Arquitectura de Software – Projeto Final

Universidade de Lisboa

Autores

Bernardo De Andrade Félix – IST90788

Matheus de Souza Trindade – IST1105471

08 de Janeiro de 2023

- The primary functional requirements:

1. Upload Video

Actors:

- Authenticated users.

Inputs:

- Focus on small and medium size videos. Maximum allowed video size is 1GB.
- Supported formats: MP4, AVI, WMV, MOV, HEVC.
- Video metadata: Title, description, tags, category, privacy settings.

Outputs:

- Video uploaded to the user's channel rapidly.
- Video stored in formats: MP4, AVI, WMV, MOV, HEVC.
- Video stored in definitions: 360p, 480p, 720p, 1080p, 4K.
- Metadata, Thumbnails and Watermarks stored.

Description:

- Authenticated users can upload videos to their channels.
- The system supports a variety of video resolutions and formats, demonstrating its modifiability.
- During the upload process, users can update video metadata.
- The upload module also encompasses user profile functionalities like creating a profile and user signup, as these are essential for a user to be authenticated and upload content.

2. Watch Video

Actors:

- Registered and unregistered users.

Inputs:

- Video selection by the user.
- Video definition selection: 360p, 480p, 720p, 1080p, 4K.

Outputs:

- Smooth video streaming.
- Video metadata, Thumbnails and Watermarks integrated and displayed.

Description:

- This functionality allows both registered and unregistered users to watch videos.
- The video stream module is optimized for various resolutions and formats, ensuring accessibility and user satisfaction.

- **The primary quality attributes (QA):**

1. **Security (Addressed by scenarios 1 and 2):**

If the system is attacked, the confidentiality and integrity of the videos will be preserved. It is considered the following attacks:

1.1 Denial of Service: Can disrupt the availability of videos, prevent users from uploading content, or even take the entire site offline. Attackers typically overload the server with a flood of internet traffic.

1.2 Unauthorized Access: Can lead to the loss of user trust, legal issues, and the compromise of confidential information, including both user data and proprietary content. Attackers can exploit vulnerabilities in the system's software, deceive users into providing access credentials (like phishing), or gain unauthorized access through other means.

1.3 Interception Attack. An unauthorized entity attempts to intercept and access the data being transmitted.

2. **Performance (Including Usability - addressed by scenario 3):**

2.1 Video Stream Performance

Latency:

Target: 95% of videos should start playing within 500ms of the user's request.

Variations: For the remaining 5%, a higher latency threshold can be allowed, considering factors like user location, internet speed, and server load. This threshold can be divided as follows:
4% - up to 1 second.

1% - up to 5 seconds.

Throughput:

The system should be capable of handling a high volume of simultaneous video streams without degradation in quality. This includes peak times with heavy user traffic, considering 5 million daily active users and 5 videos watched per day, that leads to 25 million videos watched per day, and 1.1 million videos uploaded per hour in average.

2.2 Video Upload Performance

Latency:

Target: 90% of video uploads are expected to complete processing within 5 minutes.

Variations: For the remaining 10%, a higher latency threshold can be allowed, considering factors like video size and resolution, internet speed, and server load. This threshold can be divided as follows:

8% - up to 10 minutes.

2% - up to 15 minutes.

2.3 Performance & Usability – A large percentage of requests come from international users.

Throughput:

The system should support a high volume of simultaneous video uploads. This includes efficiently managing periods of high upload activity. Considering 5 million daily active users, and that 10% of users upload 1 video per day, an average of 500.000 videos are uploaded per day and 21.000 videos are uploaded per hour.

3. **Modifiability (Including Scalability - addressed by scenarios 4 and 5):**

   Considerations:

   The product has 5 million daily active users (DAU).

   Total daily storage space needed: 5 million * 10% * 300 MB = 150TB

   Users watch 5 videos per day.

   3.1 If the bandwidth decreases, indicating a slower connection, the system dynamically reduces the video's resolution to maintain smooth video streaming. In the other hand, if the user's network bandwidth improves, the system increases the video resolution, enhancing the viewing experience with higher video quality.

   3.2 **(Scalability)** With the increase in the number of users, a static scaling won't be able to process all requests efficiently while keeping a low infrastructure cost, mainly during peak times, for example, and that could affect both outputs from the primary requirements, a smooth video stream and a fast video upload. The system will dynamically insert or remove servers depending on the demand of the system.

4. **Availability (Including Reliability - addressed by scenario 6):**

   4.1 If the system crashes, the videos will be available to the users. While addressing this quality, the quality of **reliability** will also be considered, so if the mapped internal components crash the videos won't be corrupted or deleted.

Constraints

1. Usage of CDN to serve videos.
2. Support the following clients: mobile apps, web browsers, and smart TV.

Concerns

1. It is recommended to leverage some of the existing cloud services.
2. Low infrastructure cost

Design Round 1

The purpose of the design round 1:

A video streaming system that supports uploading and streaming of videos. Covers all functionalities, and the quality attribute of security. Performance, usability, modifiability, scalability, reliability and availability will be addressed in the next rounds.

Architectural drivers – Round 1

Constraints

1. Support the following clients: mobile apps, web browsers, and smart TV.

Primary Functional Requirements

1. Upload Video (as detailed initially);
2. Watch Video (as detailed initially);

Primary Quality Attributes

1. **Scenario 1 - Security**

   Source: Another System;

Stimulus: Attack. The system detects an unusually high number of requests per second from various sources, trying to overwhelm the server (denial of service attack);

Artifact: The system;

Environment: Normal Operation;

Response: Data and service are protected from unauthorized access;

Response Measure: 100% of attacks detected and resisted, and no data made vulnerable;

Solution: Implement Rate Limiter (Requests per second are limited), and proper authentication and authorizations.

2. **Scenario 2 - Security**
   Source: Data transmission between the system and external entities;

   Stimulus: Interception Attack. An unauthorized entity attempts to intercept and access the data being transmitted.

   Artifact: The system;

   .

   Environment: Normal Operation. The scenario occurs while the system is actively sending or receiving data over the internet.

   Response: Data is protected from unauthorized access;

   Response Measure: 100% of attacks detected and resisted, and no data made vulnerable;

   Solution: Implement and enforce authentication, authorization and strong encryption (such as TLS/SSL for web traffic, AES for data encryption).

First View – Decomposition

The decomposition starts with the three modules:

1. Stream video – primary requirement.
2. Upload video – primary requirement.
   - 2.1 Video Transcoding – As detailed by the outputs of the primary requirements: upload video and stream video, a video transcoding operation is added to the system. The introduction of a video transcoding, between the clients and the database will allow the system to convert a video format into other formats, to provide the best video stream possible to different devices and bandwidth capabilities. In this way, the video will be available, for example, also in low qualities, even if the video owner uploads it in a high quality, and allows a second user with a low bandwidth to have a smooth stream experience. It is detailed in the following topics:
     - 2.1.1 Inspection – Make sure videos have a good quality and are not malformed. This functionality addresses the output of the primary requirement 'Stream Video', since it supports a smooth video streaming for the user.
     - 2.1.2 Metadata Generator – Metadata can come in containers with video file and audio. It involves title, description, tags, category, privacy settings. The metadata needs to be properly stored to be used in the videos, as part of the output of both primary requirements.
     - 2.1.3 Watermark Generator – An optional feature for the user. The image needs to be properly stored to be used in the videos, as part of the output of both primary requirements.
     - 2.1.4 Thumbnail Generator – An optional feature for the user. Thmbnails can either be uploaded by the user or generated by the system. It needs to be properly stored to be used in the videos, as part of the output of both primary requirements.
     - 2.1.5 Video Encoding – This responsibility is related to the output of the primary requirement 'Upload Video' since the video needs to be available in different formats and definitions. It also influences the input of the primary requirement 'Stream Video', since it allows the user to select the desired format to watch

the video, as well as the output, since a smooth video stream may depend on the automatic adjustment of the video quality of the video based on the bandwidth available, and that requires a proper encoding.

3. User profile management feature - part of primary requirements. Only authenticated users can upload and edit videos in their channels.

    3.1 Create profile, Authentication and Authorization – This is a partial solution for the security issues that may arise, based on Security Scenarios 1 and 2. Unauthorized Access, Data Interception and DOS attacks can take place when proper authentication and authorization are not well designed for the system. The security tactic of **'Authenticate Actors'**, **'Authorize Actors'**, and '**Restrict Login'** are applied. This solution aims to address security issues such as unauthorized access to data. This solution can be bought apart from the rest of the project, from a specialized security company.

