Immutability Changes Everything

A Subset Of Pat Hellands Work

Roland Gritzer, Andreas Waida

June 1, 2016

1 Introduction

In this subset of Pat Hellands article "Immutability Changes Everything"[3], we are reflecting on immutable data, based on his thoughts. Not only trying to pin down what immutable data actually is, we rather give a short insight of how immutable data is developing and in how far it may influence the future.

There are a few key words, we might have to give a closer look at for this reason. What has immutable data got to do with "Big Data"? What does Helland mean by "Inside Data" and "Outside Data"? How is immutability related to common hardware components? Let's find out about its benefits and its disadvantages.

2 Appending - That's it, That's all

Helland compares immutable data principle with the book of an accountant. He never deletes entries. Corrections are new entries. Based on the entries the new balance is calculated. Transaction logs in DBMS can be seen as an immutable base and the log is just a cache for the actual data. "The truth is the log"[3]

There is a variety of applications of the *Append-only* principle, where changes are logged and results are calculated on request. In Append-only systems, changes have to be serialized.

Back in the days, people used multi-layered forms where each party had to fill in a section, sign the form and tear off one layer. Each layer is an immutable version of the form. You can only append data to it[3]. This example shows the main concept of a distributed Append-only system.

3 It's always nice to play Outside

Also referring to one of Helland's other articles "Data on the inside vs. Data on the outside"[2], he, once again outlines the distinction between data being held in classical

relational databases and data being dealt with in form of messages, files or web pages. On the basis of outside data's immutability, he exposes its advantages.

According to him, "immutable data is the same no matter when it is referenced and no matter where it is referenced."[2] Whereas inside data is called by its value, every time outside data is created, it obtains a unique identifier and has it when sent outside. As a result data may get lost, but there is no chance of referencing the wrong set of data.

Of course, it has to be ensured that data does not get lost. Therefore Helland reflects on the model of retrying messages[2]. Again immutability plays into our hands as data is not supposed to change for the retry.

4 Turning the Inside out?

Before presenting two concepts of how to combine immutable data with classical relational DBMS, Helland gives an introduction to what he calls an immutable DataSet. By creating a DataSet with its unique ID as "a fixed and immutable set of tables", together with a schema describing those tables, it is already provided with information that is not changeable from then on.

Getting back to the idea of each immutable DataSet having a unique ID, we could consider relational databases as holders of references to immutable DataSets with their meta-data being visible to the DBMS. As the treated data is not to be changed, there is no worrying about locking the data itself at all. It will always be the up-to-date version being referenced.

Helland also presents a concept of performing JOINs on tables of relational DBMSs and immutable DataSets. He talks about a "tailoring" of the schemas of both, the one "describing its data when written" [3] and the other "describing its data as of the snapshot" [3]. "This sidesteps the notion of identity within the DataSet and focuses exclusively on the tables as interpreted as a set of values held within rows and columns."[3]

5 DataSet Consumption

From the consumers point of view these DataSets appear to be immutable, even if the data is changed locally to optimize read access. You can for example store a value in two places to achieve faster access this does not collide with the immutable nature of the DataSets.

Immutability is also essential for parallel computations in Big Data applications. The input has to be immutable to make the functional calculations idempotent.

In a relational database, the information about what is the content of our table is stored as *prescriptive meta-data* (SQL-DDL), which may be changed for updates. Immutable DataSets in contrast store *descriptive meta-data*, unchangeable and just showing what's there.

Another positive aspect is that "*Normalization is not necessary in an immutable DataSet.*" [3] Denormalized DataSets may also be faster to process. If you want to

save storage space you can normalize DataSets but you may notice longer access times.

6 Versioning

Take a look at Versions, they are immutable. The main concept is that every version has its unique ID and one version is replaced by the other. Each has only one father/child.

In extension, you may have version history in form of a directed acyclic graph, where one version may have multiple children to allow concurrent access. Every version is stored in a Key-Value store as a change of the earlier version. This allows easy traveling through versions.

Log Structured Merge trees (LSM) is a data structure which provides low-cost indexing while expecting frequently added records based on immutable files using copy-on-write techniques[1].

7 Log Structured File Systems

Log Structured File Systems were first mentioned in 1992 [6] but they are getting more and more important for nowadays' calculations.

GFS (Google File System) [2]. and HDFS (Headoop File Systems) [3]. are similar file systems created for big immutable files being highly available in a clustered data center. Files are replicated multiple times to ensure availability in case of the failure of a node.

These file systems provide a filename to the user that may be changed. But the global unique ID of the unchangeable data behind it will not change. Consistent hash ring techniques are used to find files by their name. Again there are a number of advantages, brought to us by immutable data.

8 And Hardware Goes Along

The resounding growth of immutability also affects hardware areas.

There is a system known as wear leveling [4] that is meant to extend the operational life span of SSDs. As a result of this mechanism, once a block is used, it won't be overwritten again until every other block of the chip was used ("in a circular fashion"[3]). Hence the data written to a block appears to be immutable as it is not changed until it is deleted/overwritten again.

Current hard drive architectures often use a system called "Shingled Disk Systems"[1]. Similar to a roof, where each shindle is topping the one put down before, the data is written to the band of the hard drive. By accessing some middle shindle, the rest would break. By using log-structured file systems implemented directly on the disk controller[5], the user is unaware of the shingles though. Thats why the data remains unchanged and thus complies with immutability.

[1] `https://en.wikipedia.org/w/index.php?title=Copy-on-write&oldid=721800011`
[2] `https://en.wikipedia.org/w/index.php?title=Google_File_System&oldid=720811688`
[3] `https://en.wikipedia.org/w/index.php?title=Apache_Hadoop&oldid=721269142`

9 Tripwires that need a Skipping

As many advantages immutability may have, there are certain issues to it that have to be mentioned. Copy-on-write mechanisms, of course, consume a lot of storage. So we gain on efficiency for an expense of storage cost. Although " immutable data has more choices for its representation. We can normalize for space optimization or denormalize for read usage." [3]

SSD developers are also confronted with an issue related to as write amplification[4]. Writing new data results in reading and rewriting a lot more storage than the new data actually requires.

10 Conclusion

All in all, we can say that even if there are a few drawbacks such as cost of storage, that there are ways to clear the way from obstacles for immutable data. For example, the price for more storage on smaller room declining.

"Versioning gives us a changing view of things while the underlying data is expressed with new contents bound to a unique identifier."[3] Losing a previous version will be a thing of the past and immutability will simplify replication. Also, creating hybrids out of classical relational DBMSs and systems using immutable DataSets opens up great new possibilities.

For Helland, and we totally agree, immutability rather is an unstoppable phenomenon than something that has to be purposely driven forward.

References

[1] Garth Gibson and Greg Ganger. Principles of operation for shingled disk devices, 2011.

[2] Pat Helland. Data on the outside versus data on the inside. In *CIDR*, pages 144–153, 2005.

[3] Pat Helland. Immutability changes everything. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[4] K.M.J. Lofgren, R.D. Norman, G.B. Thelin, and A. Gupta. Wear leveling techniques for flash eeprom systems, February 1 2005. US Patent 6,850,443.

[5] R.M.H. New and M.L. Williams. Log-structured file system for disk drives with shingled writing, August 9 2011. US Patent 7,996,645.

[6] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.*, 10(1):26–52, 1992.

[4] `https://en.wikipedia.org/w/index.php?title=Document-oriented_database&oldid=719568689`