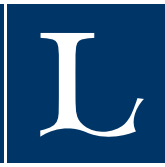
BACHELOR THESIS



MapCloud

A Distributed, Combinatory SOA-Method for 3D Map Data Delivery

Khashayar Farmanbar
Tomas Karlsson
2015

Bachelor of Science
Computer Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Abstract

MapCloud is a solution combing four different Software Oriented Architecture (SOA) methods to obtain a reliable and efficient delivery method for digital map data, more precisely 3D map data.

Mapping software use large amount of data including mapping data, aerial imagery and 3D models. Pre-processing data is a cumbersome task due the sheer amount of data that needs to be handled. The challenge is also on the other end, when the data is accessed in real-time. This process needs to be fast and reliable.

This theses is done at Agency9. Agency9 is a Swedish company offering mapping and Geographic Information System (GIS) software and services worldwide.

Agency9's current system and workflow is limited to a set of single high performing servers for processing and preparation. In the workflow there are also manual steps that constitute a bottleneck for administration due to the need of copying huge amounts of data. From a design standpoint MapCloud overcomes these shortcomings. The workflow of MapCloud utilizes parallel data processing for preparation and distributed, resilient, access method with high-throughput for web-centric map applications.

Index

1 Background

Mapping software has rapidly increased in popularity as mass-market products. Increased access to Internet and mapping data increases the number of products and services utilizing connectivity and geographic information. During the last years a range of consumer services has gained popularity in this area. Tech giants have launched their own services such as Google Maps and Google Earth, Apple Maps and Microsoft Bing Maps; also Nokia remains in this business by licensing their Here Maps to Yahoo!, Microsoft and others.

In 2008, hitta.se[1], the premier Swedish directory and online mapping service, launched a web embedded 3D mapping service, powered by Agency9[2] 3DMaps[3]. The service supplied aerial imagery including 3D height information spanning from satellite imagery down to city level aerial imagery with a resolution of 25 cm per pixel. One of the challenges in the preparations for launch turned out to be how to keep track of all the images, including version handling, when new imagery became available and yet offer the speed required for a consumer market web service like hitta.se. The work presented in this thesis has risen from the need to find efficient ways to deal with aerial images and 3d meshes for use with Agency9 3DMaps.

The Agency9 3DMaps is a map software capable of streaming large 3D maps directly from hard drive using Java or over http to any browser, including mobile platforms such as iOS and Android, using JavaScript and WebGL. Streaming is achieved using a tile based quad tree (Finkel & Bentley, 1974) structure. In a quad tree structure a map view is created from several different levels of details. The top level, level 0, contains the entire area of interest with a fixed resolution. In the case of 3D Maps the default resolution of level 0 is an aerial image of 512x512 pixel. Since 3DMaps support any Cartesian (Descartes, 2001) projection (Craig, 1882) the actual meter/pixel resolution of the level 0 is arbitrary depending on the valid projected bounds (Ihde, et al., 2000).

In the original design from 2006 streaming was done using static files sitting on an http connection using a custom quad tree tile-naming schema. Similar quad trees can also be found in the structure used by TMS from 2010 (Masó, Pomakis, & Julià, 2010) or in Bing from 2010 (Schwarz, 2010). Agency9 3DMaps have since then been extended and can now handle many quad tree layouts as well as services such as a Web Map Service (WMS)[4] and Web Coverage Service (WCS)[5].

The production of a Agency9 3DMaps tile structure is done using a set of tools delivered with Agency9 3DMaps SDK. The tools take as input a set of aerial images and deliver as output a pre-tiled quad tree on disk. Input can be any number of images with varying resolution, size and geographical overlap. When processing, the tools splits the images according to a quad tree layout, and assures that each tile has the correct meter per pixel

[1] www.hitta.se owned by hittapunktse AB, part of the Schibstedt group
[2] Agency 9 AB is a Swedish company, with headquarters in Luleå, Sweden. www.agency9.com
[3] 3DMaps is a product by Agency 9 AB used for visualisation and interaction with 3D maps.
[4] Standard for requesting map images via HTTP http://www.opengeospatial.org/standards/wms
[5] Standard for coverage data access over the Internet http://www.opengeospatial.org/standards/wcs

resolution and handles any conflicts made by potential image overlaps. Data processing can be done by the customer using the tools or as part of a service offered by Agency9. An internal goal has been a processing time of three to five days with a delivery window of seven days from the receiving of data to final product accessible online.

There have been several attempts at moving towards a database driven model for storage and more scalable processing pipeline. Main motivation for this has been to more easily add capabilities for partial update and centralized management of resources. Software designs for this use case has been tested internally based on PostGIS[6] 2009. They were also tested for management of dataset related to the SEMANCO[7] project in 2011. However the added time and complexity of managing and setting up a PostGIS database, especially for new SDK users, have far outweighed the benefit of partial updates. Data is acquired and updated on a per customer range from an interval of a couple of time per year to once every other year with full reprocessing of dataset as service with a delivery window on average of seven days.

The file based data production model has been proven in production and have scaled well for the past nine years. Larger dataset have been tackled by increasing CPU power and storage. The current generation toolset have been proven to scale-up (Bondi, 2000) linearly to 72 ht-cores running on top of a dual XenonE5-2699 (Intel Corporation, 2014) Haswell processor with a raid 0 (Chen, Lee, Gibson, Katz, & Patterson, 1994) Solid State Drive (SSD)[8] configuration.

