NextCloud FaceRecognition app

Design notes

Date	Summary	Author/Editor
2018-07-29	Initial version	Branko Kokanovic <branko at kokanovic dot org> Matias De lellis <mati86dl at gmail dot com>

Contents

1 Introduction 2
2 Terminology 2
3 North star 2
4 Architecture 3
4.1 Library support 3
5 Background processing 4
5.1 DNN models and algorithms 6
5.2 Processing order 6
5.3 Estimates 6
5.4 Cluster stability 7
5.4.1 Continuous cluster rebuild 7
5.4.1.1 Algorithm implementation 8
5.4.2 Incremental cluster build 9
5.5 New/change image processing 10
5.5.1 File addition callback 10
5.5.2 File modification callback 11
5.5.3 File deletion callback 11
5.6 Resource governance 11
6 User operations 11
7 Administrative operations 12
8 Extended functionalities 12
9 Proposed DB schema 12

1 Introduction

Nextcloud FaceRecognition is/will be Nextcloud application with a goal of recognizing, analyzing and aggregating face data in users' images, and providing additional functionalities on top of these information, all with built-in privacy of Nextcloud. Imagine Google Photos, but only for faces (not detecting objects…) and in such way that your images never leave your Nextcloud instance.

Goal of this document is to sums up all thinking around this topic, present various possible implementations with their respective tradeoffs (so next guy coming is aware some problem/solution is being already thought of) and to serve as a poor-man functional specification.

Any comments and all yellow highlights are open questions and invites for further discussions (but, of course, feel free to add your thoughts to anything, everything is open for discussion! ☺).

2 Terminology

User – in this document means Nextcloud user

Face descriptor – 128D vector representing face data

Person/cluster – person from images. Person is not the same as user (although it could be). One person should have one cluster. One person can have multiple faces.

Face – image of a face. It is represented by image and coordinates of face in that image. It has associated face descriptor. It either belongs to single person or person is not known.

Pdlib – PHP extension, binding around DLib library.

Model/DNN model – currently active neural network model on which face detection and face description is based.

3 North star

Serves as a "vision". Not necessarily reflects reality, just gives long-term guidance

User downloads FaceRecognition app from NC app store. Do we need some "cooler" name?

After installation, app immediately start working in background, crunching images. During this period (no clusters yet), user is presented with progress bar – number of images processed, total number of images and estimated time of finish. This progress bar is OK to exist even afterwards (when clusters are present).

After enough data is gathered, user is given all possible clusters of people in images (ideally – no errors☺). Errors could be:

- One person is part of multiple clusters (merge is correction action)
- One cluster contains multiple persons

User than can do various functionalities, on top of these clusters, as explained later.

Any changes, like:

- Upgrades,
- changes in algorithms (DNN model, clustering…),
- functionalities (adding support for new image type…)

Komentar [MI1]: First comment but more off topic..
I've been thinking about this for a long time.
This library.. should be named like that?
Will it really be a full wrapper of dlib? or should we target recognition functions. I think that calling it php-dlib is very ambitious, and if it moves away from our requirements it can be difficult to maintain.
Always imagine it as https://github.com/ageitgey/face_recognition but in php.
To discuss it with goodspb or fork the project.-

Komentar [BK2R1]: Not offtopic! You are right. However, there are IMHO two different libraries here:

- Pdlib – call it ambigous or not, full dlib coverage or just for us, it doesn't matter. It is stupid „binding", same as „php-gtk" is, there is zero php code there! From our (faceRecognition app) perspective, it is just a wrapper that „gets the job done":) As long as „goodspb" user is merging what *we* need, I would ignore the fact it is not having full Dlib coverage;). If „goodspb" don't want to merge stuff that we need, well... we could just fork:) But, yes, calling it „php-dlib" is very ambitous (and we don't care covering whole dlib, just stuff we need:), why would we?) I tend to have pragmatic approach – do not touch if it works. In current state, we are „almost" there
- When you say https://github.com/ageitgey/face_recognition, you are probably not thinking of pdlib (as it is simple binding), I think you are reffering more of what I call below „libFaceRecognition" (in section 4.1) – wrapper for pdlib and bunch of helper functions. And more I think abou this, and now your comment, more I think we should go this way – split responsibility and have separate libFaceRecognition project.

