BON: Bootstrap-enabled Overlay Network for Distributed Application Management

April 29, 2008

Abstract

Distributed systems need to be deployed before they can be use, and many systems and tools were proposed to help deploy and manage large-scale distributed applications. However, an interesting question is: what if the distributed systems can deploy and manage, or bootstrap, themselves without help from external tools or systems.

We present BON (Bootstrap-enable Overlay Network) to explore how to build the bootstrap capability within distributed systems. BON is a novel system that deploys and manages itself without help from external systems. It operates reliably and securely in large-scale distributed environments involving inevitable machine and network failures. The evaluation results show that the design achieves good performance and scalability.

1 Introduction

Managing distributed applications on distributed computation infrastructures [19, 9] is a time-consuming and error-prone process. It involves deploying, monitoring and control of the applications on a set of distributed resources. After the initial deployment of the software, the applications must be monitored continually for detecting and recovering from inevitable machine and network failures. For availability and reliability considerations, the applications must be carefully monitored and controlled to provide continued operation and sustained performance under dynamic fluctuations in distributed environments.

A number of tools has been proposed and developed to address various aspects of management process in distributed environments: resource monitoring and discovering, content distribution, application monitoring and control, etc. They can be categorized into two approaches: centralized approach and peer-to-peer approach. While the centralized approach is easier to design, develop and use, they cannot scale well for large systems. The peer-to-peer approach scales better than the centralized one. And it can adapt to fluctuations in distributed environment dynamically, if carefully designed.

It is interesting to notice that management tools for distributed applications are themselves distributed applications in essential, and they need to be managed too. For example, to use BitTorrent to deploy contents, the clients of BitTorrent has to be deployed to each target node first. After starting all the BitTorrent clients, the deployment progress of software packages on each BitTorrent client has to be monitored. And the BitTorrent clients have to be stopped after the deployment is completed. It will be more complex when some clients cannot be monitored or controlled properly because of machine and network failures. The situation is similar for the centralized management tools. The central controller and every clients have to be deployed and managed. Some centralized tools make use of services provided by operating systems as clients—e.g. sshd of *nix systems—and avoid the deployment and management of these services but the management functions are limited. It is a choice to use management services that are already deployed and maintained by others, which kicks the ball back to the operators of the services.

This is a bootstrap problem of distributed application management. To state the problem formally: to use management tool A, it may need to be managed by another management tool B, recursively, B may need to be managed by C, and so on. In the end, a tool Z is needed to bootstrap the whole process. If A has bootstrap capability, we have $A = Z$, and it is unnecessary to iterate the whole alphabet (or other character and number set) to name so many tools.

A management tool can be considered as bootstrap if it is quite simple and easy to be managed by users and doesn't require other management tools. We call it the bootstrap for management tools. Management scripts based on ssh/scp on *nix platforms can be

considered as bootstrap tools. However, they have limited scalability and functions.

In this paper, we propose the design of Bootstrap-enabled Overlay Network (BON). BON is a distributed management tool with unstructured topology. The bootstrap capability is enabled by building self-management functionality into the overlay with certain security guarantees. BON can deploy, update, monitor and control itself in distributed approach with little user interference. It achieves good reliability and performance by using gossip-style communication extensively within the system.

We will discuss bootstrap problem and present an overview on BON in section 2. The design and implementation of BON is presented in section 3 and evaluate the performance of BON in section 4. The related work is given in section 5 and we conclude in section 6.

2 Bootstrap and BON

Tools for managing distributed systems can be categorized into two approaches: centralized and distributed. The centralized tools [2] are preferred and commonly used by developers and researchers because of their simplicity. Once copied or installed to a machine box, they are ready to use. The distributed tools need much more effort before they can really be used.

A management tool can be considered as bootstrap if it is quite simple and easy to be managed by users and doesn't require other management tools. An example is vxargs. It is a script that makes use of ssh/scp to deploy contents and running shell commands or programs remotely. It needs little management work: copying the script to the control node is all of the work.

There is a trade-off when using management tools currently available. The ones in centralized approach needs little management work but has limited scalability and features. The ones in distributed approach provide more features and are more scalable, but users have to do extra management work on these tools.

We have designed and implemented a management tool BON which combines the above two virtues. It uses the distributed approach, which means it is scalable and can adapt to network environment changes dynamically. And it needs little management, because it is designed with built-in self-management capability. A BON system can deploy, monitor and update itself automatically in a distributed approach.