However the last two years data growth have even outpaced what Moore's law (Moore, 1965) predicted for transistors, where a typical large-scale city model has grown from 100 GB to 1TB. The trend has also been further escalated by heterogeneous mobile market that requires different image formats leading up to a 4x duplication of all image data. Scale-up solution with RAID arrays coupled with increased size of spinning disk have been able to keep pace with the ever increasing storage need, current maximum disk size is at 8TB and 10TB disk is not far off. Increase in disk size has however not been matched by the same increase in disk IO (Ruemmler & Wilkes, 1994). The risk of an additional disk failure during RAID rebuild is increasing and the work of moving 100 TB data during hardware updates is a cumbersome task. Some of these storage challenges have been solved by moving data distribution over to an Amazon Cloud solution. This process ensures full offsite backup of data and redundancy at the cost of an additional step in the delivery process.

[6] PostGIS is a spatial database extender for PostgreSQL object-relational database. http://postgis.net
[7] Semantic Tools For Reduction in Urban Planning. http://semanco-project.eu
[8] SSD is persistent memory such as flash. http://www.zurich.ibm.com/sto/systems/solidstate.html

2 Research topic and demarcations

2.1 Research topic

The goal of this report is to overcome the scale-up (Michael, Moreira, Shiloach, & Winiewski, 2007) limitation of the current production setup and present an alternative workflow and software that unifies production and delivery. The suggested system design needs to offer full redundancy and high availability, offer a scale-out (Michael, Moreira, Shiloach, & Winiewski, 2007) solution capable of petabyte storage and processing while still being cost effective.

2.2 Demarcation

In this report we will focus on the management of aerial images, a similar approach albeit more complex, can be taken for 3D mesh data but for clarity this have been omitted. The report will also only focus on the internal workflow and not look at external customer workflow utilizing the SDK. Transaction conflicts and resolution due to concurrent usage is also omitted.

3 Current workflow

Data deliveries of aerial imagery are usually done using FTP or by physically sending USB hard drives via mail. The customer is often a municipality. Normally there is a low-resolution dataset with coverage over the entire municipality and one or more high detail dataset with coverage over the central city area or key areas for an infrastructure project. Most dataset are delivered in GeoTIFF[9] format in a local projection. The aerial imagery delivery is typically around 200GB in size and is often accompanied with various 3D data and the total delivery can vary from 300GB up to 1TB. Each individual image file is normally around 2GB in size. The data is then copied from the FTP or USB disk to a processing server where command line versions of the tools are available. Input data is stored on spinning disk with a RAID 5 setup and output data is stored on RAID 0 SSD setup. While processing, the tools also store intermediate data on the SSD setup in a work folder.

A processed data set usually consists of tens of millions of individual files. The processed data is compressed and stored on spinning disk on a local server and then uploaded to Amazon using FTP or rsync[10], rsync is used for very large dataset where a partial delivery is needed before the entire dataset have been fully processed. The separation of datacentres between data processing and end user data storage is important to guarantee a reliable quality of service (QoS) for customer when streaming.

The workflow is limited to a set of single high performing server for processing with a series of manual sequential steps for copy, processing, compression, deliver, decompression and backup. Internally all raw- and processed data is stored on cost effective storage in RAID setups. With the current workflow all resource available in the datacentre are not fully utilised at all times and valuable CPU cycles are lost during peak processing. Network bandwidth is also not fully utilized since data transfer is a sequential step before- and after processing.

Administration also becomes a bottleneck for internal storage due to copying of huge amount of data during hardware upgrade cycles.

[9] GeoTIFF is an image file containing georeferenced information
http://www.gisdevelopment.net/technology/ip/mi03117pf.htm
[10] rsync is a file copying tool. https://download.samba.org/pub/rsync/rsync.html

4 Future workflow

An updated workflow must maintain the separation data processing and data access while removing the sequential steps. Processing resources should ideally be shared between processing servers for maximum CPU utilization at all time. Ideally the setup should also be able to work with partial updates and dataset versioning. To maintain the full redundancy and high availability the setup must be able to work in a multiple datacentre setup. Storage should ideally be scalable and automatically handle upgrades, failure, recovery and rebalancing with limited manual work and zero downtime (see figure 1).

