2011 31st International Conference on Distributed Computing Systems

Cloud4Home — Enhancing Data Services with @Home Clouds

Sudarsun Kannan, Ada Gavrilovska, Karsten Schwan
Center for Experimental Research in Computer Systems
Georgia Institute of Technology
sudarsun@gatech.edu, {ada, schwan}@cc.gatech.edu

***Abstract*—Mobile devices, netbooks and laptops, and powerful home PCs are creating ever-growing computational capacity at the periphery of the Internet, and this capacity is supporting an increasingly rich set of services, including media-rich entertainment and social networks, gaming, home security applications, flexible data access and storage, and others. Such 'at the edge' capacity raises the question, however, about how to combine it with the capabilities present in the cloud computing infrastructures residing in datacenter systems and reachable via the Internet. The Cloud4Home project and approach presented in this paper addresses this topic, by enabling and exploring the aggregate use of @home and @datacenter computational and storage capabilities. Cloud4Home uses virtualization technologies to create content storage, access, and sharing services that are fungible both in terms of where stored objects are located and in terms of where they are manipulated. In this fashion, data services can provide low latency response to @home events as well as high throughput response when the higher and less predictable latencies of datacenter access can be tolerated. Cloud4Home is implemented with the Xen open source hypervisors for standard x86-based mobile to server platforms, and is evaluated using sample applications based on home security and video conversion services.**

***Keywords*-Objects; Virtualization; Hand-held Platforms; @Home Services; Distributed Hash Table; Public Cloud**

I. Introduction

Concurrent with the rapid gains in popularity of cloud computing facilities and infrastructure is an even more impressive continued increase in the capabilities of end devices used in the Internet's periphery. In fact, major hardware manufacturers are releasing new end devices as frequently as every six months, whereas server systems typically experience a multi-year replacement cycle. These facts raise the interesting challenge of how to best combine server provider cloud services like those offered by Google Docs, Amazon's EC2, VMware's VSphere, Microsoft's Azure, Eucalyptus, etc. with services that can efficiently run on consumer devices like smartphones, iPads or netbooks, laptops and desktops, game consoles, etc. In particular, purely end-point based solutions cannot take advantage of the large storage and computational capacities present in large scale datacenters. Conversely, current 'thin client' models in which end devices 'simply access the Internet' can suffer from high and variable delays in accessing and using remote resources – as evident for services like DropBox – and they are subject to challenges when devices must operate in disconnected mode. Preferable to either extreme would be a solution that (1) can leverage the lower costs of using local resources and exploiting locally available state, avoid potential issues with data privacy or security for cloud-based operation, while at the same time (2) exploit Internet resources when those are not encumbered by undue costs like high latency or undue communication overheads.

The Cloud4Home project and approach described in this paper can exploit the efficiencies and opportunities presented by flexible and combined @home and @datacenter operation. The data services explored in the paper can tap into the aggregate resources offered by remote clouds, and they can leverage 'nearby' devices in home or office settings. The outcome is quality in service delivery that exceeds that of the pure 'in the cloud' or 'at the edge' service realizations. Data services investigated include data access and storage, data and media manipulation, and services like real-time image recognition – face detection – for images captured by a home security system, for example. For such services, sensitivity to response-time variation experienced with the use of public clouds and the Internet indicate the usefulness of using nearby or home devices. At the same time, increased processing requirements cause an increased need to exploit the larger capacities of remote cloud services. Given these tradeoffs, we formulate the following design principles for Cloud4Home systems:

- *Fungibility for dynamic flexibility*: the physical resources on which services run should be 'fungible', so as to create dynamic options in the mappings from the resources applications believe they are using – virtual resources – to the physical resources actually being used.
- *Augmentation*: services should not be constrained to operate within the boundaries of private devices vs. public cloud platforms; instead, they should be able to run so that the two types of resources can augment each other, in a manner best suited for the given service and current operating constraints.
- *Guided active management*: since the 'best' mappings of virtual to physical resources depend on current context, user needs, and resource availabilities, active management of these mappings must have continuous inputs from methods that monitor these factors.
- *Automation and independence*: guided management

1063-6927/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCS.2011.74

should not require end user participation, i.e., it should be automated, and in addition, management should be independent of specific operating systems or application frameworks present on mobile devices.

