Resource-Efficient Software Delivery Using Volunteer Assistance

Purvi Shah[*]
Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX

Jeffrey Morgan
John Schettino
Chandrasekar Venkatraman
Hewlett-Packard Labs
Palo Alto, CA

Miranda Mowbray
Hewlett-Packard Labs
Bristol, UK

ABSTRACT
We propose to apply Peer-to-Peer (P2P) technology to resolve the scalability problems observed in current techniques used to provide management and maintenance services in enterprise networks. We aim to create a content delivery infrastructure that can be used by managed services organizations, on the basis of donated servers, to help download and maintain software packages.

Keywords
Content delivery networks, trace analysis, synchronization, peer-to-peer, data integrity, load balancing

1 INTRODUCTION

While existing content delivery networks (CDNs) such as *Akamai* [1] can provide very large service capacity, the cost of maintaining such CDNs is very high. Previous work on volunteer computing proves that distributed computing projects such as *SETI@Home* [2] using donated machines can work as well or even better than the largest supercomputers. While CDNs such as *CORAL* [4] offer an attractive solution, CDNs including donated resources need to address additional security and load balancing issues.

2 TRACE ANALYSIS

We began by analyzing 10 days worth of logs associated with a software package delivery system supporting Linux installations and updates in an enterprise. We made the following important observations regarding the package downloads,

- Our repository served system software and updates for approximately 10 different Linux distributions and consisted of roughly 2.2 million files, 91% of which are smaller than one megabyte.

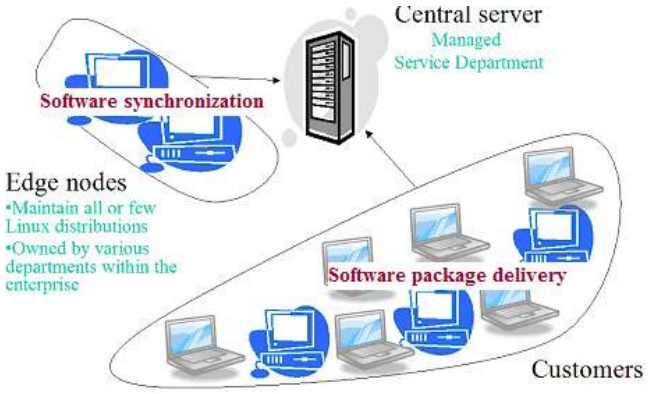

Fig. 1. Workload on the Existing System.

- The total size of the repository is approximately 2.86 TB. Image (*.iso* files) downloads comprises 71% of the total upload compared to the other packages. Nevertheless the percentage of downloads for update packages (for instance *.rpm* files) is large.
- We observed interesting access patterns such as flash crowds in the event of new package release.
- We found out that 17% of large files were identical and differ from other files only in name. This is because a large number of software packages are the same for different Linux distributions.
- We also observed there was considerable similarity between different versions of the same package.

Based on our observations we believe a software package delivery system needs to be optimized for efficient delivery of small and large sized files. It should be capable of managing flash crowd scenarios and be able to exploit the similarity between files.

We also observed that various departments within the enterprise manage around forty edge nodes that maintain complete or partial mirrors of the software repository for serving updates to a small set of machines. By studying data traces collected over a period of 35 days by an edge node we observed that it spent on average 1.81 hours daily (max.: 20.53, min.: 0.54) synchronizing its repository with the server.

In summary Fig. 1 gives an idea about the current system design and its workload.

[*] Student author (purvi@cs.uh.edu). Most of this work was performed while this author was visiting HP Labs in Palo Alto.

CoNEXT'07, Dec 10–13, 2007, New York City, New York, USA.

3 SOFTWARE SYNCHRONIZATION BY THE EDGE NODES

With insights from our trace analysis we first developed a tool that uses file-swarming techniques based on the *BitTorrent* [3] protocol to efficiently synchronize the repository on the edge nodes. It integrates P2P technology into the existing synchronization system and is also able to exploit the presence of identical files.

This integration was feasible because all edge nodes use the same *rsync* tool [5] to synchronize their repository with the central server. Our results using the *Emulab* [6] network testbed indicate that the integration of file-swarming techniques with the synchronization software reduces the server workload of synchronizing the edge nodes significantly.