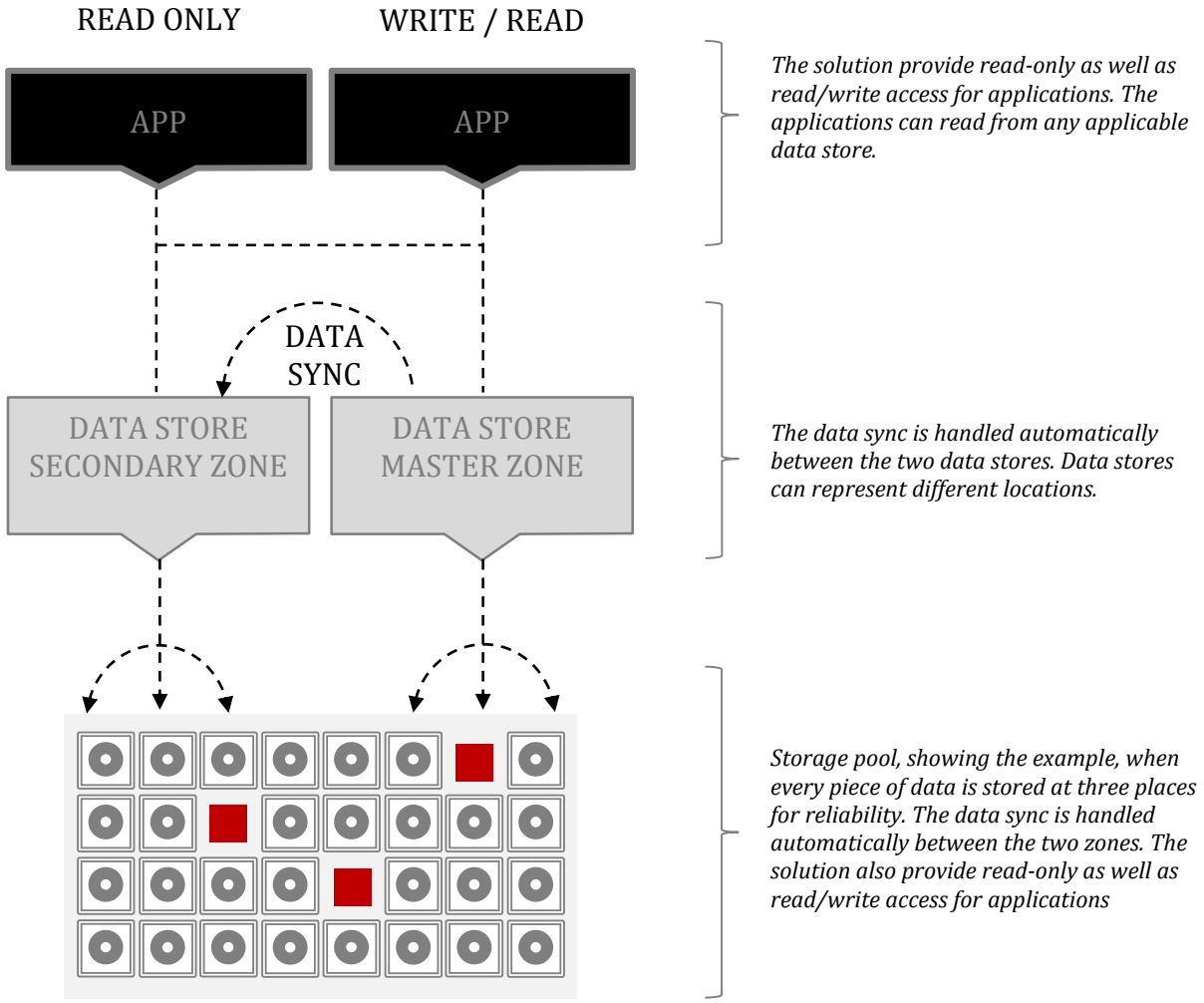


Figure 1 shows a tentative setup where storage pool is shared between two zones where the user access is read online and where the processing side is has both read and write access. Synchronization is automatically managed between zones.

5 System design

By adapting a service-oriented architecture (SOA) (Bell, 2008), the design can be enhanced as a set of support services. A service is a self-contained unit of functionality. The SOA service approach brings loose coupling to the plate, which minimizes dependencies for the system. Key identified functionalities are:

- MapDB
    - Key functionality of the MapDB is to manage versioning and spatial indices. The MapDB should also contain metadata, commit logs and information about dataset and sceneries. This functionality requires efficient and sharded (CodeFutures, 2014) read and write access on small amount of data, typically below 1k.

- MapStore
    - Key functionality is to offer low cost storage of binary data per tile or per raw image file. The system is also responsible for managing storage and access quotas for billing. This functionality requires efficient and sharded read and write of large chunks of data varying from 256k up to 2GB per file.

- MapStream
    - Key functionality is to stream tiles to 3DMaps clients using http. An auxiliary function is cache and CDN (Hofmann & Beaumont, 2005) functionality for hot data.

- MapCompute
    - Key functionality is data processing. Main tasks are tiling large images, quad tree construction, geometry processing and image compression. This functionality requires horizontal scaling, load balancing and fault tolerance

