COMTILES: A CASE STUDY OF A CLOUD OPTIMIZED TILE ARCHIVE FORMAT FOR DEPLOYING PLANET-SCALE TILSETS IN THE CLOUD

M. Tremmel

Bahnhofstraße 31, 9429 Bodenmais, Germany - markus.tremmel23@gmail.com

KEY WORDS: Cloud Optimized File Format, Cloud Optimized Tile Archive, Cloud Storage, Web Map, Tiles, Cloud-native Geospatial.

ABSTRACT:

The container formats commonly used for managing map tiles, such as MBTiles and GeoPackage, were originally designed with only POSIX filesystem access in mind. This makes these file formats inefficient to use in a cloud native environment, especially in combination with large tilesets. The Cloud Optimized GeoTIFF format solves the problem of providing large satellite data in the cloud, creating a new category of so-called cloud optimized data formats. This type of format allows geospatial data to be deployed as a single file on a cheap and scalable cloud object storage such as AWS S3 and accessed directly from a browser without the need for a dedicated backend. Based on the concepts of the COG format, this contribution proposes a new cloud optimized file format called COMTiles, specially designed for planet-scale tilesets. This format has the potential to simplify the deployment workflow of large tilesets in a cloud-native environment, while simultaneously reducing the hosting costs. In comparison to PMTiles, another cloud-optimized tile archive solution, COMTiles can reduce the number of transferred data and the performance of decoding portions of the file. COMTiles also adds support for different coordinate systems.

1. INTRODUCTION

The state-of-the-art container formats for managing map tiles are the Mapbox MBTiles specification (Mapbox, 2018) and the OGC GeoPackage standard (Open Geospatial Consortium, 2021). These formats enable the storage of large tilesets containing hundreds of millions of map tiles in a single file. When used in conjunction with a tileserver, map tiles can be requested over a network or directly accessed by client applications. Since both formats are based on an SQLite database, they are mainly designed for a block-oriented POSIX-conform file system access. This design approach makes these file formats inefficient to use in a cloud native environment, especially in combination with large tilesets. To deploy a MBTiles database in the cloud, the tiles must be extracted and either uploaded individually to an object storage or imported in a cloud database and accessed by an additional dedicated tileserver. The main disadvantages of both options are the complex workflow for the deployment and the expensive hosting costs. The Cloud Optimized GeoTIFF (COG) format already solves the problem for providing large satellite data in the cloud, creating a new category of so-called cloud optimized data formats (Cloud Optimized GeoTIFF, 2018). This type of format allows geospatial data to be deployed as a single file on a cheap and scalable cloud object storage such as AWS S3 and accessed directly from a browser without the need for a dedicated backend.

Based on the concepts of the COG format, this contribution proposes a new cloud optimized file format called COMTiles, specially designed for planet-scale tilesets. This format is designed as a streamable and read optimized single file archive format for storing large raster and in particular vector tilesets in the cloud. The COMTiles format has the potential to simplify the deployment workflow of large tilesets in a cloud-native environment, while simultaneously reducing the hosting costs. The proposed format aims to achieve this by minimizing the latency to preserve the user experience commonly associated with web maps, such as Google Maps. In addition, a novel tile request batching approach is presented, which can significantly reduce the number of tile requests. This enables the deployment of a planet-scale OSM tileset with 90 gigabytes in size on a Cloudflare R2 storage and access to approximately 35 million tiles of this dataset through a browser at a remarkably low cost of only $1.35 per month.

To evaluate the efficiency of the design, this work compares COMTiles with another cloud native tileset format PMTiles for planet-scale tilesets based on different metrics (Liu, 2022). The evaluation shows that COMTiles can reduce the number of transferred data and the performance of decoding portions of the file by accepting a larger total index size. COMTiles also adds support for different coordinate systems.