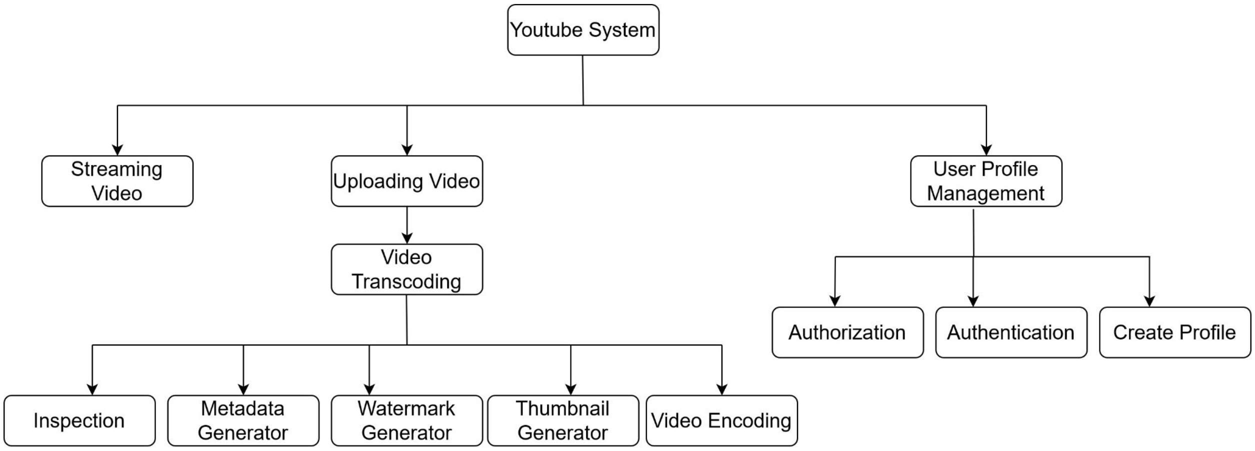
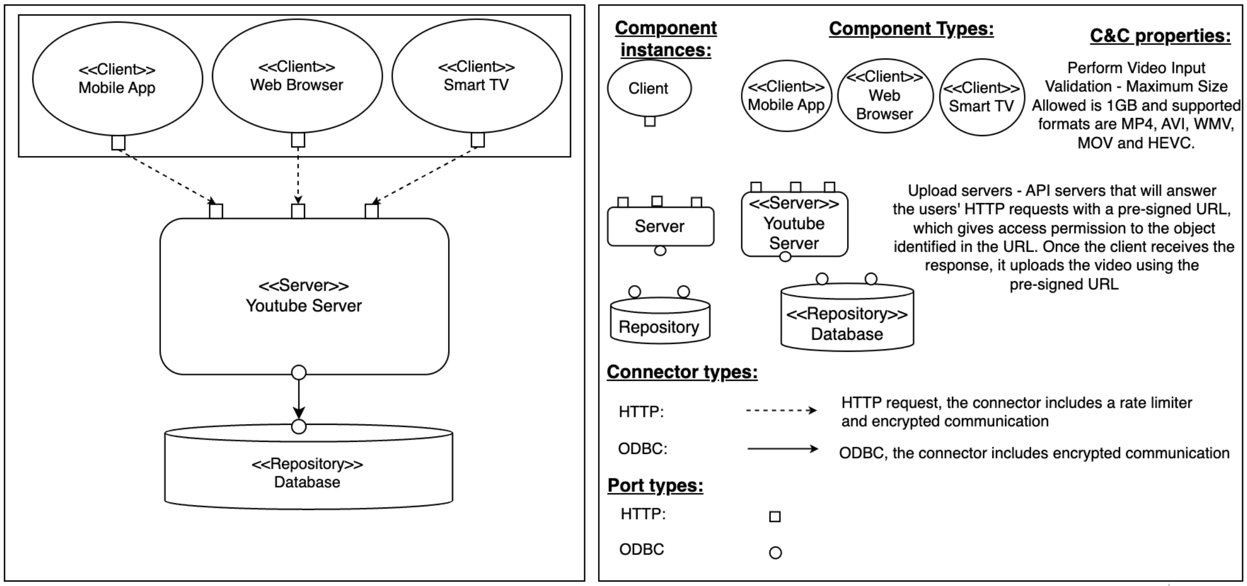


Figure 1. Decomposition viewtype - YouTube System.

Second View – Component-and-Connector

- The instance 'client' has three component types, Mobile App Client, Web Browser Client and Smart TV Client, and it comprehends the constraint that the system should support these three types of clients. The clients will perform an input validation to check the size and format of the videos being uploaded, to make sure that they are within the limit of 1GB established by the input of the primary requirement ‘Upload Video’, and that they are of one of the accepted formats MP4, AVI, WMV, MOV or HEVC.

- The instance of Youtube Server, has one component type, Youtube Server, that will be further explored in the performance scenario. One important aspect involving the Youtube server is that it includes API servers for video upload. These API servers will answer the users' HTTP requests with a pre-signed URL, which gives access permission to the object identified in the URL. Once the client receives the response, it uploads the video using the pre-signed URL. It solves the issue of unauthorized access detailed by the security scenarios 1 and 2, by applying the tactics **'Identify Actors'** and **'Authorize Actors'**. It ensures only authorized users upload videos in the right location.
- The instance Youtube Database will have initially one component type, and this component will be further detailed in the next views.
- The instance that represents the connectors between clients and servers consists of HTTP requests, GET requests for the Stream Primary Requirements and POST requests for the Upload Primary Requirements. A possible issue is a DOS attack through these requests. The security tactics of ‘Detect Intrusion’, ‘Detect Service Denial’ and ‘Limit Access’ are applied. The connector includes a rate limiter as a solution for the Security Scenario 1. This rate limiter will monitor and limit high number of requests per second from various sources (when reaching > 10.000 requests per second, limit to 2.000 requests per second). Another possible issue are interception attacks, for example, user information could be stolen, such as passwords, or even authentication tokens, like session tokens, and this be used to edit videos by unauthorized users. The security tactic of ‘Encrypt Data’ is applied. The connector includes data encryption, to guarantee safe communication between clients and servers.
- The instance that represents the connectors between servers and databases consists of ODBC (Open Database Connectivity) a common protocol for connectivity between servers and databases for various programming environments. One possible issue is interception attacks, for example, database information could be stolen. The security tactic of ‘Encrypt Data’ is applied. The connector includes data encryption, to guarantee safe communication between servers and database.


<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
<<Server>> Youtube Server
<<Repository>> Database
Component instances:
Client
Server
Repository
Component Types:
<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
<<Server>> Youtube Server
<<Repository>> Database
C&C properties:
Connector types:
HTTP:
ODBC:
Port types:
HTTP: □
ODBC ○

Perform Video Input Validation - Maximum Size Allowed is 1GB and supported formats are MP4, AVI, WMV, MOV and HEVC.

Upload servers - API servers that will answer the users' HTTP requests with a pre-signed URL, which gives access permission to the object identified in the URL. Once the client receives the response, it uploads the video using the pre-signed URL

HTTP request, the connector includes a rate limiter and encrypted communication

ODBC, the connector includes encrypted communication

Figure 2. C&C viewtype - Round 1 (left). Component and connectors instances, types and properties (right)

Summary of C&C viewtype of Round 1:

- Shows the different types of clients supported, the servers that support video functionalities and the database to store the uploaded videos.
- Differentiates the connectors used between Client-Server (HTTP) and Server-Database (ODBC) and the ports that accept those connections.
- Details the security aspects of the implementation of API Servers within the Youtube Server.

Design Round 2

The purpose of the design - Round 2:

Ensure performance by achieving latency and throughput for video streaming and uploading as specified initially.

Architectural drivers – Round 2

Primary Functional Requirements

1. Upload Video;
2. Watch Video;

Primary Quality Attributes

Scenario 3 - Performance & Usability:

- **Source:** User Request;
- **Stimulus:** Arrival of an event. The system receives sporadic requests to upload or stream a video from *international users (usability)*;
- **Artifact:** The system;
- **Environment:** Normal Operation;
- **Response:** The system streams or uploads the video, based on the request;
- **Response Measure:** For video streaming, the latency should have as maximum limit 500 ms for 95% of videos played, considering a variation of 1 second for 4% of videos played and 5 seconds for 1% of videos played. For video uploading, the latency should have as maximum limit 5 minutes for 90% of videos uploaded, considering a variation of 10 minutes for 8% of videos uploaded and 15 seconds for 2% of videos uploaded.

Constraints

1. Usage of CDN to serve videos.

Concerns

1. It is recommended to leverage some of the existing cloud services.
2. Low infrastructure cost

Third View – Component-and-Connector

As detailed by the performance scenario, the system receives sporadic requests from international users and has strict latency targets to deliver the primary requirements of streaming and uploading videos efficiently. In this context, it is presented the constraint that CDN should be used to serve videos and the concerns that it is recommended to leverage some of the existing cloud services, aiming a low infrastructure cost.

Iteration 1 - The problems that arise from this scenario are the following:

- A centralized database might not be enough to handle efficiently all the requests from the international users, making both video upload and stream processes slow, that affects the primary requirement output and the latency target established by the performance scenario. The performance tactic **'Maintain multiple copies of data'** is applied, as follows:
    - Videos are cached in CDN, so they can be streamed directly from a CDN close to the user, guaranteeing a smooth video stream, the output of the primary requirement. In order to achieve the low infrastructure cost concern, it will be used Amazon Cloud Service. An eviction policy is established for the CDNs considering that Youtube video streams follow a long-tail distribution, that means that a few popular videos are accessed frequently but many others have few or no viewers. So the CDN will only serve the most popular videos (10%), and the other videos (90%) will be streamed from a high capacity video storage servers. This will consider also video popularity per region, and the statistics will be updated daily.
    - Implement multiple upload center across the globe, so the users can upload their videos to the closest upload center, making sure that they have a fast video upload experience, as detailed by the output of the primary requirement. To achieve this, the CDN will be used as upload centers.
    - A sharded separate database will be implemented for the metadata and for better performance, video metadata and user object will be cached. The eviction policy consists of keeping the metadata associated to the most viewed videos, as detailed by the CDN eviction policy. It will be implemented one master database, for writing, and multiple slave databases, for reading. If one master goes down, a new