The resulting outcome, described in this paper, is the Cloud4Home approach and system for enhancing cloud services through the use of local, private resources. We specifically focus on services for storage, access, and manipulation of data for the home environment and when doing so, we leverage the VStore++ [1] system, a virtualized object storage system that abstracts from an application where the objects it accesses are stored. Further, it permits the enhancement of data access services with custom data manipulation functions that can be run on the machines used by data providers and/or consumers. VStore++ is independent of operating systems or middleware by operating at the virtualization level of systems. It is fungible in that data can be stored on local disks, on remote machine's stores, or at Internet-connected storage sites, in ways that are transparent to end users, independent of application frameworks, and even the operating systems running on the @home devices. At the same time, since the context in which an end-user device operates can change dynamically, as can end user requirements, VStore++ will track resource availability in order to direct requests to appropriate destinations based on their needs and/or resource availability, using a global indexing and monitoring infrastructure maintained during its operation. Interactions with VStore++ may take place across wireless networks, across the Internet, when using Internet-based resources like cloud storage, or across a mix of wired and wireless links when operating in a user's home. The outcome is a Cloud4Home comprised of dynamically varying sets of devices that cooperate to provide end users with seamless storage, access, and data manipulation services, including interactions with remote, publically available cloud platforms.

The technical contributions of this paper include the following. (1) VStore++ is a set of Cloud4Home services implementing methods for data storage and manipulation that enhance what can be provided by solely @home or @datacenter service realizations. (2) VStore++'s implementation for a prototypical home environment, with desktop and handheld devices, uses the Xen hypervisor to attain operational independence from vendor-specific solutions. (3) Home security and video streaming services realized with VStore++ exhibit improved performance properties compared to prior service realization, including services realized in Amazon's EC2 cloud. (4) For home device cooperation and active resource management, VStore++ uses (i) a dynamic overlay layer implemented with the lightweight Chimera [2] peer-to-peer overlay system, and (ii) a distributed key-value store for data accesses and dynamic resource monitoring. (5) Experimental evaluations of the VStore++ system and approach demonstrate multiple interesting facts, including (i) the tradeoffs in using @home vs. @datacenter resources, (ii) the advantages derived from judiciously using both, and (iii) future work in terms of scaling Cloud4Home functionality to larger systems and to other sets of services.

II. Services in the Home

Before continuing with details regarding VStore++ and our Cloud4Home approach, we provide additional motivation for the need to enhance the capabilities of remote clouds by using private and nearby resources. Several classes of services are discussed.

Home surveillance. Surveillance companies, such as Dropcam Echo, have started providing specialized hardware that not only captures images and video, but also has the ability to offload content to a public cloud. The devices are expensive and consumers need to bear storage cost in the cloud, with limited control over the data being generated. In addition, the quality of the service is highly dependent on the available connectivity to the public cloud, exposing it to potentially substantial levels of variability.

A Cloud4Home solution like the one described in this paper can eliminate several limitations associated with this class of services. First, with Cloud4Home, both the data being acquired and stored and the computations performed on such data can be mapped to appropriate target nodes, including nearby nodes in home environment. This flexibility in mapping can be used to maintain desired levels of service quality, despite known variability in cloud connectivity. The outcome is lower response times resulting in improved timeliness for detecting potentially critical events. Guided management can be used to control which and how much state is stored in the public vs. the home cloud, thereby limiting the service delays, costs and providing desired privacy guarantees. At the same time, resources available in the public cloud continue to be available, including large amounts of storage, public databases of image training sets, or computational resources for parallel execution of face detection and recognition algorithms. In particular, the surveillance service described in this paper is implemented as an application that uses VStore++ interfaces [3] to store and manipulate captured images. Surveillance images are processed first by a face detection algorithm, followed by face recognition [4]. These algorithms may be run at nodes in the home environment, or on instances at Amazon's EC2 public cloud.