2. CLOUD OPTIMIZED GEOSPATIAL FORMATS

All major cloud platforms provide a object storage like AWS S3, Azure Blob Storage or Cloudflare R2 as the cheapest and most scalable way to store large amount of data. At the time of writing, 100 GB of data is charged between $1 and $5 per month, depending on the provider. However, the problem with most geospatial data formats like MBTiles or Shapefiles is that they were developed before cloud object storage was prevalent. These files are designed for the usage in local disks and filesystems which offer low latency. Compared to local filesystems an object storage service rely on HTTP requests with a relatively high latency (100 ms or more) (Abernathey et al., 2021). Therefore, the internal structure of a file has must be specifically designed to be efficiently consumed in networked environments. The first widely adapted geospatial format that could be efficiently used in a cloud-native environment was the Cloud Optimized GeoTIFF format. Based on the ideas of this format, new cloud optimized geospatial formats, often built on top of existing formats, have been created for various use cases. These formats are based on and make use of the following cloud native geospatial concepts:

Category	Formats
Raster	Cloud Optimized GeoTiff (COG)
Point Clouds	Cloud Optimized Point Cloud (COPC)
N-dimensional Arrays	Zarr, TileDB
Vector	FlatGeobuf, GeoParquet
Tile Archive	PMTiles, COMTiles

Table 1. Categories of cloud optimized formats for storing geospatial data.

- Cheap and scalable cloud object storage for storing massive datasets
- HTTP GET range requests (IETF RFC7233) for partial reads
- Spatial index for referencing parts of the file
- Metadata for describing the properties, content and structure of the file
- Read optimized-access

Based on the metadata and the spatial index, clients can request portions of a file via HTTP GET range requests. The index indicates the location of specific chunks of data within the file and allows random reads. Since the index for massive datasets is too large to be fully downloaded with low latency, the index must also be streamable. Cloud optimized formats must be designed in a way that minimizes the number of HTTP requests for requesting parts of the index. This reduces both latency and costs, since each request and the data size transferred is billed by the cloud providers. To achieve this, most cloud optimized formats are optimized for read access, which makes updates to the file expensive compared to transactions in a database.

As shown in Table 1, there are several types of cloud optimized formats for the different use cases within the geospatial domain. The previously mentioned COG format focuses on storing and serving georeferenced raster images on the web. In addition, also specialized formats for point clouds like Cloud Optimized Point Clouds and for multidimensional arrays like Zarr or TileDB exists. Formats intended for storing large vector datasets in the cloud are GeoParquet and FlatGeobuf. GeoParquet with its column-oriented layout has a special focus on tabular data and can be used for analytical workflows in cloud data warehouses like BigQuery or Snowflake (GeoParquet, 2022). FlatGeobuf is a performant binary encoding for spatial vector data based on flatbuffers that can hold a collection of Simple Features (Harrtell, 2021). In contrast to GeoParquet, a spatial index for fast spatial bounding box filtering based on a packed Hilbert R-Tree can be part of a FlateGeobuf file. Both formats are well suited for analytical processing and workflow of large vector datasets. However, both formats lack the concept of overviews for visualizing large basemaps such as the planet-scale OpenStreetMap dataset (Holmes, 2021). Overviews are crucial for web maps to allow a fluent user experience known from slippy map[1] applications like Google Maps when zooming across multiple scales.

As a result, since map tiles are based on the concepts of overviews, they are the most effective way to display large vector maps in the browser. To simplify the management of large basemaps in the cloud, the tiles can be stored in a cloud optimized tile archive. Besides COMTiles, there are other cloud-optimized formats for tilesets with different design approaches and trade-offs like PMTiles, TileBase and Cotar. Due to the broadest adaptation, the strong tool support and the most advanced concepts, the focus in the following is on PMTiles in version 3.

A PMTiles archive is divided into the following five main sections: header, root directory, JSON metadata, leaf directories and the tiled data. The root directory and leaf directories consists of a list of entries and are used to address the tiles in the archive. A directory entry is defined by a TileId, Offset, Length, and RunLength. A TileId corresponds to a cumulative position on the series of square Hilbert curves (Liu, 2022). To reduce the size of the archive, each directory is compressed using a combination of Run-length encoding (RLE), Delta coding and Varint encoding, as well as a general purpose compression such as Gzip. The compressions used lead to a remarkable reduction in the size of the index. As a result, the index of a planet-scale vector base map is only about 91 megabytes in size.

3. FILE LAYOUT

The COMTiles format is designed as a streamable and read optimized file archive for hosting map tiles at global scale on a cloud object storage. The basic concepts of COMTiles are based on the ideas of the Cloud Optimized GeoTIFF format and extended for the management of map tiles. As COMTiles is designed to be agnostic to the tile content, both raster and in particular vector tiles can be stored in the archive.