Overall, my take is – have three project with following dependencies: FaceRecognition app -> libFaceRecognition -> pdlib. First two are „ours", pdlib is binding. Does this makes sense?

Komentar [MI3R1]: In general my doubts lie in that, the times I work with premature libraries, just because someone did something that is useful to me, half of the time, they end up abandoning, and I ended up giving supports my users, or others who used the same library,. In this case it is worse, because dlib is much more than facial recognition.. and surely people will ask for support of mor …

Komentar [BK4]: Other than "FaceRecognition"? I don't see other Nextcloud apps are very imaginative when it comes to naming, they mostly just explain what they do (Nextcloud Calendar, Nextcloud Notes, Nextcloud Antivirus…) But, for example, when we have all of these infrastructure in place, next logical step would be to detect objects, facial expressions… Thinking in that direction, does we need another name or not? …

Komentar [MI5R4]: It could be called recognition alone, but I think we have to focus on something. Try to do well, and with experience expand it into something generic. For now FaceRecignition I think it is correct.

Komentar [BK6R4]: Agree!

Komentar [BK7]: what is corrective action here (asking regarding section 1.5 Cluster stability)

should not:

- affect existing cluster sets (person in one cluster stays in that cluster)
- need to regenerate all the data again (like it is first-time run)
- need to recreate any of additional data user did afterwards (like merging clusters, assigning names to clusters…)

4 Architecture

FaceRecognition follows regular design, same as any other Nextcloud application, e.g. Nextcloud app framework is the one dictating overall architecture. One can call it "principle of least surprise"☺. There is no other languages involved except PHP (and JS, of course), no need for advanced message queues, no need for separate database, no need for special docker images… We should strive to work with whatever is available in Nextcloud and to try to fit into Nextcloud ecosystem nicely.

Only notable example where this is not possible is access to Dlib models (in pdlib). If there is no pdlib extension (common case), nothing is happening, and main page just show this info and link to github wiki how to install pdlib extension. First time experience can be crucial here, so info message is important.

App should work in most setups:

- App should work regardless of GPU presence
- App should work without any other dependencies (installed softwares, servers, other PHP libraries, even pdlib.so should come bundled, if possible)
- App should work with PHP 7 (should PHP5 be ever supported, does NC supports PHP 5?)
- App should work without cron jobs functionality being set up on Nextcloud
- Owncloud support (what is delta to support both?)

App will probably never work in following setups and that should be noted clearly in info message in main admin view of the app:

- With PHP5
- Without cron job support
- With encryption

One additional architectural question is how want to structure apps. Do we want to have:

- one app with all functionalities (including support for object detection, emotions…),
- app exclusive for face detection and recognition/clustering, or
- bunch of smaller apps/modules – one for dealing with images, one dependent on it to detect faces, one for face analysis, one for face clustering…

In my opinion, first or second approach seem more appropriate, as pdlib dictates our architecture. Between first or second, I have no strong opinion (first if we find better name☺, second if we continue as "FaceRecognition" and if app turns out to be successful).

4.1 Library support

Ideally, we should structure app to consist of GUI layer along with bindings with Nextcloud (callbacks, DB layer…) and standalone library layer, named "libFaceRecognition" library. It is written in PHP and can be used in other PHP systems that respect user privacy, like image galleries (Piwigo, Koken…).

Komentar [BK8]: Can app provide pdlib.so for generic dimensions (windows/linux, x64/arm, gpu/no gpu) and dynamically load it? As far as I can tell, PHP7 doesn't have ability to dynamically load extensions, without them being already specified in php.ini?

Komentar [MI9R8]: http://php.net/manual/es/function.dl.php
Is deprecated and there are no alternatives..
This is going to be an important limitation for the acceptance of the application..
But if the library works correctly it will take time but we can compile it in most distributions (and windows ;).

Komentar [BK10R8]: Yes, too bad. Least we can do is to bundle .so/.dll files in app and, on admin page, show something like:

„add following lines to /etc/we_can_figure_path/php.ini:

[pdlib]
Extension=/path/to/nextcloud/app/pdlib.so
"

Komentar [MI11R8]: Perfect. and beyond the administration page, it is essential, a good set of README, WIKI and FAQ
Also, I will try to keep a fedora and debian repo with this library,.. ;)

Komentar [MI12]: Let's wait for the nextcloud 14 requirements.
And can it be a good time to show a preview?