Users of BON need to deploy and start only one instance of the BON peer and give it a list of target nodes on which BON should be deployed. The running BON peers will actively deploy new peers to other target nodes automatically. The self-deployment process uses the epidemic approach and is very fast. The self-update capability of BON allows the deployed BON peers to update themselves to a new version even when the whole BON system is running.

A set of RPC calls is implemented by BON peers and users can use them to communicate and control each BON peer directly. BON can also be used to manage other applications. The applications can be deployed, monitored and updated automatically by BON, and a similar set of RPC calls are ready implemented to control individual application peers on demand.

The management power of BON can be extended in two ways: leveraging self-update capability or through a managed application. For example, users may want to improve the monitoring functions of BON. One way is to rewrite the monitoring part of BON, adding more features and improving its performance. After that, BON can be updated to a new version with more powerful monitoring functions. Or the users can deploy an existing monitor application satisfying their requirements with BON.

BON can be also used as an add-on to distributed applications, which turns any distributed application to a bootstrap-enabled one. If we consider BON and the managed application as a whole, any distributed application becomes bootstrap-enabled. The application gains the self-management capability through BON, or to say, it can bootstrap itself.

3 BON Design and Implementation

BON is designed as a peer-to-peer system with one peer running on each target node. Each peer has a dynamic partial view of target nodes that BON should be deployed. The partial view of a BON peer is of size $O(\log N)$, and N is the total number of the target nodes. The running BON peers form an unstructured overlay which ensures that the whole system is well connected with high probability and can be recovered from network partitions easily.

The architecture of a BON peer is shown in Figure 1. The peer-monitor component is responsible for deploying, monitoring and updating BON peers on other nodes. It periodically detects peers on other nodes and makes sure there are BON peers deployed and running on the detected nodes. It also exchanges

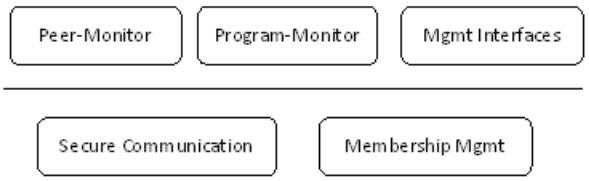

Figure 1: Architecture of a BON peer

and compares the version number of itself with other peers and schedules a self-update when a new version of BON peer is available. The overall effect implied by peer-monitor component is keeping BON peers running on all the managed nodes with the newest version as long as possible. The program-monitor component monitors the application it manages and updates the application to the newest version. A set of RPC calls is provided to control the BON peers and the managed applications.

When two peers need to send messages with each other, the secure communication components help to authenticate each other peer and provide a secure communication channel. It also help to authenticate with a remote node when peer-monitor component tries to deploy a BON peer remotely. The membership management component maintains a partial view of all the target nodes that BON should be deployed. The nodes in the partial view is a random subset of all the target nodes.

Epidemic algorithm is used extensively in BON system. While epidemic algorithm is efficient at disseminating data and maintaining replicated states consistently, it is simple, elegant and easy to implement. It is quite a useful technique in designing a reliable, scalable and efficient distributed system.

Some challenges in designing the BON system is presented below.

3.1 Peer-Monitor

BON relies upon peer-monitor component to deploy, monitor and update BON peers on the managed nodes. A BON peer periodically contacts peers on other nodes which are randomly selected from the partial view. If it fails to contact the peer on a node, it will try to deploy an instance of BON peer on the node and start it. The deployment is skipped if a copy of BON peer has already been deployed and the stopped peer is started directly.

BON peers have associated version numbers with them, which are stored persistently along with each peer. The peers actively communicate with each other and compare the version numbers they have. When a difference is found, an self-update action is scheduled and initialized by the peer with lower version number.

The function of peer-monitor component can be described in two scenarios.

Self-Deployment Process

At the very beginning, there is not any BON peer deployed yet. At least one peer must be deployed to a node and started manually. The first peer then detects other nodes in its partial view and deploy new peers if necessary. The new peers repeat the similar detect-and-deploy action. While there are more and more BON peers deployed and running, this self-deployment process speeds up exponentially. It is proved that with probability 1 all the reachable nodes can be deployed eventually[8].