Media conversion. The diversity of end-user devices gives rise to increased amounts and types of multimedia content being generated or used by end-users. A purely public-cloud-based solution for interacting with such content can be limited by the available connectivity to the public cloud. Furthermore, for a given device form factor and multimedia resolution, the performance (i.e., delay) and operating costs

(i.e., dollars) associated with the public cloud interactions may not be necessary.

A more flexible solution supported by the Cloud4Home approach is one in which content can be retrieved from either home or remote cloud resources, and where appropriate format conversion services can be applied transparently so as to customize desired multimedia content for diverse end-user devices. This can be done so as to better meet desired cost functions. We use a prototypical multimedia service in the experimental analysis of our VStore++ system.

Other services. Obvious examples are those that pertain to content sharing in college dorms or apartment homes, as evident from recent work on peer-based solutions for IPTV services. More interestingly, there are other examples of hybrid @home and @datacenter services. Consider data services in hospital environment, where privacy concerns may require purely office based solutions, but where convenience in remote data access or data use by researchers suggests the need for datacenter storage and manipulation for select data elements [5]. Another example occurs for gaming applications where camera-guided home games require low latency home processing but where long term analysis needed for improving camera-based control methods can benefit from the extensive data collected in millions of game-playing homes (provided that privacy concerns are adequately addressed). More generally, there is a plethora of sensor-based applications in which mobile or home devices can be used to pre-analyze and rapidly inspect captured data, raise local alarms, but where at the same time, global data mining methods can benefit from the many inputs received from spatially or time-offset distributed end systems.

III. VStore++ Architecture

We next describe the VStore++ architecture, which realizes the Cloud4Home approach for providing data storage, access, and manipulation services that are transparently performed across any of the resources available in the home or remote public clouds.

The overall VStore++ architecture is shown in Figure 1. VStore++ is a virtualized storage service exposing an object-based file system interface, similar to other object-based storage interfaces [6, 7], including popular cloud stores such as Amazon's S3 storage service. Internally, it uses a standard file system to represent objects, using a one-to-one mapping of objects to files. In addition to object *fetch* and *store* operations, it supports an explicit *process* operation, which permits object manipulation functions to be associated with the object access.

Applications using VStore++ API reside in guest virtual machines (VMs) running on nodes in the home environment, which is virtualized with the hypervisor in our current implementation. All requests are passed to the VStore++ component residing in the control domain (i.e., dom0 in Xen) via shared memory-based communication channels. On

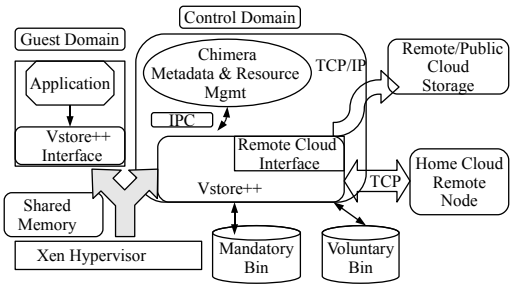

Figure 1. Cloud4home architecture.

each node, a set of mandatory resources is available for the execution of services (e.g. storage or computation) on behalf of applications deployed on that node. In addition, nodes can contribute voluntary resources to the aggregate storage pool available to any node in the VStore++ home cloud.

The metadata layer provides object lookup and transparent access to storage and services distributed across nodes in the home or remote clouds. To deal with the potential dynamism of the home environment, where nodes may periodically go off-line and become unavailable, the metadata management layer is built as a distributed key-value store on top of a peer-to-peer overlay across all control domains in the home cloud. At least one of these nodes must provide an interface among the home and remote cloud services.

In order to enable location transparency for object accesses and for execution of object manipulation functions, all VStore++ operations specify only the object and/or service, where the actual location is determined at the metadata layer. Since our goal is to ensure that the request routing can be performed in a manner that depends on the resource availability and operating conditions, it is necessary to also maintain resource monitoring state at runtime. We use the same distributed key-value store for both metadata management and distributed resource monitoring, and support dynamic request routing decisions based on customizable policies.