The following primary requirements are taken into consideration in the design of the format:

- Support of different coordinate reference systems (CRS)
- Minimize the transferred amount of data and the number of requests to reduce costs and latency
- Every tile in the archive can be requested with at most one additional request
- Fast decoding of the index

The file layout shown in Figure 1 was chosen to support the requirements. A COMTiles layout is basically divided into the following four sections: header, metadata, index and data

3.1 Metadata

The metadata section describes the properties and structure of a tileset encoded as a UTF-8 encoded JSON document, as shown in Figure 2. To support different predefined tiled coordinate reference systems, the metadata document is based on the OGC “OGC Two Dimensional Tile Matrix Set and Tile Set Metadata” specification. In this OGC specification, the basic concept for describing the structure of a tileset is a tile matrix set that is defined on top of a CRS. A fundamental part of the definition of tile matrix set is a tile matrix, which is composed of a collection of tile matrices. A tile matrix is associated to a specific scale and divides the space into regular conterminous tiles with a unique identifier (Open Geospatial Consortium, 2022). Based on the tile matrix limits, a limited coverage of a tile matrix can be defined. Different coordinate systems are supported

[1] https://wiki.openstreetmap.org/wiki/Slippy_map.

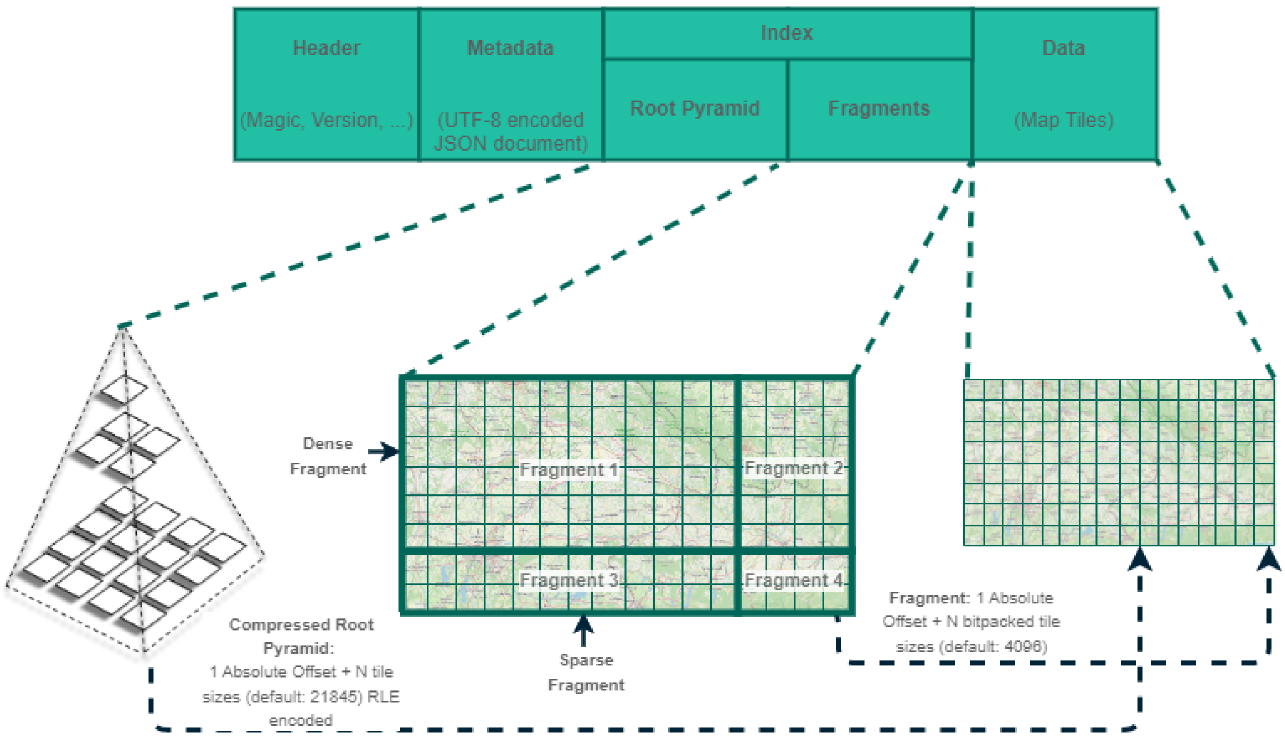
Header
(Magic, Version, ...)
Metadata
(UTF-8 encoded JSON document)
Index
Root Pyramid
Fragments
Data
(Map Tiles)
Dense Fragment
Fragment 1
Fragment 2
Fragment 3
Fragment 4
Sparse Fragment
Compressed Root Pyramid: 1 Absolute Offset + N tile sizes (default: 21845) RLE encoded
Fragment: 1 Absolute Offset + N bitpacked tile sizes (default: 4096)