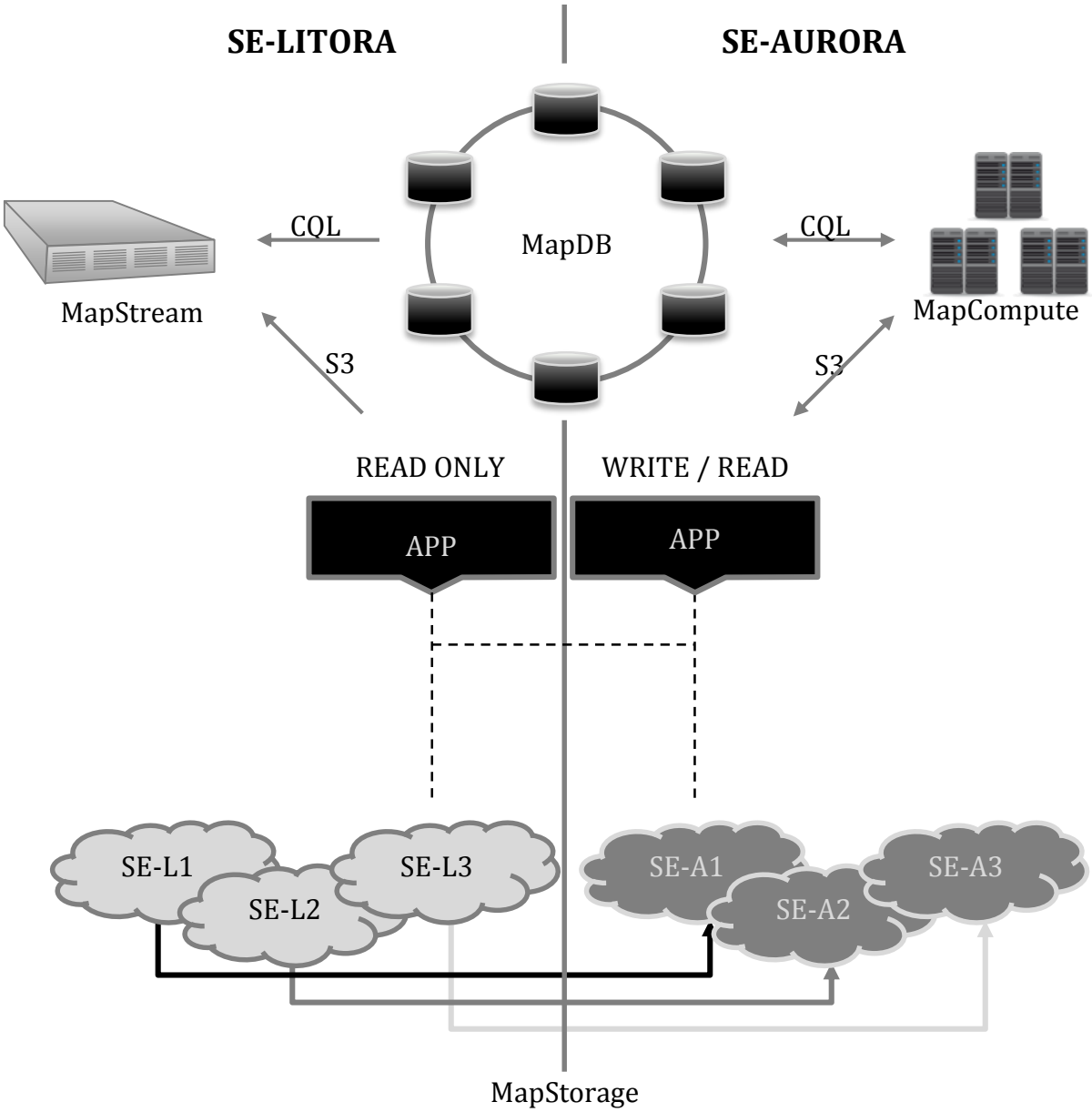


Figure 2 Schematic view of combination of the MapCloud solution. The data is stored at two sites, in this example denoted SE-Litora and SE-Aurora. At every site there can be arbitrary number of storage units (SE-L1, SE-A1, etc).

The combination of the four services, the MapCloud, form a complete solution for processing, storage, versioning and streaming of aerial images (see figure 2). MapCloud is built around a multi-site setup at two different geographical locations *SE-LITORA*, *SE-AURORA*. The two sites offer both full redundancy and high availability, as well as disaster recovery.

6 Component design

6.1 MapDB

The MapDB component is built around Apache Cassandra. Several open source databases solutions have been considered among other PostgresSQL[11], MySQL[12] as well as NOSQL solutions such as HBase (Strauch, 2012). However none of these offer the same automatic built in scalability without a single point of failure.

6.2 Cassandra

Apache Cassandra is an open source NOSQL database that can handle large amount of data and provide high availability with no single point of failure. Cassandra started out as Facebook project but was released as Open Source in 2008 and has been an Apache top level project since 2010. According to Rable (Rable, Sadoghi, & Jacobsen, 2012), Cassandra is the top performing NOSQL database as of 2012. *"In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments."*

Cassandra has also been proven in production at several IT giants. One of the largest production deployments is Apple's, with over 75,000 nodes storing over 10 PB of data (Wikipedia, 2015), another large Cassandra installations is Netflix with over 2,500 nodes, 420 TB, serving over 1 trillion requests per day (Cockcroft, 2011). Some of Cassandra key features are that it is; Decentralized, Supports Replication and Multi Data Centre Replication, Scalability, Fault-tolerant, Tuneable consistency and its Query language.

Cassandra has a master-less architecture with no single point of failure. All nodes in a cluster are the same. Fault tolerance is offered by storing redundant copies of that data that is automatically distributed across all nodes using peer-to-peer, if a node goes down, one or more copies of the data are still available in the cluster. Cassandra works with a strong/eventual consistency model, Basically Available, Soft state, Eventual consistency (BASE) (Pritchett, 2008), compared to traditional linearizable consistency; Atomicity, Consistency, Isolation, Durability (ACID) (Pritchett, 2008). The consistency level can be tuned on both read and writes using different quorums level (Jonathan, 2013) based on the Paxos protocol (Lamport, 2001).

Cassandra Query Language (CQL) (Apache Cassandra, 2015) is the default and primary interface into the Cassandra DBMS. Using CQL is similar to using Structured Query Language (SQL) (Chamberlin & Boyce, 1974). SQL and CQL share the same abstract idea of a table constructed of tables and rows. The main difference from SQL is that Cassandra does not support joins or sub queries. Instead, Cassandra is built around denormalization requiring tables that are optimized based on the applications query pattern. Denormalization is similar to materialized views (Date, 2006) which provide a snapshot

[11] An open source object-relational database system. http://www.postgresql.org/about
[12] An open source database. http://www.mysql.com/about

for read performance, but unlike materialized views a denormalization strategy offer equal performance advantage on both read and writes. Another key difference of CQL compared to SQL is the primary key. A CQL primary key can consists of two parts, the partition key and one or more cluster keys.

```
CREATE TABLE keyexample(

        empID int,
        deptID int,
        first_name varchar,
        last_name varchar,
        PRIMARY KEY (empID, deptID)
)
```

A partition key empID points to a specific node and the cluster key deptID is a seek operation on that node. This approach allows a data model to be optimized for different cluster scenarios based on access patterns.