Komentar [BK13R12]: Not sure when 14 is out, so I guess it is ok☺ (I wouldn't block project on the fact that PHP5 is not supported☺)
Regarding good times to show preview – well, every time something works is a good time☺ As far as I am concerned, you can show preview with what you got up to now. However, some alpha working preview with libFaceRecognition and whatsnot (this „different" …

Komentar [MI14R12]: Until now the release date is: 2018-09-06, but still not test a single beta. Maybe we will have to keep both versions for a good time .. …

Komentar [MI15]: In general it is not difficult. The api are still very similar. But we need someone to continue using …

Komentar [BK16R15]: Me neither. It is out then, unless some poor soul really like it and want it supported☺

Komentar [MI17]: Of course, we can show it to the administrator

Komentar [MI18R17]: I'm working on this.

Komentar [BK19R17]: Yes, you are right, admin view is more proper place, corrected

Komentar [MI20]: In general I prefer this option. and then it can be extended to the first. The last one we need more knowledge, at least from my part.

Komentar [BK21R20]: Yap, I am inclined more into this direction too, but just wanted to write all possible options (that's why these documents are about – when someone, …

libFaceRecognition exposes various APIs (add image, do cluster, merge clusters…) and do not have persistence layer (just simply returns results back). One big uncharted/non-straightforward API to expose would be background processing.

With standalone library, core parts of FaceRecognition could be much, much better tested (as there is a clear API boundary, and Nextcloud is not dependency).

However, I think that, since this project is already pretty large, extracting library this early can get quite messy. I do think that library should exist, but I have no strong opinion that it should exist from the start. Thoughts?

5 Background processing

This is part of work of application than happens without any user interaction.

All background operations should be guarded with a check that pdlib (or any other global requirement) is present. If they are not present, we just bail out.

Background processing should be started with cron jobs. If cron jobs are not available, there should be NC command which is equivalent of what cron job would execute. This would allow for easier testing and ability for this app to be used by admins without cron job support, as well as better control of resource governance, see section Resource governance).

Should we inspect type of cron jobs and allow only system cron jobs (and possible webcron), but never AJAX. Looks to me that this can yield more problems than benefits. For people using AJAX, we should treat AJAX cron job type same as if cron is not set up, and just mention that in info message at main app view.

Nextcloud cron.php have rudimentary support to cap cron job execution to 15 minutes (and 15 min is hardcoded), but it is per app (once $job->execute() method is called, it can last as long as indefinitely). It is our job to cap ourselves and to bail after 10-14 minutes of work.

Background processing should be split into separate smaller tasks, where each task is:

- idempotent (yields same results for same inputs)
- detectable to be finished (so we can skip it in next run and not execute it again) (this is provided by DB, but we need to investigate whether all DB vendors on top of NC can work do support proper transaction support. If not, this assumption needs to be changed), and
- is guaranteed to progress (any error in any task should never block or stop processing – errors are just logged, and next step is executed)

In theory, we could parallelize some stuff, like obtaining face descriptor. Do we want to?

Background processing should look something like this:

1. Check if pdlib is installed. Bail out if it isn't
2. Check if we are started from cron job, or from app command. If former, **and** if cron job is disabled, bail out.
3. Take global lock using flock to protect us that there is only one thread doing background processing.
4. Iterate for all users. Check if we should create cluster for someone (enough images processed, but there is no clusters for that user). Enough is defined as "min(count(all user images), 1000 faces)". Cluster is created with following tasks:

Komentar [MI22]: As you mentioned before, the nextcloud applications are very structured, and although you can make a more generic library, I think it would make the code more dirty than what it brings. What part do you imagine that can be generic?

