Matt Wilson
Assignment 3
CS5750 Fall 2015

A Proposal for vidsig™

Vidsig is a product used to identify the video playing on a TV or computer screen. It consists of a mobile app that connects to a backend system in the cloud.

User Experience

The user experience will be simply to open the app on their camera-equipped mobile device, record what they see on the screen, and the app will give them the title of the video, whether it be a movie, tv show, music video, commercial, web video, and possibly other video content. The aim will be to keep the interface simple, and keep the number of functions the app performs to a minimum.

Architecture

As stated above, the user's interaction with vidsig will be through a **Smartphone Application**. That application will send data directly to a vidsig **Application Server**, which will return the information about the video to the smartphone app. The application server will interface with a **Database Server**, which will hold all known video signatures. The application server It will look for a match in the known video database, compared to the signature of the uploaded video.

A useful expansion to the original specification for vidsig would include the ability to hold on to unrecognized user uploads for a period of time, on a **Storage Server** and using a **Submission Database Server**. These videos could be checked periodically against newly inserted signatures in the known video database, and the users notified if a match is found later. The users should also be able to submit titles for their unrecognized videos if they are later able to discover the source material, by navigating to the app's Unrecognized tab. This will allow some of the videos to be crowdsourced, potentially saving the time and money resources of the company. Users would also be allowed to choose to remove the unrecognized videos from the service.

Smartphone Application

There will be a version of the vidsig app for Android, iOS, and Windows mobile. Ultimately, the data sent by the app is platform-independent, so only the user interface will need to be specialized to each mobile platform. The underlying processing code within the app should be identical. Within the app, a user may capture video - or choose a locally stored video - to have vidsig identify. The user will have the choice to compress the data sent to the cloud, either always, never, or conditional upon whether they are using their carrier's data plan or are on a

wifi connection. Certain wifi networks may be chosen as full-data or compressed-data, if the user wishes to limit the bandwidth they use on certain networks. This customization is important because depending on a user's mobile carrier and home internet provider, they may have a data cap on their home wifi, such as Comcast home internet, but unlimited mobile data, such as most T-mobile plans. It is also important to make this optional because compression will use the processing and most importantly, battery resources of the mobile device. Some users may find this to be a reason not to use the application.

Whether compressed or not, the data sent to the cloud will need to be encrypted. A user may attempt to take a video of a tv screen across the room, but also capture people in the room. Or, they may accidentally upload a video they intended to keep private. While most videos really would not contain any private information and therefore would not require any encryption, to protect the company from any potential privacy concerns, the encryption is necessary on every video. This will make the app slightly slower and battery intensive, but it is a worthwhile step. An asymmetric cryptographic system will be used, with apps encrypting the videos using vidsig's public key. Then, vidsig's application servers will decrypt using the private key. Key management will also need to be handled in the app, in order to change keys if one is cracked, and prevent malicious intruders from getting their own public keys into the app, allowing them to decrypt the videos.

Application Server

The application servers will be clones from a single master image. They will use Amazon Web Services Elastic Compute Cloud (EC2). This is where most of the work for this product will be done. As user connections increase, additional servers can be allocated. In this way, the applications running on the server can be designed and written once, but scale out to a massive number of users. This will be a platform-as-a-service configuration, with a very barebones platform of a linux image on the servers.

When a video is uploaded to an application server instance, several steps will be executed. First, the uploaded information will be decrypted using our private key. Then, if the video is compressed, the video will be uncompressed. The compression algorithm will need to be chosen based on several factors. If the compression is too lossy, it may be more difficult to match the video to its original source. So, a low-loss or no-loss algorithm should be used. We will need to strike a balance in the speed of the algorithm and the compression ratio of the algorithm chosen. A fast but low-compression algorithm will not save very much data, costing bandwidth and possibly resulting in a long transmission time, or transmissions dropped before they can finish. A high-compression algorithm that takes longer to execute is likely using excessive CPU time on the user's mobile device. It may be instructive to gauge usage patterns, or take surveys of users, to determine which of these priorities is more important and will drive more users to the application if implemented.

Next, the decrypted, decompressed video will run through the proprietary algorithm that determines the signature of the video. This will probably be the most CPU-intensive step, even above decryption and decompression. The signature generated will then be used in a query to the database server, which will return the closest matches to the application server. If one particular video matches much better than any others, that information will be sent to the user. However, if there are several close matches, they will all be sent back to the user. If there is no sufficiently close match, the user will be notified of that fact. The video will be sent to the Storage Server, and logged in the Submission Database. The user's view in the application will allow them to later submit the source of the video if they find it within a reasonable period of time, or they may simply delete the video.

The primary resource we will need for the application server is cpu time, so EC2's Compute Optimized configuration, most likely C4, will be our servers of choice. C3 servers offer large amounts of SSD storage, which is very fast, but we won't really need it. The application servers will only hold on to a relatively small amount of data at a time.

Database Server

Because Amazon Web Services products are designed to work together, the Database Server will be an AWS Relational Database Service, or RDS Server. The size of the database is scalable, so as video signatures are added to the database, larger instances can be used. This will allow for low costs at the start of the project, only increasing as the content available on the app increases.