6.3 MapDB data model

The MapDB main responsibility is to act as the metadata server for map tiles and handle versioning through a commit log. Data committed to the MapDB only contains information about the commit log, bounds of the commit and its version and references to the committed raw data. The actual raw data is stored inside of the MapStore system using an object storage model. MapDB also maintains the spatial indexes for managing tiles need for streaming. Spatial indexing is achieved using simple quad tree hashing. The MapDB layout is based on a concept of datasets and data view. A dataset table contains metadata about dataset such as type of data, srs, creation date and update time etc.

```
CREATE TABLE dataset(
        id uuid,                        # ID per dataset
        account uuid,                   # Account this dataset belongs to
        name text,                      # Name of the dataset
        created_by uuid,                # User that create the dataset
        created timestamp,              # Creation date
        updated timestamp,               # Last updated
        data_type text,                 # Only support one type (Ortho, DTM, MESH, LAS etc)

        srs text,                       # EPSG code
        unit text,                      # Degree, Cartesian
        srs_maxx double,                # SRS bounds
        srs_maxy double,
        srs_maxz double,
        srs_minx double,
        srs_miny double,
        srs_minz double,
        qtree_cx double,                # Quadtree tiling bounds
        qtree_cy double,
        qtree_cz double,
        qtree_hx double,
        qtree_hy double,
        qtree_hz double,

        PRIMARY KEY (account, id)
);
```

Data is added to the system using commits. A commit can contain one or more aerial image sources. Metadata about the commit is stored in a commit log table and the metadata about the commit data is stored in the commit part table.

```
CREATE TABLE datasetUUID_user_commit_log(
        commit_id uuid,                 # Comit ID, Can be used to search dataset_commit_tile_log
        commit_time timestamp,          # Commit time. Should also set updated time on dataset
        message text                    # Message regarding this commit
        user_id uuid,                   # User who did the commit
        bytes bigint,                   # Size in bytes of this commit.
        filenames List<String>,         # Original filename of this commit.
        cx double,                      # BBOX of the commit.
        cy double,
        cz double,
        hx double,
        hy double,
        hz double,

        PRIMARY KEY(userid, commit_time)
);
```

The commit part table contains the reference to the imagery split on a per tile basis. The imagery data is stored in the MapStore as binary blob. A primary key based on the (tile, level) as partionkey and with the coloums commit_id and file source for efficient clustering. Generally, Cassandra will store columns having the same tile but a different level on different nodes, and columns having the same tile and level on the same node according to the CQL 3.0 specification (Apache Cassandra, Apache Software Foundation, 2015). This enables quick access for merging multiple commits per tile level.

```
CREATE TABLE datasetUUID_commit_tile_part(
        id uuid,                        # ID of this part
        tile bigint,                    # A,B,C,D encoded as two byte pairs. 32^level
        level int,                      # Index per level
        commit_time timestamp,          #  When the commit initiated .
        filesource int,                 # Reference filename in the user_commit_log
        blob_id text,                   # reference to blobstore
        mime_type text,                 # Filetype
        scalefactor double,             # The scaling that need to be applied meet the target m/pixel.
        cx double,                      # BBOX of the blob. Usually the full commit bounds
        cy double,
        cz double,
        hx double,
        hy double,
        hz double,
        commit_id uuid,                 # Uppdate from which comitt

        PRIMARY KEY ((tile, level), commit_id, filesource)
);
```

The tile affected by this commit is also recorded in a commit tile log that tracks the tiles that got dirty from this commit.

```
CREATE TABLE datasetUUID_commit_tile_log(
        commit_id uuid,
        tile bigint,
        int level,
        PRIMARY KEY(commit_id,  tile, level )
)
```

A dataview contains a renderable view of a dataset. Several datasets can be combined to create a dataview. A dataview also have time slider support with the capability to show different versions of the same view. The table dataview contains the metadata about the actual view.

```
CREATE TABLE dataview(
                id uuid,                        # One ID per view.
                account uuid,                   # Account this dataview belongs to
                name text,                      # Name of the dataview
                datasets List<uuid>,            #Datasets that this dataview was created from
                platform List<String>,          # List of supported platforms
                created_by uuid,                # User create the dataview
                created timestamp,              # Creation date
                updated timestamp,              # Last updated
                time_slider timestamp,          # Timeslider Each a version of the data view
                data_number int,                # Can be  a flightnumber or verision
                type text,                      # Ortho
                mime_type text,                 # PNG, A3X…
                srs text,                       # EPSG, Only used from the first view
                levels int,                     # Quadtree depth
                cx double,                      # BBOX of the view.
                cy double,
                cz double,
                hx double,
                hy double,
                hz double,

                PRIMARY KEY (account, id, time_slider, number)
);
```

A dataview tile contains the actual renderable data. The primarykey is based (tile, level) as partionkey and the coloums time_slider, data_number and platform.

```
CREATE TABLE dataview_tile(
                tile bigint,
                level int,
                id blob,                        # Usually A3X when not a LAS or Ortho only
                mime_type text,                 # PNG, A3X,
                platform text,                  # Crunch, ETC, DXT
                cx double,                      # BBOX of the blob. Manly used to get info about height.
                cy double,
                cz double,
                hx double,
                hy double,
                hz double,
                updated timestamp,              # When the tile was updated, actual time when update happend

                time_slider timestamp,          # Each timeslider stamp represnets a version of the dataview
                data_number int,                #Can be  a flightnumber or verision

                PRIMARY KEY ((tile,level), timeslider, data_number, platform)
);
```

6.4 MapStore

The MapStore component is built around a subset of the Amazon S3 API. Using the S3 API as target decouples storage implementation from the underlying storage system. The Amazon S3 API is a REST based API offering an object storage model. The API has become

a de-facto standard for object storage and is supported by many cloud vendors as well as open source initiatives.

During system design phase the tests have mainly been conducted against Amazon S3 cloud service and the final implementation is intended to run on top of Ceph (Inktank Storage, Inc., 2015). Other solutions that have been considered are SWIFT (OpenStack Foundation, 2015), GlusterFS (Red Hat Inc., 2014) and XtreemFS (Quobyte Inc., 2014) however Ceph are now a part of Red Hat and seem to be the strongest of those three offering both community and commercial support.

