VHub: An Open Multimedia Hub for Building Cloud Services and Mobile Apps

<Authors Removed for double-blind review>

Abstract

VHub aims to build an open multimedia hub to allow the entire multimedia/vision community in both academia and industry to easily build cloud services and mobile multimedia applications. VHub is built on top of a to-be open sourced distributed functional programming platform OneNet [14]. The VHub Client Suite and VHub App Library are also to be open sourced, pending corporate legal review and management approval. With VHub, a multimedia researcher or developer can quickly turn his/her single-machine implementation of a multimedia application, e.g., an image/video recognizer, into a service hosted in the cloud with multiple service endpoints. This provides the capability to scale by increasing the number of worker machines, balance the load among endpoints, and automatically deal with failure. VHub is capable of concurrently supporting multiple service providers, each of which may support services with different schemas and domains. VHub can also support multiple versions of multimedia services on the same schema and domain. This multi-version hosting feature allows the developer to seamlessly test and deploy a new implementation of the service without any risk in disrupting the existing service (the old version). VHub service endpoints may be simultaneously deployed in a public cloud and / or in a private cluster. We also provide tools for obtaining detailed operating telemetry of the multimedia service endpoints, with capability for developers to arbitrarily customize their telemetry. The open sourced VHub App Library implementation is based on the Portable Class Library [15] and can support mobile Apps across Windows Phone, Android and iOS platforms. The VHub suite of libraries will greatly reduce the development and deployment effort for multimedia researchers and developers to bring their work to a Cloud service and encourages them to build mobile applications that easily showcase the multimedia service to a worldwide audience.

Categories and Subject Descriptors D.1.3 [**Concurrent Programming**]: Distributed programming; D.4.7 [**Organization and Design**]: Distributed systems, interactive systems; H.5.1 [**Multimedia Information Systems**].

Keywords VHub, cloud services, image recognition, video recognition, multimedia services, mobile applications.

ACM MM'2015 Oct 26–30, 2015, Brisbane, Australia.

1. Introduction

With the rapid evolution of cloud computation, GPGPU and cluster based machine learning, and the availability of datasets such as ImageNet [1] and Clickture [2], the dawn of fast development of image/video recognition technologies has come. Developers and researchers around the globe are working feverishly to improve the recognition results, with new technologies, algorithms, and platforms being developed yearly, and with significant improvement in the accuracy, coverage, and performance of recognizers [3]. Innovative multimedia applications, such as SnapChat, Instagram, and How-Old [16], are also being rapidly developed, and are catching worldwide attention.

However, so far, a majority of the multimedia services, particular those incorporating the latest technologies, are confined to university and corporate labs, accessible only in the form of research publications. Only a tiny percentage of them make it to cloud services that can be easily consumed by a large number of users. It is a non-trivial obstacle for multimedia developers to turn multimedia applications that they have made from running on a single machine into a cloud based service, with fail over support and scale-out capability. It also requires tremendous effort and a highly prized skillset to build an online multimedia presence that can be operated 24x7, with high performance and low cost of operation. The required engineering resource and technical know-how are usually not accessible to ordinary multimedia developers.

VHub aims to bridge this gap and to build an open multimedia hub that allows the entire multimedia/vision community to easily build cloud services as well as mobile multimedia applications to consume them. From the ground up, VHub is built as a central hub that supports multiple providers, with different schemas and operating domains, and possibly multiple versions of cloud services in each domain. VHub is a managed cloud service, which offers the following benefits to multimedia service developers and application developers.

Multimedia Service Developers: We use multimedia service developers to refer to the group of people who are developing new multimedia algorithms in academia and/or industry labs. Many of them are well-versed in the latest machine learning, computer vision and multimedia algorithms. However, distributed system design and cloud deployment are usually not the strong suite of this group. We plan to provide an open sourced VHub Client Suite (including VHub Client Library, toolkit and sample code) that allows a multimedia developer to quickly turn his/her testing multimedia application into a hosted multimedia service endpoint with less than one

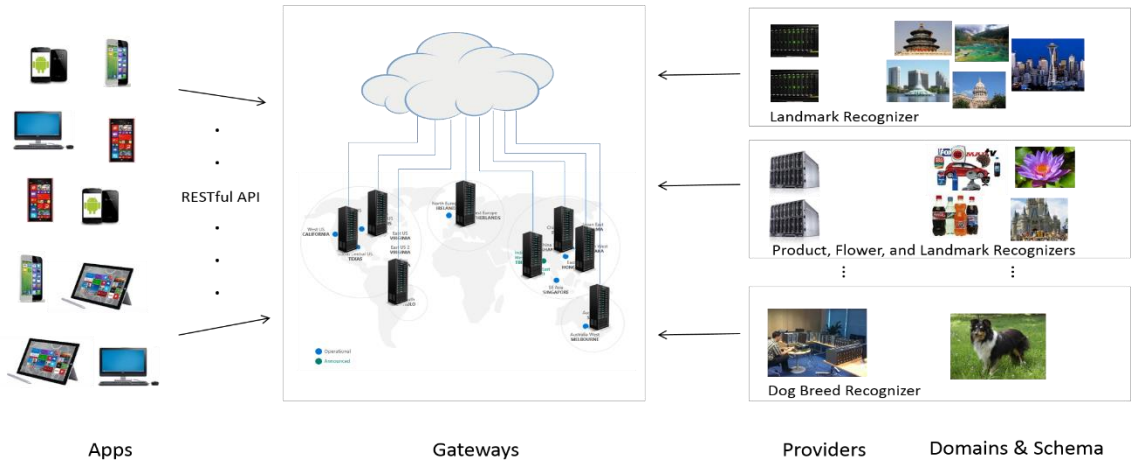

Figure 1 VHub architecture diagram: Apps, gateways, providers, domains and schema.

day of programming work (estimated). Once the multimedia application is turned into a service endpoint, it has the following benefits:

- Integration for cloud service.
- Launching one or multiple instances of service endpoints, either on a local machine, a local computer cluster, or in a public cloud.
- Automatic load balancing among service endpoints.
- Failover support: if a certain endpoint fails, the request will be automatically redirected to another live endpoint.
- Versioning support: the developer may implement a newer version of the multimedia service, and the newer version may run concurrently with the old version of the service. This allows the developer to seamlessly test and upgrade his/her service without disruption.

