Self-Organizing Subsets: From Each According to His Abilities, To Each According to His Needs

Amin Vahdat, Jeff Chase, Rebecca Braynard,
Dejan Kostić, Patrick Reynolds, and Adolfo Rodriguez*
Department of Computer Science
Duke University
{*vahdat,chase,rebecca,dkostic,reynolds,razor*}*@cs.duke.edu*

Abstract

The key principles behind current peer-to-peer research include fully distributing service functionality among all nodes participating in the system and routing individual requests based on a small amount of locally maintained state. The goals extend much further than just improving raw system performance: such systems must survive massive concurrent failures, denial of service attacks, etc. These efforts are uncovering fundamental issues in the design and deployment of distributed services. However, the work ignores a number of practical issues with the deployment of general peer-to-peer systems, including i) the overhead of maintaining consistency among peers replicating mutable data and ii) the resource waste incurred by the replication necessary to counteract the loss in locality that results from random content distribution. This position paper argues that the key challenge in peer-to-peer research is not to distribute service functions among all participants, but rather to distribute functions to meet target levels of availability, survivability, and performance. In many cases, only a subset of participating hosts should take on server roles. The benefit of peer-to-peer architectures then comes from massive diversity rather than massive decentralization: with high probability, there is always some node able to step in to provide the required functionality should the need arise.

*This research is supported in part by the National Science Foundation (EIA-9972879, ITR-0082912), Hewlett-Packard, IBM, Intel, and Microsoft. Braynard and Reynolds are supported by an NSF graduate fellowship and Vahdat is also supported by an NSF CAREER award (CCR-9984328).

1 Introduction

Peer-to-peer principles are fundamental to the concept of survivable, massive-scale Internet services incorporating large numbers—potentially billions—of heterogeneous hosts. Most recent peer-to-peer research systems distribute service functions (such as storage or indexing) evenly across all participating nodes [3, 4, 5, 6, 7, 9, 10]. At a high level, many of these efforts use a distributed hash table, with regions of the table mapped to each participant. The challenge then is to locate the remote host responsible for a target region of the hash space in a scalable manner, while: i) adapting to changes in group membership, ii) achieving locality with the underlying IP network, and iii) caching content and/or request routing state so as to minimize the average number of hops to satisfy a request.

These recent efforts constitute important basic research in massively decentralized systems, and they have produced elegant solutions to challenging and interesting problems. However, these approaches seek massive decentralization as an end in itself, rather than as a means to the end of devising practical service architectures that are scalable, available, and survivable. From a practical standpoint, they address the wrong set of issues in peer-to-peer computing.

We suggest that distributing service functions across a carefully selected subset of nodes will yield better performance, availability, and scalability than massively decentralized approaches. The true opportunity afforded by peer-to-peer systems is not the ability to put everything everywhere. Rather, it

is the opportunity to put anything anywhere. Why distribute an index across one million nodes when a well-chosen subset of one thousand can provide the resources to meet target levels of service performance and availability?

Given n participants in a peer-to-peer system, we argue that the best approach is not to evenly spread functionality across all n nodes, but rather to select a minimal subset of m nodes to host the service functions. This choice should reflect service load, node resources, predicted stability, and network characteristics, as well as overall system performance and availability targets. While it may turn out that $m = n$ in some cases, we believe that $m \ll n$ in most cases. Membership in the service subset and the mapping of service functions must adapt automatically to changes in load, participant set, node status, and network conditions, all of which may be highly dynamic. Thus we refer to this approach for peer-to-peer systems as *self-organizing subsets*.