The Amazon S3 API offers basic functionality in form of creation, deletion and listing of buckets as well as put, get, list and delete operations of object. In the object storage model there are only buckets and objects, there is no concept of a file system, a bucket can contain an arbitrary number of object all identified by a unique key. The Amazon S3 cloud service offers both high availability and redundancy but at a high cost per terabyte of storage. A Ceph deployment can be built at a substantially lower cost per terabyte of storage based on commodity hardware.

A Ceph storage cluster is built consisting of two types of daemons a Ceph OSD Daemon (OSD) stores data as objects on a storage node; and a Ceph Monitor (MON) (Inktank Storage Inc., 2014). Ceph, similar to Cassandra, is based on Paxos but the monitor enforces a strong consistent policy for a quorum. User access of a Ceph deployment is done through the Ceph Block Devices [13], Ceph Object Storage (radosgw)[14] and/or the Ceph File system[15] services.

The Ceph block device service offers block device storage cloud systems such as OpenStack, Ceph file system service offers a POSIX (Andrew, 2013) compliant file system and the Ceph Object Store offers an object store gateway to the underlying storage utilizing either S3 or SWIFT API. A multi-site setup as shown in Figure 1 and Figure 2 can be deployed as a federated configuration[16], in a federated configuration the master region has read and write access, while the slave region has only read.

The radosgw offers full user admin and quota admin through command line[17] or Admin Ops API[18]. Creating a new user is done by

```
radosgw-admin user create --uid=johndoe --display-name="John Doe" --email=john@example.com
```

or through the Admin OPS API as http post:

```
POST /{admin}/user?format=json HTTP/1.1
```

Where the parameters are supplied using json.

[13] http://ceph.com/docs/master/rbd/rbd
[14] http://docs.ceph.com/docs/master/radosgw
[15] http://docs.ceph.com/docs/master/cephfs
[16] http://ceph.com/docs/master/radosgw/federated-config
[17] http://ceph.com/docs/master/radosgw/admin
[18] http://ceph.com/docs/master/radosgw/adminops

For the S3 API there is also a Java library providing easy management of basic operations. First a connection is made to the server using the right credentials.

```
AWSCredentials credentials = new BasicAWSCredentials(accessKey, secretKey);
AmazonS3 conn = new AmazonS3Client(credentials);
conn.setEndpoint("objects.mapcloud.agency9.com");
```

And then an object can be created using:

```
ByteArrayInputStream input = new ByteArrayInputStream("Hello World!".getBytes());
conn.putObject(bucket.getName(), "hello.txt", input, new ObjectMetadata());
```

The S3 interface also supports multipart upload. This is a vital functionality to enable large upload of raw aerial images over a client interface. A full delivery is close to 200GB or more and each single file is up to 2GB in size thus the system need cope with network failure. By using multipart upload it is possible to recover from failure during upload of raw images. Multipart upload is managed by the Java Low-level API[19].

```
List<PartETag> parts = new ArrayList<PartETag>();
InitiateMultipartUploadRequest initRequest = new InitiateMultipartUploadRequest(
                                                    bucketName,keyName);
InitiateMultipartUploadResult initResponse =
                               s3Client.initiateMultipartUpload(initRequest);

File file = new File(filePath);
long contentLength = file.length();
long partSize = 5 * 1024 * 1024;

try {
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++) {

            partSize = Math.min(partSize, (contentLength - filePosition));

        UploadPartRequest uploadRequest = new UploadPartRequest()
            .withBucketName(bucketName).withKey(keyName)
            .withUploadId(initResponse.getUploadId()).withPartNumber(i)
            .withFileOffset(filePosition)
            .withFile(file)
            .withPartSize(partSize);

        parts.add(s3Client.uploadPart(uploadRequest).getPartETag());
        filePosition += partSize;
    }

    CompleteMultipartUploadRequest compRequest = new
                CompleteMultipartUploadRequest(bucketName,
                                               keyName,
                                               initResponse.getUploadId(),
                                               parts);

    s3Client.completeMultipartUpload(compRequest);
} catch (Exception e) {
    s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
              bucketName, keyName, initResponse.getUploadId()));
}
```

[19] http://docs.aws.amazon.com/AmazonS3/latest/dev/llJavaUploadFile.html

Retrieving files are equally important and is easily done directly from the 3DMaps client using:

```
http://objects.mapcloud.agency9.com /[bucket-name]/[key-name]
```

A dataset and data view will always be represented as bucket and the key-name is based on the primary key of the table. Thus accessing a binary blob containing an aerial image can be done directly without accessing the MapDB first using:

```
http://objects.mapcloud.agency9.com /[dataview]/tile/level/timeslider/ data_number/platform
```

6.5 MapCompute

The MapCompute component is built around Apache Spark[20]. Apache Spark is a fast and general engine for large-scale data processing. Spark provides automatic cluster processing using functional lambda statements in the form of resilient distributed datasets or RDD. Hadoop[21] was also considered as an alternative but external benchmark indicates that Spark can be up 10-100x faster (Xin, 2015) for certain use-cases. Spark also offers the upper hand when it comes to code simplicity, compared to Hadoop which usually requires huge amount of boilerplate code for a simple problem. In Spark a parallel job is modelled function on a RDD stream similar to that of Java 8 stream API.

RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset. All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset. The code below shows how to count the number of lines in a text file using a parallel RDD lamda function[22].

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

The RDD are resilient meaning they are a fault-tolerant collection of elements. If any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it. If a node in a cluster goes down the job is rescheduled to a new node, which recreates the transformation. Processing of images are done first per aerial image and second per tile base.