It is not easy to define when self-deployment is finished globally. Ideally, it is finished when all the target nodes are reached and deployed with BON peers. However, considering some nodes may be unreachable or shutdown temporarily, the 'finished-globally' situation is rare. Even after a nodes is deployed with BON peer, it may be failed or unreachable again after some time. The peers then have to continue the detect-and-deploy action as long as they are live and leave the 'finish' definition to user. The user can query how many or which peers are deployed, and determine whether it is appropriate to consider the situation as a finish.

If node H has already been deployed with a peer but the peer is not running, the peer will be simply re-started without extra deployment. Starting multiple instances of BON peer on the same node is idempotent to start only one instance.

There are race conditions cases among BON peers in the self-deployment process. There're possibilities that two or more peers find the same node (noted as H) on which no BON peer has been deployed. And they may take the deployment action on it simultaneously. This race condition problem is solved without direct coordination among peers. When multiple peers try to deploy on H simultaneously, the peers copy the software package of BON peer to different directories on H independently. The directories names they choose are based on the hostnames of nodes on which they are running, and cannot be conflict for all the time. After that, a local post-deployment script is started which finishes the remaining deployment and start the deployed BON peer. It is designed that only one instance of the script can be running at a time, so only one peer can finish the deployment

process finally. In solving the race condition, some overheads are introduced because multiple BON peer packages may be copied, which wastes network bandwidth and storage resources. Through the evaluation, it is shown that the overhead is low on most nodes. And the size of BON peer package is small (122KB), so the overhead can be considered as insignificant.

Self-Update Process

Self-update happens when two peers exchange version numbers about themselves and find a difference. An self-update action is scheduled and initialized by the peer with lower version number. To update the whole BON system to a new version, user only has to update one peer to the new version and all the other BON peers will emerge to the newest versions eventually.

It is important to ensure that communication interfaces for different versions of BON peers are consistent. If the condition is ensured, the whole BON system can be updated gracefully while most of peers are running and the system functions well. Otherwise, the new BON system can only be deployed after the old one is stopped.

3.2 Application Management

One or more distributed applications can be managed by BON. Each BON peer manages the application peers that run in the same node. Each of the managed application peers has an associated version number. The version numbers of application peers are maintained by managing BON peers. The similar epidemic algorithm is used for keeping application peers up-to-date. A trick used in application management is, when an application peer has not been deployed on a node yet, its version number is set to 0.0, which is smaller than any real version numbers. In this way, application deployment problem is turned to the application update problem.

BON peers knows the existence of a new application or a new version of application through periodically communicating with each other. A new application entry is created if it is known for the first time. The peers download the application with specified version from other peers. There is a set of implemented RPC calls for control and monitoring of managed applications.

3.3 Security Guarantees

It is necessary for different BON peers to authenticate each others and keep the communication confidential among them. In addition, when a BON peer deploys another BON peer on a remote node, it need to authenticate itself to the remote node to deploy contents and execute commands. The authentication credential is owned by user and should be protected confidentially. From another point of view, the BON peers relies upon the authentication credentials provided by user to deploy and manage each others automatically in a distributed approach.

BON system is designed for distributed infrastructures that are semi-open—like Planet-Lab or grid, or for proprietary ones—such as large machine rooms or data centers. It assumes that: 1) the network between nodes in a distributed infrastructure is not trusted, 2) the users that can access the infrastructures are not adversarial, 3) the applications running in the infrastructures are not adversarial. Based on assumptions above, BON designed an ad-hoc mechanism which guarantees that 1) communications among BON peers are authenticated and confidential, 2) authentication credentials needed to authenticate with remote nodes are not leaked under any circumstances. The mechanism incorporated in current design supports challenge based authentications like public-key authentication scheme, which is extensively used.

To achieve that goal, BON includes an separate authentication agent acting as the authentication bootstrap for BON. Whenever a BON peer needs to login into a node, it will indirect the authentication challenges to the agent and the agent will solve the challenges and send the result back. In this way, the authentication credentials are kept secret for all the times. A symmetric encryption key K_E is shared among all the BON peers and the authentication agent. It is used to authenticate peers and agent within the same BON system, and establish confidential communication channels among them. The shared key K_E is included within the software package of the BON peer. When a BON peer is deployed, K_E is deployed also. Note that K_E is never leaked during deployment. K_E is safe to be stored within local nodes based on assumption 2) and 3), and further local security policies can be leveraged to protect K_E.