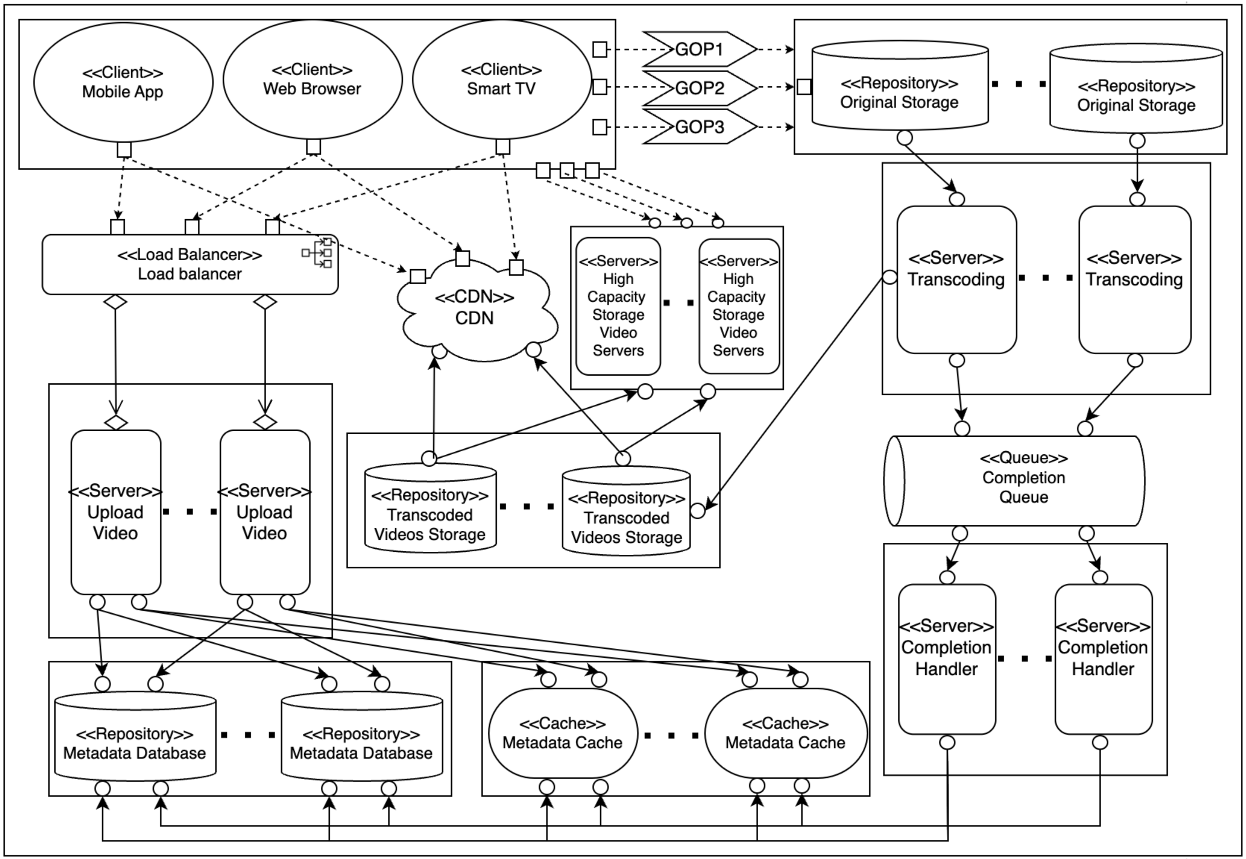
slave will be promoted to master. If one slave goes down, another slave can be used for reads and a spare will be brought to replace the dead one, as will be described in the availability scenario.

- One database won't be enough to store all the data being stored in the original stored and transcoded storage. This performance issue can affect the system as a whole, making it unavailable for uploading and streaming, that affects directly the primary requirements. The tactic introduced is:
    - **'Maintain multiple copies of data'**, by implementing the sharding of the database. The goal is to use many small databases instead of one large database. The User-ID will be used as a sharding key.
- Uploading a video as a whole unit is inefficient. For example, if an upload fails, the system will take a lot of time to resume the whole operation, what affects the primary requirement output and the latency established by the performance scenario. To handle this problem, the performance tactic **'Introduce Concurrency'** is applied, as follows:
    - Video will be split into chunks by GOP alignment, and this can be implemented by the client to improve the upload speed. This allows fast resumable uploads when the previous upload failed.
- One server won't be enough for processing all requests efficiently, and that could affect both outputs from the primary requirements, a smooth video stream and a fast video upload. The following tactics are applied:
    - **'Introduce Concurrency'** - dividing servers per cohesive responsibilities to be able to work in parallel. Servers will be separated to allow independent scaling for video upload and video stream, in order to cover iteration 1 of performance scenario.
    - After that, the performance tactics **'Multiple copies of computation'** is applied. Horizontal scaling is implemented instead of vertical scaling, since buying multiple servers with a considerable low price has a higher cost-benefit than buying an extremely powerful server, beside the fact that no single server could process this incredibly big amount of requests (as described in the performance scenario). In this context, stateless servers are used in this system, and the state information is

saved in the databases. This change supports the concern of ‘Low infrastructure cost’.

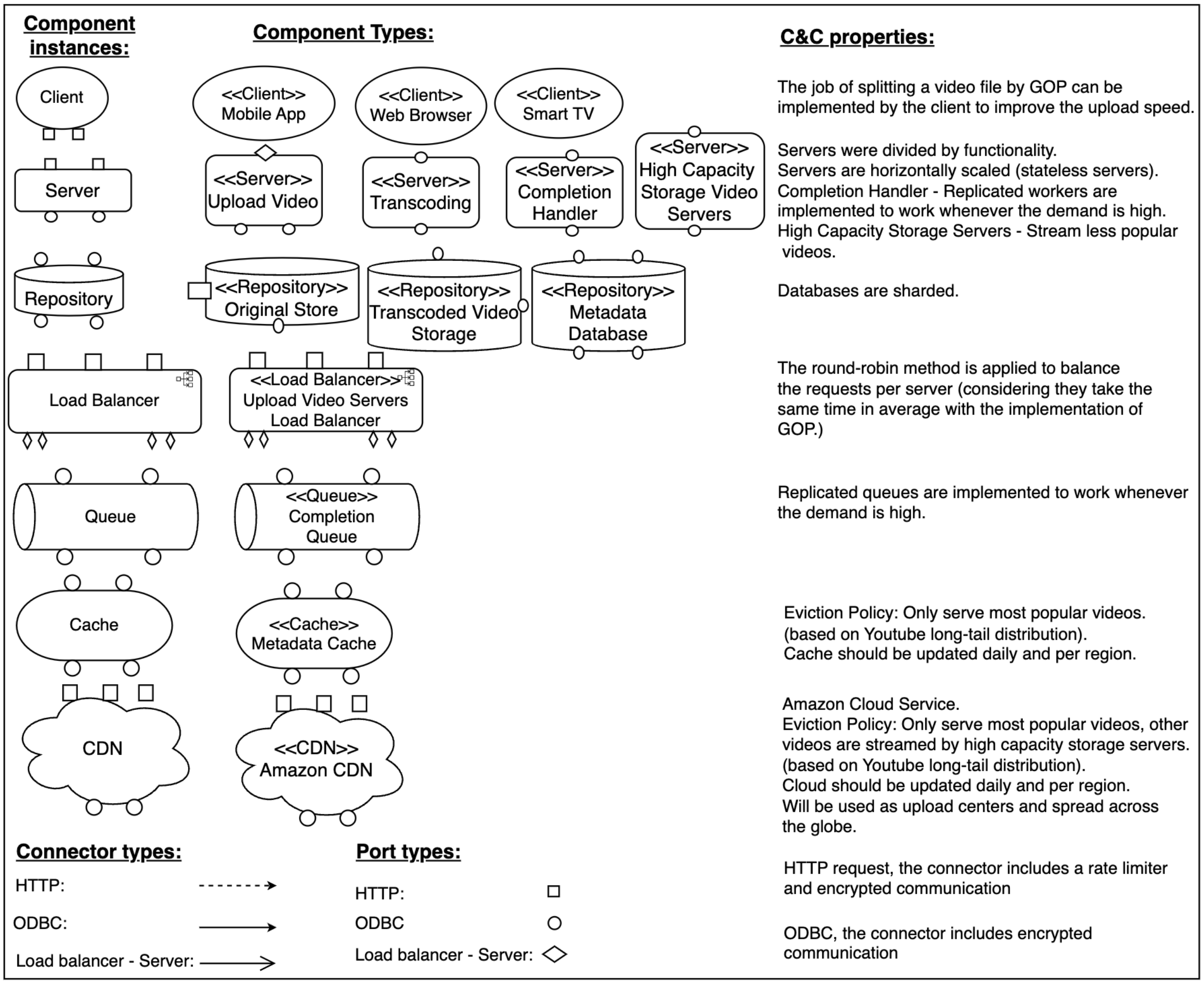
  - A load balancer is introduced, to verify the load of servers and send requests to the servers with smaller load, avoiding the overload of servers and compromising the performance. The round-robin method is applied to balance the requests per server, since requests take more or less the same time in average, considering the implementation of the GOP, as previously described..

- Transcoding a video is computationally expensive and time-consuming. Then, the video transcoding module can become a bottleneck, since it has multiple responsibilities happening in a single thread, what can make the upload process slow, that affects the output of the primary requirement and the latency target established by the performance scenario. The performance tactic **‘Introduce Concurrency’** is applied as follows:
  - To optimize the performance during the storage of metadata, a message queue (completion queue) is implemented to store information about video completion events, so the transcoding servers don’t need to wait until the video data storage process is finished, the job can be executed in parallel. A list of workers (completion handler) is implemented to pull event data from the completion queue and update the metadata cache and database. Parallelism brings speed optimization and makes sure that the system delivers the latency requirements established in the performance scenario. After this step, the tactic **‘Maintain Multiple Copies of Computation’** can be applied as follows:
  - Implement replicated queues and workers that are set to work whenever the work demand is high, so the final latencies can be achieved.



