

Peer-to-Peer Computing for Secure High Performance Data Copying

Andrew Hanushevsky, Artem Trunov, Les Cottrell

Presented at the International Conference on Computing in High-Energy and
Nuclear Physics (CHEP'01), 9/3/2001—9/7/2001, Beijing, China

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

Work supported by Department of Energy contract DE-AC03-76SF00515.

Peer-to-Peer Computing for Secure High Performance Data Copying

Andrew Hanushevsky, SLAC, Artem Trunov, SLAC, Les Cottrell, SLAC

Produced under contract DE-AC03-76SF00515 between Stanford University and the Dept. of Energy

Abstract

The BaBar Copy Program (bbcp) is an excellent representative of peer-to-peer (P2P) computing. It is also a pioneering application of its type in the P2P arena. Built upon the foundation of its predecessor, Secure Fast Copy (sfc), bbcP incorporates significant improvements performance and usability. As with sfc, bbcP uses ssh for authentication; providing an elegant and simple working model -- if you can ssh to a location, you can copy files to or from that location. To fully support this notion, bbcP transparently supports 3rd party copy operations. The program also incorporates several mechanisms to deal with firewall security; the bane of P2P computing. To achieve high performance in a wide area network, bbcP allows a user to independently specify, the number of parallel network streams, tcp window size, and the file I/O blocking factor. Using these parameters, data is pipelined from source to target to provide a uniform traffic pattern that maximizes router efficiency. For improved recoverability, bbcP also keeps track of copy operations so that an operation can be restarted from the point of failure at a later time; minimizing the amount of network traffic in the event of a copy failure. Here, we present the bbcP architecture, its various features, and the reasons for their inclusion..

The Peer-to-Peer Approach

BaBar Copy, bbcP, is a true peer-to-peer (P2P) program. In a P2P architecture there is no discernable client or server, even though two nodes communicate with each other. Either node can act as a server or as a client. P2P architectures are well suited to environments where information flow is equal. For instance, in most instances, when copying data from one place to another it does not matter who initiates the copy operation. Additionally, it could very well be the case that desired data exists on both nodes and each node wishes to swap some portion of the data. In situations such as these, it makes very little sense to differentiate nodes into client-side and server-side applications.

P2P also carries with it a very low administrative overhead. In fact, all one needs to do to run a P2P application is to have the same program available in each of the nodes that is to run the application. There is no administrative installation process. This allows P2P applications to be deployed in a minimum amount of time.

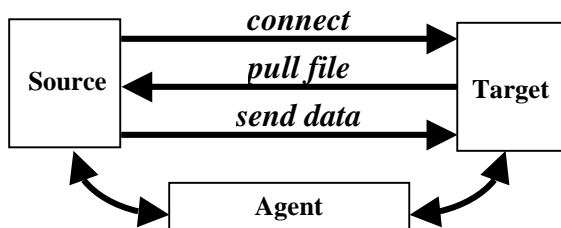


Figure 1: Peer-to-peer architecture

The predecessor of bbcP, sfc, proved the utility of the P2P approach. In bbcP, that approach has been further refined to better support 3rd copy operations. In this mode, a node that is neither the source nor the target of the copy operation initiates the copy. In bbcP 3rd-party copy is supported using an agent that initiates the copy operation by instantiating a source node and a target node

and then requesting the target to “suck” the data from the source; as illustrated in figure 1. This eliminates special case handling, bbcP always employs an agent whether or not the agent is co-located with the source or target nodes. The agent is responsible for relaying all the necessary parameters to the source and target in order to accomplish the copy operation.

Peer-to-Peer and Firewalls

Firewalls are a natural impediment to P2P architectures. Since P2P applications run outside any administrative oversight, they necessarily use arbitrary ports. Many firewalls block incoming and,

perhaps even outgoing, ports that are not known to site administrators. This usually prevents a P2P application from successfully executing. These problems are addressed in bbcp in two ways: 1) allowing the agent to instruct the target to connect to the source (the default is for the source to connect to the target), and 2) restricting the port usage to a well defined range.

The first solution, known as protocol reversal, attempts to overcome incoming or outgoing firewall restrictions and is accomplished by a simple command line option. In most cases, reversing the protocol connection method is sufficient to avoid restrictive firewalls. In cases where the source and sink firewalls present onerous restrictions, the user may co-ordinate with source or target node administrator to define a set of port numbers that will be allowed through the firewall and the administrator enters this range into the operating system services file. The presence of restricted port numbers is automatically detected by bbcp and it restricts its port usage to the predefined range. While this necessarily involves administrative overhead, this may be the only possible solution in high security environments.