A. Metadata and Resource Management

The metadata and resource management layer provides (i) lookups for data objects and services available in the distributed store (ii) routes VStore++ requests to the appropriate location and (iii) tracks information regarding available resources in the local/home environment and between the home and the remote cloud platform. The layer is organized as a key-value store where unique keys correspond to object names, service names, and as node identifiers. This allows us to maintain a uniform interface for access and manipulation of meta information regarding objects, services, and infrastructure available in the VStore++ cloud.

The current implementation of this key-value store is based on a distributed hash table (DHT), built on top of the Chimera peer-to-peer system [2]. Chimera is a lightweight

C implementation of a structured overlay that provides functionality to that of prefix routing protocols like Tapestry [8] and Pastry [9]. When an application creates and stores an object using VStore++, the corresponding metadata entry is also created and updated. The object name is hashed, and the object information is routed to a node with an ID closest to the hash value. Updates to Chimera have an overwrite policy value that determines if the metadata needs to be overwritten, if newer version of metadata is to be added by chaining, or if an error should be returned.

In our current implementation, every object's key is a 40 bit hash value generated by the object name. The value entry in the key-value store is a serialized data containing object location and metadata, such as tags, access information, etc. The location field can map to a node in the local home cloud or to a remote cloud.

Similarly, information regarding services deployed in VStore++ is also stored in key-value store. For services, our current prototype uses unique keys derived from the service name and identifier. The value associated with each entry in key-value store is a string identifying the nodes where the service is currently available. Additional service information is maintained in *service profiles*, which encode the minimum resource requirements for a service for a given SLA for the different types of nodes.Our current assumption is that such profiles are determined *a priori* and made available to VStore++ when services are deployed.

Finally, we use the same key-value store to maintain resource information for the Cloud4Home overlay. A key-value entry exists for each physical resource, with keys derived based on the nodes' IP address in the home cloud. The structure of resource monitoring mechanism along with its pseudocode is shown in Figure 2. On each node, Chimera provides a logical tree view of other nodes in the overlay, implemented as a red-black tree. Nodes periodically update their current resource usage in the key-value store using their node ID as key and serialized resource information structure as value. The updates are performed through a resource monitoring utility module. When an object needs to be stored or processed, VStore++ makes a *chimeraGetDecision()* call to obtain a list of nodes and for each node, queries the key-value store for the node's resource information. This information is used to determine the most suitable target node for a service request, as described in the following subsection. The 'policy' parameter in this operation makes it possible to support multiple decision policies, where requests are routed to target nodes depending on overall service performance, vs. achieving balanced resource utilization or improved battery lives for portable devices.

In addition to the DHT-based key-value store, we have enhanced Chimera with capabilities for dynamic overlay reconfiguration, caching, and replication [10, 11]. Our Cloud4Home prototype uses a simple metadata caching and replication functionality built into the metadata management

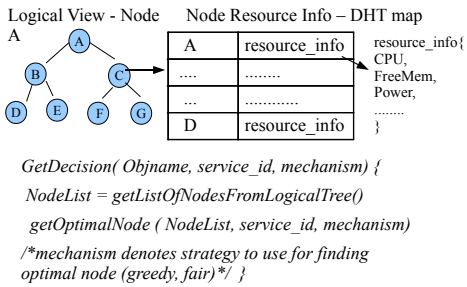

Figure 2. Resource Monitoring

layer to support improved availability and reliability. Key-value entries are cached onto intermediate hops on each request's path through the DHT overlay, and state can be replicated using a fixed replication factor. Whenever a key-value entry is modified, the corresponding caches are also updated. This approach is suitable for the current scale and dynamism in home environments considered in our research, but more scalable and robust methods may be necessary for larger home or office deployments. Regarding overlay formation and reconfiguration, our implementation permits nodes to dynamically join or leave the Cloud4Home infrastructure. Whenever a node enters or exits, it sends a message to its right and left nodes in the logical tree structure. A departing node's keys are always redistributed among the available set of nodes.

Naturally, there exist many alternative implementations of this layer for VStore++, including centralized ones or those that distribute the key-value store in a manner that is proportional to the local resources/capabilities of each node in the distributed overlay [12]. Our future work will investigate such alternatives. We next describe in more detail the basic operation and components of VStore++.

