activeimage

Deduplication: pros, cons, what to look for and what to expect from it.

Today we attempt to take a closer (or maybe higher) look at Deduplication, in particular – at Data Deduplication.

We will not try to get down to the molecular level. The main intention of this article is to shed some light on deduplication-relevant pros and cons and to broaden view of this topic.

In computing, the simplest and the most effective definition (by Wikipedia) of Data Deduplication goes as "a specialized data compression technique for eliminating duplicate copies of repeating data".

So, if you poke around the Internet long enough – you see various examples and analogies which would try to help us to understand what's going on.
One of the examples would compare a book library with a concise deduplicated collection made out of it.
In this deduplicated collection each word would be stored only once with a lot of references made to all places where in the books this word is actually placed.

Another example would compare a house with a set of elements from which this house is built.
In the deduplicated set we would only need x1 instance of each element (brick, hinge, door nub, roof-tile, window's frame etc.) with a precise list of references where each element goes to (blueprint of the house).

In both above cases we deal with two main portions of deduplicated data –

1. the main collection of unique items ("one of each") and
2. some sort of a list which has all references recorded, i.e. what element of data belongs to what spot in the data set itself ("index" of some kind).

In real live it would also be cool if a size of the index turned to be relatively small compared to a size of unique items collection and a size of the source data.

So, in the book library example, if we changed our design and decided to store only a unique letter in our "unique items collection" – then we would get only the alphabet stored there. But the index list would grow as big as the book library itself.

Not quite practical. The main reason why Deduplication exists is to reduce a storage space consumed.

In binary-based computing, the same problem would be if we decide to store only 1 and 0 into our unique items collection. A size of corresponding index will be times bigger than a size of a source data.

Any way you think about it – the Deduplication design deals with a standard set of challenges:

1. What size of data block to consider as potential "unique block"?
2. What size of index will we get and will it be acceptable?
3. What reduction in size of the data itself will it give us?
4. What resources will be needed for this data transformation?

In conjunction with "Backup and Recovery" topic Deduplication is demanded for the same purpose: reduction of storage consumption.
However, this is not the only requirement, and perhaps not the most important one. The requirement #1 in this case is:

5. **How quickly can we retrieve the stored data or any portions of it, what resources do we need for that?**

…because the only sane purpose of producing and storing backups is ability to use the stored data when it's needed, where it's needed and, in a form, which is needed.
Naturally, on a way of retrieving data from deduplicated collection we need to reconstruct it back to its original form, re-instantiating all references with the corresponding blocks of data. This process is often called "rehydration".
How quickly? Depends on our design.

Deduplication is classified in many different categories:

- **Software-based vs. hardware-based** – "who is responsible for executing deduplication algorithm?".
  Hardware deduplication is not uncommon these days, works well… until you run out of space and hit the specs limits of that particular hardware piece. It tends to be priced higher too.

- **Source-side vs. target-side** (destination) vs. both – where exactly deduplication algorithm is executed – on the protected machine, on the backup-data receiving side, on both.
  This will affect the corresponding side's resources (CPU, memory, network bandwidth). This might also cause a storage consumption spikes occasionally (with target-side deduplication, when backup data has already arrived but had not been processed yet).
  Target-side deduplication had been around for quite a while and up until recent times had been considered as "industry standard". Why? Due to the most of pure-source-side algorithms been inefficient or overly resource-consuming.

  Don't be surprised to read the typical target-side machine's specs: yes, this machine has to be Godzilla of computing, capable of not only crunching large amounts of data but perhaps tons of bricks, steel rails and pipes and who knows what else…

A mixed approach – source and target side deduplication combined – had been observed to prevail in recent years. This by itself indicated the main intention with a time to get rid of a heavily-specked receiving backend.

Finally within last couple of years we see pure-source-side algorithms capable of doing it effectively which lets us think the future is bright 😊

- **Global vs. single-source** – how many protected machines participate in forming a unique blocks collection.

  This will affect a reported ratio number - reduction in storage size consumption. Global deduplication sounds the way to go with, many people enjoy it for a while…until the first corruption occurs. Then many of them have been forced to recreate the entire global repository from scratch.

  Just because the corrupted block of data had been associated with hundreds of protected machines. Oops!!! "when good Global goes bad".

  Even if everything works as expected – it might not be too easy to retrieve the data in timely manner, especially when the system is busy with accepting and accommodating several incoming streams of backups. And it often happens when you need this data now, immediately, for Disaster Recovery.

  Many vendors offer some smarts to replicate the data from one site to another. If data resides in deduplicated repository – it will be rehydrated at the first step, then compared with the destination, then the changes will be calculated and sent over. The same massive resource consumption is expected.

  Single-source on the other hand is portable, easily retrieved, the dataset can be copied/pasted to an external media and sent over to a site where a compromised machine is in need of DR. The smarts required: to be able to use left and right mouse buttons. Replication between sites is easy too with single-source – the same data gets replicated as the one collected from the last backup. No transformation, no calculation, no overhead.

- **Content-agnostic vs. content-aware** – this assumes an absence or existence of some intimate knowledge of certain file formats by the software which performs deduplication. Such knowledge might increase storage savings but also decreases reliability of recovery process, just another smart component to rely on to rehydrate the data.

- **Inline vs. post-process** – just a variation of the question "when does deduplication happen?". **Inline** deduplication is designed to handle data before it's sent to a destination. By doing so the process decreases amount of data being sent, also decreasing resource consumption by the receiving end. If the algorithm is implemented well – the source machine's resources will not be consumed excessively. **Post-process** concept is often prerogative of old-fashioned "industry-standard" approach, also can be seeing with hardware-based deduplication. Here the data is sent to the destination as-is, compressed in the best-case scenario. Then the receiving side starts its magic. This approach may impact

network bandwidth, requires more storage available on the receiving side, requires uncompromised specs of the hardware to sustain a “data-crunching feast”.

The last few words are to cover what to expect in sense of reduction of storage consumtion.

1. The deduplication ratio greatly depends on a source data type:

DATA TYPE	MAX Expected Ratio
Unified virtual environment, core VM’s system	40:1
File & print server(s)	30:1
MS Exchange, SAP HANA	20:1
Oracle RMAN	14:1
CAD/Video/Medical	10:1
Lotus Notes	9:1
TSM	4:1
SQL/Oracle transaction logs	1.5:1
Encrypted data (any)	1:1

2. Dissimilar data types (mix between any of the above listed types) pointed to the same Global pool will not increase but rather decrease the ratio.

3. Deduplication ratio after a single backup will resemble to the one achieved with a plain compression. The longer backups have been pointed to the same deduplicated collection – the better ration becomes. The ratio even with a single source increases dramatically already with the second backup. Hence deduplication shines with a long-term retention policy.

4. Pay attention to how the ratio is conveyed to you. “10:1” can be described as “90% of savings”. Both are accurate, which one wins a deal?

Conclusion:
10 years from now someone will find this article and will laugh and laugh and laugh…
Because 10 years from now we will probably use much more efficient storage media, something like… I don’t know, maybe some graphene-based, or some other kind of carbon-based, like coal or something…

And who knows, maybe the history will repeat itself and the next generations and civilizations after ours will find large deposits of coal under ground … and will heat their dwellings with it …
I wonder what was stored on a coal which we burn today? Perhaps nothing important, some deduplicated data with “zillions to1” ratio 😊

Quiz (if you read the above material):

- *Can Noah’s life-hack with his Ark be considered as deduplication? Why?*
- *What deduplication ratio Singularity had? Where the heck did they hide the index?*