Data-Oriented Transfer: Architecture and Techniques for Internet Data Transfers

Himabindu Pucha
Carnegie Mellon University

Michael Kaminsky
Intel Research Pittsburgh

David G. Andersen
Carnegie Mellon University

ABSTRACT

Our demo illustrates the benefit of *dsync*, a file transfer tool built within the Data-Oriented Transfer service, in a **software synchronization** scenario; *dsync* updates hundreds of PlanetLab nodes to the desired software version using only a single original source in real time while outperforming today's state-of-the-art systems, *rsync* and BitTorrent. Our demo will also highlight the effectiveness of *dsync* in dealing with a diverse range of operating conditions.

Categories and Subject Descriptors: C.2.4 Computer Communication Networks: Distributed Systems

General Terms: Design, Experimentation, Performance.

Keywords: Network, rsync, BitTorrent, synchronization.

1. OVERVIEW

File transfers are a nearly universal concern among computer users. Home users download software updates and upload backup images (or delta images), share multimedia content among peers, and enterprise users often distribute software packages to cluster or client machines. Consequently, a number of techniques have been proposed to address file transfer, including simple direct mechanisms such as FTP, "swarming" peer-to-peer systems such as BitTorrent,and tools such as that attempt to transfer only the small delta needed to re-create a file at the receiver.

Unfortunately, these systems fail to deliver optimal performance due to two related problems. First, the solutions typically focus on one particular resource strategy to the exclusion of others. For example, *rsync*'s delta approach will accelerate transfers in low-bandwidth environments when a previous version of the file exists in the current directory, but not when a useful version exists in a sibling directory or, e.g., `/tmp`. Second, existing solutions typically do not adapt to unexpected environments. As an example, *rsync*, by default, always inspects previous file versions to "accelerate" the transfer—even on fast networks when such inspections contend with the write portion of the transfer and *degrade* overall performance.

This demo presents *dsync* [2], a transfer tool that overcomes these drawbacks. To address the first problem, *dsync* opportunistically uses all available sources of data: the sender, network peers, and similar data on the receiver's local disk. *dsync* addresses the second problem, the failure of existing file transfer tools to accommodate diverse environments, by constantly monitoring resource usage and adapting when necessary. For example, *dsync* includes mechanisms to throttle aggressive disk usage if either the disk or CPU becomes a bottleneck in accepting data from the network.

Copyright is held by the author/owner(s).
SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.
ACM 978-1-60558-175-0/08/08.

The file transfer tool, *dsync*, is built within DOT [3], a flexible transfer service. DOT's architecture separates content negotiation from the data transfer itself, enabling new transfer mechanisms to be easily deployed without modifying existing applications. In DOT, applications determine what data they need to send and then use DOT's transfer service to send it.

DOT operates by first receiving a description of the file or files to be transferred from the file transfer application at the receiver. DOT receiver then uses this description to fetch the "recipe" for the file transfer. Like many other systems, DOT source divides each file or set of files (file tree) into roughly 16 KB chunks using Rabin fingerprinting, and it hashes each chunk to compute a chunk ID. Given a recipe (usually provided by the sender), DOT attempts to fetch the chunks described in it using *dsync*'s transfer techniques.

dsync attempts to obtain data from two sources: the network and the disk. *dsync* obtains content from network peers and the original sender using the SET [1] sub-system. SET discovers network peers downloading content that is identical or similar to the on-going download using handprinting, a technique with constant lookups and mappings. *dsync* also opportunistically locates chunks on the receiver's local disk. First, *dsync* consults a pre-computed index if one is available. If the index is not available or if the chunk ID is not in the index (or the index is out-of-date), *dsync* tries to locate and hash files that are most likely to contain chunks that the receiver is trying to fetch.

Our demo of *dsync* will highlight the following:

- *dsync* leverages the Data-Oriented Transfer (DOT) architecture to provide an application-independent file transfer system.
- *dsync* uses SET to benefit from additional network peers downloading identical or similar content.
- *dsync*'s mechanisms enable it to locate and leverage similar content if it exists on the receiver's disk, adapt to varying disk loads, and cope with receivers in different initial states.

2. REFERENCES

[1] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, Apr. 2007.

[2] H. Pucha, M. Kaminsky, D. G. Andersen, and M. A. Kozuch. Adaptive file transfers for diverse environments. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2008. (To appear).

[3] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for Internet data transfer. In *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006.