Securing Copy Operations

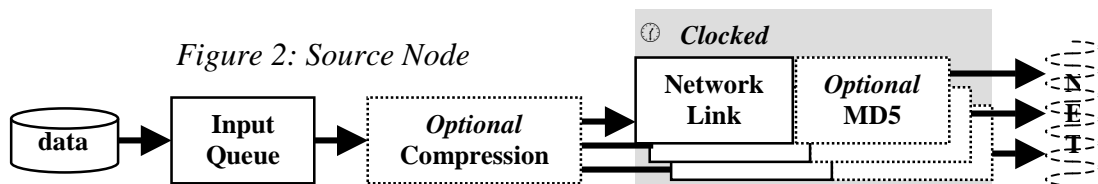
As with sftp, bbcp employs an embarrassingly simple authentication method: if a user can ssh to a site then that site can serve as a source or target node. There is no need for additional certificates, accounts, passwords, or permissions. The ability to use an existing and widely deployed authentication infrastructure allows bbcp to be used without much thought. In fact, in most cases it can be used as conveniently as ssh's scp command.

To further secure the copy operation, the agent sends all control information on an encrypted channel and prevents the source and target nodes from discovering each other's file system structure. Additionally, the agent constructs random one-time passwords that must be used to initiate communications on any unencrypted connection.

All of these mechanisms allow bbcp to maintain the highest level of security with a minimum amount of burden on the user.

Accelerating Copy Operations

The main method used by bbcp, as with sftp, to accelerate the movement of data is the use of parallel streams[1]. This is not a new concept and many other programs use this method[2][5]. However, in the course of developing sftp we discovered that other mechanisms come into play and are equally important in maintaining high performance. These are: 1) TCP window size and buffer management coordination, 2) buffer alignment, 3) data pipelining, 4) serialization, 5) file system feature utilization, and 6) compression. The architecture employing all of these features is shown¹ in figure 2.



It is well known that choice of the appropriate TCP window size can substantially improve data transfer rates[3]. However, bbcp takes this concept in two new directions: a) all data buffers are sized to be the same as the window size, and b) the actual window size is automatically negotiated between the source and target nodes to be identical in size. Forcing all units of transfer to be the same size from start to end avoids the under-or over-filling of any system buffer queue. This simple concept

¹Only the source node is shown. The target is a mirror image of a source node.

usually allows the operating system's socket stream and TCP stack to run at peak efficiency with consequent improvements in performance. We call this window size and buffer management coordination and accounts for more than 20% improvement in overall performance.

Buffer alignment is also important. Data buffers are always aligned to operating system page boundaries. This allows the operating system to directly fill data buffers; avoiding unnecessary memory copy operations. While this is operating system implementation dependent, many advanced operating systems give preferential treatment to page aligned buffers. This is especially important for certain file systems that require buffers to be page aligned in order to bypass memory copy operations (see below).

The concept of data pipelining is relatively new in WAN data transfer operations. The concept arose from the recognition that in practice most parallel connections are routed the same way, once a route is established it rarely changes, and when a route changes all streams are usually rerouted. Additionally, many modern-day routers perform traffic-shaping by simply dropping packets on certain virtual circuits[6]. Data pipelining attempts to make a router's job easier and avoid the pitfalls of traffic-shaping by clocking out the data on all the streams in parallel. This necessarily means that should one stream run into problems, all of the other streams are equally affected. While this may sound disadvantageous, the counter-intuitive reality is that if a router is performing traffic shaping it is to one's advantage to back-off the data transfer rate on all of the streams going through that router if one of the streams is being shaped. This usually alleviates congestion and decreases the number of dropped packets on all of the streams. In the end, the transfer rate settles to a uniform high rate across all of the parallel streams. In short, one converges to an optimal transfer rate by simply balancing the transfer rate across all of the parallel streams.

Unlike sftp, bcp serializes the data from source to target. We found that serialization had three important advantages: a) data is sequentially read from and written to disk, b) the range of input and output devices substantially increased, and c) recovery from failed transfers. Sequential (i.e., non-random) disk I/O can substantially increase disk data transfer rates[4]. This is especially true when direct I/O is possible. Because the data transfer is sequential, non-random devices (e.g., tape) can be directly used as the source and target devices. Finally, because a deterministic high watermark is maintained throughout the copy, a failed copy operation can be trivially resumed because the amount of successfully transferred data is always known. We should point out that data serialization is only practically possible in a pipelined architecture. Even then, bcp must make allowances for the elasticity of operating system and router buffer queues.