[20] http://spark.apache.org
[21] http://hadoop.apache.org
[22] https://spark.apache.org/docs/latest/programming-guide.html#basics

6.5.1 Processing a delivery using Spark

First a new datasetable and a dataset_commit_tile_part is created to represent a new dataset. A dataset delivery consists of a set of images in a bucket and a delivery should also always have the same meter/pixel properties. The bucket is listed and mapped to a `JavaRDD<String>`. A map job is invoked which reads an image and calculates in bounding box (BBOX), the image is scaled to fit the closest level in the quad tree pyramid, the image is split and written to the dataset_commit_tile_part table and to the commit_tile_log, and the bounding box is returned. On the returned `JavaRDD<BBOX>` a reduce job is invoked which calculates the final commit BBOX. A pseudo code listing is as follows.

```
JavaRDD<String> imageFiles = listbucket(deliveryBucket);
JavaRDD<BBOX> bboxes = imageFiles.map(s -> {

                                        AerialImage image = readImage(s);
                                        BBOX bbox = createBBOX(image);
                                        AerialImage[] tiles = splitByGrid(image);
                                        for(AerilImage part: tiles)
                                                    writeToTile(part);

                                        return bbox;
                                        }
);


BBOX commitBounds = imageFiles.reduce((a, b) -> a.add(b));
```

The BBOX, commit time, and file list is then written to the dataset UUID_user_commit_log and the commit is completed.

To create a view dataview a table is created. Merging of a dataset commit with a tile view create the actual ready to use tile data. The commits are sorted in a meter/pixel with the low to high resolution. Merging is then done with the help of Painters algorithm (Foley, van Dam, Feiner, & Hughes, 1995).

Based on the commit with the lowest resolution the commit log is mapped to a `JavaRDD<Pair<Tile, Level>>` with dirty tiles. The data is transferred using a `forEach()` action. A pseudo code listing is as follows.

```
JavaRDD<Pair<Tile, Level>> dirty = queryCommitLog(commitID);
dirty.forEach(f -> {
            AerialImage image = readDatasetPart(f);
            writeToView(image);
});
```

When a commit parts have been compiled to the view a simple box filter (Tech-Algorithm, 2007) is applied fills the complete pyramid with data. A pseudo code listing is as follows.

```
JavaRDD<Pair<Tile, Level>> dirty = queryCommitLog(commitID);

while(level >= 0){
            JavaRDD<Pair<Tile, Level>> parents = findParents(dirty);
            dirty.forEach(f -> {
                                        AerialImage[] images = readChildren(f);
```

```
                                        AerialImage child = reduce(images)
                                        writeToView(child);
                                });
            dirty = parents;
            level --;
};
```

This process is then repeated with each commit, from low- to high resolution.

6.6 MapStream

The Mapstream is a very simple web service sitting on a Tomcat[23] server. The services list the available data views, their metadata and their access point in the MapStore. Streaming of the data view is done directly from the MapStore using the S3 API.

[23] http://tomcat.apache.org

7 Discussions

The MapCloud system has not been tested in a production environment. Isolated parts of the system have been verified separately in simulation. However work remains before all services are fully implemented and integrated. From a design standpoint the MapCloud overcomes all shortcomings of the current workflow. The MapDB component has added versioning control as well traceability of commits and data updates. The data model has been implemented on a virtual Cassandra cluster and have been tested for a limited use case. The current model should be seen as a proof of concept as well as an exercise to verify the feasibility of modelling of spatial data in CQL.

A production ready data model needs to consider not only aerial images but also a combination of Digital Surface Model (DSM) / Digital Terrain Model (DTM), Triangulated Irregular Network (TIN) based terrains and 3D buildings. The data model also need to handle a combination and merging of different datasets with different datatypes as well as filtering and clipping of unwanted data.

The MapStore service can offer low cost per TB by utilizing cost effective commodity hardware when deployed on Ceph. The S3 API offers all functionality needed for fully automate the pipeline and storage. Performance and streaming capabilities of the S3 API have already been proven in production for both aerial images and 3D models in the implementation of Slagboom and Peeter's map viewer[24]. The Ceph storage system also offers scalability and can handle petabyte of data reliably.

By utilizing Apache Spark, the current parallel image processing algorithms have been transformed from a scale-up to a scale-out solution. A test has verified the feasibility of such as system by running in a simulated environment; further work is needed to prove the actual performance in a real production cluster. Questions that could not be answered in simulated environment remains; how to best scale the workloads across a cluster and when and where the network become and bottleneck.

[24] http://viewer.slagboomenpeeters.com

8 Conclusions

The MapCloud system design answers the research question and provides an architecture that solves previous workflow problems. A design has been created that identifies four key SOA components. The SOA components have been verified through implementation in isolation in a simulated production environment. Future work remains to test each component in a production environment.

9 References

Amazon Inc. (2014). *Amazon Web Services - Overview of Security Processes.*

Andrew, J. (2013, October 4). *POSIZ.1 FAQ*. Retrieved from Open Group: http://www.opengroup.org/austin/papers/posix_faq.html

Apache Cassandra. (2015, April 12). *Introduction to Cassandra Query Language*. Retrieved from Introduction to Cassandra Query Language | DataStax CQL 3.1.x Documentation: http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html

Apache Cassandra, Apache Software Foundation. (2015). *CREATE TABLE | DataStax CQL 3.0 Specification*. Retrieved from http://docs.datastax.com/en/cql/3.0/cql/cql_reference/create_table_r.html?scroll=reference_ds_v3f_vfk_xj__p_sks_qcr_2k

Bell, M. (2008). *"Introduction to Service-Oriented Modeling". Service-Oriented Modeling: Service Analysis, Design, and Architecture.* Wiley & Sons.