One goal of our work is to determine the appropriate subset, m, of replicas required to deliver target levels of application performance and availability. The ratio of subset size m to the total number of nodes n can approximately be characterized by:

$$\frac{m}{n} = \frac{u}{dE}$$

where u is the sum of all service resources consumed by the n total hosts, d is the sum of all service resources provided by the m hosts in the subset, and E is the efficiency — the fraction of resources in use when the system as a whole begins to become overloaded. Efficiency is a function of the system's ability to properly assign functionality to an appropriate set of sites and of load balancing, better load balancing results in values of E approaching one [6]. In a few systems, such as SETI@home, all available service resources will be used; thus, u approaches d, and it makes sense for m to equal n. However, in most systems one node can provide far more resources than it is likely to consume; thus, $u \ll d$, and given reasonable efficiency, $m \ll n$.

Self-organizing subsets address key problems of scale and adaptation that are inherent in the massively decentralized approach. For example, routing state and hop count for request routing in existing peer-to-peer systems typically grow with $O(\lg n)$ at best. While this may qualify as "scalable," it still imposes significant overhead even for systems of modest size. Using only a subset of the available hosts reduces this overhead. More importantly, massively decentralized structures may be limited to services with little or no mutable state (e.g., immutable file sharing), since coordination of updates quickly becomes intractable. Our recent study of availability [11] shows that services with mutable state may suffer from too much replication: adding replicas may compromise availability rather than improve it. Random distribution of functionality among replicas means that more replicas are required to deliver the same level of performance and availability. Thus, there is an interesting tension between the locality and availability improvements on the one hand and the degradation on the other that comes from replication of mutable data in peer to peer systems. A primary goal of our work is to show the *resource waste* that comes from random distribution, i.e., the inflation in the number of randomly placed replicas required to deliver the same levels of performance and availability as a smaller number of "well-placed" replicas. Finally, massively decentralized approaches are not sufficiently sensitive to the rapid status changes of a large subset of the client population. We propose to restrict service functions to subsets of nodes with significantly better connectivity and availability than the median, leading to improved stability of group membership.

Our approach adapts Marxist ideology—"from each according to his abilities, to each according to his needs"—to peer-to-peer systems. The first challenge is to gather information about the *abilities* of the participants, e.g., network connectivity, available storage and CPU power, and the *needs* of the application, e.g., demand levels, distribution of content popularity, and network location. The second challenge is to apply this information to select a subset of the participants to host service functions, and a network overlay topology allowing them to coordinate. Status monitoring and reconfiguration must occur automatically and in a decentralized fashion.

Thus, we are guided by the following design philosophies in building scalable, highly available peer-to-peer systems:

- It is important to dynamically select subsets of participants to host service functions in a decentralized manner. In the wide area, it is not necessary to make optimal choices, rather, it is sufficient to make good choices in a majority of cases and to avoid poor decisions. For example, current research efforts place functionality randomly and use replication to probabilistically deliver acceptable performance for individual requests. Our approach is to deterministically place functionality and to replicate it as necessary based on network and application characteristics. This requires methods to evaluate expected performance and availability of candidate configurations to determine if they meet the targets.

- A key challenge to coordinating peer-to-peer systems is collecting metadata about system characteristics. Configuring a peer-to-peer system requires tracking the available storage, bandwidth, memory, stability (in terms of uptime and availability), computational power, and network location of a large number of hosts. At first glance, maintaining global state about potentially billions of hosts is intractable. However, good (rather than optimal) choices require only approximate information: aggressive use of hierarchy and aggregation can limit the amount of state that any node must maintain. Once a subset is selected, the system must track only a small set of candidate "replacement" nodes to address failures or evolving system characteristics. Similarly, clients maintain enough system metadata to choose the replica likely to deliver the best quality of service (where QoS is an application-specific measure). Once again, the key is to make appropriate request routing decisions almost all of the time, without global state.

- Service availability is at least as important as average-case performance. Thus, we are designing and building algorithms to replicate data and code in response to changing client access patterns and desired levels of availability. Some important questions include determining the level of replication and placement of replicas needed to achieve a given minimum level of availability as a function of workload and failure characteristics. Once again, a key idea is that a few well-placed replicas will deliver higher availability than a larger number of randomly placed replicas because of the control overhead incurred by coordination among replicas.