<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
GOP1
GOP2
GOP3
<<Repository>> Original Storage
<<Repository>> Original Storage
<<Load Balancer>> Load balancer
<<CDN>> CDN
<<Server>> High Capacity Storage Video Servers
<<Server>> High Capacity Storage Video Servers
<<Server>> Transcoding
<<Server>> Transcoding
<<Server>> Upload Video
<<Server>> Upload Video
<<Repository>> Transcoded Videos Storage
<<Repository>> Transcoded Videos Storage
<<Queue>> Completion Queue
<<Server>> Completion Handler
<<Server>> Completion Handler
<<Repository>> Metadata Database
<<Repository>> Metadata Database
<<Cache>> Metadata Cache
<<Cache>> Metadata Cache

Figure 3. C&C viewtype - Round 2 - Iteration 1



C&C properties:

The job of splitting a video file by GOP can be implemented by the client to improve the upload speed.

Servers were divided by functionality.
Servers are horizontally scaled (stateless servers).
Completion Handler - Replicated workers are implemented to work whenever the demand is high.
High Capacity Storage Servers - Stream less popular videos.

Databases are sharded.

The round-robin method is applied to balance the requests per server (considering they take the same time in average with the implementation of GOP.)

Replicated queues are implemented to work whenever the demand is high.

Eviction Policy: Only serve most popular videos. (based on Youtube long-tail distribution).
Cache should be updated daily and per region.

Amazon Cloud Service.
Eviction Policy: Only serve most popular videos, other videos are streamed by high capacity storage servers. (based on Youtube long-tail distribution).
Cloud should be updated daily and per region.
Will be used as upload centers and spread across the globe.

HTTP request, the connector includes a rate limiter and encrypted communication

ODBC, the connector includes encrypted communication

Figure 4. C&C viewtype - Round 2 - Iteration 1 (Component and connectors instances, types and properties)

Summary of C&C viewtype of Round 2:

- Include a “Load Balancer” to distribute work efficiently to the different “Upload Video” servers.
- Substitute the “Stream Video Server” instance with an instance of the “CDN” component and High Capacity Storage Video Servers.
- Substitute the “Database” instance with a storage system that supports the existence of different video formats of the same video. The original files are stored in “Original

Storage", transcoded into multiple video formats by the "Transcoding Server" and then stored in "Transcoded Videos Storage".

- Add a "Metadata Database" and "Metadata Cache" to more efficiently serve videos to the "Client".
- To parallelize the work of the "Transcoding Server" while it performs the actual transcoding, a "Completion Queue" and "Completion Handlers" are implemented to update the newly added "Metadata Database" and "Metadata Cache".
- All <<Repository>> instances are scaled to implement the "Multiple Copies of Data" tactic.
- All <<Server>> instances are scaled to implement the "Multiple Copies of Computation tactic".
- Client implements job of splitting video into group of pictures.

Iteration 2 of Performance - Decomposition of the Video Transcoding Module

The bottleneck we are focusing on during this iteration is the inability of the video transcoding module to perform multiple tasks at once, considering the set goals for how quickly it should operate. To manage various video processing tasks simultaneously and keep the system efficient, we need to introduce a way to separate concerns and allow the client's developers to specify the tasks to be done. We have chosen to use a programming approach known as a directed acyclic graph (DAG), which helps us to be adaptable and perform many tasks at the same time. The planned design for converting video formats, which makes use of online cloud services and is a key aspect of this project, is shown in the diagram that follows.

Fourth View - DAG Decomposition

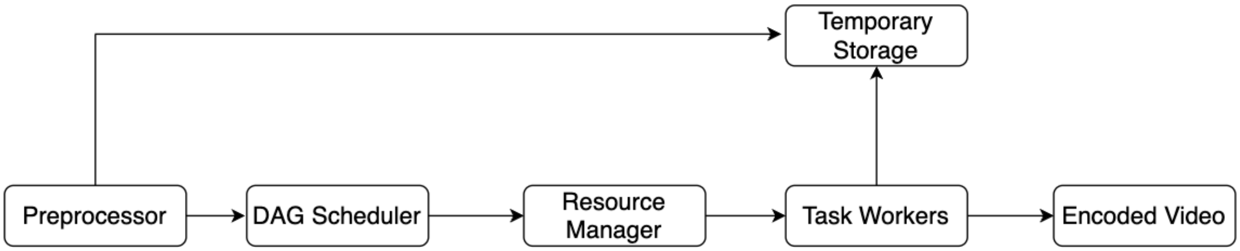


Figure 5. Round 2 - Iteration 2: Decomposition viewtype - Directed Acyclic Graph. Fourth View.

- **Preprocessor -** The preprocessor has the following responsibilities:
  - **Video Splitting -** Video stream is split or further split into smaller Group of Pictures alignment (GOP), an independent playable group of frames arranged in a specific order. The introduction of GOP supports performance, as detailed previously. (Preprocessor split videos by GOP alignment for old clients)
  - **DAG Generation -** The preprocessor generates DAG based on configuration files client programmers write.
  - **Cache Data -** The preprocessor is a cache for segmented videos. For better reliability, the preprocessor stores GOPs and metadata in temporary storage. If video encoding fails, the system could use persisted data for retry operations.
- **DAG Scheduler -** The DAG scheduler splits a DAG graph into stages of tasks and puts them in the task queue in the resource manager. This allows the process to be broken and to have different parts running in parallel, supporting concurrency and improving performance. The original video is split into three strages:
  - **Stage 1:** video, audio and metadata. The video file is further split in two tasks in stage 2:
  - **Stage 2:** video encoding, watermark and thumbnail. The audio file requires audio encoding as part of the stage 2 task.

Fifth View - Resource manager and Task Workers Components & Connectors

- **Resource Manager -** The resource manager is responsible for managing the efficiency of resource allocation. It contains 3 queues and a task scheduler, that introduces concurrency and improves performance, as discussed previously. It is detailed as follows:
  - **Task queue:** It is a priority queue that contains tasks to be executed.
  - **Worker queue:** It is a priority queue that contains worker utilization information.
  - **Running queue:** It contains information about the currently running tasks and workers running the tasks.

  - **Task scheduler:** It picks the optimal task/ worker, and instructs the chosen task worker to execute the job. Then it binds the task/worker info and puts it in the running queue, and removes the job from the running queue once it is done.
- **Task Workers -** Run the tasks that are defined in the DAG. Different task workers may run different tasks, creating concurrency in the system.

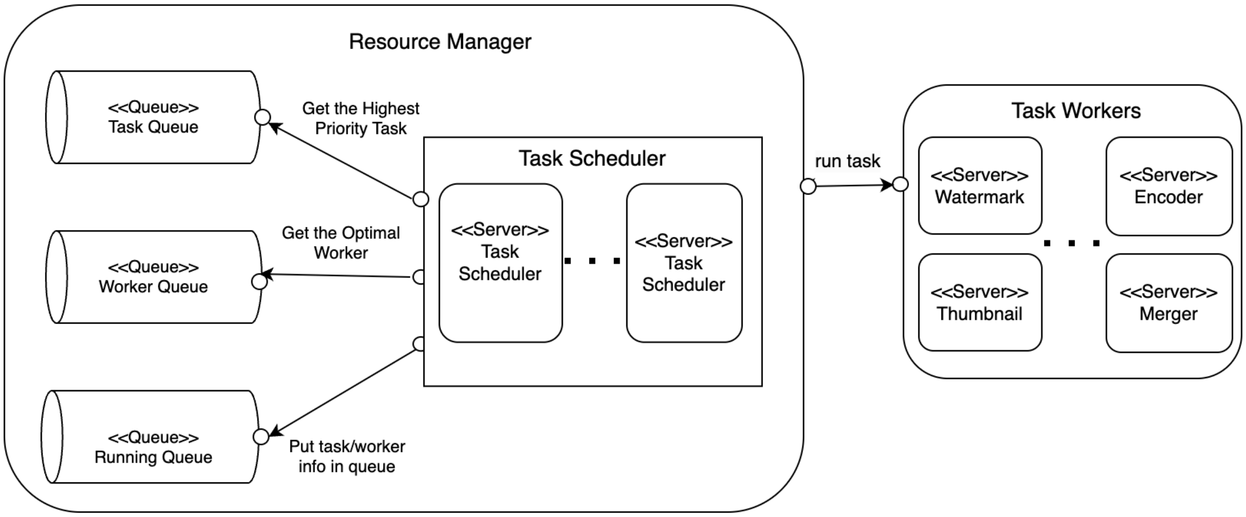


Figure 6. Round 2 - Iteration 2: C&C viewtype - Resource Manager & Task Workers. Fifth View.

- **Temporary Storage -** Metadata is cached in memory, since it is usually accessed by workers and its data size is usually small. Video and audio data are stored in blob storage. Data in temporary storage is freed up once the corresponding video processing is complete.
- **Encoded Video -** Final output of the encoding pipeline.

Iteration 3 of Performance - Improving parallelism in the upload of videos

Sixth View - Component & Connectors (Implementing Additional Queues in Video Upload Flow)

In order to improve the system's parallelism even more, this iteration will focus on a loosely coupled system and the introduction of more concurrency with additional queues implemented within modules.

