by embedded systems and special-function devices. Individuals and the devices they use will represent tens or even hundreds of processors, most of which will be small and of limited functionality.

It will be useful for many, if not most, of these computers to communicate with each other. For example, vehicles on a freeway could share information with their neighbors to form a global picture of traffic

*This work was partially supported by the Advanced Research Projects Agency under contract N00174-91-C-0107.

†The authors are affiliated with the Computer Science Department, University of California, Los Angeles. Gerald Popek is also affiliated with Platinum *technology*. E-mail: {geoff,ratner,reiher,rguy}@fmg.cs.ucla.edu popek@platinum.com

conditions that could then be used to improve flow or reroute drivers around traffic jams. Disaster-response teams, or soldiers on a battlefield, could coordinate and plan their actions on both the local and global level. Mobile knowledge workers could easily share work and synchronize their meetings. A large proportion of these communications will take place over wireless links with limited range and bandwidth.

1.1 Cooperative Chaos

The idea of many computers communicating local information to nearby machines is very attractive until one considers the scale involved. With rare exceptions, current research has tended to concentrate on communication patterns in the dozens, not the hundreds or thousands. The exceptions address scaling by discarding difficult features[1] or exploiting organizational regularity that is externally imposed (e.g., Internet naming.) But many existing and emerging scenarios, such as those given above, could benefit from communicating with extremely large numbers of partners.

Adding to the challenge, the communications patterns in many scenarios are changing constantly. On the freeway, for instance, cars are constantly moving into and out of communication range. On the battlefield or during a disaster response, individual units change their configurations, lose and regain their communication capability, and change their desired communication partners. Such an environment is best labeled *chaotic*. External control is limited and communication patterns are unpredictable. Incompleteness and inaccuracies in the shared data may arise from dropped messages or incorrect observations. If computers in such an environment wish to communicate, they must not be dependent on constant behavior or a static configuration. Instead, they must be able to adapt and react almost instantly to the current global configuration, without slowing, losing data, or becoming confused. They must also be tolerant of incorrect data, integrating information from multiple, possibly unreliable, sources using algorithms that will converge to a correct result.

[1] For example, Usenet is based on an inefficient flooding algorithm and prohibits updates to previously posted articles.

We envision a design that will support *cooperative chaos*. By this we mean an uncontrolled, constantly changing environment in which large numbers of computers share information. Some of the information might be transmitted by request-response protocols, but we expect that most data will be flooded throughout the immediate vicinity. Highly flexible and adaptive protocols will be necessary; it is likely that the concept of timeouts and retries will be discarded. The system must never depend on the participation of any particular machine or the arrival of any particular message. Distance-based deterioration will also be required, so that machines far from a particular neighborhood are not overwhelmed with unnecessary information. At the same time, there must be support for transferring targeted messages to remote machines regardless of the intervening path, and for mechanisms that will allow far-away machines to develop a global overview as well as local details.

Our proposal is related to *epidemic algorithms* [1], but differs from that work in the proposed scale and in our increased willingness to tolerate inconsistency between replicas.

1.2 Inaccuracy and Inconsistency

A critical difference between prior systems and Cooperative Chaos is that earlier research concentrated on providing all machines with a consistent global view of their shared data. Optimistic systems such as Ficus [7] and Coda [4] are designed to tolerate temporary inconsistencies, but their correctness depends on making progress toward an eventual globally consistent state.

Ficus was designed so that although the underlying network may be partially unavailable, eventual global consistency will be achieved (barring permanent network partitions). Other replication systems such as Coda make similar guarantees. This is a significant achievement, but the requirement for global agreement limits the scalability of the system. By contrast, Cooperative Chaos extends the broken-network assumption further, designing for the situation where communications are so variable, and the scale so large, that even eventual consistency is not a reasonable goal.

1.3 Characteristics of a Solution

A good solution to these emerging problems should have the following characteristics:

- Scale to very large numbers of participants
- Be completely distributed, requiring no central authority
- Support dynamic and rapidly changing communications patterns and group membership
- Tolerate rapid and dynamic update patterns
- Tolerate approximate and occasionally inaccurate data
- Limit the computational and data-storage burdens placed on participating sites
- Support flexible data storage patterns (a site may choose whether to store any individual datum)
- Allow "aging" of data with time or distance

In addition, the new paradigm will require rethinking of many traditional problems and their solutions, including routing, congestion control, and security models.

2 Supporting Chaos

We believe that the Cooperative Chaos environment will require information to be transferred primarily by near-neighbor communications. There will be many sources of information, many consumers, and many small pieces of data. In such a system, it is not practical to tag each item with a large amount of identifying information, nor is it practical for every site to try to remain aware of all the information provided by other sites (in many cases, it isn't even desirable).

One approach to scaling distributed systems is to divide nearby machines into groups, and then to treat the groups themselves as "super-machines." However, for extremely large numbers of machines this requires multiple levels of grouping, which introduces serious difficulties in an environment as dynamic as that we envision. Also, the machines managing or representing the higher-level groups become critical bottlenecks, and often must maintain some sort of information about all of the machines beneath them, which can become an unmanageable quantity of data at extremely large scales.