Multimedia Application Developers: VHub provides RESTful web services [4], and is consumable across a versatile array of mobile and desktop platforms. Multimedia developers may easily use the reliably hosted VHub service to develop their own innovative applications. To reduce the learning curve of VHub, we plan to further provide an open sourced VHub App Library and sample App implementations to allow Multimedia Application developers to quickly develop their own apps across different mobile platforms (Windows Phone, Android and iOS).

The rest of the paper is organized as follows. We discuss the VHub design and architecture in Section. 2. The entire functionality and implementation of the VHub Client Suite is described in Section. 3. We explain the VHub gateway deployment and implementation in Section 4. Experimental results are reported in Section 5.

2. VHub: Architecture

From the ground up, VHub is built as a managed cloud service that supports multiple service providers, with potentially different operating schemas and domains, and with possible multiple versions of the same service running concurrently at the same time. The VHub architecture diagram is shown in Figure 1. We will examine each VHub component in details below.

2.1 Schema

In VHub, a schema is a particular multimedia service call with specific input and output signature. For example, for an object recognition multimedia task, the input of the multimedia service is a byte array of a JPEG coded image, while the output is a list of recognized entity with confidence score. Each schema supported in VHub is represented by a unique 128-bit GUID [5]. VHub can easily extend supported schemas by inserting new GUIDs.

2.2 Domain

A VHub domain provides additional information that narrows down and quantifies the multimedia service requested. For example, within the VHub schema of object recognition, domains can be defined for multimedia services to recognize: dog breed, certain landmark (e.g., Disney, Seattle, Austin,), flower, brand, etc. These domains all share the same schema (input and output data format). Nevertheless, each domain may be served by a different service provider, with specially trained recognizer. A VHub domain is also identified by a 128-bit domain GUID.

Although examples given in the paper are for object recognition, VHub is schema and domain agnostic, and may channel through services a wide variety of schemas and domains. In fact, potentially any web service can be constructed as a VHub service, initiated by a certain application and serviced by a certain provider.

2.3 Provider

A VHub provider is an organization (e.g., a university lab, a corporation, a startup, or even an individual person) that signs up to provide multimedia services in the cloud. Each provider is associated with a provider account, which authenticates the provider. The provider account may also allow the provider to receive credit for the multimedia services rendered. The VHub provider registers schemas and domains of services that it will provide.

2.4 Service endpoints.

A VHub provider may run one or more service endpoints. Generally, the number of endpoints should scale out when more service requests are received, and scale back when the number of service requests reduces. When multiple endpoints are providing the service, VHub automatically balance the load among the service endpoints, and provide the capability for failover recovery so that if

one endpoint fails, the request can be channeled to another active service endpoint.

VHub Client Suite includes tools and scripts to launch service endpoints in either a local machine, in a private server cluster, or in Virtual Machines hosted in a public cloud. The service provider is responsible to find available machines or cloud resources to operate the service endpoints. If 24x7 uninterrupted service is desired, it is advisable that the provider runs service endpoints in more than one geographical region, so that even if the power supply, network, or operating environment of one geographical region experiences a catastrophic failure, the service can still be run by redirecting the requests to other active endpoints in another geographical region.

Most university labs may only have access to a local computer cluster. In such a case, a solution to provide uninterrupted 24x7 service is to employ a small cluster of service endpoints in a public cloud. Many large public cloud providers have established education programs, e.g., [6] [7], which allow the educators to operate cloud service either free or with a significant discount.

2.5 Versioning.

Each VHub endpoint is associated with a deployment version, which is a numerical number that timestamp the code and the data used to provide the multimedia service. VHub allows a provider to deploy multiple versions of service endpoints to the same domain. In Section 4, we will show how the VHub front end may use distribution rules to harmonize multiple versions of the same service in the same domain. When VHub gateway receives a request with both an older version of service endpoints and a newer version of service endpoints, the gateway may distribute request to the endpoints of both versions. Thus, if there are bugs in the newer version of the service endpoint, the older version of the service endpoint can still step in and service the request. This ensure a smooth transition when the developer upgrades his/her multimedia service.

3. VHub Client Suite

VHub Client Suite includes a set of to-be open sourced library, toolkit and script that enables multimedia researchers and developers to quickly turn a multimedia application into a VHub service that can be launched in either one local machine, or on multiple machines in either a public cloud and/or a private cluster.

3.1 Service endpoint programming

We provide sample code to allow the multimedia developer to convert a multimedia application into a VHub service. The operation involved is as follows.

3.1.1 Specify a remote execution roster.

VHub uses OneNet [14], a distributed functional programming platform, to turn a standalone multimedia application into service endpoints that can be launched in a cluster. A core function enabled by OneNet is native remote execution of an arbitrary closure in a remote container. For the case of VHub, this remotely executed function is the constructor of a service class that is derived from `VHubBackEndInstance`. This service class contains logic to register multimedia service to VHub gateways, and when a request is received from gateway, calling a developer provided callback function to service the request.

