Interactive Silicon

Compression Takes Aim at I/O Bottlenecks

August 2001

Compression Takes Aim at I/O Bottlenecks

Introduction

Although compression has been used widely in the IT world for the purpose of maximizing storage space, only recently has it begun to be applied for the purpose of reducing I/O bottlenecks. So far, these applications have been limited to specialized hardware solutions at the link and physical layers and specialized software solutions at the application layer.

The concepts of compressed caching and compressed memory have been studied in academia for many years, but it is the necessity to address the disk I/O bottleneck in the real world that has finally mothered invention and produced a practical solution. Despite improvements in hard drive technology, the performance-robbing disparity between CPU speed and disk drive speed continues to grow. The improvement to system performance provided by compressed caching is so great that it promises to become standard fare not just for server farms and across networks, but eventually for laptops and desktops as well.

Some Background

The modern development of compression algorithms is made possible by Claude Shannon's rigorous development of information theory in the late 1940s, and benefits from much development work that began in the 1970s when computer use began to boom. It continues to evolve in directions dictated by the needs of the information age.

Lossless Compression

Initially, compression applications were confined to the use of lossless algorithms. Lossless refers to the fact that after the compressed file is decompressed it returns to its original state, with no loss of information. The compression and decompression of important data must of course always be lossless. Current algorithms for tape backup and other archiving, as well as popular space–saving utilities such as WinZIP, all use lossless compression. While most of these applications were designed to save disk space, others, such as the CCITT Group 4 standard for fax machines, were designed to shorten the amount of time required to transmit the information.

Lossy Compression

As computers embraced different media, the need to archive and transfer very large files containing audio, video, or still image information drove the development of lossy algorithms. With lossy compression, the original master file is not recovered after decompression, and some information is lost as a result. This allows a higher rate of compression and is acceptable because the information that is lost is determined to be information that, for the most part, is not noticed or perceived by the brain to be important. So it can be left out with little or no perceived degradation.

Lossy algorithms have achieved more fame and fortune in recent years as they have been under the Internet spotlight, making possible real-time audio and video streaming. The industry has developed many standards for compression including GIF and JPEG for compression of still images and MPEG for compression of audio and video files. Many of these video and audio compression schemes are implemented in real time with specialized hardware. Cellular phone transmission of audio data, for example, is made possible by a real-time CODEC (compressor, decompressor) in the cell phone itself. It compresses digitized audio in a lossy manner before transmission, and decompresses the compressed data as it is received.

Compression Ratios

Lossless algorithms tend to achieve compression ratios approaching 2:1 on average, though much higher ratios can be achieved with certain kinds of data. Lossy algorithms make use of highly mathematical techniques such as DCTs (Discrete Cosine Transforms) for stripping out unimportant spatial or temporal frequencies, allowing them to achieve higher ratios, depending on the degree of degradation the user is willing to accept. Close examination of a JPEG file, for example, will show that higher compression ratios produce lower resolution images. All lossy techniques make use of lossless compression algorithms once the extraneous and unimportant information has been stripped.

Identification of Data Structure

The identification of structure in data is vital to achieving good compression ratios. Different data has different patterns and distributions of characters and thus compresses differently. It is never completely random, unless intentionally made so. If so, it could not be compressed. An analysis of the data structure cannot be ignored without limiting the amount of compression that could otherwise be achieved by using a more efficient algorithm, which is designed to compress data that has been identified as containing a particular pattern or structure. As the data that resides on a server can contain different kinds of structure, it is important to either have an a priori knowledge of the data structure so that compression algorithms can be tuned for optimum performance or utilize technology that can do this automatically in real time.

Latency Mismatch and Bottlenecks

Bottlenecks occur in the transmission of data whenever there is a difference in data access time, or latency mismatch, between two devices. When one device has to slow down or wait on another, the system is reduced to operating at the speed of the slowest device. This is the number-one problem in computer performance today.

We now have CPUs that operate at speeds well in excess of 1 GHz, and as we all know, this will soon seem slow. This power is only being fully utilized when the application is processor-bound. That is, it is working with data that resides in internal registers and not in external memory or storage devices. Unfortunately, main memory that can be accessed at this rate is expensive and power hungry. Thus, small amounts of fast memory are utilized as L1 and L2 cache to ease the worst-case scenarios.

In a typical design, L1 cache might have an access time of 1 nsec, L2 cache might have an access time of 10 nsec, and main memory might have an access time of 100 nsec. Note that the latency mismatch between adjoining layers of storage is one order of magnitude. This fits in well with the concept of how cache should be designed and utilized. Under most conditions, more accesses occur in an adjoining faster layer than a slower one, helping to compensate for the fact that the adjoining storage is ten times slower.

Now let’s look at the next layer of storage in the latency hierarchy, the hard disk. A typical access time for hard disk is five msec. The mismatch now is almost five orders of magnitude. Any time virtual memory experiences a page fault or an I/O call to disk is issued, performance takes a huge hit. The obvious place to insert another cache is at this level, between main memory and hard disk. To perform as a suitable cache, the storage media should be at least an order of magnitude faster than the slowest adjoining layer, which in this instance is disk. So, the cache should have an access time of 500 usec (5 msec ÷ 10) or less, and can be as slow as 1 usec, which is an order of magnitude slower than the faster adjoining layer of main memory.

