Better Bonded Ethernet Load Balancing

Jason Gabler, NERSC / LBL
jygabler@lbl.gov

When a High Performance Storage System's mover shuttles large amounts of data to storage over a single Ethernet device that single channel can rapidly become saturated. Using Linux Ethernet channel bonding to address this and similar situations was not, until now, a viable solution. The various modes in which channel bonding could be configured always offered some benefit but only under strict conditions or at a system resource cost that was greater than the benefit gained by using channel bonding. Newer bonding modes designed by various networking hardware companies, helpful in such networking scenarios, were already present in their own switches. However, Linux-based systems were unable to take advantage of those new modes as they had not yet been implemented in the Linux kernel bonding driver. So, except for basic fault tolerance, Linux channel bonding could not positively combine separate Ethernet devices to provide the necessary bandwidth.

Problems with Bonding

The Linux Channel Bonding Project[1], source of the official bonding driver in the standard Linux kernel distribution, offers six bonding modes which fall into four categories:

The first type of bonding stripes a connection across the pool of bonded slaves (Ethernet devices) in a round-robin fashion. While this provides the most complete use of each slave device and gives the potential to use more than a single device's worth of bandwidth per connection, it results in the destination host receiving packets out of order. To correct this problem TCP/IP's congestion control system must work very hard at reconstructing packet order at a very heavy processor cost. Even with kernel parameter tweaking, "for a four interface balance-rr [*round robin*] bond, expect that a single TCP/IP stream will utilize no more than approximately 2.3 interfaces worth of throughput."[2]

The second type of bonding load balances by evaluating the current load of a slave. While this would seem sufficient for the needs of the described HSM scenario, the mode is dependent upon the ARP protocol. This makes such bonding modes only useful between hosts on the same subnet. This can be difficult or impossible to arrange as it would most likely force otherwise unwanted changes in already established network topologies.

The third type provides fault tolerance with minimal or no load balancing features. While this might be helpful in a HSM scenario or similar cases as second-tier bonding[3], a balanced load and, more poignantly, over all maximum

1 http://sourceforge.net/projects/bonding/
2 http://www.ibiblio.org/peanut/Kernel-2.6.12/networking/bonding.txt, T. Davis, J. Vosburgh

throughput is what we are after.

The fourth bonding type considers aspects of the traffic itself, specifically the OSI layer 2 data to provide balancing while also providing fault tolerance. The slave device (or *channel*) is chosen with the following hashing algorithm:

```
channel = (src MAC XOR dst MAC) modulo (number of channels)
```

Using static MAC addresses, this algorithm will place multiple simultaneous streams from the same hosts on the same slave device, saturating it. This results in essentially the same per-client host performance as though there was no bond at all.

Better Hashing

Any additions or modifications to bonding functionality needed to maintain some level of determinism when choosing the slave and also continue to include the fault tolerance already programmed into many of the bonding modes. This was accomplished by altering the slave choosing algorithm from a layer 2-based mode to an algorithm based on Layers 3 and 4 data. The new algorithm is:

```
channel = ( (src port XOR dst port) XOR (src IP XOR dst IP) )
                modulo (number of channels)
```
[4]

Now the slave device is chosen according to parameters for each individual connection. As a source hosts provides unused port numbers to the algorithm for new connections, there is now the possibility that connections from this same source host will not land on the same slave device. Likewise, port numbers have less chance of producing an "unlucky" combination as IP addresses provide more variability to the hash. While there is still the chance that a single slave may become disproportionately overloaded while communicating with the same remote host, it is no longer guaranteed.[5]

The following graphs compare performance between the two hashing algorithms. All tests were performed with Iperf[6] between two hosts, each with two bonded Ethernet devices. The Figure 1 shows the effective bandwidth usage between two static hosts.

---

[3] That is, the first "tier" is two bonds, each using a load balancing bonding mode. These are then related through second tier comprised of a single bond providing only fault tolerance. As of the writing of this paper, the enslavement of bonds (or "bonding bonds") is not officially supported. Although there has been some reported success, there are still ARP-related issues during failover.

4 A documented source showing this algorithm in current use by major network switch manufacturers such as Foundry and Cisco: http://www.foundrynet.com/services/documentation/sribcg/Trunking.html#112750. Note – this has also been implemented into the AIX operating system by IBM thanks to the efforts of Brent Draney at NERSC.

5The reader might wonder at this point why the bonding mode should not instead keep a counter and simply cycle through the available slaves on a per-connection basis instead of this more involved algorithm. That sort of method would guarantee a single remote host to domination of the entire bond if it were to transfer enough data over multiple connections. Keeping the method at least somewhat deterministic gives all remote hosts an equal chance at using each slave device,.