The rest of this position paper elaborates on some of the challenges we see in fully distributing service functionality among a large number of nodes and describes Opus, a framework we are using to explore the structure and organization of peer-to-peer systems.

2 Challenges to Massive Decentralization

In this section, we further elaborate on our view of why fully distributing functionality among a large number of Internet nodes is the wrong way to build peer-to-peer systems. While a number of techniques have been proposed to minimize per-node control traffic and state requirements, it still remains true that in a fully decentralized system with millions of nodes, the characteristics of all million nodes have to be maintained somewhere in the system. To pick one example, each node in a million node Pastry system must track the characteristics of 75 (given the suggested representative tuning parameters) individual nodes [6], potentially randomly spread across the Internet. We believe that by choosing an appropriate subset of global hosts (m of n) to provide application functionality and by leveraging hierarchy, the vast majority of nodes will maintain state about a constant (small) number of nearby *agents*. Sets of agents are aggregated to form hierarchies and in turn maintain state about a subset of the m nodes and perhaps approximate information on the full set of m nodes[1]. Thus, to route a request to an appropriate server, nodes forward requests to their agent, which in turn determines the appropriate replica (member of m) to send the request to. In summary, massive

[1]For simplicity, this discussion assumes a two-level hierarchy, which should be sufficient for most applications. Our approach extends directly to hierarchies of arbitrary depth.

decentralization requires each system node to maintain state about $log(n)$ other global nodes. If successful in carefully placing functionality at strategic network points, the vast majority of nodes maintain state about a constant and small number of peers (one or more agents), and each agent maintains state about a subset of the m nodes providing application functionality.

Another issue with massive decentralization is dealing with dynamic group membership. Assuming a heavy-tailed distribution for both host uptime and session length, significant network overhead may be required to address host entry or departure of the large group of hosts that exhibit limited or intermittent connectivity (some evidence for this is presented in [8]). This is especially problematic if $O(log(n))$ other hosts must be contacted to properly insert or remove a host. In our approach, we advocate focusing on the subset of hosts (again, m of n) that exhibit strong uptime and good connectivity—the head of the heavy-tailed distribution rather than the tail. In this way, we are able to focus our attention on hosts that are likely to remain a part of the system, rather than being in a constant state of instability where connectivity is *always* changing in some region of the network. Of course, nodes will be constantly entering and leaving in our proposed system as well. However, entering nodes must contact only a small constant number of nodes upon joining (their agents) and can often leave silently (especially if they never achieved the level of uptime or performance to be considered for future promotion to an agent or one of the m nodes that deliver application-level functionality).

Finally, a key approach to massive decentralization is randomly distributing functionality among a large set of nodes. The problem then becomes routing requests to appropriate hosts in a small number of steps (e.g., $O(\log n)$ hops). Because these systems effectively build random application-layer overlays, it can be difficult to match the topology of the underlying IP network in routing requests. Thus, replication and aggressive caching [3, 5, 7] must be leveraged to achieve acceptable performance relative to routing in the underlying IP network. While this approach results in small inflation in "network stress" relative to IP, application-layer store and forward delays can significantly impact end-to-end latency (even when only $O(\lg n)$ such hops are taken). While such inflation of latency is perhaps not noticeable when performing a lookup to download a multi-megabyte file, it can become the bottleneck for a more general class of applications. With our approach, requests can typically be routed in a small and constant number of steps (depending on the chosen depth of the hierarchy). Further, because we have explicit control over connectivity, hierarchy, and placement of functionality, we can ensure that requests from end hosts are routed to a nearby agent, which is in turn routed to an active replica. The random distribution of functionality in massively decentralized systems makes it more difficult to impose any meaningful hierarchy.