Figure 1. Layout of a COMTiles archive.

based on the Common TileMatrixSet definitions contained in the OGC specification as informative annex. These definitions propose different tile matrix set definitions for Mercator, Transverse Mercator, Polar Stereographic, Lambert Azimuthal Equal Area, and Lambert Conformal Conic projections. In addition, there are also additional properties part of the metadata document such as the content of the tiles (raster, vector) or properties about the structure index. In combination with the file header, the metadata document can be used to define requests for portions of the index.

3.2 Index

The basic concept of the COMTiles format is to create an additional streamable index that stores the offset and size of the actual map tiles in the data section of the archive as so-called index entries. In combination with a metadata document, the index can be used to define a request for a specific map tile in the archive stored on a cloud object storage based on HTTP range requests. Since the index for a planet-scale tileset is too large to be fully downloaded with an acceptable latency, only portions of the index need to be queryable. The main design goal of the index is to enable low cloud access charges and a similar latency as compared to a deployment with a dedicated tile backend. Therefore, the transferred amount of data and the number of requests for portions of the index have to be minimized. Every tile in the archive should be requested with at most one request.

The combination of two different approaches for the index layout showed the best results in tests: a root tile pyramid which serves as an overview for the lower zoom levels and index fragments which are lazy loaded on higher zoom levels. Since lower zoom levels are accessed more frequently and the number of tiles is manageable up to a specific zoom level, all index entries can be fetched at once in an acceptable time when the map

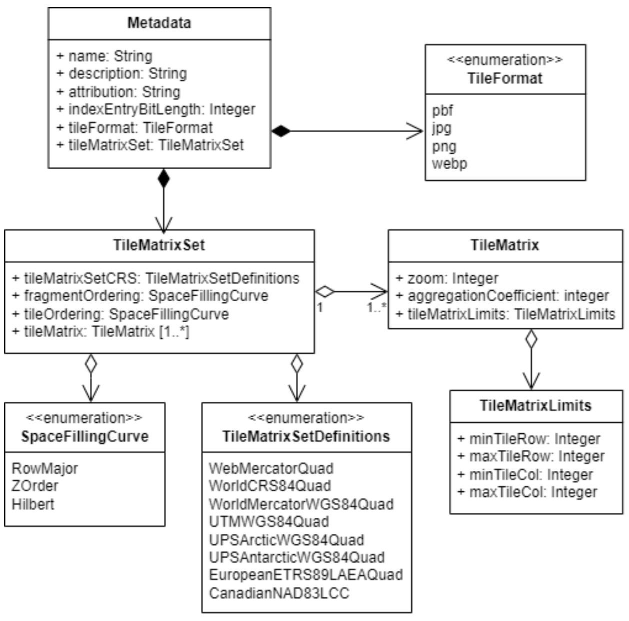

Figure 2. Structure of the metadata document of a COMTiles archive.

Encoding	Row-major order	Hilbert order
Varint	32.83 (19.93)	32.83 (19.93)
Gzip	25.05	25.08
ORC RLE V1	22.96 (20.43)	23.04 (20.37)
ORC RLE V2	21.43 (19.92)	21.46 (19.95)
Parquet RLE/Bit-Packing	25.44 (19.11)	24.83 (19.19)
Modified ORC RLE V1	22.17 (20.12)	22.36 (20.11)

Table 2. Results of comparing different compression algorithms for a index pyramid. Results are in kilobytes, with results of additional Gzip compression in parentheses.