It is not necessary to keep authentication agent running for all the time. When the self-deployment process is finished, the agent can be closed, and reopen timely for maintennace purpose. It is also a security concern to offline authentication agent to protect user authentication credentials.

The authenticate agent can be also replicated if required. When a BON system is deployed at large enough scales, there can be multiple instances of authenticate agent for performance considerations.

3.4 Stopping BON System

The BON system has a global state indicating it is active or not. When BON is active, the peer-monitoring component is enabled. Every BON peers actively monitor each others and keep them running as possible as they can. While BON is not active, the peer-monitoring function is disable and BON acts just like an usual management tools. The active state is an essential part of BON system design. Without it, BON is just another usual management tool for distributed applications. If the state is designed as active all the time, it is a hard problem to stop an active BON system. If we try to stop the BON peers one-by-one, we will noticed that the stopped peers will soon be restarted by other peers. And the only way to stop an active BON system is to stop all the peers near simultaneously, which is quite hard. The active state also has an associated version number and each BON peer replicates and stores the state with version locally. The epidemic algorithm is used to maintain the state consistently and efficiently among all peers.

3.5 Handling Failures

Two kinds of failures may happen and must be handled by BON, namely, nodes failures and network partitions. When a node fails, the BON peer running on it is stopped also. As soon as the node recovers, it will be noticed by some BON nodes quickly and the deployed BON peer will be restarted.

Network partitions can also be handled by BON easily. The topology formed by BON peers is a random graph which ensures that all the nodes are connected with high probability. When network partition occurs, BON peers are splitted into several subsystems, each of which is connected internally. Implied by epidemic algorithm, each partitioned part will eventually emerges to the consistent state regarding to the version of BON peers, the version of managed applications, and the vertion of active states. When two partitions rejoin, the peers in different part will contact and communicate with each other, leading to the emergence of the whole system.

3.6 Membership maintenance

The membership maintenance of BON system is different from other distributed systems. While other distributed systems maintains a list of peers that are currently running, BON maintains a list nodes on which there should be peers running. And an important requirement of BON is, even when there is only one BON peer running, it should be able to know all the target nodes that BON need to be deployed. The implication is every running BON peer should be able to know the full list of target nodes.

In current design, a simple scheme is used that each BON peer will store a replica of the full list of target management nodes persistently in compressed format. The partial view of a BON peer is chosen randomly from the whole list. The list also has an associated version number and is maintained in epidemic approach among peers. To further reduce the communication overhead, only the differences between two version are propagated during node list update process. This scheme is enough for handling node list containing even tens of thousands of nodes because only the node names or IPs are stored in the list. An estimation shows that, storing 15000+ nodes names in zip format takes about 100K bytes, which is an acceptable result.

3.7 Support for Indirect Communication

The one-hop source routing[11] algorithm is implemented by BON. When a peer failed to communicate directly to another peer, it randomly selected k ($k = 4$ in current design) other peers for sending the messages indirectly. This feature is useful for managing large number of nodes. We implement this mechanism because we notice a lot of connection problems caused by name resolution error. It happens between our laptop and PlanetLab nodes, and also among some PlanetLab nodes. In the experiment, we noticed a node cannot connect to other two nodes because of name resolution error for more than one day. But through one-hop source routing, they can communicate without problem. This feature is useful for users to control or monitor nodes at large scale. As the user control or monitor other nodes through a peer as proxy, one-hop source routing can increase the number of connected peers.

3.8 Implementation

We have implemented a BON system for running on PlanetLab platform. The source is written in Python, and the final software package size of a BON peer is about 122KB, with 1000+ lines of code.