Instead, we believe a solution will require an entirely new paradigm, one that is not dependent on any sort of arbitrary scale-reduction tricks. One proposal we are considering is based on the way information travels though large groups of people; for this reason we call it the "Murmuring Crowd" design.

2.1 Neighbor Communication

In a large crowd, messages generally don't travel via long-distance hops. Instead, communication is limited to immediate or nearby neighbors. Nevertheless, information is successfully transferred far beyond the immediate vicinity of its source.

In itself, the idea of moving information via a relay is not new; indeed, this is the basic transport mechanism of the Internet. However, unlike normal Internet routing, our information is not transmitted with a specific destination in mind. Instead, it is provided gratis to neighboring sites, who may do with it what they wish, including ignoring it, forwarding it verbatim, or integrating it with other information and forwarding only the result.[2] This is similar to the way IP routing information is passed, but in our case the neighborhood is constantly changing.

The idea of gratuitously multicasting information to neighbors underlies the Usenet flooding mechanism, and in some ways our system is an extension of that design. However, Usenet depends heavily on unique identification and on a lack of updates. Every message (new Usenet article) is assigned a unique identifier at the source (the Message-ID) [2, 3]. This allows a receiving site to detect and discard duplicate messages, so that the flooding algorithm does not result in explosive message growth. In addition,

[2] The design must be careful to avoid overloading receivers with information that they must examine before discarding.

since updates are prohibited, every message has a unique instantiation, so there is no need to detect updated versions or conflicting updates. These restrictions make the Usenet model inappropriate for the problems addressed by Cooperative Chaos.

Often, it will also be desirable to support broadcasts from some special sites. For example, consider the way we learn of world news events. Our first inkling of a major development may come from a co-worker's comment, "Hey, did you hear about..." But we take these casual reports with a grain of salt, because word of mouth is often unreliable. Instead, we turn to a more trusted reporter, such as a local news outlet, to learn more, on the assumption that the professionals will have more accurate and timely information. Similarly, a machine participating in Cooperative Chaos may wish to listen to wide-area broadcasts from an authoritative source and give these more weight in developing its own global picture. Our algorithm allows each machine to independently integrate the various sources of data as it sees fit.

2.2 Integrating Information

The first concept of Cooperative Chaos, then, is neighbor multicast of information, but multicasts are not enough. Updates must be supported in an efficient, general, and flexible manner. Furthermore, an individual site must be able, without extensive computation or communication, to resolve conflicting information received from different neighbors.

In the past, the standard approach to updates has been to tag every datum with a summary of its entire update history, usually via some form of *version vector* [8], which are themselves a refined form of Lamport's clocks [6]. Using the summary, an incoming message can be compared against the current value of a datum to determine whether it is newer, outdated, or *conflicting* (updated by a parallel thread), and appropriate action can then be taken. But version vectors potentially require space proportional to the number of updating sites. Algorithms have been proposed that compress version vectors by eliminating entries for sites that have not generated an update in some time, but these do not work well in the face of frequent changes in the group of updating sites. This forces the tagged datum to be relatively large (to reduce overhead), and does not scale well.

A second problem is the difficulty of integrating multiple closely related data items. On the battlefield, for example, multiple soldiers may submit their observations regarding an enemy position. Some of these observations may be incomplete or inaccurate. An application-specific algorithm, similar to the application-specific resolvers used in distributed file systems [5, 9], must attempt to determine the best approximation to the true situation.

A third difficulty is the need to select only a relevant subset of the available information, or to summarize information into a higher-level picture. The commuter does not need to know anything about the traffic situation behind him or in a distant city, but he may require a general picture of what is happening twenty minutes down the road. This requires a means of selecting a level of detail and of causing information to selectively "decay" as it travels farther from its source.

To address these problems, we propose dispensing with version vectors and their associated guarantee of global knowledge of a datum's update history. Rather than promise that a particular datum will travel long distances, and that a given site will eventually see the latest version of that datum, we have chosen a looser assurance. We will make a strong effort to communicate a datum widely, and will also attempt to achieve a consistent view of its value. But we value efficiency and dissemination of data over precise accuracy, especially since many participants only require a global summary and don't want to be bothered by precise updates.

2.3 The Murmuring Crowd

The key to our current idea is to provide a method for tracking the history of data in an approximate fashion. Each datum can be tagged with the following minimal information:

- The identity of the datum itself
- The identity of the site that originated the information

- Optionally, the identity of the site immediately relaying the information
- A *hop count*, incremented whenever the data is relayed unchanged
- A *fan-in count*, reflecting the number of incoming messages that were included in calculating this datum

These tags are designed to incur relatively low overhead while still providing enough information to allow the integration of incoming data and prevent network overload.