Before VHub can execute all functionality of the service class remotely, it needs to construct a proper remote execution environment. All the code and data that constitutes the remote execution environment is called the remote execution roster, and needs to be specified by the developer.

Managed Assemblies: For managed code, VHub Client Library has the capability to automatically discover the managed assemblies that the service class depends upon. Moreover, it can automatically traverse referenced assemblies needed for execution. Therefore, if the multimedia service is written in managed .Net code, there is no need for the programmer to specify the dependent assemblies, as they will be automatically discovered and included.

Unmanaged DLLs, Libraries, Executables and data files: If the multimedia service calls into unmanaged DLLs, libraries, and executables, or uses data stored in data files, those files (or its containing folder) need to be specified by the programmer. The most convenient way is for the multimedia developer to specify a local directory that contains all the DLLs, EXEs and data files to be used by the service class. This directory will be automatically copied to the remote service endpoints that can be consumed by the service class executed remotely.

Other dependencies: The programmer may specify additional dependencies that are sometimes needed in properly setting up the remote execution environment. They may include: environment variable settings, customized data serialization and deserialization delegates, and the working directory.

3.1.2 Setup service class

In the multimedia service class, the developer should specify the schema and domains of multimedia services that they provide. The developer may also specify a list of Cacheable Objects that are used by the multimedia applications that consume the service (see Section. 4.6). Finally, the developer should register a callback function, which will be invoked when a service request is received. VHub supports a diverse set of execution modes, as shown in Section 8.2.

3.1.3 Launch parameter

VHub service endpoints may be launched with customized operating parameters. A selected set of parameters that can be passed in include:

- Time interval to resolve for VHub gateways
- Timeout if a request hasn't been completed in the set time
- Time interval to group the request-reply statistics for the telemetry operation
- Collection of VHub gateways.

3.2 Launch of service endpoints : behind the scene

At each of the remote endpoints that the VHub service will run, a daemon needs to be deployed first. VHub Client Suite includes a script to launch daemons in both a public cloud environment and in a private cluster. Once launched, a VHub daemon monitors requests to start new service endpoints. In the first step to launch a service endpoint, the VHub client computes a strong hash (SHA-256) signature for each file (including managed assemblies, unmanaged DLLs, libraries, executables and data) included in the remote execution roster. The list of hashes are sent first. The daemon checks if the corresponding file already exists in the remote endpoint by looking into a common cache folder (a directory containing the content of the cache files with the filename being the hash). If the file already exists, the daemon simply links the file to its final location, and "touches" the original file in the cache by updating its modification timestamp. If the corresponding file is not present in the remote node, it is sent by the client to the daemon. Upon receiving the file, the daemon first writes the file to the cache directory,

Figure 2 vHub Front End Deployment location.

and then links the file to its final location required by the remote execution container.

After all files in the remote execution roster are accounted for, a separate process is launched by the daemon that constitutes the remote execution environment. In this remote execution environment, the service class instance will be constructed. The service class instance will register domains and schemas of services to the VHub gateways. The registration goes through a traffic manager DNS (domain name service) point `imhubr.traffic-manager.net`, which resolves to all active VHub gateways in a round-robin fashion.

Each VHub service class instance maintains a service queue. Whenever a request is dispatched by the VHub gateway to a service class instance, it is served in either synchronous, asynchronous or call back mode specified by the programmer (see 8.2). Each request is assigned a time budget (in milliseconds). If the request fails to exit the service queue before the time budget, it will be discarded with a timeout exception, and will never be serviced. This way, a VHub application can set an upper bound on the time elapsed before a request is served, so that if the VHub service endpoints are flooded with requests, the requests will be dropped instead of incurring a long wait in the service queue.

3.3 Telemetry of VHub back end

VHub has built-in capability to provide detailed and customized telemetry of all service endpoints. Currently, we provide capability to monitor the RTT (network round-trip delay time) from the collection of service endpoints to all VHub gateways, the average, median, 90-percentile and 99.9-percentile latencies to service a request in each of the endpoints, and the average, median, 90-percentile and 99.9-percentile latencies for the requests pending in the service queue in each endpoint.

VHub provides a unique capability to instrument customized telemetry among all VHub service endpoints that are launched. To do so, the multimedia developer needs to export the customized telemetry measures from the backend service classes that he runs. Then he will need to write a data analytical program that is launched against all of the service endpoints. The data analytical program will first import the customized telemetry data, and span it into a virtual distributed dataset that spans across all of the service endpoints. Then, the developer can employ a rich data semantics that is designed similar to the `Collections.Seq` operator in F# [38] plus additional distributed data process semantics such as MapReduce [21], distributed sort, and distributed fold to aggregate the telemetry. The developer can obtain real-time interactive telemetry against all of the service endpoints with sub-second latency, as the entire distributed data processing is done in-memory without the need to first persist the data to a persistent store. We leave the programming details of customized telemetry to Section 8.3.

4. VHub Gateways

We discuss the deployment and implementation of VHub gateways in this section.

4.1 Deployment

VHub gateways are deployed in each geographical location of a major public cloud provider (blue dots in Figure 2~~Figure 2~~). Each of the VHub gateway runs a RESTful web service [8] that listens to incoming multimedia requests, and can be consumed by a wide range of platforms, from desktop applications (on Windows, Linux, Mac platform) to Apps (on Windows Phone, Android, and iOS). For ease of use, we include a VHub Application Library that implements the consumption of VHub web services in the form of a Portable Class Library [15], which can be used by both Windows Phone and Windows Store Apps. It can also be used via Xamarin Suite [23] to develop Android and iOS Apps. VHub gateway also includes a set of human readable Web interfaces to monitor the health of the gateway operations and the attached service endpoints.

