Extracting Guarantees from CHAOS

By John Kubiatowicz

The P2P revolution promises freedom from boundaries, censorship, and centralized control. P2P proponents claim the vast untapped resource of personal computers owned by ordinary people can be combined together to build something greater and more reliable than the sum of its parts.

Each participating computer or node in a P2P system is called a "peer," meaning that participants interact as equals. Peers play a variety of roles. When accessing information, they are clients. When serving information to other clients, they are servers. When forwarding information for others, they are routers.

According to the P2P vision, thousands, millions, or even billions of peers may interact in a seething, transient pattern of communication. Architects of these systems claim this chaos can lead to properties such as "durability," "anonymity," "scalability," and "security"—the list goes on. Can we really achieve guarantees in the chaotic P2P environment?

The attainment of guarantees is a central concern of the OceanStore [7] project at Berkeley. As the chief architect of OceanStore, I present some principles that can be used to achieve guarantees in P2P systems. Although I use distributed file service to frame my ideas, the concepts are generally applicable.

Distributed File Service

Distributed file service is a "typical" P2P application. The idea is straightforward: replace the local hard disk of a computer with pools of storage spread throughout the Internet; the computer interacts with a vast web of peers to read or write information. Figure 1 illustrates this idea. Examples include FreeNet [6], Gnutella [6], FreeHaven [6], Publius [6], Intermemory [1], and OceanStore [7], to name a few.

Let us start by highlighting some properties we might want to guarantee for our distributed file service:

Architects of P2P systems claim thousands, millions, even billions will interact in a seething, transient pattern of communication. Can we really achieve guarantees in the chaotic P2P environment?

- *Availability*. Information can be accessed 24 hours a day, seven days a week.
- *Durability*. Information entered into the system will last virtually forever. OceanStore contemplates 1,000-year durability [9].
- *Access control*. Information is protected. Access control includes privacy (unauthorized entities cannot read information) and write-integrity (unauthorized entities cannot change information).
- *Authenticity*. Adversaries cannot substitute a forged document for a requested document.
- *Denial-of-service (DOS) resilience*. It is difficult for an adversary to compromise availability.

Additionally, although recent gains in CPU speed, storage capacity, and network bandwidth have allowed designers to sacrifice efficiency for greater functionality, systems must still provide sufficient *performance* to be usable.

Several interesting new goals have arisen in the P2P domain [6]. Some are difficult to achieve in traditional environments:

- *Massive scalability*. The system works well with thousands, millions, or even billions of clients.
- *Anonymity*. It is impossible or very difficult for an outside observer to ascertain who has produced a document and who has examined it.
- *Deniability*. Users can deny knowledge of data stored on their machines.
- *Resistance to censorship*. No one can censor information once it is written to the system.

It is an open question whether all of these properties can coexist.

The Challenge: Untrusted Components

Let us contrast P2P file service with centralized network file service. Centralized file service has been around since the advent of NFS in the early 1980s. In a centralized system, files are stored remotely, but on professionally managed servers in locked machine rooms. Professional staff can quickly remedy failures and security breaches. Performance problems can be addressed by upgrading centralized resources. In short, centralized systems have components trusted to behave well.

In contrast, P2P systems must deal with an unreliable and untrusted infrastructure. By "unreliable" we mean systems not professionally managed that may crash or fail at any time. Since failure rate grows linearly with system size, large P2P systems are almost guaranteed to have malfunctioning components. By "untrusted," we mean participants could be adversarial, attempting to exploit vulnerabilities, compromise privacy, or damage the system. For example, peers might substitute their own information in place of legitimate data. Or, malicious routers might prevent functioning components from cooperating.

Since individual components are not trustworthy, P2P designers must invoke new design principles to achieve guarantees. Only the aggregate behavior of many peers can be trusted. In the following sections we intentionally blur the distinctions between unreliable and untrusted. Techniques effective for untrusted infrastructures invariably solve issues with unreliable ones (although the converse is not true).

Figure 1. Distributed File Service: Millions of peers around the world manipulate "pools" of data, which provide data storage for workstations, PDAs, and cell phones.

Taming the Chaos

Here we develop a set of mechanisms that can be combined to provide guarantees in an untrusted infrastructure. We start with replication, data location, and cryptography. We finish with a computer system's version of thermodynamics.

Fault tolerance through replication

Redundancy—the use of multiple resources when a single one would suffice—is a powerful mechanism. Redundancy can help both unreliable and untrusted infrastructures by providing online replacements for faulty resources. Naturally, systems exploiting redundancy must provide ways to filter bad resources from good ones. We touch upon cryptographic validation later.