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. Proceedings of the second international workshop on Software and performance. *WOSP '00* (pp. 195-203). New York: ACM.

Chamberlin, D. D., & Boyce, R. F. (1974). SEQUEL: A structured English query language. *Proceeding SIGFIDET '74 Proceedings of the 1974 ACM SIGFIDET workshop on Data description, access and control* (pp. 249-264). New York: ACM.

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., & Patterson, D. A. (1994). RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys 26* (pp. 145-185). ACM.

Cockcroft, A. (2011). *Migrating Netflix from Oracle to global Cassandra*. Retrieved from Slideshare: http://www.slideshare.net/adrianco/migrating-netflix-from-oracle-to-global-cassandra

CodeFutures. (2014). *Database Sharding - CodeFutures*. Retrieved from Database Sharding - CodeFutures: http://codefutures.com/database-sharding/

Craig, T. (1882). *A treatise on projections, by Thomas Craig.* Michigan: University of Michigan.

Date, C. J. (2006). Materialization. In C. J. Date, *The Relational Database Dictionary: A Comprehensive Glossary of Relational Terms and Concepts, with Illustrative Examples* (p. 59). O'Reilly Media, Inc.

Davison, W. (den 22 July 2014). *rsync.* Hämtat från rsync: https://download.samba.org/pub/rsync/rsync.html

Descartes, R. (2001). *Discourse on Method, Optics, Geometry, and Meteorology.* Indianapolis / Cambridge: Hacket Publishing Company.

Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, Volume 4, Issue 1, 1-9.

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. (1995). *Computer Graphics: Principles and Practice.* Addison-Wesley.

Gilmore, J., & Fenlason, J. (den 22 July 2014). *GNU TAR.* Hämtat från GNU tar: an archiver tool: http://www.gnu.org/software/tar/manual/tar.pdf

Hofmann, M., & Beaumont, L. R. (2005). *Content Networking: Architecture, Protocols, and Practice.* Morgan Kaufmann Publishers.

Ihde, J., Boucher, C., Dunkley, P., Farrell, B., Gubler, E., Luthardt, J., & Torres, J. (2000). *European Spatial Reference Systems–Frames for Geoinformation System.* Albacete: Sitopcar.

Inktank Storage Inc. (2014). *Intro to Ceph - Ceph Documentation*. Retrieved from http://docs.ceph.com/docs/master/start/intro

Inktank Storage, Inc. (2015). *Home Ceph*. Retrieved from http://ceph.com

Intel Corporation. (2014). *Intel® Xeon® Processor E5-1600 and E5-2600 v3 Product Families.* USA: Intel Corporation.

Jonathan, E. (2013, July 23). *Lightweight transactions in Cassandra 2.0*. Retrieved from DataStax: http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0

Lamport, L. (2001). *Paxos Made Simple.*

Masó, J., Pomakis, K., & Julià, N. (2010, April 6). OpenGIS® Web Map Tile Service Implementation Standard. *OGC 07-057r7*. 2010: Open Geospatial Consortium Inc.

Michael, M., Moreira, J. E., Shiloach, D., & Winiewski, R. W. (2007). Scale-up x Scale-out: A Case Study using Nutch/Lucene. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International* (pp. 1-8). Long Beach, CA: IEEE.

Moore, G. E. (1965, April 19). Cramming more components onto integrated circuits. *Electronics Magazine*, p. 4.

OpenStack Foundation. (2015, April 16). *Welcome to Swift's documentation!* Retrieved from http://docs.openstack.org/developer/swift/

Oracle Corporation. (u.d.). *MySQL :: About MySQL*. Hämtat från MySQL.com: http://www.mysql.com/about/

Pritchett, D. (2008, vol. 6). Base: An Acid Alternative. *ACM Queue*, pp. 48-55.

Quobyte Inc. (2014). *XtreemFS - fault-tolerant distributed file system*. Retrieved from http://xtreemfs.org

Rable, T., Sadoghi, M., & Jacobsen, H.-A. (2012). *Solving Big Data Challenges for Enterprise Application Performance Management.* VLDB.

Red Hat Inc. (2014). *Write once, read everywhere*. Retrieved from Write once, read everywhere - Gluster: http://www.gluster.org

Rouse, M. (2005, September). *What is JBOD (just a bunch of disks or just a bunch of drives)*. Retrieved from TechTarget: http://searchstorage.techtarget.com/definition/JBOD

Ruemmler, C., & Wilkes, J. (1994, March). An introduction to disk drive modelling. *IEEE Computer*, pp. 17-29.

Schwarz, J. (2010). *Bing Maps Tile System*. Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/bb259689.aspx

Strauch, C. (2012). *NoSQL Databases.* Stuttgart: Hochschule der Medien, Stuttgart.

Tech-Algorithm. (2007, October 7). *Box filtering*. Retrieved from Tech-Algorithm.com: http://tech-algorithm.com/articles/boxfiltering

The PostgreSQL Global Development Group. (2015). *PostgreSQL: About*. Hämtat från PostgreSQL: http://www.postgresql.org/about/

Weigold, T. (u.d.). *IBM Research - Zurich | Storage Technologies | Storage systems*. Hämtat från IBM Research - Zurich: http://www.zurich.ibm.com/sto/systems/solidstate.html

Wikipedia. (2015). *Apache Cassandra - Wikipedia, the free encyclopedia*. Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Apache_Cassandra

Xin, R. (2015, November 5). *Spark officially sets a new record in large-scale sorting*. Retrieved from http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html