The global cluster of VHub gateways are aggregated through two traffic manager DNS addresses `imhub.traffic-manager.net` and `imhubr.trafficmanager.net`. The first DNS address routes the connection request to the closest VHub gateway in the region, and should be used by the applications that consume the VHub service. The second DNS address routes the connection request to active VHub gateways in a round-robin fashion, and should be used by VHub service endpoints so that each endpoint can register services to all active VHub gateways.

4.2 Request format

VHub receives a multimedia request with the following information:

Table 1. VHub Request

Parameter	Description
`ProviderID`	GUID that defines provider
`DomainID`	GUID that defines the operating domain
`SchemaID`	GUID that defines the request schema
`DistID`	GUID that defines the distribution policy
`AggreID`	GUID that defines the aggregation policy
`Reqbuf`	A byte array that contain JSON coded request

Either `distID` or `aggreID` can be an all zero GUID, in which case the default distribution policy or aggregation policy installed at the VHub gateway will be used to process the request. `Reqbuf` is a byte array that contains JSON coded multimedia request. The `SchemaID` defines request-reply format, so that the VHub application and the VHub service endpoints can correctly parse the request and reply that they send to each other. The VHub gateway does not attempt to decode the information stored in `Reqbuf`, and simply passes the information to the service endpoints. This design allows VHub to have the capability to support a large number of different providers and applications with potentially diverse request-reply schemas.

4.3 Request distribution

When receiving a request, VHub gateway generates a 128-bit GUID that uniquely tracks the request in the service pipeline. It then uses the `domainID` and `schemaID` to filter and identify which VHub endpoints are capable of serving the current request. Among the VHub endpoints that are capable, a distribution function that is associated with `distID` is employed to decide which one or more

VHub endpoints should be used to service the request. If no installed functional delegate can be found associated with `distID`, a predetermined distribution policy (least-random distribution) is used. As the VHub operators, we will routinely publish the current set of distribution policies (and aggregation policies) that are available to use. The request distribution system is flexible (can enlist the service of one or more service endpoints) and extensible. We may install new distribution policies over the time. A selected set of distribution policies are described as follows.

Least-random distribution: Least-random is the default distribution policy. In this approach, we keep a count of number of outstanding requests (not serviced and not yet timed out) for each of the service endpoints. The request is distributed to the service endpoints with the lowest outstanding request count. While among the service endpoints with equal number of outstanding requests, one random endpoint is drawn to service the request.

Least-random 2: This policy is similar to the least-random distribution policy described above but instead of one service endpoint, two service endpoints with the lowest number of outstanding requests are selected to service the request.

Fastest responding endpoint: In this policy, we calculate an expected service latency for each service endpoint. This service latency takes into consideration: 1) average network round-trip time (RTT) between the gateway and the service endpoint, 2) average time for the service endpoint to process one request, 3) number of requests assigned to the endpoint from the current gateway, 4) number of requests that are pending in queue at the endpoints. We use information of both 3) and 4) as there are other VHub gateways, and the gateways do not coordinate in assigning request. The service endpoint with the lowest expected service latency is used to service the current request.

Multi-version: In this policy, if there are multiple versions of the service endpoints, one endpoint is selected using the least-random distribution policy among each version. This policy allows the developer to seamlessly test a new version of a multimedia service without disrupting the old service.

4.4 Request aggregation

Aggregation refers to the action of determining if a multimedia service request has been completed, and whether we should deliver the result back to the requesting application. If the request is dispatched to only a single service endpoint in the request distribution stage, the decision is straightforward. If a reply is received, its content is immediately delivered to the requesting application as there are no other requests to wait. If a service endpoint is disconnected before the request times out, another service endpoint will be selected to service the current request. If a reply is timed out (no reply is received within a set time period), the requesting application is informed of the timeout of the request. Any future replies of the request are discarded.

If the request is dispatched to more than one single service endpoints, the aggregation becomes non-trivial, and that is where the aggregation policy kicks in. The aggregation policy is a set of predefined functional delegates with a customized function associated with each `AggreID` that are called upon when a reply comes in, and when a service endpoint that is assigned with request gets disconnected. Like the distribution policy above, we as VHub operator, publish and implement the set of aggregation policy implemented at gateways. VHub has a flexible and extensible aggregation policy system as well. Some selected aggregation policies are described as follows.

First reply: The first reply triggers a reply to the calling application. Subsequent replies are discarded.

First reply plus 500ms: Upon receiving the first reply, a clock is initiated that waits for an additional 500 milliseconds. All replies during this interval are examined, and a score value (`Confidence` shown in Table. 2) is extracted from each reply. The reply with the highest score value is returned to the calling application.

4.5 VHub Reply Object

A VHub reply is encoded as a JSON object [9] and contains the following information.

Table 2. VHub Reply Object

Member	Information
`Confidence`	A value between 0.0 and 1.0. During aggregation, reply of higher score is preferred.
`Description`	Text portion of the reply.
`PerfInformation`	Text coded information of performance.
`Result`	An byte array of coded reply
`AuxData`	Additional cacheable objects referred

In the VHub Reply Object, `Confidence` is a score between 0.0 and 1.0 that is used by the gateway to aggregate the reply. If multiple replies arrive during the request interval, the object with higher score is preferred. VHub gateway provides detailed performance information of the request, in term of the time that it takes for the gateway to assign the request to the service endpoint, the time that the request and reply spend on the network, the time that the request waits in the service queue, and the time that the request is served. These information is coded as a JSON object in `PerfInformation.`

