Block and Inode Allocation Policy

ext4 recognizes (better than ext3, anyway) that data locality is generally a desirably quality of a filesystem. On a spinning disk, keeping related blocks near each other reduces the amount of movement that the head actuator and disk must perform to access a data block, thus speeding up disk IO. On an SSD there of course are no moving parts, but locality can increase the size of each transfer request while reducing the total number of requests. This locality may also have the effect of concentrating writes on a single erase block, which can speed up file rewrites significantly. Therefore, it is useful to reduce fragmentation whenever possible.

The first tool that ext4 uses to combat fragmentation is the multi-block allocator. When a file is first created, the block allocator speculatively allocates 8KiB of disk space to the file on the assumption that the space will get written soon. When the file is closed, the unused speculative allocations are of course freed, but if the speculation is correct (typically the case for full writes of small files) then the file data gets written out in a single multi-block extent. A second related trick that ext4 uses is delayed allocation. Under this scheme, when a file needs more blocks to absorb file writes, the filesystem defers deciding the exact placement on the disk until all the dirty buffers are being written out to disk. By not committing to a particular placement until it's absolutely necessary (the commit timeout is hit, or sync() is called, or the kernel runs out of memory), the hope is that the filesystem can make better location decisions.

The third trick that ext4 (and ext3) uses is that it tries to keep a file's data blocks in the same block group as its inode. This cuts down on the seek penalty when the filesystem first has to read a file's inode to learn where the file's data blocks live and then seek over to the file's data blocks to begin I/O operations.

The fourth trick is that all the inodes in a directory are placed in the same block group as the directory, when feasible. The working assumption here is that all the files in a directory might be related, therefore it is useful to try to keep them all together.

The fifth trick is that the disk volume is cut up into 128MB block groups; these mini-containers are used as outlined above to try to maintain data locality. However, there is a deliberate quirk -- when a directory is created in the root directory, the inode allocator scans the block groups and puts that directory into the least heavily loaded block group that it can find. This encourages directories to spread out over a disk; as the top-level directory/file blobs fill up one block group, the allocators simply move on to the next block group. Allegedly this scheme evens out the loading on the block groups, though the author suspects that the directories which are so unlucky as to land towards the end of a spinning drive get a raw deal performance-wise.

Of course if all of these mechanisms fail, one can always use e4defrag to defragment files.