Komentar [BK23R22]: Well, if you remove dependencies to DB and NC API, everything can be in libFaceRecognition. Think of it as stateless library. You provide to it some persons/faces and let it crunch the rest. For example, libFaceRecgontion API can be (just example):

```
fr = new libFaceRecogntion()
fr.insertExisting(persons, faces)
fr.addFace(face)
fr.addAllFacesFromImage(image)
fr.do_processing(15min timeout)
fr.create_new_clusters()
```

So, Nextcloud Facerecogntion app will get persons/faces from DB and feed everything to libFaceRecognition. It then read data from there and updates DB. Somewhat similar to https://github.com/ageitgey/face_recognition. I still didn't think a lot of API, but I hope you see where I am going with this idea.

Komentar [MI24R22]: Yeah that's right. I would like to avoid the pdlib layer, for the reasons explained above, but I'll wait to see how it turns out..

Komentar [MI25]: Perfect. Ajax is discarded. webcron is questionable and cron should consider some operating limit. as you mention
I'm looking at alternatives such as mailing the activity application.

Komentar [MI26R25]: I refer only to the configuration to send the summary mail..

Komentar [BK27R25]: Can you explain „maining the activity application"? I don't get it? Maybe this is something to be in 8. Extended functionalities)?

Komentar [MI28]: Well..
Clustering may give different results, but in the current approach it is only used once as an initial grouping. and then it evolves according to the changes of the user

Komentar [BK29R28]: UserCluster.is_valid (in DB) should block „regeneration" of clusters if nothing is changed

Komentar [MI30]: We have to use the Settings or in the same database as a flag to know the state

Komentar [MI31]: How to want, it would be interesting, but a complexity that I will not prioritize. What I was thinking, is to analyze every certain amount of photos, an

Komentar [BK32R31]: Of course, first thing first. First make it work, than make it correct, than make it fast☺ I am also worried that if we are „too good", we can hog whole machine☺

Komentar [BK33]: Any other implementation of global lock in PHP?

Komentar [BK34]: Fix this, this is not correct, it doesn't make sense to compare images and faces☺

    a. Start chinese whispers clustering on what we got up to now
    b. Once clusters are found, insert them into DB, attach generic names to them ("person-<N>"). Find N by getting all „person-<N>" for a given user, finding maximum and using N+1 for next one. This seems OK solution, as it doesn't fill the gaps (maybe user was getting used that „person-36" is best friend). If we can make N same as primary key for person table, that would be better solution, but person DB table have composite primary key.
5. Iterate for all files, for all users. Find those files which are eligible images, which are not in DB (for currently selected model) and insert them in DB. Split commits to, let's say 1000-10000, so we don't waste transaction log and we are guaranteed to progress. What are file types we will support?
6. Enumerate all eligible images (from DB) to find faces in them (these are all images that do not have face descriptor *and* do not have errors during processing).
7. Shuffles all those images (shuffling is important, as we don't want to generate clusters from user's images if we process only directory "Alabama 2016" and "Athens 2017", or from only images from one period of life (only baby images, and no images as toddler/kid). See also section about "Processing order".
8. For each image from previous set:
    a. Get image dimensions. If any dimension is >1024px, shrink image up to 1024 pixels keeping aspect ratio (this is dependent of memory of the host). Of course, do this in /tmp (or whatever host OS equivalent is). Double check this requirement, maybe when I did this, I did this for nothing, maybe dlib handles images of 10000x10000.
    b. Get image EXIF data (if there is and file type supports it). Rotate image if EXIF says so (I am pretty sure that dlib doesn't do this under the hood, and I am fairly confident that this can only help underlying neural network, but I didn't experiment, maybe I is not needed)
    c. Detect all faces on image by doing face detection and shape prediction. Do we need upsampling ever? Dlib recommends it, but it is not intuitive for me why is it needed
    d. BEGIN DB TRANSACTION
    e. For each found face:
        i. Transfer coordinates back (if image was resized and/or rotated)
        ii. Obtain face descriptor.
        iii. Write face descriptor to DB
    f. Write that image is processed (even if there is no faces found). If there is any error during image processing, write that too.
    g. COMMIT. Only now we can commit, as we rely on DB, to provide us state (we can split this even more atomically, so each face descriptor is task on its own, but not sure if there is a need to complicate stuff here, unless proven that it is not working for someone).
9. Iterate for all users. For each user:
    a. Find all faces that are in DB, but are not in user's clusters. If we detect more than 10 faces like this, or if more than 2h since any of these is passed, or if "is_valid" (UserCluster table) is *false*, start new round of clustering for that user.
    b. When new clusters are obtained, merge them to existing cluster, using receipt from section Cluster stability.