4.6 Cacheable Objects

A VHub reply may refer to Cacheable Objects, each of which is represented by a 128-bit GUID, which is constructed via SHA256 hash of the content of the cacheable object, and then extract the first 128 bit. The use of Cacheable Objects is to allow the multimedia applications to efficiently communicate with the service endpoints, and cache information that does not change. E.g., in object recognition, the reply may contains significant amount of metadata, e.g., Wikipedia page of the object, representative pictures of the object, metadata information returned by Google's Knowledge Graph [22] or Microsoft's Satori [39]. Such information is usually the same for the identical objects, and it is redundant if multiple objects are queried. The use of Cacheable Objects allows such information to be sent through the network only once. If the calling application does not have one or more Cacheable Objects associated with the reply, it can query the gateways to retrieve the content of the objects. Once the calling application has the Cacheable Objects, it can simply reuse them from the local store. The Cacheable Objects are usually hold and served by gateways, which reduces the repeated traffic from the gateway to the service endpoints. As the gateway is usually close to the applications with lower network latency and higher bandwidth, this also reduces the serving latency of the reply. VHub's capability to associate each object with a unique ID also enables caching layer to be implemented at all levels of the applications and/or services. E.g., VHub application may implement its own Caching algorithm in the mobile application to take advantage of the Cacheable Objects and reduce the network traffic and the service latency required to get the reply of the multimedia service.

Figure 3 Deployed vHub back end recognizers.

5. Experimental Results

We wrap and deploy a back end image recognition service designed by our colleagues to a set of service endpoints. The specific implementation uses Caffe recognition engine [10] and has the capability to recognize 11 categories of image objects: dog breed, Orlando/Austin/Seattle/Beijing/Sichuan landmarks (one category for each landmark), Disney attractions, bottled drinks, photo tagging, flowers, and office products. The list of objects recognized are shown in Figure 3~~Figure 3~~.

The recognizer is implemented mostly in unmanaged C++, and includes components such as the Cuda [11], Open CV library [12] and LevelDB [13]. Moreover, each recognition category includes a model directory. All together, the remote execution container of the recognition service includes 40 files. They include 7 files in the model directory per recognition category (the Caffe recognition model is usually around 200MB, and takes bulk of the size of the remote execution container), 21 DLLs, 3 executables. For proper execution, the code also expects a particular directory structure at the remote end for the model file. We deployed 20 instances of the recognizer to a private cluster of machines, which are behind firewalls. Although our particular test deployment uses Caffe recognition engine, we expect that other image recognizer can be easily deployed to VHub service endpoints.

We have deployed 13 VHub gateways in virtual machines, with one for each major geographical location of a tier-one cloud service provider, as shown in Figure 2~~Figure 2~~. The machine configuration is shown in Table. 4. Notice that the machines for the service endpoints are far more powerful than the gateways, as they are the one doing the majority of the hard work for the multimedia service.

Table 3. VHub: Gateway / Service endpoint configuration

Type	Specification
`Gateway`	VM, 2 cores, 3.5GB
`Endpoint`	Dual Proc Xeon® E5-2450L, 8 cores per Proc, 192GB RAM

Using the telemetry tool described in Section. 8.3, we have obtained the application layer network round trip time between the gateways at different geographic region and the service endpoint clusters and shown results in Table 4. In our particular deployment, the service endpoints are all co-located in one geographical location (in northwest US), thus, we only show aggregated network statistics. Please also note that the roundtrip time shown includes not only network ping time, but also processing time to schedule the network and the application thread to parse the packet. Comparing with network ping latency, the information gives a more accurate measure as it is the total time that the request and response bounce between the gateway and the service endpoints.

Table 4. Application layer RTT (Round Trip Time, in millisecond) between gateway and service endpoints

Gateway Location	Network Round Trip Time (in millisecond)		
	Average	Median	90-percentile
New South Wales, Australia	1606	184	6130
Victoria, Australia	272	218	574
Iowa, US	372	63	65
Hong Kong, China	183	182	186
Virginia 1, US	162	95	550
Virginia 2, US	98	98	101
Netherlands	240	189	510
Tokyo, Japan	188	122	558
Osaka, Japan	171	122	460
Illinois, US	219	126	485
Texas, US	568	96	96
California, US	88	88	95

We observe that median latency is well correlated with the distance (and the network ping latency) between the gateway and service endpoints. The gateway that is farthest from the service endpoints is the one located at Victoria, Australia, and the next three farthest gateways are located in the Netherlands, New South Wales, Australia and Hong Kong, China, respectively. However, when measuring the 90-percentile latency or the average latency, a different picture emerges. The gateway server at New South Wales, Australia, performs poorly, with a huge 90-percentile latency (at 6.13 seconds) and a large jump of the average latency (to 1.6 seconds). The gateways located at Iowa and Texas, both give good median and 90-percentile latency, yields poor average latency of 372 ms (milliseconds) and 568 ms, respectively. Both indicate that there are occasional instances when it takes the gateway a long time to communicate with the service endpoints. We are investigating

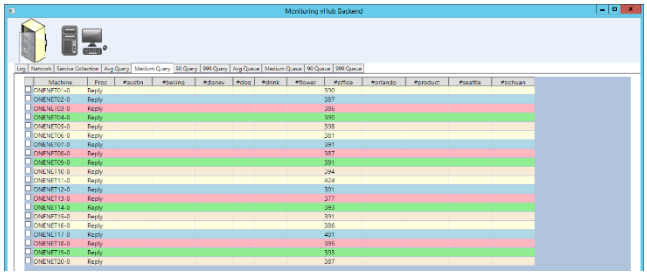

Figure 4 Screenshot of VHub telemetry tool.

the performance discrepancies, but do observe that the VMs we have used for the gateway have only two cores and are rather underpowered. From our past experience of working with cloud services, we do not think that it is possible to completely eliminate the performance issue of every single gateway, as the gateways are VMs deployed in a complicated cloud operating environment, with performance impacted by shared network infrastructure and/or computer infrastructure. However, it is feasible for a time sensitive VHub application to send multiple requests to different gateways (not only the closest gateway). We do observe certain gateways (e.g., the one in California and in Virginia 2) seem to give a more consistent performance. They may be employed as backup for the VHub application.