is initially loaded. For the compression of the tile pyramid, a lightweight compression technique should be used that allows for fast decoding. Lightweight compression techqniques can be applied at logical and physical level. The reduction of the number of values and the mapping to smaller values is applied on the logical level. Basic logical level techqniques are Run-length encoding (RLE), delta coding or dictionary encoding. These techniques are used in combination with null suppression (NS) algorithms that eliminate the leading zeros in the binary representation of small integers. Widely used null suppression techqniques are Varint encoding and bitpacking. In planet-scale tilsets duplicate water tiles are a dominant part of the map, since about 71% of the earth is covered with water. Thus, the index pyramid references many tiles of the same size. Therefore, to reduce the size of the pyramid, RLE coding is the most important logical level technique for compressing the size of the pyramid. There are ready to use and widely adapted lightweight compression algorithms used in big data formats that combine RLE encoding with a null suppression technique. For instance, the ORC format employs RLE V1 and RLE V2 encoding, while the Parquet format utilizes an RLE/Bit-Packing Hybrid encoding. For the comparison, also only the usage of Varint encoding was evaluated. To find the best integer compression algorithms, a zoom level 0 to 7 index pyramid with 21845 index entries based on a planet-scale vector tileset was compressed. As illustrated in Figure 2, the ORC RLE V2 encoding exhibited the most optimal results, while an adapted verson of the ORC RLE V1 encoding was identified as the second-best performing encoding method. According to the findings, the ORC RLE V2 encoding exhibited the most optimal results, while an adapted version of the ORC RLE V1 encoding was identified as the second-best encoding method. The evaluation also shows that using an additional heavy compression such as Gzip on top of the light compression algorithms results in only small improvements in the compression ratio while slowing down the decoding performance. It was also tested whether using a Hilbert curve instead of row-major ordering for the index entries of the pyramid results in a better compression ratio. The main idea behind this is that tiles of the same size, such as ocean tiles, are more likely to be spatially clustered, resulting in longer runs in the RLE encoding. However, the results show no improvements in compression ratio when the index entries of the pyramid are order on a Hilbert curve instead of a row-major. Since the adapted version of the ORC RLE V1 encoding is faster to decode and easier to implement than the ORC RLE V2 encoding, this encoding was chosen as the compression method of the root pyramid in the COMTiles format.

To fulfill the requirements that every tile can be accessed with only one additional request, a different approach called index fragments is used for lazy loading portions of the index on higher zoom levels. Fragments are a collection of index records with a default of 4096. Depending on the extent of the tileset, the boundaries of the fragments can be sparse or dense. To allow random access, all fragments have to have the same size. Therefore, the index records of a fragment are bitpacked with a default size of 20 bits per entry, resulting in a default size of 10kb per fragment for 4096 index entries. To reduce the number of requests, the fragments are ordered on a space filling curve such as the hilbert curve. Therefore, the tiles can be fetched in batches, reducing the number of total requests. The sequence for requesting batches of tiles from a cloud object storage by combining an index pyramid with fragments is shown in Figure 3. Since no advanced compression algorithms can be used to fulfill the specified requirements, the resulting index for a planet-scale tileset is about 10 times larger compared to PMTiles ($\sim$91 MB to $\sim$880 MB). The difference in size is mainly a result of the compression algorithms PMTiles uses, with deduplication of directory entries based on RLE encoding being the dominant factor. However, since cloud storage is cheap, the additional cost of the difference in the index size proved to be negligible.

Since no heavyweight compression algorithms for the complete index fragments is used, the index records can also be stream decoded and processed before the full fragment is loaded. This can reduce the latency when the corresponding index entries for the required tiles are in the front part of the fragment. In the browser small chunks from an network stream can be processed based on the Streams API and the associated ReadableStream interface. One disadvantage is that the size of the chunks can't be configured, as it is controlled by the browser. In the tests on an about average internet connection of 110 Mbps the full fragment was returned as a single chunk, omitting the possibility to process index entries before the full fragment is available on the client-side. In contrast, for a 3G connection with 1.44 Mbps bandwidth, the chunk size was by about 1.5 Kb. This allowed to process the index entries of a fragment in 7 chunks. Therefore, when the references to the tiles were part of the first chunks, the latency could be reduced.