The communication facilities provided by underlying infrastures are critical. In current implementation, BON is designed to be used on PlanetLab platform and the BON peers rely on ssh a lot to ensure security. It is quite hard to ensure security on systems which only provide telnet-like facilities. Although ssh help us a lot, there are some inconve-

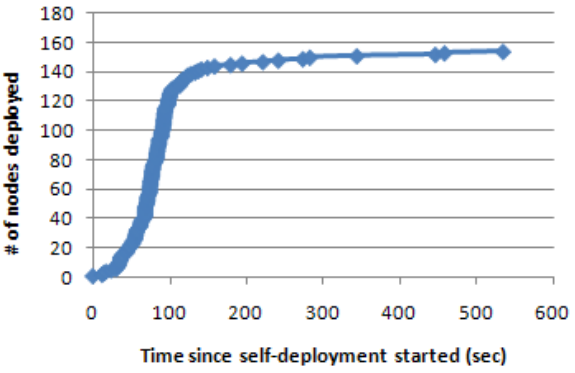

Figure 2: Progress of self-deployment process on 159 nodes

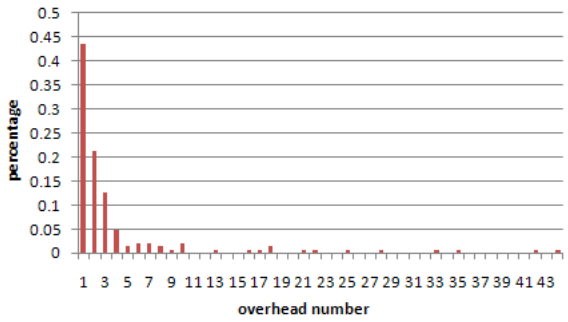

Figure 3: Overhead introduced by race condition during self-deployment process

niences. It does not provide API interface to control ssh and scp. We need to use expect-style commands to control ssh which involves a lot of expect-sendline combinations. The differences between ssh version make this method complicated. It would be nice if ssh provides a standard library.

The version numbers involved in the BON system is managed by user centrally. A client script is written to help user to communicate with BON peers and maintain different version numbers for different part of the system. When a user communicates with BON peers, he uses the same symmetric encryption key K_E shared among BON peers to authenticate himself to BON peers.

4 Evaluation

We conducted experiments on PlanetLab platform to evaluate the performance of BON. Planet-Lab is a global research network consisting of 800+ nodes at 400+ sites in the world. Specifically, we provide experimental results showing that BON is efficient at deploying itself and achieves good scalability, the overhead caused by race condition is small and the active state transition is fast.

4.1 Self-deployment Evaluation

Self-deployment process is the core of the BON system and we evaluate it extensively and shows the results in this section.

First, we evaluate the performance of self-deployment process on 159 Planet-Lab nodes. An instance of BON peer is started and the list of 159 nodes are given. A BON peer detects other 5 nodes for every 10 seconds. We finally collected timing data from 154 nodes out of 159 nodes. There are 5 nodes cannot be connected at data-collecting stage and their data are omitted in the result. This kind of failure is common for distributed systems especially when the scale is large. Figure 2 summarizes the progress of self-deployment process. We can see that 90% (143) of nodes are deployed successfully with a BON peer within 149 seconds. The median deployment time is 93 seconds while the largest one is 533 seconds.

During the self-deployment process, there are cases that multiple BON peers try to deploy an instance on the same node and this can cause extra overhead. The overhead wastes network bandwidth and the storage at the deployed node. We evaluate the overhead by count the simultaneous deployment happened at each node. The result is shown in Figure 3. We can see that for 43.71% of nodes there is exactly one deployment on each of them. Obviously, most of nodes have a deployment overhead within 10. These nodes count for 92.04% of all nodes and the average overhead of these nodes is 2.31. The average overhead for all nodes is 4.30 which is an acceptable number.

To evaluate the scalability of self-deployment process, we deploy a new instance of BON system at a different scale (24 nodes) and compare the performance of self-deployment process. Table 1 summarizes the statistics results. We can see from the table that the scalability is quite well. While scale difference between two systems is about 6.6 (159/24), the 90-percentile deployment time is only 1.75 (149/85) times of difference, while the median value is 1.41 (82/58) times of difference.