Table 5. Synchronous and asynchronous image recognition run.

Statistics	Synchronous	Asynchronous
# of images	984	984
Total time	501.8 sec	68.7 sec

We prepare a dataset of 984 images (mainly of office objects). To reduce the network bandwidth cost, we first preprocess the image by shrinking its long edge to 256 pixels and compress it to jpeg format with 75% quality setting. The resized image is then sent to the gateway at California to perform the recognition test. In the first test run, we send in the image synchronously (i.e., we wait for the recognition result of an image before sending another one), as shown in Table. 5. We observe that each image completes the recognition task in an average of 510 ms. In the second test run, we send in the image asynchronously (i.e., we do not wait for the recognition result, just continuously send in image for recognition). Because the image recognition is a very computation heavy task, the bottleneck is usually not at the gateway, but at the service endpoints. The VHub gateway has a timeout value of 31 seconds, therefore, when image pours in for the recognition task, the queues at the service endpoints will grow and eventually, the image recognition requests time out. The 20 node service endpoints complete the entire service request in 68.7 seconds, and process around ~14 request per second. There are 13 images around the tail of the batch timeout. We are working to provide more sophisticated queueing and load information logic at gateway so that the application can throttle the speed to send in the requests.

As VHub is an open platform for any researchers and developers to deploy and host service endpoints on their own computer clusters, the coverage, quality, performance and stability of these deployed recognizers and their hardware/software systems can be vastly different. In particular, (1) different service endpoints may have different network RTT to the gateways, specifically when they are hosted in different geographic region with different network and computer equipment. (2) Even in the same cluster of a certain geographic region, different nodes may have different processing capability and speed, due to difference in hardware configurations, network topologies, and load factors. Also, other researchers/developers may have run their own jobs that can consume a significant portion of the CPU, memory and network resource of the nodes that are used by VHub service endpoints. They can dramatically impact the performance of the node and make nodes temporarily busy (and slow in response). (3) Different classifiers may also use different classification model, with varying performance and memory footprints.

VHub provides powerful telemetry into the real-time operation of the service endpoints. The screenshot of the VHub telemetry tool is shown in Figure 4. In Table 6, we show the telemetry behavior of the VHub service endpoints at a run where all service endpoints are light loaded and under a run where some of the service endpoints are encountering abnormal behavior. The right side of the table shows the VHub service endpoints' performance at light load. We observe that the median time to execute an image recognition request ranges from 386-424ms, while the 99.9-percentile latency for the request ranges from 454-1520ms. We observe that the 99.9-percentile recognition latency is much higher than the median recognition latency, as some image object or some instance of run may execute slowly. The left side of the table shows the telemetry when some of the service endpoints experience some performance issues. In particular, one of our colleagues have concurrently run a job on the cluster without our knowledge. Post mortem analysis shows that he has attempted to run the job at node 16-20, succeeded to launch two jobs at node 17 and 19, and relocated one job to node 03. And then, node 16 and 20 has been automatically rebooted by the system due to a software update, and become unavailable to any jobs. The telemetry of the performance shows that node 01-02, 04-15 and 18 operates with recognition latency similar to the normal run, while node 3, 17 and 19 operates with significantly higher median and 99.9-percentile recognition latency, and node 16 and 20 shows as failure. Through the experiment, we showed that VHub is capable of showing detailed service endpoint telemetry. With a single glance, the developer can be aware of the real-time health of his service and act accordingly.

Table 6. Two Runs of VHub.

Node	Heavy/Unbalanced Load		Light Load	
	Median	99.9%	Median	99.9%
01	405	667	390	716
02	388	437	387	483
03	1008	2425	386	1520
04	407	510	390	1152
05	387	482	398	574
06	389	470	381	513
07	388	458	391	1472
08	393	469	387	472
09	407	460	391	449
10	399	459	394	525
11	392	429	424	734
12	388	426	391	415
13	390	512	377	512
14	397	423	393	1070
15	379	552	391	637
16	FAILED	FAILED	386	563
17	5829	24214	401	454
18	384	449	395	955
19	20781	23006	393	601
20	FAILED	FAILED	387	489

6. Related Works

The breakthrough of deep neural network (DNN) learning in large scale image classification [25] has greatly advanced the state of the art of visual recognition. Since 2012, many record-breaking results have been reported in academic publications on ImageNet [1] and LFW (Labeled Faces in the Wild) [24], two important benchmarks for image classification and face recognition. For example, on the ImageNet 1000 class classification task, in early 2015, Microsoft, Google, and Baidu successively reported that they achieved new records of error rate 4.94% [26], 4.8% [27], and 4.58% [28], surpassing the human level performance 5.1% on the same task. On the LFW face verification task, the best results were all achieved by using deep neural network algorithms in the past year, with the best record of mean classification accuracy 99.53% kept by Sun et al [29].

However, despite the exciting progress in academia, few research labs, especially in universities, have the skillset to build a robust system either online or offline to quickly launch their latest work as a real application. As a result, the most advanced algorithms are available deep in a production pipeline in large commercial companies such as Microsoft, Google, Facebook, and Baidu, etc.

Recognizing the great commercial potential, many startups have stepped into this field, focusing on various visual recognition problems and providing online services to customers. They mostly use RESTful APIs to significantly reduce the effort that a mobile app or web app developer needs to integrate visual recognition functions, and thus lets the developer focus on the end-to-end user scenarios while still being able to benefit from any improvement of visual recognition service performance in a transparent way.