4. EXPERIMENTS

In this section, COMTiles is compared to PMTiles for planet-scale tilesets based on different metrics. Furthermore, the efficiency of the presented tile request batching approach is evaluated. The COMTiles archive used in the following evaluation was generated with the @com-tiles/mbtiles-converter from a planet-scale MBTiles database based on the OpenMapTiles vector tiles schema (MapTiler, 2023). The file layout described in Section 3 was not yet integrated in the @com-tiles/mbtiles-converter at the time of the test. Therefore, the generated COMTiles archive were post-processed with a separate utility library (Tremmel, 2023) to match the specified file design. The post-processing stage entailed the utilization of the modified ORC RLE V1 encoding technique for compressing the root pyramid, along with the bitpacking method for compressing the index fragments. The complete source code is available online [2].

4.1 Comparison with PMTiles

The author has selected the following metrics to compare COMTiles with PMTiles, as they have a significant impact on user experience and cloud access charges:

[2] https://github.com/mactrem/com-tiles-evaluation.

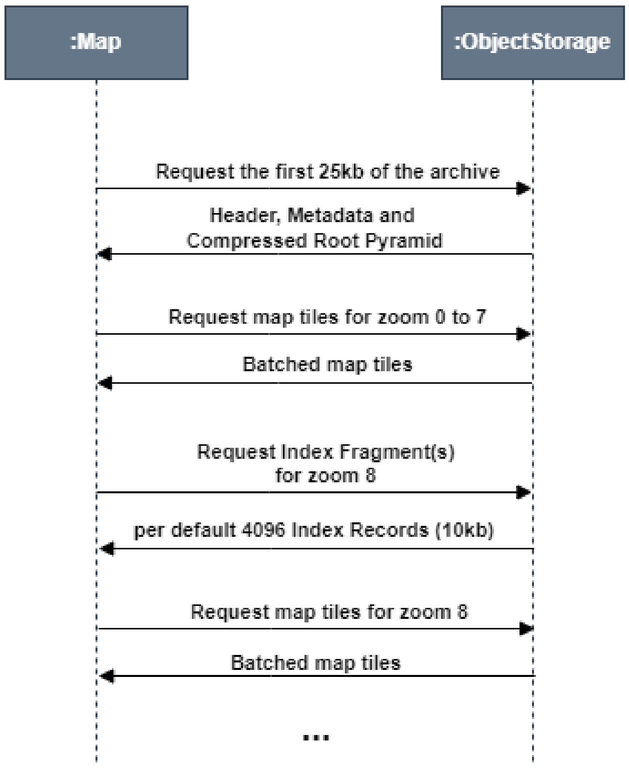
:Map
:ObjectStorage
Request the first 25kb of the archive
Header, Metadata and Compressed Root Pyramid
Request map tiles for zoom 0 to 7
Batched map tiles
Request Index Fragment(s) for zoom 8
per default 4096 Index Records (10kb)
Request map tiles for zoom 8
Batched map tiles
...

Figure 3. Sequence of requesting data from a planet-scale COMTiles archive.

- Size of downloaded data from a cloud object storage
- Number of requests to a cloud object storage
- Performance for decoding portions of the index

For testing, a planet-scale MBTiles database based on the OpenMapTiles vector tiles schema was converted into a PMTiles archive using the go-pmtiles utility tool (Liu, 2023) in version 2.0.1. To display the PMTiles archive on a map, the PMTiles js library (Liu, 2022) in version 2.7.0 were utilized. This library was slightly modified to capture the number of requests, the transferred amount of data and to benchmark the time for decoding portions of the index. The global scale PMTiles and COMTiles archives were both deployed on a AWS S3 storage.

4.1.1 Number of requests and data transferred To determine the number of requests and transferred data size, two different realistic map interactions have been automatic simulated. For the first workflow the testing method described by (Netek et al., 2020) was selected with the Department of Geoinformatics building at Palacký University in Olomouc as the default object. The map interactions shown in Table 3 are described by (Netek et al., 2020) as "simulating how users typically interacted with map applications". Since this workflow is more focused on zoom based interactions, also a second test case with a focus on a panning based map navigation pattern was conducted. The workfow of the second test is focused on exploring larger cities in Europe and is summarized in Table 4.