Excessive replication can incur high storage and bandwidth overhead. Thus, several P2P systems, such as Intermemory, OceanStore and FreeHaven, have utilized an efficient form of redundancy called erasure coding in which each chunk of data is transformed into many fragments. The essential property of this transformation is that only a fraction of the fragments must be recovered to reconstruct the data [3].

Figure 2 illustrates the power of this technique. The graph shows the probability that a block of data can be recovered, measured in Fraction of Blocks Lost Per Year (FBLPY). Each curve represents the same storage over-

head with different levels of fragmentation. This graph incorporates the assumption that server failures occur independently and assumes fragments are regenerated at regular intervals [12].

The important point of this figure is the vast difference in durability between the top and bottom curves. The top curve represents standard replication—four complete copies of data—while the bottom curve represents fragmentation into 64 fragments, any 16 of which are sufficient to reconstruct. For a repair interval of six months, the first encoding technique loses 0.03 (3%) of blocks per year while the second loses 10^{-35} of blocks per year.

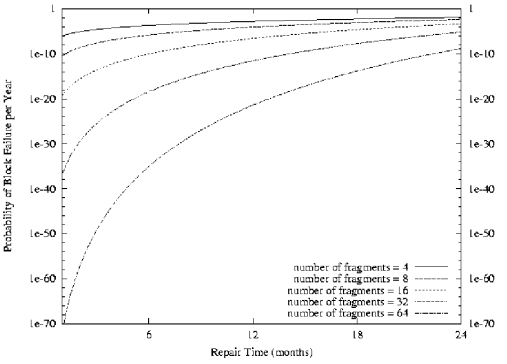

Figure 2. Fraction of Blocks Lost Per Year (FBLPY) as a function of repair interval. Each curve represents a factor of four overhead with different fragmentation. In all cases, one-fourth of fragments are required to reconstruct.

Location-independent routing

Many P2P systems allow objects to be stored anywhere, amidst thousands or millions of peers. If each unique document or endpoint in a P2P system is assigned a globally unique identifier (GUID), then the process of locating data can be viewed as a routing problem: clients construct messages addressed with GUIDs and let the infrastructure pass these messages from peer to peer until the target is located. Since this type of routing involves cooperative decision making, we classify it as decentralized object location and routing (DOLR), as shown in Figure 3. Note that GUIDs are "pure names" that encode nothing about the location of the objects to which they refer.

The DOLR abstraction is powerful and represents a fundamentally new paradigm, namely the ability to route messages directly to objects without knowing their location. Above the DOLR interface, clients can transparently replicate, destroy, and migrate data to meet application-level goals. Below the interface, the system can utilize multiple simultaneous paths to gain reliability and performance. It can tolerate broken routing links, bad servers, and inconsistent paths by retrying or replicating requests. In fact, DOLR networking is a natural way in which third-party routers in the infrastructure can improve the behavior of P2P applications.

For the remainder of this article, we will assume the routing process is deterministic—able to find at least one object with a given GUID when it exists. If more than one object posseses the same GUID (for replicas), then the network will locate one of them. DOLRs such as CAN [1], Chord [11], Pastry [10], and Tapestry [4] provide this property, while the DOLRs associated with most of the original P2P systems do not.

In addition, message routing should provide locality—the use of local resources over global ones whenever possible. Locality rewards good placement decisions with short network traversals between clients and objects. A DOLR with locality will route to the closest of a set of replicas of data. Tapestry and Pastry provide locality directly, while CAN and Chord can be adapted to provide locality.

Efficient use of communication is essential to achieving P2P guarantees. Locality improves performance and increases availability, since the probability of transmission failure increases with distance. Further, the ability of a P2P system to survive a denial-of-service attack can be viewed as its ability to efficiently dissipate traffic from attackers.

Three of the more interesting P2P goals, namely anonymity, resistance to censorship, and deniability derive from the underlying DOLR. The best techniques for anonymity employ tortuous routing paths through many nodes to obscure associations between requestors and destinations. Resistance to censorship is a form of anonymity that prevents adversaries from discovering (or interfering with) servers that export particular information. Deniability is similar, except that it often involves hiding the existence of information from servers themselves—leading to information leakage. Note that techniques to provide some or all of these properties may interfere with efficient routing.

