

Monday, February 8, 2010

To dedup or not to dedup - that results in a lot of questions

In some discussions in mail and in some community forums linking to the articles about deduplication and hashing there was a slight misunderstanding. I should explain some things.

There are two major ways to deduplicate: Synchronous and asynchronous. The synchronous variant does the deduplication while writing to the disks, so duplicates aren't written to disk, the asynchronous variant writes every block to the disk and later on it deletes duplicated blocks by searching possible candidates.

How do you find candidates: There is the brute force method: You take a block, compare it with each other block on the disk. Obviously this isn't efficient. I think early dedup versions did it this way. So you need a better way.

There are mathematical algorithms allowing you to condense a large set of data to a small set of data. Those algorithm are called hash algorithms. You calculate them for every block, put them in a table. When you write a new block, you calculate the hash, look if there is already a block with the checksum and so you have a candidate block for deduplication.

Then there are two other variants of this hash-based algorithms: Hash-only and Hash-and-compare. Hash-only is based on the assumption, that with a hash function with a low probability of hash collision, you can assume, that two blocks with equal hashes are indeed equal and thus deduplicatable. Hash-and-compare doesn't trust the hash and compares possible candidate for deduplication with the block that should end in a pointer instead of a entirely written block. Only in the case of proven duplicity it gets deduplicated.

There are many design decisions, that leads to the complete solution to implement deduplication and the the foundation may have their foundation in the past of an development.

The decision synchronous/asynchronous is made on the foundation of the time to make decision to dedup or not to dedup. If you can make this decision in a very short time, you can do it synchronously. If the decision is resource-intensive and/or takes a long time, you do it asynchronously.

The needed time for deduplication is the to find a candidate and to check it's duplicity. The brute force method would take a long time and would be implemented in an asynchronous way.

With the hashing method it's a little bit more complex: With modern CPU you can safely ignore the time to compute the hashes. More interesting than the CPU is the memory. Finding candidates for deduplication is done with tables. When the database fits in memory it's extremely fast, if you have to access your disks, it get's slower. (This is the reason why ZFS deduplication is speeded up by SSD L2ARC by a large margin with large filesystems). But let's assume you have enough memory.

So the hash gets important: Finding candidates is fast with any hash algorithm. So this phase is irrelevant for your design decision if you use synchronous or asynchronous deduplication. The second phase - checking for duplicity - is the interesting one.

When you have a weak checksum, you will find many possible candidates as each hash bucket is filled with many blocks. But only one is a real duplicate. So you have to read many blocks to check for this real duplicate. This takes time and eats away your precious resource IOPS. So you do it asynchronously in times where you have some spare IOPS.

When you have a strong checksum, the possibility is high, that a deduplication candidate is really a duplicate. The stronger the checksum, the higher the probability. When you do asynchronous deduplication with, you may find several candidates, but with strong checksums the probability is high, that's block have in fact the same content. With synchronous deduplication you will find just one, as other duplicates were deduplicated before. (aside of the case of an hash collision, but i wrote already about the probabilities of such an occurrence).

The advantage of the strong checksum is that it gets viable to do deduplication while writing. You have two possibilities here, too: You trust the checksum, and don't do a comparative read, or you read the block you want to point to, make a bit-wise comparision, and when it's a real duplicate you just store a pointer.

With this comparative read the probability of a false duplicate isn't zero. It's the probability that a undetected error occurs on the way from the rotating rust to the cpu register in the process of reading the block, resulting in a byte-for-byte comparison that signals duplicity, although it was just a read error. But we starting to split hairs ...

Asynchronous and Synchronous deduplication can be combined freely with hash-only and hash-and-compare deduplication. Reusing weak hashes, which were introduced as checksums only, mandate asynchronous (because of potentially many dedup candidate) hash-and-compare (because of high probability of hash collisions) checksums.

ZFS dedup is strong checksum (256-bit) synchronous deduplication with selectable hash-only and hash-and-compare modes of operation. So if your are in fear of hash collisions, you can switch on the hash-and-compare mode of operation. Given the low probability of a hash collision with current storage sizes it's a viable option to forget about the compare run.

NetApp dedup is weak checksum (16-bit) asynchronous deduplication with mandatory hash-and-compare mode of operation. Why mandatory? Let's just look in the math:

So when even it's okay for you have a one percent probability of having a false positive, you would be just able to dedup a 36 blocks device with a 16 bit checksum. There is no way to work with hash-only dedup when your hash is just 16 bit.

Posted by Joerg Moellenkamp in English, Solaris, Sun/Oracle, Technology at 20:42

Let me dedup first 2 sentences of this post
Anonymous on Feb 9 2010, 07:25

Did somebody try to compare the difference in power consumption (if there is some) on non-ZFS-deduped / ZFS-deduped systems? Simply, if it's not more (power/storage_response_time) efficient to add extra HDD into zpool instead, forget about "Do I have enough spare IOPs?", and simply go and take the data directly from the HDD. And yes, it should depend on the stored content (dedup percentage vs. storage capacity vs. etc.). Thanks.
Anonymous on Feb 9 2010, 08:40

ZFS computes the checksums anyway. The difference with hash-only dedup is just the lookup to a table, with hash-and-compare you need an additional IOPS, but that isn't much of a problem as i explain later.

For reads it makes no difference, if the data is deduplicated or not. As deduplicated data is only cached once for potentially many blocks, the cache in storage arrays will be used more efficiently, thus potentially resulting in a lower IOPS count getting to the disks

The spare IOPs problem isn't a problem with ZFS hash-only, a ignorable one for ZFS hash-and-compare (you would do the IOPS without dedup anyway, you have a read io instead of a write io, and just in the case of a false positive you have to issue a write io, but the probability of that is pretty low, you can't lose, but you can win a lot) and a big one for weak-checksum dedup, as you have to the problem to check many dedup candidates.
Anonymous on Feb 9 2010, 09:32

Interesting read and it inspired me to check upon the dedupe features in TSM6.1. Seems that it uses SHA-1, non-compare.

However, regarding the performance impact of synchronous dedupe: Even if the hash table is in memory, shouldn't every alteration be flushed to persistant storage(disk or NVRAM)? If that is the case my immediate thought is that synchronous dedupe may come at a significant performance penalty or at least would require a lot extra of NVRAM and computing power.
Anonymous on Feb 9 2010, 11:25

Dedup your brain!
Anonymous on Feb 9 2010, 11:40

Is there anything this comment should tell me ?
Anonymous on Feb 9 2010, 11:44

Do you mean Andy or Iparvirt? I guess Iparvirt. So here is what I think: Maybe Iparvirt oversaw that ZFS is able to leverage SSDs for L2ARC and therefore the hastable is already in nonvolatile Read Cache. What speeds up the dedupe is the performance on querying

Blog Export: c0t0d0s0.org, <http://www.c0t0d0s0.org/>

the hashtable not writing it. So having a copy of the hashtable in L2ARC should give you viable performance speedup.
Anonymous on Feb 9 2010, 12:11

no ... this was a response to andys comment ...
Anonymous on Feb 9 2010, 12:50