Table 5 shows the results of the comparison between the two formats in terms of the amount of data transferred and the total number of requests. The results reveal that the COMTiles solution outperforms PMTiles, resulting in a 3.3x and 3.1x reduction in data transfer for the first and second test case, respectively. Regarding the total number of requests, both approaches

Interaction	**Zoom before/after**	**Displayed Extent/Level**
Interaction	Zoom before/after	Displayed Extent/Level
Initial load	-/6.5	Czech Republic
Zoom in on the Department	6.5/17	Street level
Pan to the Square	17/17	Street level
Zoom out 3x	17/14	City of Olomouc
Zoom out 1x	14/13	City of Olomouc

Table 3. List of interactions for the first test case. Adapted from (Netek et al., 2020).

Interaction	**Zoom before/after**	**Displayed Extent/Level**
Initial load	-/1	Planet level
Zoom in Munich center	1/18	City of Berlin
Zoom out	18/5	Country level
Pan to Berlin	5/5	Country level
Zoom in Berlin center	5/18	City of Berlin
Zoom out	18/6	Country level
Pan to Paris	6/6	Country level
Zoom in Paris center	6/18	City of Paris
Zoom out	18/6	Country level
Pan to London	6/6	Country level
Zoom in London center	14/13	City of London

Table 4. List of map interactions for the second explorative test.

did not exhibit a statistically significant difference for both interaction pattern. However, the data revealed that a COMTiles based client made approximately 13% fewer requests in the first test case, while a PMTiles based client made approximately 7% fewer requests in the second explorative test case.

4.1.2 Decoding performance For comparing the decoding performance of both formats, data for the index at zoom level 14 in the center of Munich was selected. The benchmarking were carried out on a Windows 10 workstation equipped with Intel Core i7-8665U (1.90 GHz) and 32 GB of RAM. Benchmark.js were used as benchmarking library because it supports high-resolution timers. The benchmark suites were repeated 15 times and the average was calculated. For benchmarking COMTiles a fragment with the default size of 4096 index records were selected. For PMTiles, in addition to a planet-scale directory of 16384 entries, also a second directory in the size of the used COMTiles fragment with 4094 entries was generated.

As shown in Table 6, COMTiles outperforms PMTiles for a planet-scale directory by about factor 63. When both approaches use corresponding fragment respectively directory sizes, COMTiles is about 19 times faster compared to PMTiles for decoding portions of the index. Thus, the faster decoding performance of COMTiles for global tilesets results from the lightweight encoding combined with the smaller downloaded index portions.

Test Case	Format	Transferred data	Requests
1	COMTiles	1.0	1.0
	PMTiles	3.29	1.13
2	COMTiles	1.0	1.10
	PMTiles	3.1	1.0

Table 5. Results of comparing COMTiles and PMTiles on a planet-scale tileset in terms of the amount of data transferred and the number of requests.

Format	Num Entries	Decoding Performance
COMTiles fragment	4096	1
PMTiles country-scale	4096	19
PMTiles planet-scale	16384	63

Table 6. Results of comparing the decoding performance of COMTiles and PMTiles.

4.2 Batching Tile Requests

For testing the effectiveness of batching tile requests, the realistic map interaction pattern of exploring the city of Olomouc already introduced in section 4.1.1 and shown in Table 3 was used. The test was conducted by displaying the map in full-screen mode on a Full HD (1920x1080 pixels) display. The vector tiles stored in the COMTiles archive had a resolution of 512x512 pixels and were sorted in row-major order.

In the test the number of tile requests was reduced by approximately 77%, resulting in a decrease from 137 individual tile requests to 32 batched requests. As a result, this leads to a considerable reduction in cloud access charges since each request made to a cloud object storage is charged. In addition, batching the tile requests also reduces the network latency as at the time of writing, most major cloud providers like AWS or Microsoft only offer HTTP/1 support for their object storage. The number of parallel HTTP/1 connections and thus the number of parallel requests is limited to 6 for most modern browsers (MDN, 2023). For example, during the tests, most of the time 15 tiles were requested for the current viewport of the map. As a result, when no batching was implemented two times 6 and one time 3 consecutive tile requests were made to the object storage. However, when the tile requests were batched, only 3 parallel requests were required for the 15 tiles. If the object storage is combined with a Content Delivery Network (CDN), this latency advantage is not applicable, as HTTP/2 enables the request for multiple resources on a single TCP connection. A disadvantage of the tile batching approach is that it takes longer until the first tile can be displayed, since a certain number of tiles now have to be downloaded completely at once.

