A Note on Guard Tag Calculation Algorithms

Martin K. Petersen, Oracle
martin.petersen@oracle.com

February 10th 2007

Introduction

When we did the initial feasibility study of whether T10 DIF was a suitable successor to HARD I did some rudimentary benchmarking of the impact of having the host processor calculate the CRC on each I/O.

I have followed up this week by doing a simple benchmark using a program which will perform either a checksum or a CRC on buffers of various size.

One test runs exclusively in memory giving us the maximum throughput of the processor. Another test writes the buffer to a disk array which consists of 10 15Krpm drives. The array is hooked up to a 2Gbps fabric using Emulex LP11002 controllers. The drives are striped in 512KB chunks using Linux software RAID.

All tests results are an average of 10 runs.

1 Algorithms

The tests were done using the following algorithms:

Algorithm	Description
XOR	A simple XOR of all bytes in the buffer
TCP Ref.	The TCP/IP reference checksum as found in RFC1071
TCP Opt.	The TCP/IP checksum optimized to work on 32-bit words
T10 Ref.	The T10 reference CRC as found in SBC-3
T10 Opt.	The T10 CRC optimized using a lookup table

Table 1: Algorithms

All my optimizations were platform-independent and written in C. A colleague and I are exploring optimizing both the T10 CRC and the TCP checksums using the SSE instructions found in modern x86 / x86_64 processors.

SSE is a set of SIMD (single instruction, multiple data) instructions used especially in 3D applications. SSE can be leveraged when working on chunks of data bigger than 32 bits (64 or 128 bit). SSE is currently used for RAID5 and RAID6 parity calculations within the Linux kernel.

Another point worth mentioning is that intel recently released specifications for the upcoming SSE4 instruction set. And there is a dedicated CRC operation included (supporting any polynomium). This means that a year or two down the road we will be able to perform the CRC calculation entirely in hardware.

2 In-memory performance

The numbers below indicate the bandwidth of each algorithm on a selection of host processors. All tests are single-threaded so only one core was busy on the multicore processors. In all cases a 1GB buffer was used:

Processor	XOR	TCP Ref.	TCP Opt.	T10 Ref.	T10 Opt.
1.3 GHz Itanium 2	397.89	338.37	1115.87	28.03	152.51
2.2 GHz Opteron	554.51	549.93	889.27	58.30	220.04
3.0 GHz Xeon MP	363.38	504.49	577.66	22.73	222.37
2.13 GHz Xeon 3050	392.24	589.55	1033.55	31.47	221.43

Table 2: Single-threaded in-memory performance in MB/s. 100% CPU

The T10 reference CRC is the slowest. My table-optimized implementation yields 5 times the performance on average, depending on processor.

The TCP reference implementation and the XOR are comparable on most processor types. This is the overhead we are paying today using HARD. Interestingly enough, XOR is fairly slow on the Xeon 3050 which is the most recent processor in the mix.

The optimized TCP checksum function is by far the fastest in all cases.

3 Disk Performance

The single-threaded disk performance graphs look like this:

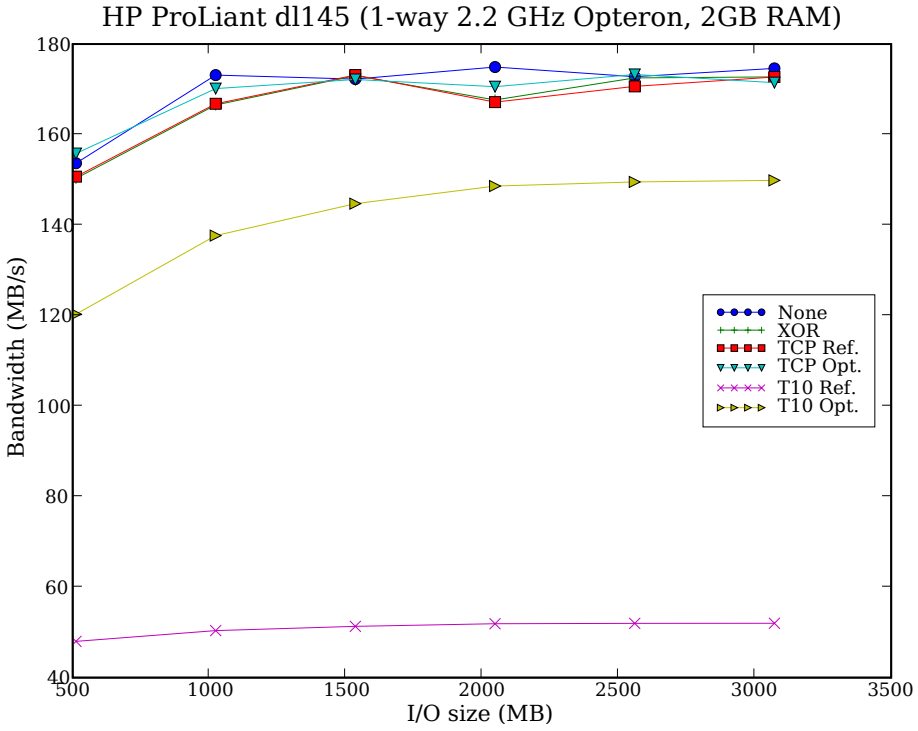

Figure 1: I/O bandwidth as a function of I/O size

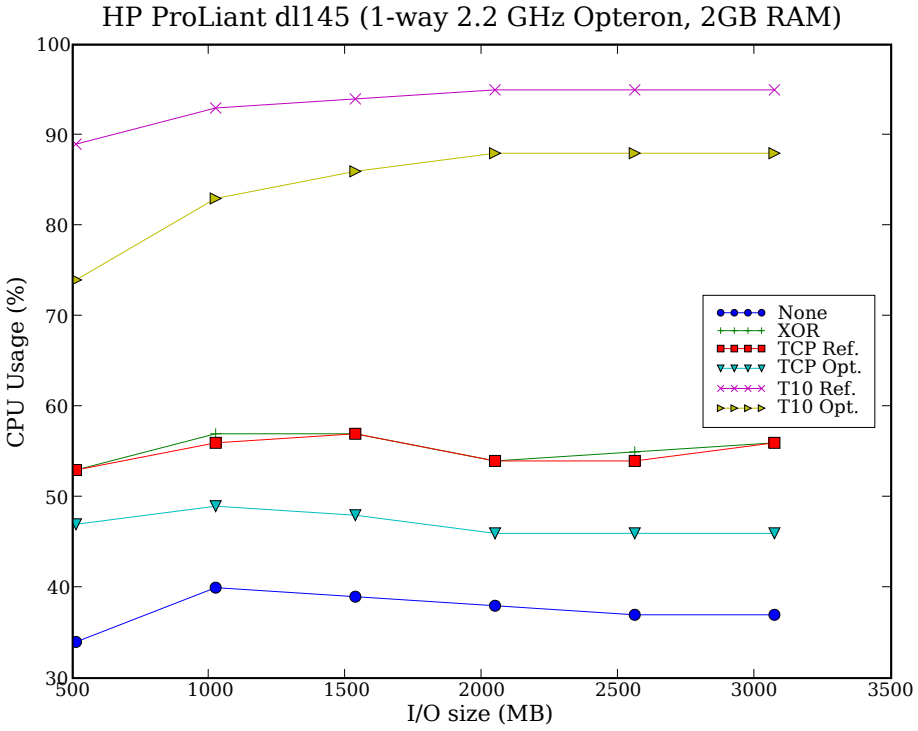

Figure 2: Processor load in %

The “None” graph represents the performance of doing the I/O without any checksumming. T10 Ref. is the slowest performer and also the biggest consumer of processing power. The table-optimized T10 CRC yields much better bandwidth but still has a high CPU usage. And just like in the pure memory test, XOR and the reference TCP checksums are comparable.

The interesting part, of course, is that the optimized TCP checksum algorithm is not only the best performer. It also has the lowest processor load (45 – 48%), only 15 – 18% above the baseline.