Milliways: the storage at the back-end of the Multiverse

Marco Pantaleoni, J Cube Inc.
Bo Zhou, J Cube Inc.
Paolo Berto Durante, J Cube Inc.
Aghiles Kheffache, The 3Delight Team
Douglas Adams

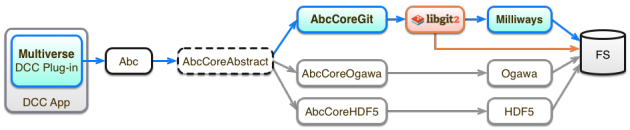

Figure 1: *Multiverse with Milliways architecture.*

1 Abstract and Overview

When we intrduced [JCube 2015], we extended the popular *Alembic* [ILM and SPI 2009] file format (a data representation scheme for storing graphics scenes) adding *Git* distributed version control capabilities.

Git, however, was conceived with textual source code in mind, performing suboptimally with very large numbers of files, especially binaries. Since it's common for *Alembic* scenes to contain hundreds of thousands of properties/samples, it's easy to suffer from heavy filesystem overhead[1]. We explored alternatives to store *Git* objects to circumvent filesystem degrading performances, using the LIBGIT2 [libgit2 2009] API to change *Git* storage back-end.

Existing ones turned to be inadequate, therefore we implemented *Milliways*, a general purpose storage mechanism, and integrated it in *Multiverse* as a new LIBGIT2 back-end, proving to be much faster and almost as space efficient. Our solution is not restricted to *Alembic* or even *Git* but has much wider applicability in IT.

2 Duties of a *Git* back-end

In *Git* the vast majority of data is stored inside the "object database". Every "git object" is characterized by its type (eg. `commit` or `blob`), by its size in bytes and by its contents. Each object is uniquely identified by the SHA-1 hash of the contents (plus header). So the object database is essentially a *key-value store*, supporting the operations *list*, *get*, *put*, *delete*.

3 Existing technologies

At first we tried two storage solutions as LIBGIT2 back-ends: *Memcached* and *SQLite*. *Memcached* showed a substantial speed-up, but was unsuitable in practice being based in RAM. *SQLite* was slower perhaps due to its highly generic nature: it's among the fastest SQL database systems, but couldn't compete with traditional filesystems for raw data storage. Other popular fast key-value stores either had too stringent limits for the size of values or were limited to RAM or both.

4 Our Solution

Being the *Git* object database a key-value store, we wrote *Milliways*, a fast disk-based key-value store. We wrote it as a header-only C++ library, so that it can be easily integrated in other projects.

[1] This is particularly evident for the Windows filesystem

The underlying data structures are *B+-trees*, balanced search trees particulary suitable for disk storage, and, because all code is inlined and there is no metadata, we end up to be faster than the filesystem.

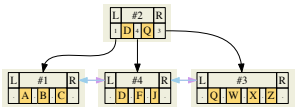

Figure 2: *Example of a B+-tree of degree 3.*

Milliways doesn't impose arbitrary limits on the number of the elements or the size of the values (currently set to 4GiB per value - can be brought to 256 TiB). Large values are transparently compressed with LZ4 [Collet 2016]. The storage is architecture-independend, and resides in a single file, so can easily be moved across machines and there are no filesystem overheads related to multiple files. *Milliways* is *fast*: on an old 2.3GHz i5 (with SSD) can write between 350k and 600k words/s[2].

Milliways is standalone and open source: it can be used in any project that needs portable, fast, efficient storage of key-value pairs.

4.1 Results

Milliways improves *Multiverse* speed usually in the range $\approx$ 30% – 800%[3]. In terms of space *Milliways* needs $\sim$ ⅕ – ¼ more space than "plain" *Multiverse* (will be further optimized shortly).

Table 1: I/O *Write comparison for Alembic backends (Time–Size). Further tests and results are available at* `http://multi-verse.io/tech/tests`

Scene Type	*Ogawa*	*Git*	***Git Milliways***
Anim (deform)	12s–809MB	42s–413MB	18s–512MB
Deep hierarchy	44s–805MB	198s–432MB	120s–570MB
Dense mesh	6s–816MB	34s–398MB	12s–577MB

5 Conclusion and Future Work

Milliways provides a huge improvement to *Multiverse* performance and resulted in a winning choice. We plan to continue improving it, implementing more efficient block allocation, bindings for languages and a companion server. Make it a solid open source standalone technology. Further optimizations are also possible on the *Multiverse* side: in the future we plan to work on providing a better integration of the two, improving both space efficiency and speed.

References

COLLET, Y., 2016. Lz4. https://cyan4973.github.io/lz4.

ILM AND SPI. 2009. Alembic format description.

JCUBE. 2015. Multiverse: Next generation storage for alembic.

LIBGIT2. 2009. libgit2 description.

[2] `dict` words, speed depending on options and compression
[3] speed improvement is highly dependent on the scene