4.2 Active State Transition Performance

State transition between active state and inactive state is an important part of BON system and we

	24 nodes	159 nodes
median	58	82
90% percentile	85	149
Final	103	533

Table 1: Comparison of self-deployment process at different scales in seconds

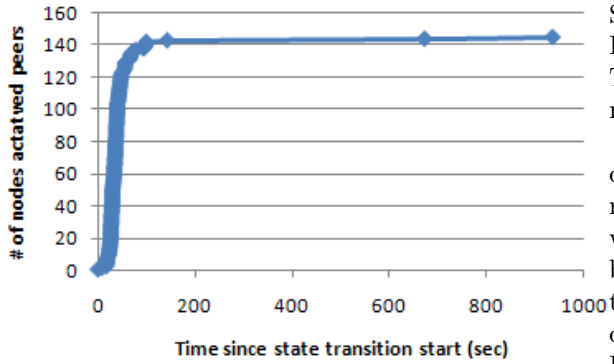

Figure 4: Progress of state transition from inactive to active

evaluate its performance here. We turn the state of BON system deployed on 159 Planet-Lab nodes from inactive to active. The progress of state transition time is shown in Figure 4 and some statistics is summarized in Table 2 for convenience. It can be concluded that the state-transition is effective and efficient. For 90% percentile of peers, the states are changed after 143 seconds and the median state transition time is 37 seconds.

5 Related Works

A lot of works have been done to help managing distributed systems. PSSH[1] and vxargs[2] are two scripts that can be used to manage a distributed system from a central host using ssh/scp. Plush[3] is another tool in centralized approach. It has extensions that support using existing tools on PlanetLab. MON[15] utilizes on-demand overlays for distributed system management.

There're some works addressing part of management works. Ganglia[16], Astrolab[20], Sophia[21], SWORD[17] SDIMS[22] and PIER[12] are some information planes that can be used to monitor systems and applications status. Ganglia is designed for clusters but was still able to run when PlanetLab is not so large. Astrolab is an information management system using gossip protocol. It supports on-the-fly attribute aggregation. Sophia uses declarative logic language to express and evaluate distributed queries. Sword is focused on resource discovery using DHTs. PIER has a broad vision of 'Querying the Internet'. The issue of supporting distributed trigger mechanism is also discussed in [13]

median	37
90% percentile	143

Table 2: Statistics for state transition experiment in seconds

An important problem in management is how to disseminate contents to all others efficiently. BitTorrent is a great tool for this job. A lot of research works[6, 14, 7, 5] proposed use overlay network for broadcasting contents. Coral[10] and CoDeeN[18] are two publically accessible content distribution network deployed on Planet-Lab platform. Researchers using Planet-Lab can use them to deploy applications and data with the two systems. Shark[4] is a distributed file system designed for large-scale, wide-area deployment. It introduces a novel cooperative-caching mechanism, in which mutually-distrustful clients can exploit each others' file caches to reduce load on an origin file server.

6 Conclusion

In this paper, we present the design and implementation of BON, a novel system designed as bootstrap for distributed application management. Different from existing works, BON is a system with built-in self-management capability. It can deploy, monitor and update itself automatically in a distributed approach. Through evaluation, it is shown that BON scales well to large number of nodes effectively and efficiently.

References

[1] pssh. `http://www.theether.org/pssh/`.

[2] vxargs. `http://dharma.cis.upenn.edu/planetlab/vxargs/`.

[3] J. Albrecht, C. Tuttle, A. C. Snoeren, and A. Vahdat. Planetlab application management using plush. *ACM Operating Systems Review*, 40(1):33–40, 2006.

[4] S. Annapureddy, M. J. Freedman, and D. Mazières. Shark: Scaling file servers via cooperative caching. In *NSDI*, 2005.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8):1489–1499, 2002.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP*, pages 298–313, 2003.

[7] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *SIGMETRICS*, pages 1–12, 2000.

[8] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[9] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[10] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *NSDI*, pages 239–252, 2004.

[11] P. K. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *OSDI*, pages 183–198, 2004.

[12] R. Huebsch, B. Chun, J. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *CIDR*, pages 28–43, 2005.

[13] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings of HotNets-III*, 2004.

[14] D. Kostic, A. Rodriguez, J. R. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *SOSP*, pages 282–297, 2003.

[15] J. Liang, S. Ko, I. Gupta, and K. Nahrstedt. MON: On-demand overlays for distributed system management. In *Workshop on Real, Large Distributed Systems*, 2005.

[16] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[17] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on planetlab with SWORD. In *Workshop on Real, Large Distributed Systems*, 2004.

[18] K. Park and V. S. Pai. Scale and performance in the coblitz large-file distribution service. In *NSDI*, 2006.

[19] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[20] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[21] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: an information plane for networked systems. *Computer Communication Review*, 34(1):15–20, 2004.

[22] P. Yalagandula and M. Dahlin. A scalable distributed information management system. *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390, 2004.