6 http://dast.nlanr.net/Projects/Iperf/

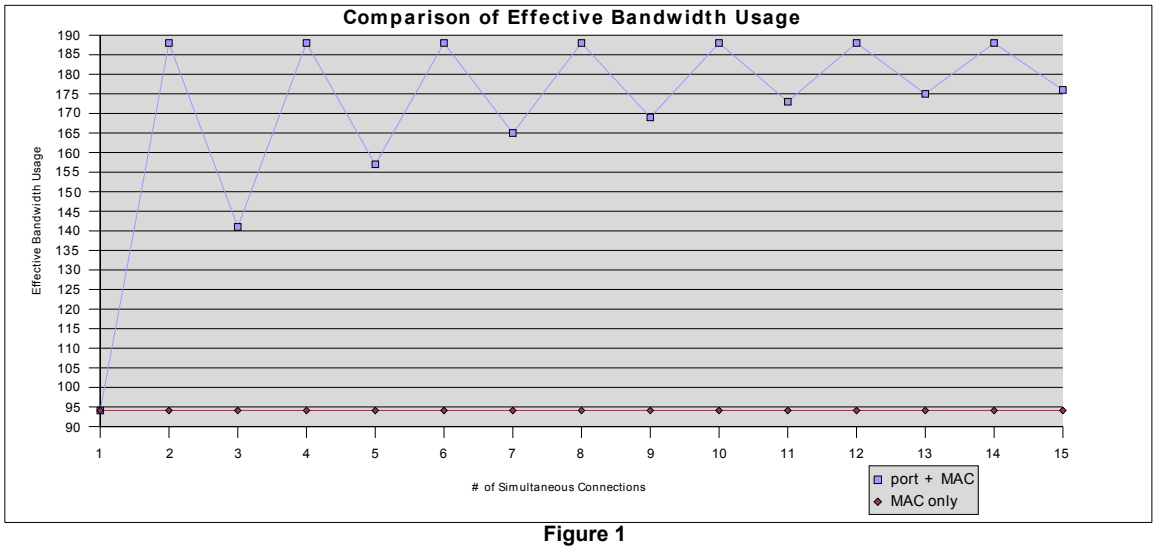

Figure 1

Figure 2 shows the true value of the new hashing algorithm by displaying the average bandwidth used for a given number of simultaneous connections between two static hosts. Essentially, data point for data point, the bandwidth is doubled for a channel pool size of 2, which should be no surprise. This benefit will continue to grow as a multiple of the channel pool size, limited only by the host architecture's capability to handle the increased effective bandwidth.

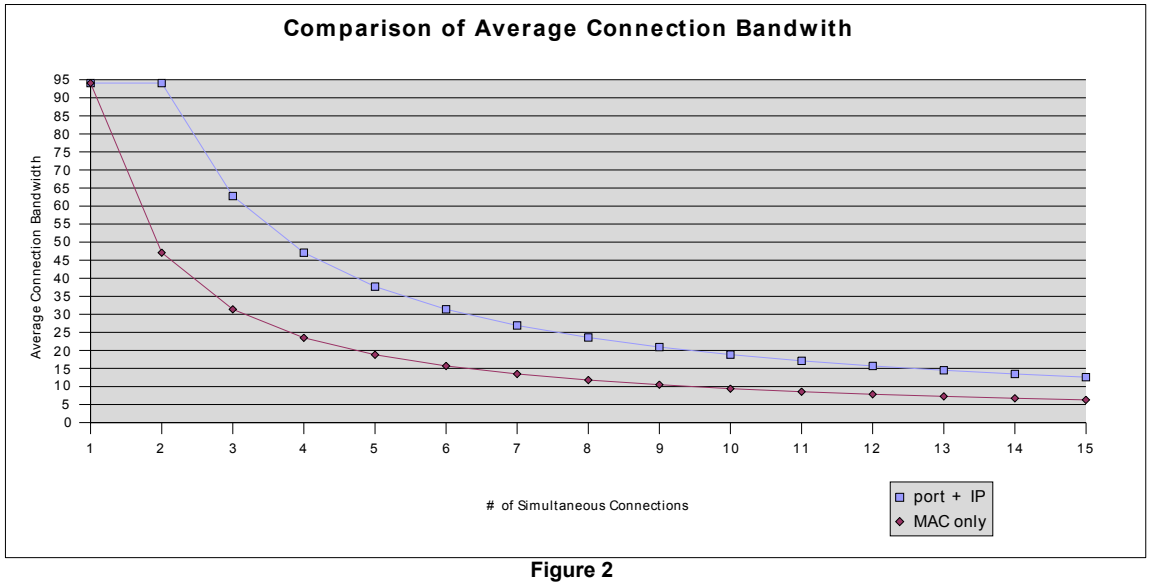

Figure 2

Other Implementation Particulars

The new algorithm shown above, being in need of layer 4 data, is not designed to handle non-TCP or non-UDP traffic. For IP traffic, which has no port data, such as ICMP or even TCP/IP fragments, the new implementation relies only on Layer 3 data to hash for the slave. In the case of non-IP traffic, the

original Layer 2 method is mimicked. It should be noted that this is not fully 802.3ad compliant and other devices which are compliant may not tolerate this behavior. However, this should not be a problem as local network bonding needs can be solved with other bonding modes and TCP fragmented is not common.

For backward compatibility with the original Layer 2 based bonding mode (called "*balance-xor*", or "*mode 2*") and to avoid the potential of creating an unwieldy group of bonding modes, the addition of this new algorithm was done by turning mode 2 into a more generic hashing mode which can select from a pool of "pluggable" hashing algorithms[7]. A second kernel option compliments the specification of mode 2 which provides the choice of which hashing algorithm to use. This makes for simpler maintenance of the bonding driver code and also permits the user to rely on the otherwise same behavior (such as fault tolerance) that mode 2 has already and continues to supply.

Conclusions

Ether Channel bonding was originally seen as a solution for local bandwidth bottlenecks, such as increasing the effective bandwidth between the nodes of a cluster to create a low-cost high performance interconnect. Once bonding usage stepped outside of the local area network it was only a tool for implementing fault tolerance. Now, with the addition of this new hashing technique, all of that fault tolerance is retained while gaining the ability to bond channel bandwidth for wide area use. HSM scenarios are not the only application. This has already been a boon to the emerging, facility-wide, distributed parallel file system at NERSC[8], bonding multiple gigabit Ethernet devices for greater access across the entire Center.

7A suggestion from Jay Vosburgh, Linux Channel Bonding Project. See the Linux Kernel source file .../Documentation/networking/bonding.txt for mode details on bonding driver modes.

[8] See http://www.nersc.gov for the Facility Wide File System