In our testing we laid more emphasis in examining the issues when using synchronization with file-swarming techniques. We learnt that there is tradeoff between exploiting similarity between the different versions of the package and the performance of the swarm for small file sizes. Synchronization software prefers smaller chunk sizes (approximately square root of the file size) to exploit the similarity between file versions while file-swarming techniques require moderately large chunk sizes (approximately 64 KB) to utilize the network socket buffers in the most efficient fashion.

4 SOFTWARE PACKAGE DELIVERY TO THE CUSTOMERS

Next we explored strategies to improve the software package delivery. A major problem in integrating pure P2P solutions in this scenario is that customers use different tools. Furthermore when providing a delivery service we desire to avoid consuming the customer bandwidth if there is an alternative way to procure the required bandwidth.

We contacted the administrators of the edge nodes and asked them whether they would be willing to volunteer their nodes for uploading software to machines not under their direct administration. Their acceptance allowed us to propose a better system design.

As shown in Fig. 2 a customer requests the server for a file. The server with the help of a file-swarming tracker then redirects the request to one of the volunteer nodes hosting the file. The customer downloads the entire file from that volunteer. Note that like existing CDNs our solution does not require custom tools.

There are several challenges involved in designing such a distributed infrastructure. Unlike CDNs that may use either dedicated connections or a private high speed networks for synchronization of their edge nodes, our system uses the same network for both synchronization and delivery tasks. Thus adaptive load balancing strategies are required that can account for both workloads. Security is another important issue, as we cannot fully trust the volunteer nodes.

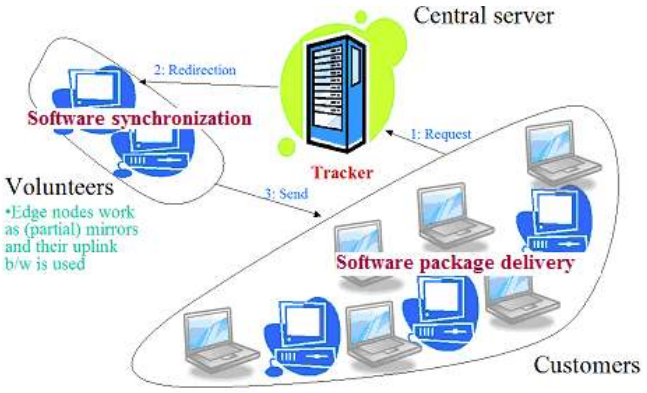

Fig. 2. System Design: Using Volunteer Assistance.

To address this issue, our system requires that its customers first communicate with the server before contacting any of the volunteer nodes. As a result, the customers will be able to authenticate the data received from the volunteers through the use of MD4 checksums provided by the server. In addition, our approach enables the server to attempt to optimize the individual workloads of the volunteer nodes.

5 ONGOING WORK

We are now in the process of implementing an adaptive system to balance the workload among the volunteers. This system will exploit the information available at the tracker and assist the server to find out which volunteers have which files. Additionally, it will provide the server with estimates of the volunteers' workloads and help make better load balancing decisions. Our simulation results indicate that by utilizing the information available at the tracker we can aptly distribute the load to the volunteer nodes.

In the future, we plan to study data placement policies that can handle volatile volunteers. This would allow individuals to donate idle machine time to improve the software package delivery process. Our trace analysis indicates that by replicating only the newly released packages we can make use of the volatile volunteers to diminish the flash crowd effects.

Our design is unique in that it combines the concept of volunteering with the P2P technology and is able to integrate with the existing system.

REFERENCES

[1] Akamai white papers. On-line at: *http://www.akamai.com/html/perspectives/whitepapers.html*

[2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11): 56–61, Nov. 2002.

[3] B. Cohen, Incentives build robustness in BitTorrent, *Proc. 1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[4] M. Freedman, E. Freudenthal and D. Mazières, Democratizing content publication with Coral, Proc. *1st USENIX/ACM Symp. on Networked Systems Design and Implementation*, Mar. 2004.

[5] Rsync: Remote file synchronization system. On-line at: *http://samba.anu.edu.au/rsync/documentation.html*

[6] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad and M. Newbold, M. Hibler, C. Barb and A. Joglekar, An integrated experimental environment for distributed systems and networks, Proc. *5th Symp. on Operating Systems Design and Implementation,* Dec. 2002.