B. VStore++ Operations

We next describe the basic types of operations supported by VStore++, including their use of the metadata and resource management layer discussed above. VStore++ supports storage and retrieval of state represented as objects via *store* and *fetch* operations. Due to the heterogeneous target environment, with end-user devices exhibiting significant differences in their storage or computational capabilities, it is often necessary for additional processing to be associated with these basic accesses. For instance, accessing a large video file via a smart phone with limited display and networking capabilities may require reformatting and compression of the original video stream. In addition, there is need to support services that explicitly operate on stored data. In the home surveillance example, captured images must be processed to first detect faces, and then to run face recognition algorithms, the latter also accessing appropriate training images before an alert can be raised. For these reasons VStore++ also supports *process* operations, which

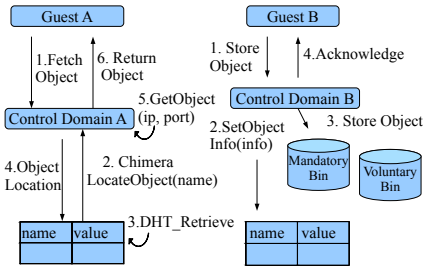

Figure 3. Operations - Store & Fetch

allow a service deployed in the home cloud to be invoked explicitly, or jointly with the object *store* or *fetch* operation.

Store. The *store* operation is represented in Figure 3. To store an object, an application must first invoke VStore++'s *CreateObject()* call to map a file to an object, which also results in the creation of the mandatory meta information, like name and type. It then invokes *StoreObject()*, which transfers the object from the application's guest domain to VStore++'s control domain. This is where the decision where to store the object is performed. By default, the object is stored in the node's *mandatory bin* – the set of resources available for applications hosted on the same node. In cases where the mandatory bin is full, or when an explicit storage policy specifies otherwise, the data is stored elsewhere, either in the voluntary resources available on other nodes in the home environment, or in a remote cloud.

It is important to note that *store* operations provide strong controls – via policies – over where data is stored, in contrast to what might be done in a distributed file system that distributes blocks and/or caches them on arbitrary nodes in a system. This provides management layers with the opportunity to control storage locations to meet privacy and/or performance demands. Specifically, the target location for the store operation is determined via the policy associated with the store. The service policy describes a set of rules which 'guide' the routing of the store request. For instance, in the home surveillance application, we may specify a service policy where objects (i.e., images) are stored on a desktop in the home cloud vs. in the remote cloud based on their size. Experimental results described in Section V discuss the benefits of one such policy. In our current implementation, these policies are represented as a set of statically encoded rules. Our future work will explore opportunities to associate learning methods and support dynamic adaptations.

Finally, VStore++ supports both blocking and non-blocking store operations, where blocking operations incur the cost of an additional acknowledgement. In all of these cases, the key-value store is updated with metadata and location of stored objects.

Fetch. As shown in Figure 3, a *fetch* uses the *FetchObject()* operation. Similarly to the *store* case, the operation is passed to VStore++ domain, where a message (IPC) is sent to the Chimera-based metadata module to determine the location of the object, whereupon the object is requested from the owner location specified in Chimera. Once the object is fetched, it is passed to the application's guest VM.

Fetch and Process. An object fetch operation may be explicitly associated with certain processing, specified via a service identifier. In that case, VStore++ performs the following steps. When the node storing the object (i.e., 'object owner') receives the request, it uses the service identifier to first determine if the requesting node is capable of executing the service itself. In that case, the object is simply returned as in the regular fetch operation, and the service processing is performed at the requesting node's VStore++ guest domain. Otherwise, the object owner checks whether it is capable of performing the required service, and if so, returns the output of the operation.

If neither of these is true, the 'value' field for the service is used to determine other possible targets, including in the remote cloud for execution of service. In the event of multiple possible locations, a decision is performed to determine the actual processing target. This step considers the time to locate the target node, the associated data movement costs for the argument and resulting object, and the service processing requirements and execution time. We maintain the latter information for each node as part of the service profile mentioned in the previous subsection. In our current implementation, we assume constant target-location time and we approximate the data movement costs by considering the movement of the argument object only (which is accurate when the resulting objects are of the same size, independent of where the service is performed). We use the key-value entries for each of the possible target nodes for resource information regarding their current processing loads and available bandwidth, and make a selection for a suitable service execution location. All results shown in Section V include the time for performing this decision process.