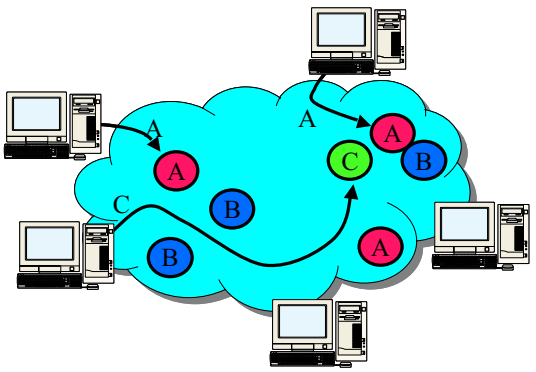

Figure 3. Decentralized Object Location and Routing (DOLR) abstraction: Messages are addressed to objects rather than to IP addresses. DOLR infrastructures provide deterministic routing finding objects if they exist. They also provide locality utilizing local resources and finding local objects whenever possible.

Cryptography

In an untrusted infrastructure, adversaries may improperly acquire information and

violate the privacy of users. Further, adversaries may substitute bad data for good data or subtly corrupt existing data. Thus, the authenticity (source) and integrity (correctness) of information is often in question.

Cryptography can address these issues. Privacy is protected through encryption, the scrambling of information such that only those with the proper key can unscramble it. Authenticity and integrity can be addressed through a combination of secure one-way hash functions and signatures. A secure hash function takes an arbitrary-size block of data and produce a fixed-size summary (for example, SHA-1 produces a 160-bit summary). The result is secure since it is computationally infeasible to find another block of data producing the same summary. Further, the result is unique, since the probability of "stumbling" on two blocks of data with the same hash value is extremely unlikely.

Consequently, summary values can be used as unforgeable names for data. If the GUIDs used by the DOLR network are secure summaries, then clients can verify the integrity of data returned to them by regenerating hashes and comparing them with the requested GUIDs. Clients can also construct recursive, self-verifying objects as trees in which interior blocks contain GUIDs corresponding to other blocks. These objects are named by the GUIDs of their top blocks. The integrity of such objects is ensured, since a substitution within a tree will alter the GUID of the top block.

Signatures utilize public-key cryptography to demonstrate that a particular user generated some piece of information. To sign data, a user maintains two different cryptographic keys, a public key and private key. Although these keys are related mathematically, it is computationally infeasible to generate the private key from the public one. When generating a new document, the user produces a signature over its GUID with his or her private key. This signature is a fixed-sized block of bits that can be verified with the public key. Users keep their private keys private and distribute their public keys to the world.

Byzantine Agreement

When immutable (read-only) data is replicated and explicitly tagged via cryptographic means, it is easy for peers to discard bad information and repair or replicate good information. The decision to use or discard information is passive and local.

In contrast, some decisions are active. For instance, the decision to allow users to change, replace, or delete information can affect the integrity of the system as a whole. This decision involves checking client credentials against an access control list. Given the untrusted nature of the infrastructure, we cannot allow these decisions to be performed by any single, possibly corrupted node.

To address active decision making, several recent systems, such as OceanStore and Farsite [2], have employed Byzantine Agreement [5]. Byzantine Agreement allows a set of peers to come to a unified decision about something, even if some of them (less than one-third) are actively attempting to compromise the process. Should the correct number of nodes agree, the result can even be signed in aggregate with threshold signatures [9] to permit others to verify the decision at a later date.

Many P2P storage systems are advertised as repositories of read-only information. The biggest barrier to providing a writable system is consistency—establishing the identity of the latest copy of data, or conversely, that a particular copy is out of date. Such consistency management usually requires a centralized resource to serialize updates. Byzantine Agreement is an ideal distributed serialization technology.

Correlated Failure Analysis

One assumption that permeates large-scale systems is the belief that components fail independently. When this assumption is violated, many purported guarantees are lost. For instance, replica placement schemes do not protect data when servers holding replicas fail together (are correlated). The extremely low FBLPY values in Figure 2 are only possible when failures occur independently. As another example, Byzantine Agreement algorithms do not function when many servers are corrupted simultaneously.

Unfortunately, correlations exist in all real systems. Peers may share the same subnet, owner, software release, operating system, or geographic location. Most P2P systems rely on random component placement and increased redundancy to combat correlated failures. While effective, these heuristics are not the best way to avoid correlations. More intelligent analyses, such as clustering based on pair-wise correlation, can be used to evaluate independence and adjust resource usage. This is an active research topic.