Each incoming datum will be separated according to its identity and hop count. For example, if a disaster response coordinator received updates from several rescuers for the same datum, it would sort them by both the source and the hop count. The assumption is that since the hop count reflects the distance the information has traveled, it will also indicate both the importance of the information to the local site and the reliability that should be accorded it. A weighting algorithm will choose between newer data with a higher hop count and older, but closer, information.

An integration algorithm will then examine all messages for that particular datum and calculate a new estimated value,[3] considering both the hop counts and the fan-in. This new value will then be broadcast to near neighbors, with a hop count of one greater than the maximum of the input hop counts, and with a fan-in consisting of the sum of the input fan-ins. Thus, the hop count reflects the distance the information has traveled (which is an indicator of its accuracy), while the fan-in indicates the number of contributors.

To prevent network overload, summaries will only be transmitted when they have a fan-in exceeding some threshold, or when a (relatively long) timeout expires. In addition, summaries with larger hop counts will be broadcast less frequently, so that information will be updated more slowly as it gets farther from the source.

[3] Many readers will think of the children's game of "telephone," which uses a similar approach to achieve an amusing result by garbling a message. The difference from our proposal is that "telephone" incorporates a single source and severe communication errors, while we assume accurate communication and multiple sources of information that can be integrated to calculate an improved estimate of the correct value of the datum.

2.4 Integration of Data

A key to the success of the Murmuring Crowd design is the correct integration of data. Just as in a crowd of humans, there is a danger that slightly incorrect information will become exaggerated and lead to overall inaccuracy and instability. To prevent this, Cooperative Chaos will require integration algorithms that are provably stable. Devising such algorithms will be a major focus of our future research; here, we merely outline some initial considerations and directions.

As an example, consider the reporting of vehicle density in a traffic system. A particular vehicle would count its neighbors and then broadcast this information as a triple of {*count, center, radius*}.[4] Nearby vehicles, upon receiving these triples, must then integrate them to calculate and rebroadcast traffic density for a wider area.

To avoid instability, a particular vehicle must not incorporate into its calculation a count that is based on a value that the vehicle itself generated earlier. This is prevented by the hop count. In addition, the count must attempt to accurately reflect the number of vehicles in a particular area, without double-counting a significant number of vehicles. This can be achieved by summing non-overlapping reports. A digital filter could also use historical information to smooth the data.

Once an integrated value is calculated, it will be broadcast with an incremented hop count and a larger value for *radius*. In this example, vehicles that are themselves very close to the broadcasting vehicle will most likely ignore this information in favor of values they have calculated for themselves. More distant machines will integrate it into a more global traffic density. At some level, determined by the application, it will become appropriate to relay the information as well as integrating it, and eventually to

[4] Since freeways are essentially linear, the radius would be (almost) one-dimensional.

simply relay without integrating or even to skip the relay altogether. This will avoid useless long-distance propagations.

A major challenge will be to determine whether each integration algorithm must be customized for the particular datum's semantics, or whether there are significant common features in many data integration algorithms that can be generalized and built into the system.

3 Conclusion

The explosion of embedded systems and cheap communication will require new paradigms of computer cooperation. For this purpose, we propose a new direction and design, one that accepts the dynamic and fundamentally chaotic nature of systems composed of huge numbers of individuals. We have identified major requirements for such a design, and proposed some example algorithms to address the difficulties inherent in such a system. We have begun to design a system that will demonstrate the efficacy of this approach. We believe that this will lead to significant new possibilities for mobile computing.

References

[1] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, B.C., August 1987. ACM.

[2] Mark R. Horton. Standard for interchange of usenet messages. RFC 850, Internet Request For Comments, June 1983.

[3] Brian Kantor and Phil Lapsley. Network News Transfer Protocol: A proposed standard for the stream-based transmission of news. RFC 977, Internet Request For Comments, February 1986.

[4] James J. Kistler and Mahadev Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[5] Puneet Kumar and Mahadev Satyanarayanan. Flexible and safe resolution of file conflicts. In *Proceedings of the USENIX Conference Proceedings*, page xxx, New Orleans, LA, January 1995. USENIX.

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

[7] Thomas W. Page, Jr., Richard G. Guy, Gerald J. Popek, and John S. Heidemann. Architecture of the Ficus scalable replicated file system. Technical Report CSD-910005, University of California, Los Angeles, March 1991.

[8] D. Stott Parker, Jr., Gerald Popek, Gerard Rudisin, Allen Stoughton, Bruce J. Walker, Evelyn Walton, Johanna M. Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. *IEEE Transactions on Software Engineering*, 9(3):240–247, May 1983.

[9] Peter Reiher, John S. Heidemann, David Ratner, Gregory Skinner, and Gerald J. Popek. Resolving file conflicts in the Ficus file system. In *USENIX Conference Proceedings*, pages 183–195. University of California, Los Angeles, USENIX, June 1994.

[10] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th Symposium on Operating Systems Principles*, pages 172–183, Copper Mountain Resort, Colorado, December 1995. ACM.

[11] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. An overview of the ParcTab ubiquitous computing experiment. *IEEE Personal Communications Magazine*, 2(6):28–43, December 1995.