Defining a Compression Solution

There are presently two different approaches to utilizing compression to minimize I/O bottlenecks – compressed memory and compressed cache. Both can be designed to use powerful and efficient compressor engines that operate completely transparent to the user. The compressed memory approach utilizes a solution that attempts to compress all of main memory on boot-up, and tells the operating system that more memory is available for use than is physically in place. The compressed cache solution dynamically allocates a portion of main memory to serve as a cache when accessing slower I/O.

It is important to realize that how a particular approach is implemented is just as important as which approach is taken. Before implementing a compression scheme the user should carefully consider the following points:

- Understand the cost and the value of compression as it affects performance in a particular solution. Even though modern hardware compression engines can compress data as fast as can be read into them (in one clock cycle), there is a cost incurred that consists of the time it takes the CPU to read data into and out of the CODEC. This kernel overhead is time that would otherwise be spent working on an application. The value of the compression comes when you compare the time saved to access a page of data that is in compressed form with the time it takes to access a page that would otherwise be on hard disk.

- Make sure the latency associated with the compression solution fits properly into the latency hierarchy. For example, it makes little sense to insert a compression solution with a latency of 50 usec between the L2 cache and main memory, because the latencies of both adjoining layers are faster than 50 usec. This would mean that every time the CPU accesses main memory it takes a performance hit, because it must access slower compressed memory instead. This is a huge cost and can be somewhat alleviated by the addition of an L3 cache. The value lies in the fact that every time the CPU accesses a page that would otherwise be on hard disk, performance is improved. But the overall value vs. cost relationship is favorable only in a very narrow set of applications, which are typically restricted to mid-range servers.

- The manner in which the compression solution is integrated into the operating system's I/O and virtual memory management operations is extremely important, and can make a huge difference in performance. For example, a compression engine might be capable of producing very high compression ratios on certain kinds of data, but if the interface defines the compression solution to the operating system as a memory doubler (2:1 compression ratio), then the remainder of memory must be zeroed out. The operating system is unable to take advantage of anything more than what it is told exists, which is twice the physical memory.

- Understand how the solution decides when to use compression. Does it attempt to compress everything, even when the data is not compressible and nothing can be gained (thus incurring a cost but no value), or does it only perform compression when the value outweighs the cost?

- Does the solution allow the user to tune the compression algorithms to the particular data being compressed? Better yet, does the solution perform continual analysis of the data and perform tuning automatically in real time?

- Understand how the solution is integrated into the operating system. What kinds of drivers are used? Are they transparent to the operating system? Do they require operating system support?

- Can the technology be retrofitted into legacy equipment, or must the user wait for it to be incorporated into new motherboard or CPU designs?

Implementing a Compressed Cache

The concept of compressed cache involves a general-purpose and flexible design that can be utilized anywhere there is a latency mismatch. It can be utilized to reduce I/O bottlenecks when transferring data to and from hard disk, across networks, within router switches, etc. Compressed data is only stored in the cache, never on the slower storage media.

One promising solution involves utilizing portions of the system's existing main memory as a disk cache, which complements both the virtual memory subsystem's page cache and the file system's cache. The effect is to make the system appear to have more memory. Although the access time of the additional storage space is slower than main memory, it is far faster than disk. The solution is completely transparent to the user, and no BIOS, operating system, or application changes are required.

The compression process is controlled by a software driver layered on top of the operating system (Windows® XP, Windows 2000, Windows NT, Linux) in such a manner that ensures cache affinity between the operating system's page and file system caches and the compressed cache. The software driver communicates with a CODEC residing on a registered SDRAM PC-133 DIMM memory module. The CODEC is a very low-latency, high-bandwidth parallel compression engine implemented on an ASIC chip. It compresses and decompresses in real time, as fast as data can be read into and out of it. This DIMM is initially asleep and behaves like all the other DIMMs in main memory. The compression engine on the onboard ASIC can compress not just the memory on its own module but on the other standard DIMM modules as well.

The intelligent software driver continuously monitors parameters such as compressibility of data, disk request rates, effective disk latencies, processor utilization, and hit rates. It then dynamically allocates memory for compressed cache as needed. The compressed cache intentionally competes with the operating system and applications for main memory as it attempts to make main memory appear as large as possible, thus increasing the likelihood that the pages needed by the application are already in main memory. Adaptive algorithms selectively cache pages that are more compressible and that are causing excessive disk activity. The result is a compressed cache that maintains a very high compression ratio (2:1 to 6:1 is typical) and only caches pages that are likely to improve performance.

The compressed cache grows and shrinks as the software driver responds to demands placed on the system by the application environment. When all of the application fits in main memory, the compressed cache is shrunk. When the application requires more memory or is performing excessive file I/O, the driver allocates more memory for cache to minimize disk activity. Access time for pages in the compressed cache is three orders of magnitude faster than for pages stored on disk.

In addition, user-friendly tuning facilities and an API allow the user or applications to monitor the status of the DIMM module and the software driver, as well as tune the environment by specifying files, directories, and partitions to be tracked as writeback, writethrough, or non-cacheable.

Potential for Improving Performance Will Grow

Compressed cache can currently increase effective system bandwidth across storage networks by a factor of two to six times. Future performance gains will be increased as the process becomes integrated onto the motherboard and eventually onto the CPU. In addition, the performance gains will only grow as CPU speeds continue to increase, simply because CPU speeds continue to increase at a faster rate than I/O speeds. And the cost of the technology will continue to scale downwards and remain less than adding more memory. Wherever additional memory will boost performance, compressed cache will be able to do it for less cost.