Most of the activity on this database will be reads instead of writes. RDS has a feature called "Read Replicas", which allows additional read-only instances of the database to be allocated when there is high demand. As the Application servers will only be reading, they can all point to one of these Read Replicas. Video signatures will be inserted into the Database only by company employees, so they will need connectivity to the full-access version of the database. There will be a small lag time as the Read Replicas catch up to the full access database, but it is unlikely a user will be searching for a video in the batch that was most recently added, making this an acceptable shortfall to increase performance.

Storage Server

When users upload videos vidsig does not recognize, they will be stored for use in expanding the vidsig database. Since the videos could contain personal or sensitive information, they cannot be made publicly available. Instead, vidsig will store the video on an AWS Simple Storage Service (S3) server, again taking advantage of the interoperability of Amazon Web Services products. At about 3 cents per gigabyte of data per month, videos could be kept for a very long time. Within the app, users will be able to see the status of videos that were unrecognized. If they later discover the source of the video, they can submit that information for review by vidsig. To avoid tainting the vidsig database with false information, crowdsourced

submissions will be reviewed before being added to the database. Users may also choose to delete videos that were not recognized, also through the app. While storage is very cheap and videos could conceivably be kept forever, most likely a time limit should be set, where unrecognized videos are removed after 14, 30, or maybe 365 days, depending on the usage patterns discovered in the initial rollout, the amount of storage being used, and the amount the company is willing to spend on storage from month to month.

Submission Database Server

A secondary database server will be allocated for the unrecognized videos. The video signatures will be stored in this database. Keeping this information in a separate instance will allow for high performance from the primary database. From time to time, the submission database signatures will be compared against the newest signatures in the primary database. This will be done by timestamping the submission database rows every time they are compared, and subsequent comparisons will only be done against known signatures timestamped later than the unknown row. This will limit the number of comparisons that need to be done. This could be done in off-peak hours if the app is more heavily used during some parts of the day. But, since the servers are scalable on demand, the comparisons could really be done anytime, allocating another Read Replica of the database server if necessary.

Auditing and Security

By using a well-known cloud provider like Amazon Web Services, vidsig will get all of the auditing capabilities they offer built-in. Amazon promises that storage will be 99.999999999% durable. Since our service losing a single video, or single signature from the database, would not be disastrous, this is well beyond our needs.

For the security of data, the stored videos will need to be stored securely, by ensuring a single-tenancy configuration for that storage. The rest of vidsig works using signatures, which should not contain any private information, allowing for multi-tenancy if necessary. Also, as previously mentioned, data transmitted from the app to the application servers will always encrypted, to prevent any data loss due to interception.

As there is no medical or other personal information, there should be no specific privacy regulations that apply to the vidsig application.

If this app is offered free with ad support, the Service Level Agreement (SLA) can be significantly tilted in favor of the company. By offering no guarantees of the privacy of uploaded videos, the company can avoid legal liability if, despite all security efforts, the storage server is compromised and the videos are released. The company will still strive to keep those videos private, of course, to maintain the trust of its users. Also, as a free app, the uptime does not need to be guaranteed. However, vidsig can promise the same uptime as Amazon Web Services, knowing that the service will be available as long as the AWS hosts are up and

running. If the app is paid, it will be very important to write the SLA as conservatively as we think customers will accept, so that it will be exceedingly unlikely to be broken.

Because we are using a PaaS system, Monitoring and Logging can mostly be left to the installed operating system. It may be worthwhile to periodically run database tools to ensure the integrity of the database, but since the application servers are commodities, if there is an unusual problem with one, such as failing hardware, it will eventually be deallocated and a different copy reallocated.

As the cloud provider will be handling disaster recovery, the most we can do is to pay the extra cost for servers in multiple availability zones or multiple regions, so that there will be redundancy in case of disaster. This is basically an Active-Active Hot configuration, the most expensive but most straightforward and resilient configuration.

Costs

During initial development of the smartphone applications, this project can be started with a single database server and a single application server. A single small RDS server can be allocated for well under $10 per month, with 40 hours of utilization per week. A reserved instance of the smallest EC2 server is $6.57 per month with no upfront cost. Later in the development process, an additional RDS for the user-submitted videos would be added, for another $10 per month. The S3 server to store videos could be allocated with a full terabyte of storage for $30 per month. That puts the total bill at less than $60 per month for cloud services during development. Even if development lasts a full year, it’s unlikely the software company could purchase sufficient hardware needed to test the app for less than the $720 usage fee for the year.

Once the application launches, more power in the EC2 and RDS servers will likely be needed. The RDS servers will have to become more powerful and allow for larger databases. The cost will scale with the power and usage. The EC2 servers can be scaled either vertically, to more powerful individual servers, or horizontally, to multiple instances, whichever appears to be more efficient and cost-effective. Those decisions can be modified over time for the same reasons. In this way, the performance of the app can be improved by spending the money it has already made, rather than spending upfront in hopes of making back the costs.