Note that cluster creation and getting faces are reversed in here. Logically, it should not matter, but user experience will be better if she/he gets cluster before all images are processed.

Once this is all developed, we should measure common case, where common case is that there is no changes whatsoever and that there is no operations needed. This common case should execute very fast and utilize only DB.

5.1 DNN models and algorithms

We use *regular*[1] dlib model for face detection. Note that we should use CNN model, not HOG model, as it is more expensive, but accuracy is far better – when it comes to use cases like Nextcloud FaceRecognition, it is far better to spend more time, but get better results.

We also use *regular*[1] dlib model for face shape prediction and *regular* dlib model for getting face descriptions.

We use Chinese whispers clustering for obtaining clusters.

We need a way to handle upgrades to new models and to new clustering algorithms (this includes both new algorithms and changes in parameters of old algorithms).

Proposal to handle changes to models is to have additional field in DB that tells with which model particular face is detected and face descriptor obtained (ideally, we could have two different columns, but this simplifies things a bit). Since administrator choose current model, model change is destructive operation that destroys all additional data that user created for old model (names of clusters…). Model should be primary key in DB, so that it can remain in DB, in case admin reverts to older model.

How to cope with changes in clustering algorithm, seems like a very hard problem, but it also seems we will change clustering algorithm almost never, do we want to bother at all handling thid?

5.2 Processing order

Images should be processed randomly (e.g. no user should have priority for image processing).

❓ Do we need fair scheduler? Like – one that tries to take images from all users in some load balanced fashion (for example – process equal amount of images from all users… or process equal percentage of images from all users)?

5.3 Estimates

Since we want to show user some estimate when app is done, or to just show progress when bulk of new images is added, we need to show something like this:

Processing 123/1456 images (ETA: 5 days 15h)

However, if we don't need all 1456 images to be able to start showing clusters to users (we need first 1000 faces only), question is how to tell that in above message? Possible idea is:

Processing 123/1456 images (ETA: 5 days 15h)

678 more faces needed to start finding people (ETA: 2d 3h)

[1] "Regular" here means that we use standard models, generously computed, provided and recommended by dlib itself, for example here and here.

There is also a question how to measure that estimated time. We need average time per image (and average time per face[2]). However, this is not per user, this is per machine running Nextcloud. There are two approaches I can think of:

- since we should just be aware of total process time and total images processed (to calculate average time to process image), it can be saved in main app settings DB table (how is this called in Nextcloud terminology). This seems like a logical solution, however, whenever image is processed, we need to update app DB table to increment total process time and total images processed, which IMHO sucks. But we do get average time in O(1).
- For each processed image, we add to corresponding DB table a column named "processing_duration". This column is than same as tagging that image is processed (although I would not use this to signify that). We don't have additional updates, but we have harder time to figure out average, in O(n). On the plus side, we could also get medians, if we ever want to show that (instead of averages). We can also get better control of what is average time if we include processing with and without errors.

I am more in favor of second one.

Do we need median times or is average OK? I think average is proper.

5.4 Cluster stability

This is, IMHO, most difficult problem in this space. We need to create clusters of faces, grouping them to "persons", but this cluster is dynamic, living thing. We need to add faces to it, and faces can get removed from it. However, concept of logical person doesn't change, it must stay the same all the time! We don't want to end up in situation where user sees a cluster of his husband, renames it to his name, and after one additional image is added, this cluster doesn't exist anymore, or it is split in two clusters, or its name is not as user renamed it…This is why this is called "cluster stability".

To simplify, we basically have two main operations that affect clusters:

- initial creation of cluster, and
- addition of faces to it.

Now, we can either have **one** algorithm to handle both operations, or **two** algorithms. In former case, we essentially recreate cluster each time new face is found (let's call this "**continuous cluster rebuild**"). In latter case, we use other algorithm to determine proper cluster to add face (or no cluster), (let's call this "**incremental cluster build**")! Both approaches have good and bad sides, and choosing here affects this whole specification, DB schema and, at the end, usability of this app!

