

DEFRAGMENTING RAID

By Mark E. Donaldson

Fragmentation on RAID is an intricate topic. Because the purpose of RAID is to offer redundancy, as well as improved disk performance by splitting the I/O load, it is a common misconception that fragmentation does not have a negative impact.

However, RAID does suffer from fragmentation, just as any single physical disk does. And this is attributed to the difference in the "logical" allocation of files versus their "physical" allocation. The file system driver handles the logical location (what the OS sees), in this case we're talking about ntfs.sys. The actual writing is then passed to the fault tolerant device driver (hardware or software, it makes no difference), which then, according to its procedures, handles the placement of files and generating parity information, and then passes the data to the disk device driver.

In this article I am assuming you already have a good grasp of the various RAID formats, and I won't detail them.

Stripe sets are created, in part, for performance reasons. Access to the data on a stripe set is usually faster than access to the same data would be on a single disk, because the I/O load is spread across more than one disk. Therefore, Windows NT can be doing seeks on more than one disk at the same time, and can even have simultaneous reads or writes occurring.

Stripe sets work well in the following environments:

1. When users need rapid access to large databases or other data structures.
2. Storing program images, DLLs or run-time libraries for rapid loading.
3. Applications using asynchronous multi-threaded I/O's.

Stripe sets are not well suited in the following situations:

1. When programs make requests for small amounts of sequentially located data. For example, if a program requests 8K at a time, it might take eight separate I/O requests to read or write all the data in a 64K strip, which is not a very good use of this storage mechanism.
2. When programs make synchronous random requests for small amounts of data. This causes I/O bottlenecks because each request requires a separate seek operation. 16-bit single-threaded programs are very prone to this problem.

It is quite obvious that RAID can exploit a well written application that can take advantage of asynchronous multi-threaded I/O techniques. Physical members in the RAID environment are not read or written to directly by an application. Even the Windows NT file system sees it as one single "virtual" drive. This virtual drive has logical cluster numbering just like any other partition supported under Windows NT. As an application reads and writes to this virtual environment (creating new files, extending existing ones, as well as deleting others) the files become fragmented. Because of this fact, fragmentation on this virtual drive WILL HAVE a substantial negative performance effect. When an I/O request is processed by the file system, there are a number of attributes that must be checked which cost valuable system time. If an application has to issue multiple "unnecessary" I/O requests, as in the case of fragmentation, not only is the processor kept busier than needed, but once the I/O request has been issued, the RAID hardware/software must process it and determine which physical member to direct the I/O request. Multiple I/O's at this level will result in multiple head movements of the disks in the array. In fact, this fragmentation can negate any and all benefits of having RAID in the first place as these unnecessary fragmented I/O requests take up extra bandwidth causing the RAID to be less and less effective.

DEFRAGMENTING RAID

By Mark E. Donaldson

So the question now becomes how does a defragmenter affect this? The defragmenter sees the RAID environment just as the file system does. That is, Diskkeeper defragments the "virtual" drive. Diskkeeper improves the speed and performance of a RAID environment by eliminating these wasteful and unnecessary I/Os from being issued by the file system. This occurs because the file system sees the files and free space as being more contiguous. The file system will spend less time checking file attributes which means more processor time can be dedicated to doing real useful work for the user/application. In addition, these I/O requests are now much more likely to fill the entire 64K chunk (RAID stripe) size with the I/O now taking full advantage of the RAID.

Next I will use an example to explain the technical information above.

As we just covered, if a given file is fragmented on the logical/virtual drive, requesting that file requires the OS to use additional I/O's for every separate fragment. These I/O requests are passed to the physical disk.

Let's take an example of an Excel spreadsheet in 100 pieces. A single physical disk would now have to perform 100 disk I/O's to retrieve it (plus some additional overhead I/O's such as reading the file record, reading in directories, etc...). Well, what if that "physical disk" was actually a stripe set of 5 disks (for simplicities sake I'll use RAID 0). The Raid controller receives the I/O's and intersperses them equally between the disks (1/5th of the file across the disks). If I was explaining RAID 5, one of the disks in each "stripe" would be reserved for parity information. Now each physical disk in our array has to do 20 I/O's to retrieve the file. Not as bad as 100 but still 20.

Now how a defragmenter affects this: Let's first go back to that individual disk. Diskkeeper has defragged the "logical file" to one piece. Accessing that file now takes 1 disk I/O. Expanding that to our 5 disk RAID set: the controller intercepts the I/O and intersperses it equally across the array. Now, each physical disk has again 1/5th of the file to retrieve, but must only perform 1 disk I/O, instead of 20!

A defragmenter (at least those that use the API's) never concern themselves with the physical storage architectures underneath it, nor does the file system. RAID controllers and device drivers are responsible for this. Whether you are using striping with parity, mirroring or combinations of RAID, makes no difference. Diskkeeper never forces files to a particular physical disk location. This is again, the job of the driver.