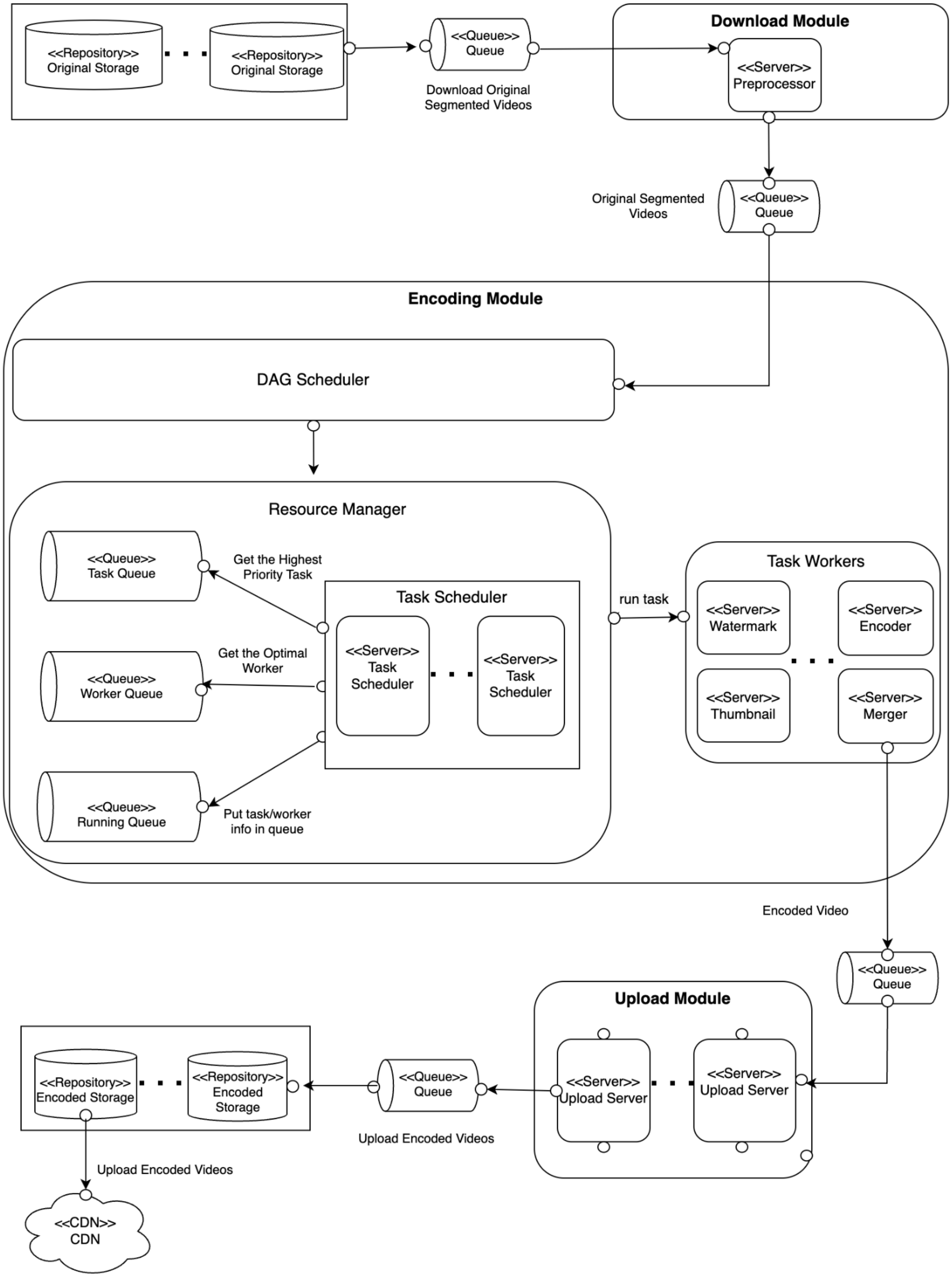


Figure 7. Round 2 - Iteration 3: C&C viewtype - Improving Parallelism. Sixth View.

The flow of how a video is transferred from original storage to the CDN is better detailed in the sixth view. The initial problem identified in this system is that the output of each module depends on the input of the previous step, what is a bottleneck for performance. The tactic applied in this case is **'Introduce Concurrency'**. To make the system more loosely coupled, message queues were introduced as described in the view above (between the original storage and the download module, between the download module and the encoding module, between the encoding module and the upload module, and between the upload module and the encoded storage). After the implementation of the queues, whenever there are events in the message queues, one module doesn't need to wait for the output of the previous module anymore, allowing tasks to be performed in parallel.

Design Round 3

The purpose of the design - Round 3:

Ensure modifiability(scalability), by reducing computational demand caused by increase in the number of users (5 million active users). The system should be able to store all videos (150TB required daily) and handle the increase in the memory demand immediately, without affecting performance. The system should also accommodate variations in the bandwidth and keep a smooth streaming (modifiability).

Primary Functional Requirements

1. Upload Video;
2. Watch Video;

Architectural drivers – Round 3

Concerns

1. Low infrastructure cost

Primary Quality Attributes

Scenario 4 - Modifiability (Scalability - Iteration 1):

- **Source:** The increasing number of requests;
- **Stimulus:** The change that the system needs to accommodate. The system experiences an increase in the number of requests to upload and stream videos. Considering that the number of video uploads is much smaller than the number of streams. (500.000 videos are uploaded per day and 25 million videos watched per day, 150TB of memory required daily)
- **Artifact:** The system;
- **Environment:** Design time;
- **Response:** Deploy modification;
- **Response Measure:** The change is accommodated immediately.
- **Solution:** Implement autoscaling for servers and databases.

Seventh View – Component-and-Connector

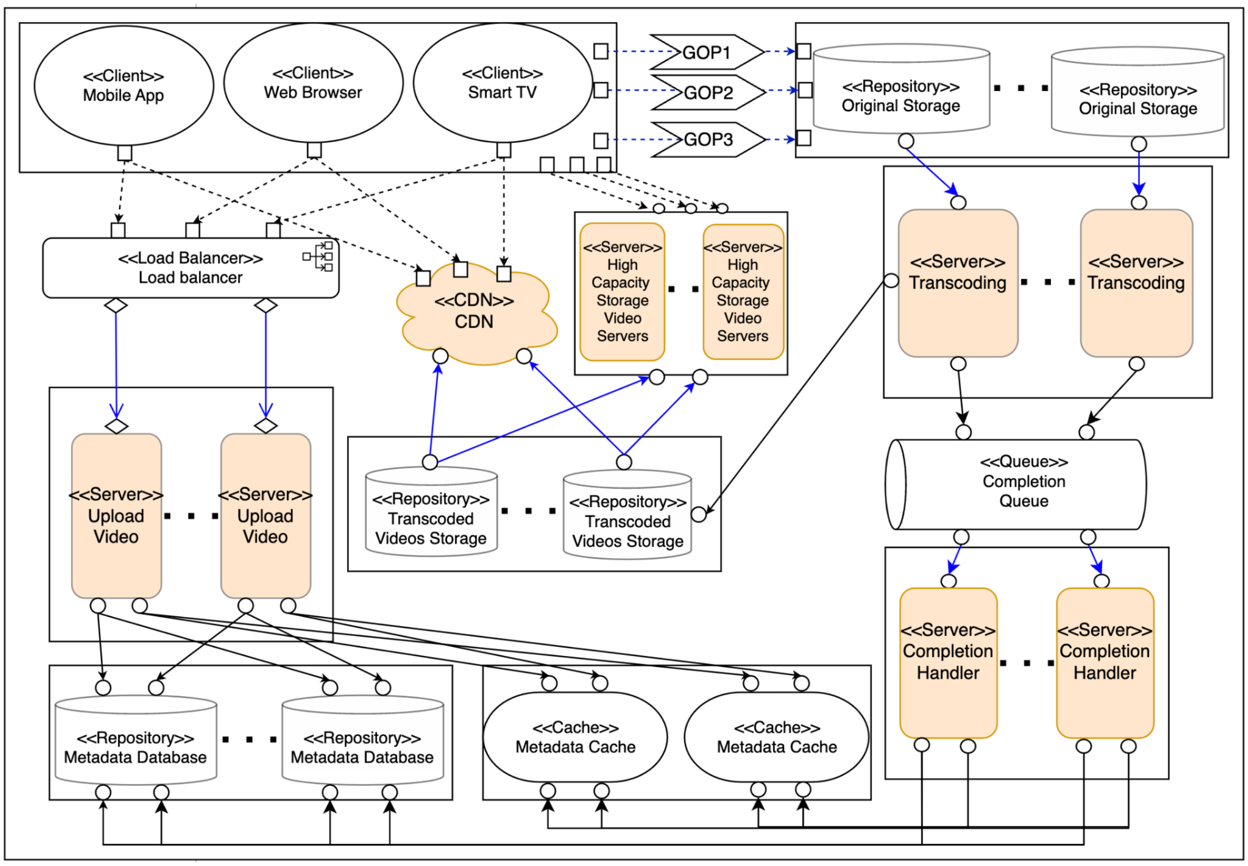
Iteration 1 - As detailed by modifiability scenario - iteration 1, the system has suffered an increased in the amount of requests, and it is noticed that the amount of video uploads is much smaller than the number of streams (500.000 videos are uploaded per day and 25 million videos watched per day, 150TB of memory required daily)

The first problem that arises from this scenario is that a static scaling of servers won't be able to process all requests efficiently with low infrastructure cost, mainly during peak times, for example, and that could affect both outputs from the primary requirements, a smooth video stream and a fast video upload. The modifiability tactic 'Use an Intermediary' is applied as follows:

- Consistent Hashing is applied as an intermediary feature to allow a smooth autoscaling of servers (when a server is added or removed, only a minimal number of requests are redirected) - Stateless servers are dynamically replicated (servers will be added or removed depending on the load). Considering that Youtube has well-known utilization peak times, what can also bring savings to the project, since all servers don't need to be active 100% of the time. In order to remove servers, a time-out of 15 minutes will be given after stop sending requests.