For example, the startups such as clarifai.com [30], metamind.io [31], rekognition.com [32], cortica.com [33], and faceplusplus.com [34] all aim to bring the best visual recognition technologies to application developers. Most of them utilize RESTful APIs to provide simple web interfaces for image classification, scene understanding, and face recognition. Metamind.io also allows users to upload their own labeled training data to train customized classifiers. Tendinsights.com [35] aims to build a Vision-as-a-Service platform for home surveillance scenarios by providing home video analysis techniques from cloud to webcams. To help developers easily add intelligent services into their solutions, Microsoft also has made available a set of RESTful APIs in Project Oxford [36] that aims to make it possible to build apps that feature face recognition, visual recognition, and speech processing.

However, all of the existing visual recognition services only host their own recognition modules. This even creates a higher barrier of entry for multimedia/vision researchers to bring their new techniques to real systems, as the hosted service is expected to have a performance and stability comparable to the commercial system. VHub aims to bridge this gap by building an open multimedia hub and hiding the complex engineering details of hosting a multimedia service to multimedia/vision researchers to allow the entire multimedia/vision community to easily build cloud services and mobile multimedia applications.

One related work to VHub is VMX, a personal vision server developed by vision.ai [37]. VMX consists of a visual object detection algorithm engineered for speed, accuracy, and a web-based interface for building vision applications. A VMX Engine runs as a server and uses RESTful APIs to provide web interface for easy online access. It can also run on a laptop or desktop computer to provide local access. However, VMX mainly aims for enabling easy setup of individual VMX engines. It lacks the gateway function in VHub which is a crucial part for service endpoint management, request distribution, and result aggregation.

7. References

[1]. Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009.

[2]. Hua, Xian-Sheng, et al. *Clickture: A large-scale real-world image dataset*. Microsoft Research Technical Report MSR-TR-2013--75, 2013.

[3]. Olga Russakovsky, et al., *ImageNet Large Scale Visual Recognition Challenge*. *arXiv:1409.0575*, 2014.

[4]. Richardson, Leonard, and Sam Ruby. RESTful web services. " O'Reilly Media, Inc.", 2008.

[5]. http://en.wikipedia.org/wiki/Globally_unique_identifier

[6]. http://azure.microsoft.com/en-us/community/education/

[7]. http://aws.amazon.com/education/

[8]. Richardson, Leonard, and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.

[9]. http://en.wikipedia.org/wiki/JSON

[10]. Jia, Yangqing, et al. "Caffe: Convolutional architecture for fast feature embedding." *Proceedings of the ACM International Conference on Multimedia*. ACM, 2014.

[11]. Nvidia, C. U. D. A. "Programming guide." (2008).

[12]. Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[13]. http://en.wikipedia.org/wiki/LevelDB

[14]. _, "OneNet: Cloud Service and Interactive Big Data Analytics (Distributed Platform Building Leverage Functional Programming)", submission under double blind review.

[15]. https://msdn.microsoft.com/en-us/library/gg597391(v=vs.110).aspx

[16]. http://how-old.net/#

[17]. Petricek, Tomas, and Jon Skeet. Real World Functional Programming: With Examples in F# and C#. Manning Publications Co., 2009.

[18]. Leijen, Daan, Wolfram Schulte, and Sebastian Burckhardt. "The design of a task parallel library." Acm Sigplan Notices. Vol. 44. No. 10. ACM, 2009.

[19]. https://msdn.microsoft.com/en-us/library/bb549151(v=vs.110).aspx

[20]. https://msdn.microsoft.com/en-us/library/system.collections.ienumerable(v=vs.110).aspx

[21]. Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[22]. Hamburger, Ellis. "Building the Star Trek computer: how Google's Knowledge Graph is changing search." The Verge (2012): 23-09.

[23]. Reynolds, Mark. Xamarin Mobile Application Development for Android. Packt Publishing Ltd, 2014.

[24]. Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. *University of Massachusetts, Amherst, Technical Report 07-49*, 2007.

[25]. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *In Advances in Neural Information Processing Systems*, 2012.

[26]. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *arXiv preprint arXiv:1502.01852*, 2015.

[27]. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv:1502.03167*, 2015.

[28]. Wu, Ren, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. "Deep Image: Scaling up Image Recognition." *arXiv:1501.02876*, 2015.

[29]. Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. DeepID3: Face Recognition with Very Deep Neural Networks. *arXiv:1502.00873*, 2014.

[30]. http://clarifai.com

[31]. http://www.metamind.io

[32]. http://rekognition.com

[33]. http://www.cortica.com

[34]. http://www.faceplusplus.com

[35]. http://tendinsights.com/vision-as-a-service

[36]. http://www.projectoxford.ai

[37]. http://vision.ai

[38]. https://msdn.microsoft.com/en-us/library/ee353635.aspx

[39]. http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/

8. Appendix

We discuss some of the detailed implementation of VHub in the Appendix.

8.1 Notations

Tuple: A tuple [17] is an ordered list of elements of specific type. E.g., `GUID*int*'Request` is a tuple of three element, where in the first element is a 128-bit GUID, the second element is a 32-bit integer, and the third element is a VHub request object.

Type signature: A type signature [17] defines the inputs and outputs of a function. E.g., `GUID*int*'Request->'Reply` defines a function that takes three input (a GUID, an integer, and a VHub request object), and returns a VHub reply object.

Function parameter: VHub Client Library and gateways are written in F# [17], and uses functional programming concept throughout the program. A core concept in functional programming is that function is a first class citizen, and can be used like variables (passed as parameter, returned as result, assigned to variable). E.g., in `GUID*int*'Request*('Reply->unit)*(string->unit)` of the 4th parameter, a function with input parameter `'Reply` and no output (`unit`) is passed as part of the calling parameter.