5. CONCLUSIONS AND FUTURE WORK

Based on the results of this paper, the author is confident that COMTiles can simplify the workflow for deploying large tilesets and significantly reduce the storage costs while preserving almost the same user experience compared to a dedicated tile backend. As only a single file must be uploaded to a cloud storage and no dedicated tile backend to be setup, COMTiles can also be deployed by non-GIS experts in a quick and easy way.

The evaluation showed that COMTiles outperforms PMTiles in an about 63 times faster decoding of portions of the index, reducing the processing time from approximately hundreds of milliseconds to a few milliseconds in the test cases. COMTiles also requested about 3 times fewer data on average from a cloud storage in the tests. Additionally, the random-access design of the COMTiles index leads to one initial roundtrip less to the server, resulting in a faster map load. The main advantage of PMTiles is an approximately 10 times smaller size for a planet-scale index. Since cloud storage is cheap, the difference in the index size proved to be negligible in terms of the storage costs. The effects of the index size when using a cloud optimized tile archive in combination with a serverless tileserver require further investigation.

However, future work may include further reduction of the index size while continuing to meet the requirements specified in Section 3. One approach could be to use a Bitvector encoding to reduce the index size of sparse tilesets. Since over 50% of the COMTiles index fragments of an OSM vector tileset are not present, this approach could significantly reduce the size of the index. This could be inspired by the implicit tiling extension of the 3DTiles spec that contains an availability section to efficiently encode sparse datasets (Cesium, 2022).

REFERENCES

Abernathey, R. P., Augspurger, T., Banihirwe, A., Blackmon-Luca, C. C., Crone, T. J., Gentemann, C. L., Hamman, J. J., Henderson, N., Lepore, C., McCaie, T. A., Robinson, N. H., Signell, R. P., 2021. Cloud-Native Repositories for Big Scientific Data. *Computing in Science & Engineering*, vol. 23(2) 26-35, 1 March-April 2021, doi.org/10.1109/MCSE.2021.3059437.

Cesium, 2022. 3D Tiles Specification. https://github.com/CesiumGS/3d-tiles (21 April 2023).

Cloud Optimized GeoTIFF, 2018. https://www.cogeo.org/ (22 April 2023).

GeoParquet, 2022. GeoParquet Specification. https://geoparquet.org/ (20 April 2023).

Harrtell, B., 2021. FlatGeobuf Specification. https://github.com/flatgeobuf/flatgeobuf (20 April 2023).

Holmes, C., 2021. Towards a Cloud-Native OGC. https://www.ogc.org/blog-article/towards-a-cloud-native-ogc/ (19 April 2023).

Liu, B., 2022. PMTiles Specification. https://github.com/protomaps/PMTiles (18 April 2023).

Liu, B., 2023. protomaps/go-pmtiles: Single-file executable tool for creating, reading and uploading PMTiles archives. https://github.com/protomaps/go-pmtiles (18 April 2023).

Mapbox, 2018. MBTiles Specification. https://github.com/mapbox/mbtiles-spec (17 April 2023).

MapTiler, 2023. Open vector tile schema for OpenStreetMap layers. https://openmaptiles.org/schema/ (17 April 2023).

MDN, 2023. Connection management in HTTP/1.x - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x (16 April 2023).

Netek, R., Masopust, J., Pavlicek, F., Pechanec, V., 2020. Performance Testing on Vector vs. Raster Map Tiles-Comparative Study on Load Metrics. *ISPRS International Journal of Geo-Information*, 9(2), 101. doi.org/10.3390/ijgi9020101.

Open Geospatial Consortium, 2021. GeoPackage Encoding Standard. https://www.ogc.org/standard/geopackage/ (20 April 2023).

Open Geospatial Consortium, 2022. OGC Two Dimensional Tile Matrix Set and Tile Set Metadata Specification. https://docs.ogc.org/is/17-083r4/17-083r4.html (16 April 2023).

Tremmel, M., 2023. Evaluating COMTiles. https://github.com/mactrem/com-tiles-evaluation (16 April 2023).