5.4.1 Continuous cluster rebuild

Chinese whispers is a stable algorithm (seems so, no data to back this up). We could fire it again and again, whenever new image is added/modified/deleted. It can be expensive, but it is bound in memory and CPU usage, so performance might not be that much of a problem (and we can mitigate it a bit by waiting for multiple faces before firing it again…). Bigger problem is how to keep the clusters stable. We need a way to detect "diff" of clusters between runs of Chinese whispers. Put it another way – to merge new run of Chinese whispers to existing clusters. Imagine we have following clusters:

Person ID	Faces IDs
1	1,2,3,4 (largest cluster)

[2] This is important, if user have 99% of scanned receipts, average time per image will give wrong estimate how long it takes to find face☺

2	5,6,7 (middle cluster)
3	8 (smallest cluster)

and we add new face (id 9). Here are some of the outputs we can get from Chinese whispers[3]:

Faces IDs	`[1,2,3,4,5,6,7,8,9]`	
Output clusters A	`[2,2,2,2,0,0,0,1,2]`	New face assigned to largest cluster
Output clusters B	`[2,2,2,2,1,1,1,3,0]`	New face forms new cluster on its own
Output clusters C	`[0,0,0,0,1,1,1,2,2]`	New face assigned to smallest cluster
Output clusters D	`[1,1,1,1,0,0,2,3,2]`	New face managed to split middle cluster
Output clusters X	`[0,1,2,3,4,5,6,7,8]`	Chinese whispers take a mind of its own☺

As one can see, there are endless options. We need algorithm that can merge new outputs to old clusters. If we can get that right, this, "continuous cluster rebuild" option could be a way to go! Downside of this approach is that, there is no easy way to manually split clusters (e.g. if cluster merge two persons together, user cannot say "hey, these are actually two persons", we are at "mercy" of Chinese whispers to do the right thing). This could be mitigated in future by introducing another layer of indirections, on top of basic persons layer.

Table above serve only as an example, we should take into account when multiple faces are added, or when faces get deleted…

5.4.1.1 Algorithm implementation

Goal of this algorithm is to preserve existing person IDs (cluster IDs) even after new runs of Chinese whispers. Note that this algorithm is best effort. Idea of the algorithm is to count which new clusters "looks like" most to old clusters (have majority of old faces) and pin them to old ones. Here is one approach how this could be possible.

Input:

- old_cluster: `dict<int, list<int>>` (list of person IDs which contains list of face IDs), example:

```
{"1": [1,2,3,4], "2": [5,6], "3": [7], "4": [8]}
```

- new_cluster: `dict<int, list<int>>` (same thing, but after Chinese whispers run, list of person IDs which contains list of face IDs), example:

```
{"1": [8,9], "2": [1,2,3], "3": [4,5,6], "4": [7]}
```

Output: `dict<int, list<int>>`, list of person IDs which contains list of face IDs (but with merging old and new person IDs, keeping old person IDs), example:

```
{"1": [1,2,3], "2": [5,6], "3": [7], "4": [8,9]}
```

In example above, one can observe two things happening:

- new face 9 (id: 9) was added
- clustering is executed

[3] For people not knowing details of Chinese whispers output, it returns vector where index is input vector (position of a given face description in input vectors), and value for that index is cluster (not person ID), starting from 0.

- face 9 was added to existing cluster where face 8 is already
- face 4 moved from cluster with faces 1 and 2, to faces 5 and 6
- new clustering mixed all existing clusters

Algorithm steps:

1. For each face, create a map of person transitions. Looking at example (where keys are face IDs, values are pairs of old and new person IDs):

```
{"1": [1,2], "2": [1,2], "3": [1,2], "4": [1,3], "5": [2,3], "6": [2,3],
"7": [3,4], "8": [4,1], "9": [null,1]}
```

2. Count obtained pairs (keys are transitions, values are counts). From example:

```
{"[1,2]": 3, "[2,3]": 2, "[1,3]": 1, "[3,4]": 1, "[4,1]": 1, "[null,1]": 1}
```

3. Create map of new person to old person transitions, based on counts obtained in step 2. If it is a tie, pick random one. Example (keys are new person IDs, values are old person IDs):

```
{2: 1, 3: 2, 4: 3, 1: 4}
```