Task: Task parallel library [18] is designed for asynchronous tasks to take advantage of the potential parallelism in a program. `Task<'Reply>` represents a task that will be asynchronously executed, and when completes, returns a VHub reply object.

Functional Delegate: `Func<T,TResult>` is a .Net delegate [19] that encapsulates a method that has an input parameter of type `T`, and return a result object of type `TResult`. `Func<TResult>` encapsulates a method with no parameters, and return an object of `TResult`.

Enumerable Collection: `IEnumerable<T>` is a .Net base interface [20] for all collections (such as array, list, dictionary, etc.). IEnumerable contains a single method, `GetEnumerator`, which returns a twin `IEnumerator<T>` object. The `IEnumerator` can be used to iterate through the collection by using the `Current` property and `MoveNext` method.

8.2 Execution mode of VHub service endpoint

VHub accommodates multiple execution forms, depending on whether the multimedia service function is implemented in synchronous mode, in callback mode, or in asynchronous form. Shown in Table 7, the first mode is synchronous execution, i.e., the service thread blocks and waits till the multimedia request completes before returns the result to the VHub gateway, and fetches a second request to execute. In the synchronous mode, each service class services no more than one request at any given time. The second mode of execution uses two call back functions. When a request arrives, the service thread calls the registered functional delegate and returns immediately. Two additional functional delegates are passed as parameter, one is invoked if the request has been successfully served (signature: `'Reply->unit`) and a second when the request fails (signature: `string->unit`). The third form of execution uses .Net task parallel library, and queued the multimedia service in a task pool, the request is served when the enclosed task completes. If the request fails to be served, an exception may be thrown to indicate the failure of the service of the request.

Table 7. Execution function in VHub service endpoint

Form	Signature
`Synchronous`	`GUID*int*'Request -> 'Reply`
`Callback`	`GUID*int*'Request*('Reply->unit)*(string->unit)->unit`
`Asynchronous`	`GUID*int*'Request -> Task<'Reply>`

8.3 Customized telemetry of the service endpoints

VHub provides a unique capability to instrument customized telemetry among the VHub service endpoints that are launched. To do so, the developer will implement a telemetry gathering function and export the function as a contract via:

```
exportSeqFunction(name,func,limit),
```

where name is the `name` of the contract to be later imported by the data analytical program, `func` is a Functional delegate implemented in the form of `Func<IEnumerable<TResult>>`, and `limit` is a size parameter that packs a certain number of telemetry inputs into an array of TResult[] for efficient transportation between the service class job and the data analytical job.

The developer can then write customized code to process and aggregate the telemetry. For example, the code below is the code that obtains the statistics of the service latency and service queue described in Section. 5.

```
let a = DSet<_>.import cl null name
let b = a.RowsReorg -1
let c = b.MapByCollection queryPerformance
let d = c.Fold perfAgg perfAgg null
```

This code is written in F# and calling the OneNet [14] data analytical library. The first line of code imports a contract from the service endpoints that returns a recent set of service requests and their service performance statistics. The amount of service requests returned can be tuned as shown in Section. 3.1.3. The code creates a virtual data collection that spans across all service endpoints, with one partition on each of the service node. The second line of code aggregates all data in a partition (i.e., in a single endpoint node in this particular case) to a single collection. The third line of code then computes an aggregated query performance statistics for all the service requests. We need to aggregate the requests because the

statistics of medium, 90-percentile and 99.9-percentile is aggregated statistics and can only be computed when the function receives the entire collection of statistics. The forth line of code aggregates the statistics across all service endpoints to the node where the monitoring function is run.

In the above example, the first three lines of data analytics code is lazily evaluated (meaning those function only constructs another virtual data collection, which is called `DSet` in OneNet). The data analytical operation is only triggered when the `Fold` function is encountered in the 4th line. At that moment, a data analytical job is launched and executed in the cluster of all endpoints. The first time that the data analytical program needs to be executed, a remote execution container will be launched by each daemon in a fashion similar to the launch of the remote execution container of the service class. For subsequent data analytical program execution (assuming we are constantly monitoring the telemetry of the service endpoints), the same remote execution container will be reused. A Fold operation request is sent to each of the remote execution container, which pulls through the analytical data. The execution of the import function triggers the callback API, which hooks up to `func` of type `IEnumerable<TResult>` exported by the service endpoints. The `GetEnumerator` method of `func` is then called, which in most of the case starts the actual data gathering process and retrieves the twin `IEnumerator` class. The callback API then alternatively calls between method `MoveNext` and `Current`, packages and sends items of `TResult[]` object to the data analytical program. When the `IEnumerator` reaches the end (`MoveNext` returns `false`), the remaining items pending to be processed (if any) are sent as a final `TResult[]`. Finally, a `null` object is sent. Because any valid collection of `TResult` items is an array of at least one, the `null` object uniquely signals the end of the data sequence.

The entire VHub telemetry is capable of dealing with failure of any back end nodes (the telemetry of the failed back end nodes are simply left out of the aggregation). The telemetry can also involve advanced big data processing operation, such as MapReduce [21]. This remote cluster data analytics can involve arbitrary code execution in the service endpoints, with most of the processing done at the service endpoints. In the example data analytical jobs above, the statistics (average, medium, 90-percentile and 99.9-percentile) of service latency and service queuing latency are all computed locally at the service endpoints. Thus, only the end result is sent back to the monitoring node, and the entire collections of observed service requests and their performance do not need to be sent over the network or be written to any disk storage. We are able to execute interactive telemetry on the VHub service endpoint cluster under a second.