Exploiting Differences

Existing P2P systems treat the majority of their components as equivalent. This purist philosophy is useful from an academic standpoint, since it simplifies algorithmic analysis. In reality, however, some peers are "more equal" than others:

- Computers have different CPUs, memory, storage capacity, network connectivity, and so forth;
- Some computers are professionally managed and highly available while others are not; and
- Physically, some computers reside at network hubs, while others are at the edges. Some are locked in

machine rooms, while others are public.

Treating all peers as equal forces us to cater to the lowest common denominator.

By exploiting differences, we can better tune performance, availability, reliability, security, and attack vulnerability. Specific examples include:

- Exploitation of "supernodes" with higher connectivity. Supernodes are common in routing infrastructure such as the Internet and can greatly reduce the number of hops a message takes (see [1, 6]).
- Use of actively managed nodes for a Byzantine Agreement, ensuring that no more than one-third of the nodes are compromised.
- Placing archival data on servers deep in mountains to survive a variety of natural disasters.

When P2P techniques move into the mainstream, successful systems will inevitably exploit such differences.

Thermodynamic Systems Design

Finally, large P2P systems must depart from conventional wisdom to achieve guarantees. One promising technique is something we might call "Stability through Statistics"—a form of thermodynamics (or, more precisely, statistical mechanics) for computer systems. Thermodynamics describes the behavior of aggregates (temperature) rather than individual elements (molecules): the temperature of a room is stable even though the kinetic energies of the individual molecules vary widely. Thermodynamics provides important understanding about stability, phase-transitions, and the latent properties of aggregates. Hence, by analogy, we suggest that properly designed systems can exhibit stable behavior by exploiting multiple components and that these systems have thermodynamic descriptions.

Pursuing this analogy for a moment, we note that interacting P2P elements (peers and documents, to name two) are like molecules. The bonds between them (links between peers, relationship between fragments) transmit forces and store energy. Locality in the DOLR provides interaction over short distances. Cryptography enforces the identity of individual elements, thereby simplifying the interaction between them (making their interaction more like a gas rather than a liquid).

Functioning P2P systems contain a wide variety of order—often buried beneath the surface. We will call this the "latent order." For instance, the web of interconnections between DOLR peers combined with directory information on those peers is a very sophisticated form of order that permits objects to be located efficiently. As another example, erasure-coded fragments in Figure 2 are related via a mathematical process, even though they are distributed to random servers.

The latent order can provide stable properties even when individual components vary in their behavior. For instance, when requesting a document, we gain faster average response time with reduced standard-deviation by requesting copies from different servers and utilizing the first returned result. The Tapestry DOLR sends multiple messages along different paths to help ameliorate packet loss and variability in routing performance. Furthermore, systems such as SETI@home ask multiple peers to perform identical computations and exclude bad results through voting. These techniques can be viewed as exploiting thermodynamic stabilization.

The behavior of peers must be peaked around some desirable norm in order to yield stable aggregate behavior. For instance, we might require no more than 10% of the nodes to be malfunctioning. Or, perhaps we require routers to provide a response time that is narrowly peeked about some value. Long-tailed distributions can be countered with a sufficiently redundancy.

Over time, distributions become skewed and the latent order is destroyed by accumulated failures. The number of copies of data, routing pointers, or peers performing computations eventually falls below threshold. This result reflects the second law of thermodynamics: the entropy of closed systems increases. Thus, the passive process of thermodynamic stabilization must be coupled with active entropy reduction, for instance, the addition of energy (through servers) to repair the latent order.

Fortunately, self-organizing behavior tends to reduce entropy. Latent order increases when corrupted elements are removed and ordered elements are added. For instance, to achieve 1,000-year data durability, servers must continuously collect, regenerate, and redistribute fragments (important since the life expectency of individual disks is five years). They may adjust routing links in the DOLR to correct for network changes. They may periodically reevaluate correlations for better resource usage. The challenge is to recognize faults and disordered elements without excessive global communication.

Active entropy reduction falls under the general heading of introspection—an architectural paradigm that mimics the continuous, online feedback that is the hallmark of living organisms. Introspection devotes spare computational resources to observing system behavior, applying analyses (such as clustering, Baysean analysis, and Markov modeling), then adapting the system accordingly. We can view introspection as adding information to the system in order to improve future behavior. Note that companies such as IBM have made adaptive systems (or autonomic computing [see www.research.ibm.com/autonomic]) a highly-visible focus of research.