4. Replace new person IDs with old person IDs in `new_cluster`. If there is new clusters (non-replaced), keep their ID (or make sure it is not in use), example:

```
{"1": [8,9], "2": [1,2,3], "3": [4,5,6], "4": [7]}

                          ↓

{"4": [8,9], "1": [1,2,3], "2": [5,6], "3": [7]}
```

5.4.2 Incremental cluster build

In this case, we have one algorithm that creates initial clusters (Chinese whispers), and another method/algorithm that continues to build clusters as faces are added/modified/deleted. It is very easy to see that we have to be very careful that second algorithm should be equivalent operation to recreating cluster from scratch. E.g. cluster to which face is assigned should be the same as that face was in initial set of faces that created all clusters. Note that it might not be hard requirement, but we should know that clustering depends in great length to initial conditions (if we start with 1000 faces and add 1000 more later, or we start with 1500 faces and add 500 more later, we could end up with completely different clusters!).

Implementation of this approach looks very tricky, as clusters seem to be very fluid. We need a metric to tell us how close new face is to existing clusters. This can be either by:

- Calculating distance to *centroid* face descriptor (average vector obtained out of all faces/face descriptors in a cluster)
- Calculating distance to each face descriptor in all clusters and taking minimum/5% percentile from each cluster
- Calculating distance to N random faces in each cluster (and doing one of the above)
- Calculating distance to one (or more) “pinned” faces (that user set)

My analysis (personal data, not included here☺) tells me that distances can get ≤0.5 in first two cases (for multiple clusters)[4]! In lot of cases, minimum distance does not even match the proper cluster! I didn't try with "pinned" faces, but I suspect that we rely on human to give us those pinned faces, and what is obvious and "best" face for human, not necessarily means it is best face for algorithm. It also adds penalty that user must be involved in this whole business of cluster creation.

Even if we get this right, next challenge is to determine whether new face should belong to any of existing clusters, or form new cluster. If my analysis is correct, if we have more than 5 clusters, probability to get any cluster to have minimum distance, or centroid distance ≤0.5 is ≥80%. This means we can not reliable figure out if this is existing person or not, and we will be biased to create giant clusters.

When face is deleted, it is also not clear should we "rebalance" clusters, or just leave it as-is. I guess latter.

IMHO, this approach ("incremental cluster build") is fool's errand and should not be taken.

5.5 New/change image processing

App should contain callbacks when file is modified, but it was debated whether we need callback when file is added.

Does this include shared images? Do we cluster those too? What happens when someone remove us from share? If image belongs to multiple users, do we have one entry for it in DB per user? Can we optimize face detection and obtaining face description, if there are?

Komentar [MI35]: If it is shown in the gallery, it should be analyzed.
If they stop sharing, it should be removed from the database..
If you can avoid analyzing twice, welcome, but one entry must remain for each user...

5.5.1 File addition callback

In background tasks, we are having step 5 (iterate filesystem and add images). We can either rely on this step for file addition, or rely on Nextcloud file addition callback.

Pros for having step 5:

- Do not rely on file addition callback (bugs, accidental misses…)
- Always eventually consistent
- No blocking of file addition callback

Cons of having step 5:

- Additional implementation burden on our side
- Heavy load every 15 minutes just to iterate for all files

Current proposal is to fully rely on Nextcloud file addition. Step 5 is required after application is installed for the first time only (and after reinstall). We are thinking to introduce step 5 if we notice flakiness on file addition callback, but with decreased frequency (not every background job, 15 minutes, but every 1 day or so).

Here is order of operations to be done in file addition callback:

1. Check file type of modified file (same code for as in background step, step 5)

[4] It is very easy to replicate this. Take your 1000s of images. Find faces/face descriptors. Put them in Chinese whispers and get clusters. Now, take any of the face from any cluster. Forget in which cluster it is (technically, move it from "training" to "test" data set☺). Now, try to find proper cluster this way☹

2. If it is image, add to to database (“Image” table). Heavy log on any error, as this is our only chance to process this file.

Note that this operation here should not involve global lock.