Process. Applications can explicitly invoke a processing service on objects already stored in VStore++. The destination of the service execution is chosen in the same manner as described above – by selecting the most suitable of all possible locations that support the service. For instance, in the home surveillance application, a process operation may be invoked on a set of stored images, to first perform face detection, and next face recognition processing on each image. Depending on image sizes, processing complexity, and resource availability, this may require movement of images among nodes in the VStore++ cloud, as shown in the examples evaluated in Section V.

C. Interfacing with Public Clouds

A key component of VStore++ is its ability to interface the home cloud infrastructure with remote public clouds. This is necessary to provide access to shared state or services available in the public cloud, or to transparently increase

the storage or computational resources available in the home cloud. As mentioned earlier, the actual location of the service execution, including for storing data, may be guided by resource availability or by other constrains, such asresponse-time SLA's, data privacy requirements, etc.

One or more nodes in the home cloud support a public cloud interface module, responsible for routing all remote cloud interactions. In our current implementation, the VStore++ domain on each node includes an interface to Amazon's S3 storage cloud, but other implementations, where the public cloud interactions areperformed only via some subset of designated nodes (e.g., nodes with sufficient Internet connectivity) are possible. The location of data object and availability of services in remote cloud are maintained in the same key value store. For data object, URL location of object in users S3 storage bucket is stored as value.

IV. Implementation

VStore++. VStore++ is implemented in C++ and majority of its components are run at user level. It currently utilizes the C++ boost library for supporting multi-threaded interfaces both at host and VM domains. Every method call in VStore++ is converted into a command. The command based interface is used for communicating between virtual machines and remote nodes. Each command packet consists of packet length, command type, the requesting service ID, VMs domain ID, shared memory reference and command data. The command data depends on command type (e.g., object name, processing command). Commands are usually less than 50 bytes and use TCP/IP sockets. The command based mechanism helps with implementing asynchronous fetch and store operations.

For object transfers between remote machines, we use the Linux zero copy mechanism using splice and tee, which provides kernel to kernel socket-based data transfer and avoids user space overheads. Larger objects are mapped to files before they are transferred. For data transfers between the host dom0 and guest VM, we utilize XenSocket, a high throughput shared memory kernel module [13] that provides a socket-based send() and recv() interface. Before every transfer, the data receiver creates a shared descriptor page and grant table reference which is sent to the sender before communication begins. The receiver allocates thirty two 4 KB pages. For better performance, the page size can be increased up to 2 MB if the devices have larger memory. For data storage across public clouds, we create a wrapper over the Amazon S3 interface [14] which is a blocking call that uses a TCP/IP-based data transfer mechanism.

Metadata management and service discovery. Metadata management is implemented by extending the DHT-based Chimera into a key-value store. This is a C-based implementation and has both Linux and Windows port. VStore++ communicates with Chimera using IPC. It has a basic put and get interface wrapped under the VStore++ interface to provide richer functionality. In our current implementation, every node registers its list of services with the key-value store using a service name concatenated with service ID as key, and a value that is a list of nodes supporting a service along with a service policy.

Resource Monitoring. In addition to the description in Section III, we added a custom resource monitoring utility to Chimera using the Linux glibtop library. The utility updates resource information in the key-value store after a configurable time period (to contain messaging overheads). A simple file system watcher component keeps track of mandatory and voluntary bin space.

Use cases. In the home security use case, face detection and recognition use OpenCV. The original code loads a training dataset to compare against images. For our prototype, we modified the code to run it as service, with training data and a set of images to be recognized as inputs, and output being ID of the best matched image. As a representative media conversion service, we use the x264 encoding [15] library. In both cases, application performance depends both on the size of input data and on its complexity. To avoid data content-related perturbations in performance measurements, care is taken to select images and videos of similar complexities, by repeated experimentation. All components of the current prototype implementation are available through svn.