Another problem highlighted by this scenario is that the system might not have enough memory to support the increase in the number of users. Regarding the scaling of databases, the tactic 'Multiple Copies of Data' was adopted in the performance scenario, that refers to the sharding of the database, and that should cover the throughput of 150TB of memory required daily. Differently from the servers, the databases are stateful and implementing autoscaling to these components wouldn't be as simple and cost-effective.

A second important aspect that supports scalability, but was already implemented in the previous performance scenario is the video transcoding feature. A raw video consumes large amounts of storage space, and by transcoding the video, the system is able to save a lot of memory, reduce the need for additional databases and scale more easily the amount of videos being uploaded daily.



<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
GOP1
GOP2
GOP3
<<Repository>> Original Storage
<<Repository>> Original Storage
<<Load Balancer>> Load balancer
<<CDN>> CDN
<<Server>> High Capacity Storage Video Servers
<<Server>> High Capacity Storage Video Servers
<<Server>> Transcoding
<<Server>> Transcoding
<<Queue>> Completion Queue
<<Server>> Upload Video
<<Server>> Upload Video
<<Repository>> Transcoded Videos Storage
<<Repository>> Transcoded Videos Storage
<<Server>> Completion Handler
<<Server>> Completion Handler
<<Repository>> Metadata Database
<<Repository>> Metadata Database
<<Cache>> Metadata Cache
<<Cache>> Metadata Cache

Figure 8. C&C viewtype - Round 3 - Modifiability - Iteration 1. Seventh View.

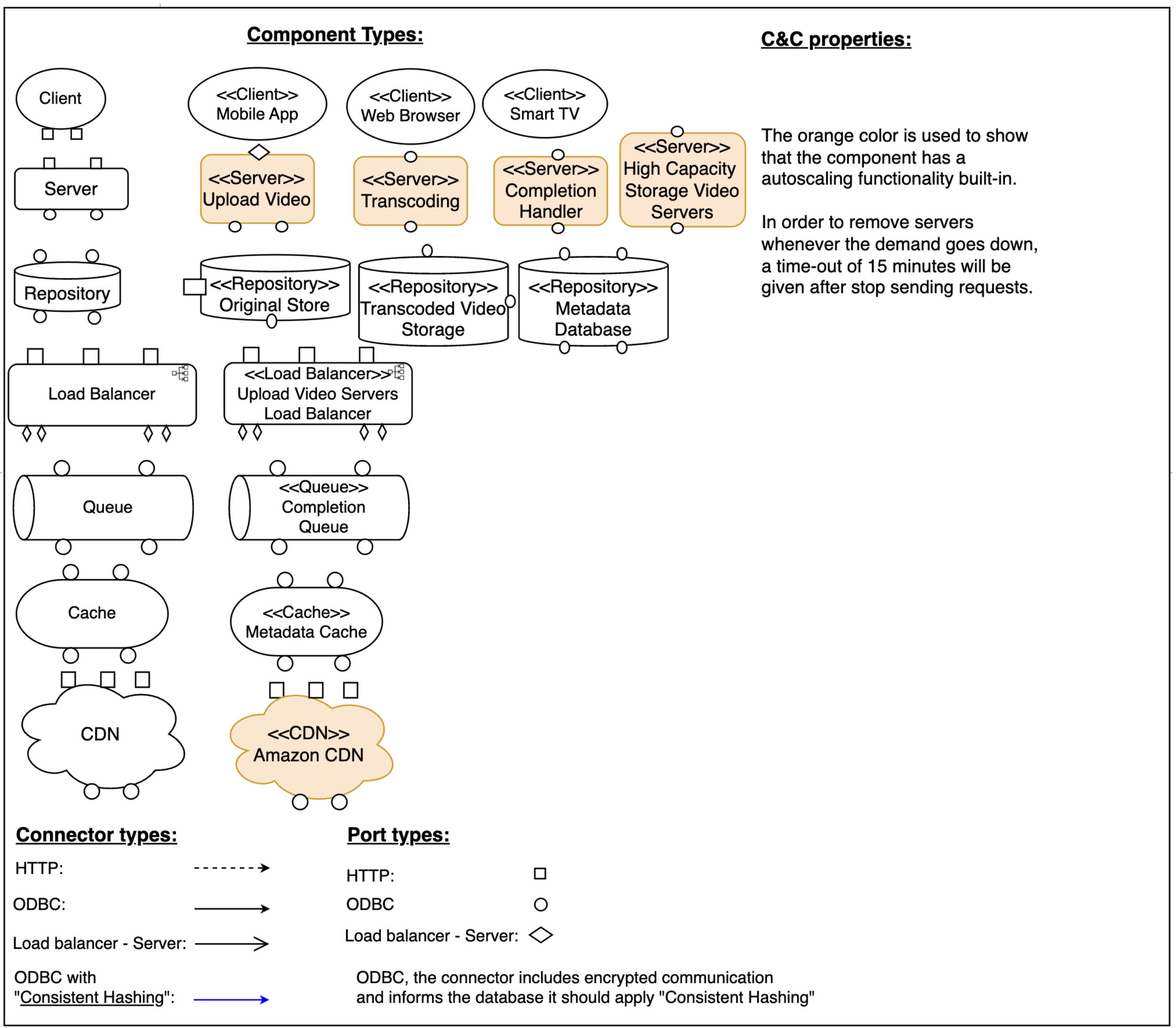

Component Types:
C&C properties:
Client
<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
Server
<<Server>> Upload Video
<<Server>> Transcoding
<<Server>> Completion Handler
<<Server>> High Capacity Storage Video Servers
The orange color is used to show that the component has a autoscaling functionality built-in.
In order to remove servers whenever the demand goes down, a time-out of 15 minutes will be given after stop sending requests.
Repository
<<Repository>> Original Store
<<Repository>> Transcoded Video Storage
<<Repository>> Metadata Database
Load Balancer
<<Load Balancer>> Upload Video Servers Load Balancer
Queue
<<Queue>> Completion Queue
Cache
<<Cache>> Metadata Cache
CDN
<<CDN>> Amazon CDN
Connector types:
HTTP:
ODBC:
Load balancer - Server:
ODBC with "Consistent Hashing":
Port types:
HTTP:
ODBC
Load balancer - Server:
ODBC, the connector includes encrypted communication and informs the database it should apply "Consistent Hashing"

Figure 9. C&C viewtype - Round 3 – Iteration 1(Component and connectors instances, types and properties)

Summary of C&C viewtype of Round 3:

- To signal the <<Repository>> instances that they should apply "Consistent Hashing" to the data being stored, the connectors connecting the servers to the databases have been changed to a blue color to depict this tactic being applied.
- In order to showcase the ability to autoscale, the components that are supposed to scale horizontally have been changed to a orange/brown color to show that more instances of these components can be added or removed depending on the load.

Scenario 5 - Modifiability (Iteration 2):

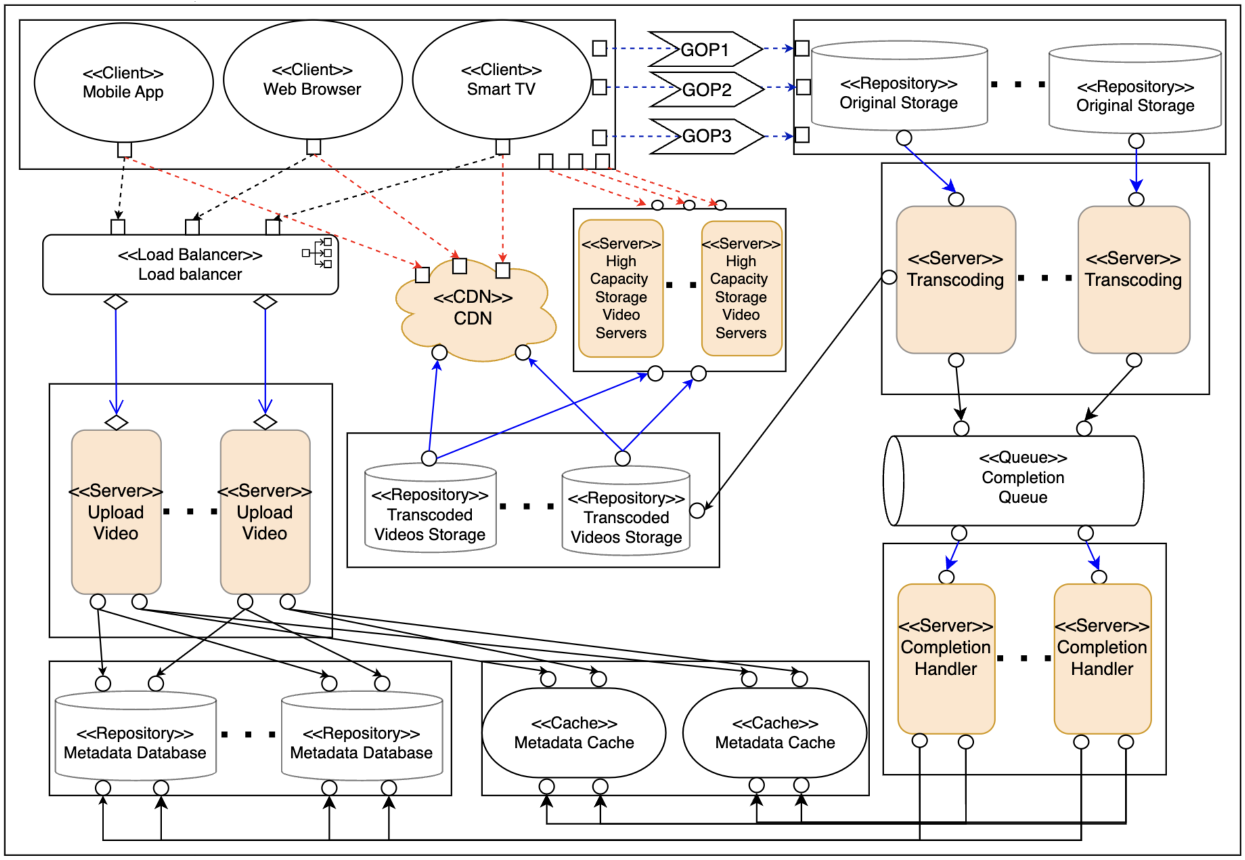
- **Source:** End user bandwidth;
- **Stimulus:** The change that the system needs to accommodate. The end user experiences changes in the bandwidth, that may decrease or increase.
- **Artifact:** The system;
- **Environment:** Runtime;
- **Response:** Deploy the modification;
- **Response Measure:** The change is accommodated immediately;
- **Solution:** Implement a video streaming protocol to support changes in bandwidth. (MPEG-DASH video streaming protocol). Videos should be encoded at multiple resolutions.