5.5.2 File modification callback

When file is modified, we need to have callback (other option is to have some hashes and always calculated hashes in background processing, but this is very expensive). Order of operations is following:

1. Check file type of modified file (same code for as in background step, step 5)
2. If it is image, check if it is in DB. If it isn’t, it means it is still not being inserted. Just bail out. Since we are not holding lock here, there is chance that, for some DB transaction isolation levels, that we miss this file modification completely! Or not☺, didn’t get into details, just wanted to turn attention.
3. If it is in DB, delete all faces assigned to it (found for this image) in DB and reset its state in DB, so that background process can pick it up again for analysis. Also reset error column in DB (rationale: modifying imaged might remove error that was present before). No need to reset “processing_duration”, I think.

Note that this operation here should not involve global lock. We could block file modification and we don’t want that.

What should we do if callback is not working for some file modification (bug in our code, bug in Nextcloud…). We lose those changes forever?, this doesn’t seems like a smartest approach. Can we do better here?

Komentar [MI36]: All nextcloud relies on hooks, but also has scheduled tasks for cleaning links, and updating files, etc.
We can leave another task for cleaning..

5.5.3 File deletion callback

When image is deleted, we should delete image entry in DB, as well as all associated faces. We should also set “is_valid” flag in UserCluster to *false*, to signify that cluster for this user should be regenerated.

5.6 Resource governance

Most of operations mentioned below can use lot of resources (cpu , memory and IO, but not storage). This is true even if GPU is not present. Resource governance (RG) is needed, especially in shared enviroinments (hosted Nextcloud). There are several ways to achieve this:

- Offer a way for admins to turn off background processing with cron + offer Nextcloud admin command instead, and let admins craft their own RG from outside where they will RG php process itself, depending on host OS (cgroups, nice, job objects…)
- Offer some throttling on our side. We can define some speeds, like face detection/minute, face descriptors/minute and chinese whispers (clustering)/minute.

Any other approaches?

Approaches above are **not** mutually exclusive, but since we will already have first, second one can be done in some later dates (lower priority).

6 User operations

All functionalities in regular work

If there is no clusters for a user, we should just show some general info, like Estimates.

Once there are clusters for users, besides new estimates, user is presented with clusters, which consist of names and thumbnails. Who/when/how generates thumbnails? Cluster are shown by count of faces in them.

User can then:

- rename cluster
- join clusters together
- remove faces from cluster
- delete cluster

TBW, need more details

See also section Extended functionalities.

7 Administrative operations

All functionalities exposed to administrator

Administrator of the Nextcloud instance have following operations at her/his disposal:

- Changing model
- Clearing/resetting all data
- Turning support for cron job on/off

TBW, need more details

8 Extended functionalities

Explains additional functionalities that could be possible, once basic face recognition is working

TBW, need more details, just throwing ideas below

- Assign tags/hidden tags
- Leverage automatic upload rules ("when image of kid is uploaded, share with wife")
- Linking other users to our clusters (instead of naming cluster "Joe", we can link it to user "Joe" in our Nextcloud instance)
    - Replace other user's avatar with our face of it, from our images
- What can we leverage from federations?
- We have face rectangles, we can offer user ability to find portraits/large faces in images?

9 Proposed DB schema

FaceRecognition settings (key, value), is this already provided by Nextcloud.

It would be nice to add associated query for most interesting/complex operation in document above, based on this schema proposal.

```
FaceModel
  id: int,
  name: string,
  description: string

UserCluster
```

```
  id: int,
  user_id: int (FK to Nextcloud user),
  is_valid: bool,
  last_generation_time: datetime

Person
  id: int,
  user_cluster_id: int (FK to UserCluster),
  name: string,
  linked_user_id: int (FK to Nextcloud user)


Image
  id: int/guid,
  user_id: int (added later, during coding!)
  file_id/filename:int/string,
  model_id: int (FK to FaceModel),
  is_processed: bool,
  error: string,
  last_process_time: datetime,
  processing_duration: long)

Face
  id: int/guid,
  image_id: int/guid (FK to Image),
  person_id: int (FK to Person),
  left: int,
  right: int,
  top: int,
  bottom: int,
  descriptor: JSON array string/compressed array?,
  creation_time: datetime
```

Komentar [BK37]: Do we need id here, since user_id is already unique?