The utilization of specific file system features is possible in bcp because the file system object is instantiated at run-time. This allows the automatic selection of file-system dependent performance options such as memory cache bypass and direct buffer I/O (usually only to page aligned buffers). Depending on the file system, transfer rates can be dramatically increased.

The last optimization is compression. This optimization is problematic because one needs to trade-off file compressibility and available CPU resources against a decrease in network traffic. The trade-off is rarely obvious before a transfer operation starts. While we have no hard experimental data, informal tests show that in the majority of cases, compression does not yield better transfer rates.

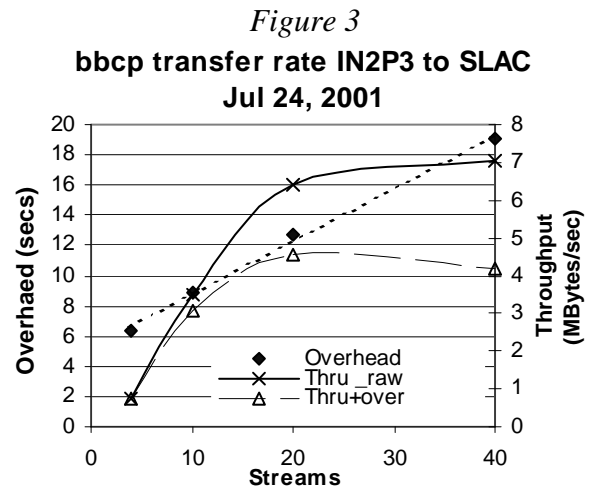
Usability Features

While transfer rate is of paramount importance in a WAN copy program; bcp was built on our experiences with sftp that usability may be equally important in many circumstances. Consequently, we added the following features: 1) the ability to copy data from multiple distributed sources, 2) allowing a list of source files to be specified in a control file, 3) continuation of partially copied files,

4) preservation of a source file's meta-data, 5) ability to specify options in a configuration file, 6) selectable level of compression, 7) MD5 data checksum, 8) configurable execution environment, 9) periodic progress messages, 10) output log file, 11) transfer rate limiting, 12) propagation of source directory hierarchies, and 13) using the well-known syntax of the Unix copy command.

Results

Our results with bbcp have been quite good, as shown in figure 3. Here we show the transfer rate achieved with bbcp between SLAC² (San Francisco, USA) using a and IN2P3³ (Lyon, France) versus the number of streams used. The dotted line shows the one-time overhead⁴ to start a data transfer operation. In the current version the overhead increases linearly with the number of streams. The overhead will be constant in future versions. The top curve is the maximum transfer rate discounting the setup overhead. In this particular test, 20 streams was optimal with a transfer rate of 6.5MB/sec. This is exceptional performance and is within a few percent of that achieved using iperf with TCP and similar streams and window sizes[7]. Other WAN routes provided comparable transfer rates. Local area rates were also outstanding with bbcp easily saturating gigabit Ethernet adapters.



Future Directions

bbcp was written with extensibility in mind using C++ with an object-component design. This allows non-disruptive additions and modifications. We are considering experimenting with the following architectural additions: 1) external network performance feedback to dynamically control the data clocking rate, number of streams, and window size, 2) internal Round Trip Time (RTT) measurements to automatically select the appropriate defaults, 3) integration of Quality of Service (QoS) networking options, 4) real-time logging (e.g., via netlogger), and 5) support of HPSS tape-based file systems.

bbcp is an open-source program with pre-compiled binaries currently available for Solaris and Linux. Additional information, including full documentation, can be found at <http://www.slac.stanford.edu/~abh/bbcp>

References

- [1] <http://www-iepm.slac.stanford.edu/monitoring/bulk/>
- [2] <http://hepwww.rl.ac.uk/Adye/talks/010402-ftp/html/>
- [3] http://www.psc.edu/networking/perf_tune.html
- [4] <http://sunsite.nstn.sk.su/sunworldonline/swol-09-1995/swol-09-raid5.html>
- [5] <http://www.isi.edu/touch/pubs/jsac95.html>
- [6] http://www.security-informer.com/english/crd_traffic_404468.html
- [7] <http://www-iepm.slac.stanford.edu/monitoring/bulk/bbcp.html>

²A 4 CPU Sun E4500 with a gigabit Ethernet adapter running Solaris 8.

³A 1 CPU Sun Ultra-4 with a 100 megabit Ethernet adapter running Solaris 6.

⁴ Once a route is established between a source and target node, any number of files may be transferred with no additional setup overhead.