Eighth View – Component-and-Connector

Iteration 2 - As detailed by Modifiability Scenario - Iteration 2, the user suffers from changes in the bandwidth, that may decrease or increase.

The problem that arises from this scenario is that whenever the bandwidth decreases, the video stream quality will be compromised, presenting delays to load, for example, and affecting the output of the primary requirement of video stream. The modifiability tactics introduced are:

- ‘Defer Binding – Interpret Parameters’ is applied. A connector between the video server and the client will have the capacity to monitor the user’s bandwidth and adapt the video quality based on it, in order to preserve a smooth video stream. It is deferring the decision about which video quality to deliver until runtime, based on the user's current bandwidth. The technology that will be applied is the MSS (Microsoft Smooth Streaming): Smooth Streaming is Microsoft's implementation of Adaptative Bitrate Streaming. It adjusts video playback in real time based on the user's current network and local PC conditions. As described in the first decomposition view, the video transcoding is implemented and supports this functionality by providing the video in different definitions.



<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
GOP1
GOP2
GOP3
<<Repository>> Original Storage
<<Repository>> Original Storage
<<Load Balancer>> Load balancer
<<CDN>> CDN
<<Server>> High Capacity Storage Video Servers
<<Server>> High Capacity Storage Video Servers
<<Server>> Transcoding
<<Server>> Transcoding
<<Server>> Upload Video
<<Server>> Upload Video
<<Repository>> Transcoded Videos Storage
<<Repository>> Transcoded Videos Storage
<<Queue>> Completion Queue
<<Server>> Completion Handler
<<Server>> Completion Handler
<<Repository>> Metadata Database
<<Repository>> Metadata Database
<<Cache>> Metadata Cache
<<Cache>> Metadata Cache

Figure 10. C&C viewtype - Round 3 – Iteration 2. Eighth View.

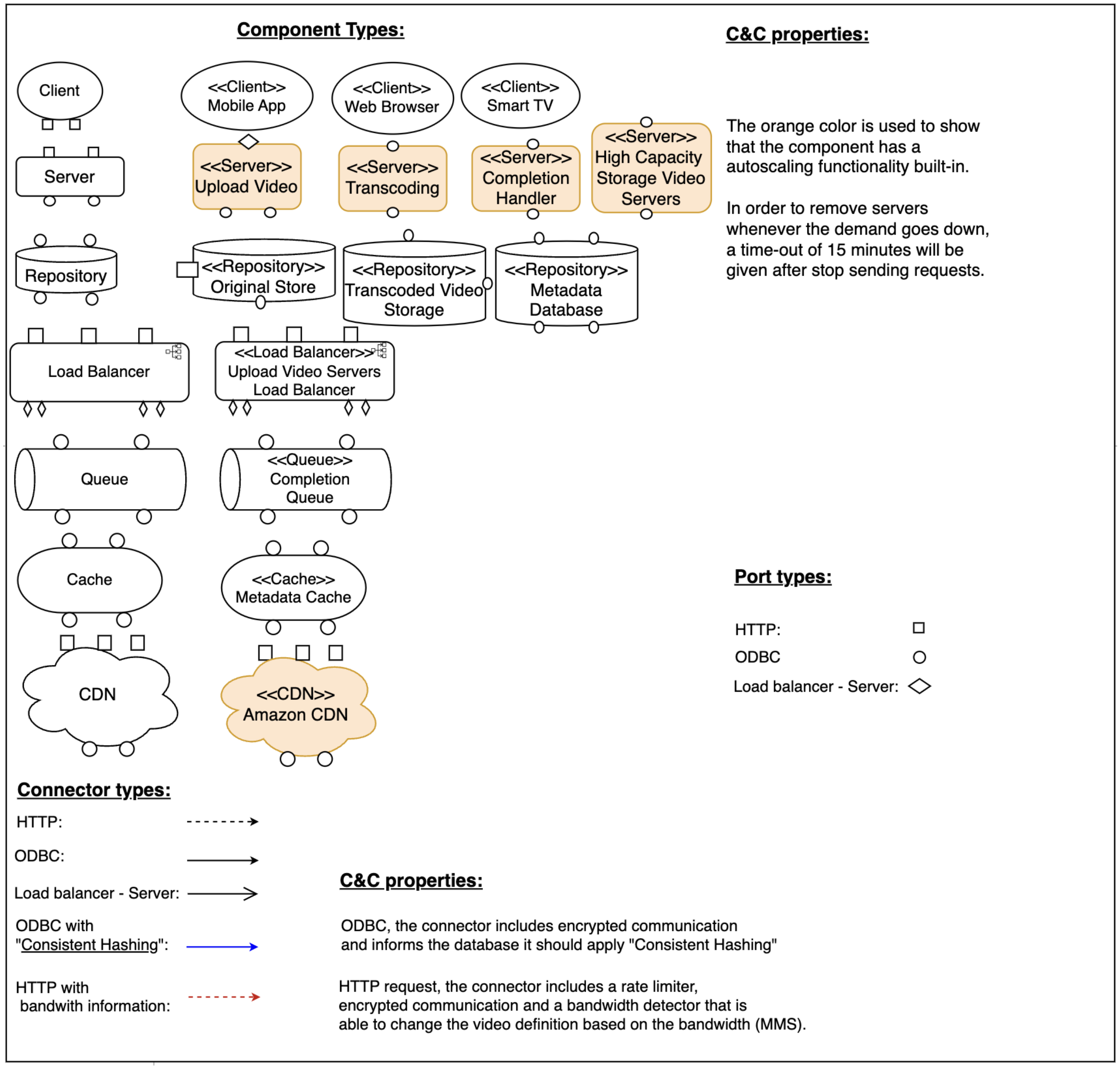

Component Types:
C&C properties:
Client
<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
Server
<<Server>> Upload Video
<<Server>> Transcoding
<<Server>> Completion Handler
<<Server>> High Capacity Storage Video Servers
The orange color is used to show that the component has a autoscaling functionality built-in.
In order to remove servers whenever the demand goes down, a time-out of 15 minutes will be given after stop sending requests.
Repository
<<Repository>> Original Store
<<Repository>> Transcoded Video Storage
<<Repository>> Metadata Database
Load Balancer
<<Load Balancer>> Upload Video Servers Load Balancer
Queue
<<Queue>> Completion Queue
Cache
<<Cache>> Metadata Cache
CDN
<<CDN>> Amazon CDN
Port types:
HTTP:
ODBC
Load balancer - Server:
Connector types:
HTTP:
ODBC:
Load balancer - Server:
ODBC with "Consistent Hashing":
HTTP with bandwith information:
C&C properties:
ODBC, the connector includes encrypted communication and informs the database it should apply "Consistent Hashing"
HTTP request, the connector includes a rate limiter, encrypted communication and a bandwidth detector that is able to change the video definition based on the bandwidth (MMS).

Figure 11. C&C viewtype - Round 3 – Iteration 2 (Component and connectors instances, types and properties)

Summary of C&C viewtype of Round 4:

- In order to apply the tactic "Defer Binding – Interpret Parameters" in the "CDN", the connection between <<Client>> and <<CDN>> should have a bandwidth detector, that is capable of changing the video definition based on the user's bandwidth to guarantee a smooth streaming. To carry this information through, the connector has been changed to a red color to show that this information is being communicated.

Design Round 4

The purpose of the design - Round 4:

Achieve reliability, as well as improve availability, with multiple copies of computation for databases, servers, queues and workers and tactics to guarantee that data won't be corrupted or lost.

Architectural drivers – Round 4

Primary Quality Attributes

a. **Scenario 6 - Availability (Including Reliability):**
    i. **Source:** Hardware crash;
    ii. **Stimulus:** A fault. Any of the identified internal components crash.
    iii. **Artifact:** The video streaming server;
    iv. **Environment:** Normal Operation;
    v. **Response:** Notify the system administrator and recover from the fault. All videos should be available to users;
    vi. **Response Measure:** Time to detect the fault should be within 1 second. Availability percentage should be 99.999%.

- **Concerns**
    1. Low infrastructure cost

Ninth View – Component-and-Connector

As detailed by the availability scenario, the system may present a fault, whenever any of the identified internal components crash (This is a meta-scenario that aims to cover availability and reliability for all mapped internal components at once).

The problem that arises from this scenario is that the crashed component may compromise the availability of the system, bringing it down, in case it doesn't have a backup plan covering it. The identified internal components are: servers, databases, CDN, workers(completion handler server) and load balancer. The availability tactic applied to this scenario is **'Redundant Spare'**. The duplicate components will take the place of a crashed component, it will be used hot components with high synchronization level, in the case of the databases, to guarantee the availability target detailed in the availability scenario. In the case of servers, they are stateless, as described by the performance scenario, what makes the replacement less complex than databases.