Thermodynamic systems design thus involves three key elements:

- *Redundancy*. More resources must be utilized than the "bare minimum" required for operation.
- *Replacement*. Some technique must be present to recognize failure and switch from faulty resources to functioning ones.
- *Restoration*. Some process must act to restore latent order (reduce entropy).

It is likely the use of randomness is important as well. Randomness can shield against a variety of systematic biases and attack methodologies.

The thermodynamic point of view can direct our design efforts. For instance, we can ask to what extent our DOLR system can absorb a denial-of-service attack. The answer could involve the DOLR equivalent of heat capacity: to what extent does the web of interconnections between nodes absorb attacks without changing the "temperature" of the DOLR (without shifting or skewing its response distribution). This point of view provides a promising framework for comparing DOLR organizations at a level beyond simple benchmarks.

Reprise—Can We Guarantee Anything?

We return to our original question: can we guarantee anything about P2P systems? The biggest challenge here is the unreliable and untrusted nature of P2P components. Guarantees require a combination of redundancy, cryptography, and thermodynamic stabilization with active repair. Thermodynamic principles suggest that P2P systems can become more stable with increasing component count. Let us briefly recap our goals.

First, scalability requires avoidance of bottlenecks and automatic integration of new peers and removal of old ones. DOLRs with locality permit scalable, flexible, and efficient use of communication.

Availability requires redundancy and continuous repair of both data and routing. Introspective techniques can be utilized to move data close to where it is needed, thereby increasing the chance it can be accessed. Denial-of-service resilience is an extreme form of availability, combined with introspection to recognize and suppress attacks. Similarly, resistance to censorship is a question of availability—making it impossible for anyone to hunt and destroy every copy of a document.

Durability is a function of redundancy and continuous repair with verification. Guaranteed long-term durability is more of a sociological issue than anything else; some entities must be responsible for performing continuous repair. This observation would argue for "service providers" that are paid to repair information.

Cryptography comes to the rescue for several guarantees. For instance, authenticity can be verified by checking cryptographic signatures; privacy can be guaranteed through encryption.

Unfortunately, access control in terms of write-integrity cannot be guaranteed without active, well-behaving components. Byzantine Agreement provides a mechanism for cooperative decision making in the face of malicious elements.

Finally, anonymity and deniability are challenging. Both require obscuring the path of requests to read or publish information, as well as obscuring where information is placed. These properties can be provided in a P2P framework. However, it remains to be seen how well they can be incorporated with other requirements.

Not withstanding, the answer to our original question would appear to be: Yes, we can guarantee interesting properties—even with faulty or malicious components. The "guarantees" in this article are probabilistic. Those uncomfortable with probabilistic arguments should consider that traditional systems fail under many circumstances. Thermodynamic, self-organized systems can provide strong guarantees. The behavior of such systems closely resembles life itself—something greater than the sum of its parts. C

References

1. Albert, R. Jeong, H. and Barabási, A. Error and attack tolerance of complex networks. *Nature 406* (July 27, 2000).
2. Bolosky, W. et al. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of Sigmetrics*, ACM Press, New York, 2000.
3. Goldberg, A.V. and Yianilos, P.N. Towards an archival intermemory. In *Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries*, 1998.
4. Hildrum, K. Kubiatowicz, J. Rao, S.and Zhao, B. Distributed object location in a dynamic network. In *Proceedings of Symposium on Parallel Algorithms and Architectures* (SPAA), August 2002.
5. Lamport, L. Shostak, R. and Pease. M. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 3 (July 1982), 382–401.
6. Oram, A. Ed. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly Books, 2001.
7. Kubiatowicz , J. et al. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of Architectural Support for Programming Languages and Operating Systems* (ASPLOS-IX), ACM Press, New York, 2000.
8. Ratnasamy, S., et al. A scalable content-addressable network. In *Proceedings of SIGCOMM*, ACM Press, 2001.
9. Rhea, S. et al. Maintenance-free global data storage. *IEEE Internet Computing 5*, 5 (Sept/Oct. 2001).
10. Rowstron, A. and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, October 2001.
11. Stoica, I., et al. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceeding of SIGCOMM*, August 2001.
12. Weatherspoon, H. and Kubiatowicz, J. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of First International Workshop on Peer-to-Peer Systems* (IPTPS 2002).

JOHN KUBIATOWICZ (kubitron@cs.berkeley.edu) is an assistant professor at the University of California, Berkeley, and the technical lead and principal investigator for OceanStore.