3 An Overlay Peer Utility Service

We are pursuing our agenda of dynamically placing functionality at appropriate points in the network in the context of Opus [1], an overlay peer utility service. While our research is specific to this service, we believe our techniques and approach are general to a broad range of peer-to-peer services. As a general compute utility, we envision Opus hosting a large set of nodes across the Internet and dynamically allocating them among competing applications based on changing system characteristics. Individual applications specify their performance and availability requirements to Opus. Based on this information, we map applications to individual nodes across the wide area. The initial mapping of applications to available resources is only a starting point. Based on observed access patterns to individual applications, Opus dynamically reallocates global resources to match application requirements. For example, if many accesses are observed for an application in a given network region, Opus may reallocate additional resources close to that location.

One key aspect of our work is the use of Service Level Agreements (*SLAs*) to specify the amount each application is willing to "pay" for a given level of performance. In general, these SLAs provide a continuous space over which per-service allocation decisions can be made, enabling prioritization among competing applications for a given system configuration. Based on an estimate of the marginal

utility of resources across a set of applications at current levels of global demand, Opus makes allocation and deallocation decisions based on the expected relative benefit of a set of target configurations [2].

Many individual components of Opus require information on dynamically changing system characteristics. Opus employs a global *service overlay* to interconnect all available service nodes and to maintain soft state about the current mapping of utility nodes to hosted applications (group membership). The service overlay is key to many individual system components, such as routing requests from individual clients to appropriate replicas, and performing resource allocation among competing applications. Individual services running on Opus employ *per-application overlays* to disseminate their own service data and metadata among individual replica sites.

Clearly, a primary concern is ensuring the scalability and reliability of the service overlay. In an overlay with n nodes, maintaining global knowledge requires $O(n^2)$ network probing overhead and $O(n^2)$ global storage requirements. Such overhead quickly becomes intractable beyond a few dozen nodes. Peer-to-peer systems can reduce this overhead to approximately $O(\lg n)$ but are unable to provide any information about global system state, even if approximate. Opus addresses scalability issues through the aggressive use of hierarchy, aggregation, and approximation in creating and maintaining scalable overlay structures. Opus then determines the proper level of hierarchy and aggregation (along with the corresponding degradation of resolution of global system state) necessary to achieve the target network overhead.

4 Conclusions

This paper argues that a principal challenge in peer-to-peer systems is determining where to place functionality in response to changing system characteristics and as a function of application-specified targets for availability, survivability, and performance. Many current peer-to-peer research efforts focus on fully distributing service functionality across all (potentially billions) of participating hosts. The resulting research fundamentally contributes to our understanding of structuring distributed services for availability, survivability, and scalability. A key challenge in peer-to-peer research is to dynamically determine the proper subset m, of n participating nodes, required to deliver *target* levels of availability, survivability, and performance, where typically $m \ll n$. For many application classes, especially those involving mutable data, increasing m will not necessarily improve performance and availability. We are using Opus, an overlay peer utility service that dynamically allocates resources among competing applications, as a testbed for experimenting with the ideas presented in this paper.

References

[1] Rebecca Braynard, Dejan Kostić, Adolfo Rodriguez, Jeffrey Chase, and Amin Vahdat. Opus: an Overlay Peer Utility Service. In *Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH)*, June 2002.

[2] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, October 2001.

[3] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area Cooperative Storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[4] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*, November 2000.

[5] Sylvia Ratnasamy, Paul Francis Mark Handley, Richard Karp, and Scott Shenker. A Content

Addressable Network. In *Proceedings of SIGCOMM 2001*, August 2001.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Middleware'2001*, November 2001.

[7] Antony Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[8] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, January 2002.

[9] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer to Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 SIGCOMM*, August 2001.

[10] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant, Web Publishing System. In *Proc. 9th USENIX Security Symposium*, pages 59–72, August 2000.

[11] Haifeng Yu and Amin Vahdat. The Costs and Limits of Availability for Replicated Services. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.