In the case of metadata cache, the data is replicated multiple times. If one node goes down, the user can still access other nodes to fetch data, a new cache server can be used to replace the dead one, and then it doesn't need to have a dedicated tactic of redundant spare covering it, since by design multiple copies of computation already covers this issue.

For metadata databases, as it was previously described in the performance scenario, it will be used the master-slave system. If one master goes down, a new slave will be promoted to master. If one slave goes down, another slave can be used for reads and a spare will be brought to replace the dead database. As mentioned, it will be used hot redundant spares, with high synchronization to guarantee the availability target of 99.999%.

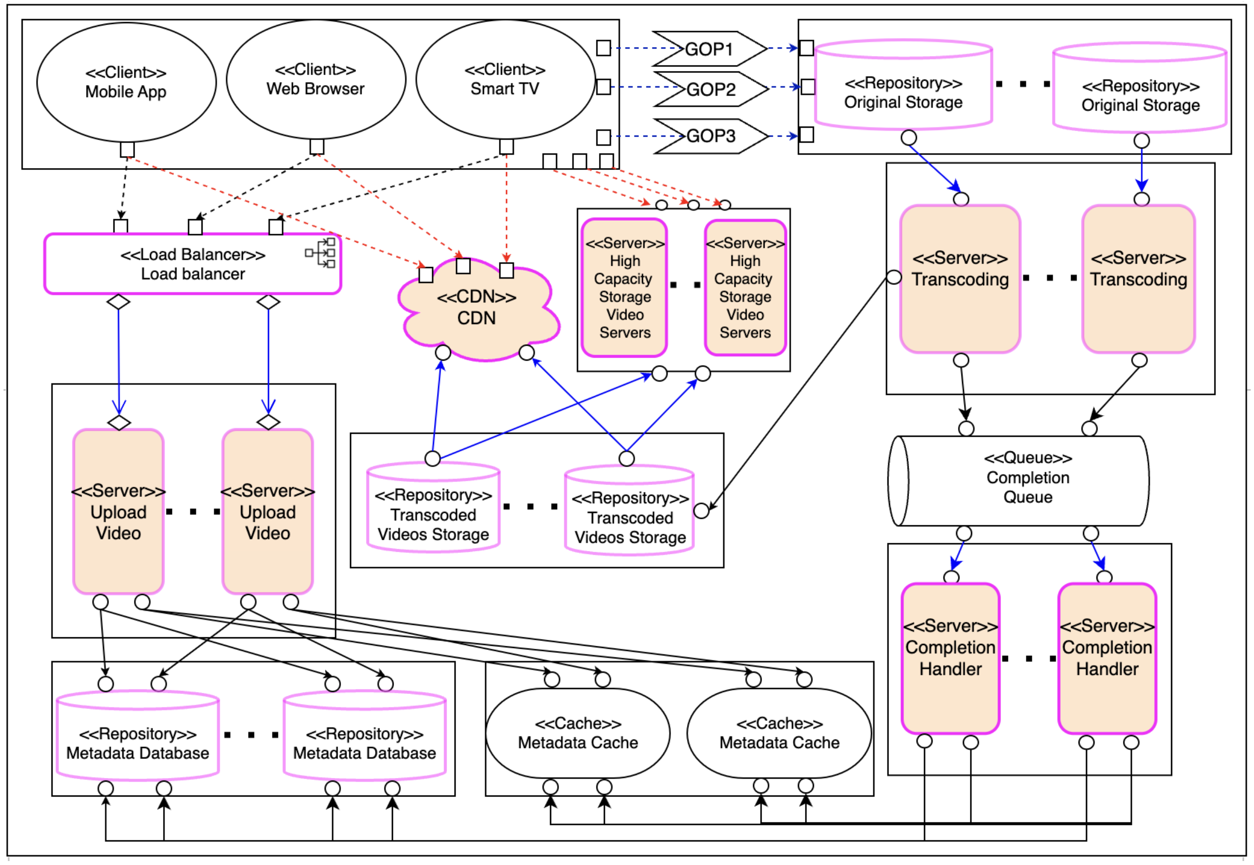
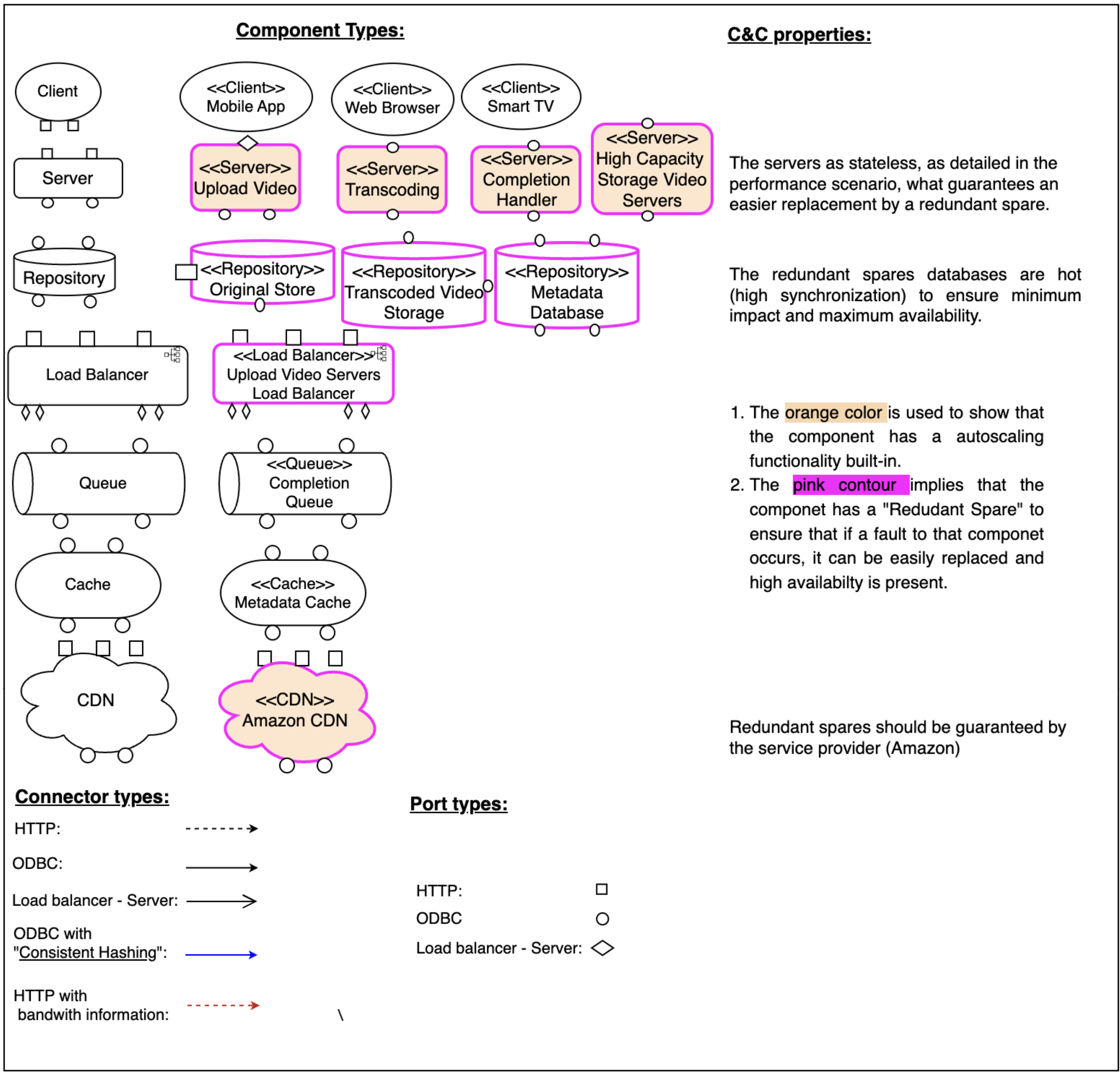


Figure 12. C&C viewtype - Round 4. Ninth View.


Component Types:
Client
Server
Repository
Load Balancer
Queue
Cache
CDN
<<Client>> Mobile App
<<Client>> Web Browser
<<Client>> Smart TV
<<Server>> Upload Video
<<Server>> Transcoding
<<Server>> Completion Handler
<<Server>> High Capacity Storage Video Servers
<<Repository>> Original Store
<<Repository>> Transcoded Video Storage
<<Repository>> Metadata Database
<<Load Balancer>> Upload Video Servers Load Balancer
<<Queue>> Completion Queue
<<Cache>> Metadata Cache
<<CDN>> Amazon CDN
Connector types:
HTTP:
ODBC:
Load balancer - Server:
ODBC with "Consistent Hashing":
HTTP with bandwith information:
\
Port types:
HTTP: □
ODBC ○
Load balancer - Server: ◇
C&C properties:

The servers as stateless, as detailed in the performance scenario, what guarantees an easier replacement by a redundant spare.

The redundant spares databases are hot (high synchronization) to ensure minimum impact and maximum availability.

1. The orange color is used to show that the component has a autoscaling functionality built-in.
2. The pink contour implies that the componet has a "Redudant Spare" to ensure that if a fault to that componet occurs, it can be easily replaced and high availabilty is present.

Redundant spares should be guaranteed by the service provider (Amazon)

Figure 13. C&C viewtype - Round 4 (Component and connectors instances, types and properties)

Summary of C&C viewtype of Round 5:

- To showcase the instances where the tactic ‘Redundant Spare’ is being applied, a pink contour has been added to them.