V. Experimental Evaluation

The Cloud4Home prototype is evaluated in a prototypical home environment with high end connectivity to the Internet and thus, to remote cloud resources. Experiments use an implementation on Xen-3.3.0 as the virtualization platform, with VStore++ as the object-based storage service, and customized Chimera 1.20 with support for DHT, peer-to-peer communication services, and resource monitoring. The experimental testbed consists of 5 dual-core 1.66 GHz Intel Atom N280 netbooks and a 2.3 GHZ 32 bit Intel Quad core desktop machine, running Linux 2.6.28 on Xen. Internal home communication capabilities exceed the connectivity available to remote cloud facilities, using a 95.5 Mbps Ethernet LAN for some of the nodes. Access to public Amazon EC2 services and to S3 cloud storage uses the Georgia Tech wireless network, offering maximum wireless bandwidth close to 6.5 Mbps for download and 4.5 Mbps for upload, with average around 1.5 Mbps.

Experimental results validate the Cloud4Home approach by demonstrating (i) the need for enhancing remote cloud services with home cloud infrastructure, (ii) the importance of flexibility in data placement on storage and location where data manipulation services run, in the home or the remote cloud, due to noticeable differences in the levels of performance and variability, (iii) the advantages of using aggregate resources offered by local and remote service instances, (iv) all the above with moderate overheads for data services and content sharing via the VStore++ implementation.

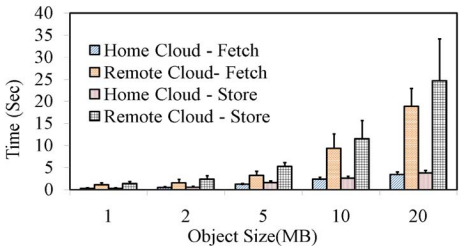

Figure 4. Home vs. remote cloud latency

File Size(MB)	Total(ms)	Inter Node(ms)	Inter Domain(ms)	DHT Lookup(ms)
1	228	103	25	12
2	454	190	37	13
5	1160	513	57	13
10	2522	1042	189	14
20	2477	2079	386	12
50	5174	4678	480	16
100	15180	13577	1603	12

Table I
HOME CLOUD FETCHES: COST ANALYSIS.

A. Importance of Home Cloud Services.

It can be important to deploy cloud services on 'nearby' resources, such as those available in a home environment. Figure 4 shows the latency and the latency variation for fetch and store accesses to data stored in nodes in a home vs. a public remote cloud. For the home cloud measurements, the dataset is distributed across all nodes in our home prototype, so data accesses are made to both on-node and off-node storage. As evident from the graph, both the absolute latency and particularly the latency variability are significantly increased when accessing public cloud storage. These increases become more significant for larger data sizes. For remote cloud accesses, additional variability exists between the two types of storage operations, due to differences in the available upload vs. download bandwidth. Even with improvements of communication resources (i.e., bandwidth) to the remote cloud, these trends will continue to exist, particularly due to the use of the shared Internet infrastructure between homes and remote public clouds. These measurements motivate the Cloud4Home approach in which services can also be run on 'nearby' resources (e.g., in the home) that are accessible with lower and less variable latencies.

VStore++ Overheads. The next set of measurements assesses the overheads associated with the basic mechanisms of VStore++. The results shown in Table I represent the costs associated with the *fetch* operation, with cost profiles for other VStore++ operations exhibiting similar trends. Inter-node transfers represents the cost for interacting with other nodes in the home environment. Inter domain costs correspond to the interactions between the application VM and the VStore++ domain via the XenStore shared memory channels, and those costs are small compared to the unavoidable costs of inter node accesses. In both, and as expected, we observe linear increases in these costs as object sizes increase. The costs of accessing the metadata management layer via the key-value store, however, remain constant for the fixed-size home cloud used in our work, independent of object sizes, and is negligible for larger objects.

In all of the measurements, the obvious differences are observed when running in a virtualized vs. non-virtualized setting: virtualization requires additional memory resources and tends to result in higher CPU utilization. Our prior work has already investigated the overheads associated with virtualizing the different types of devices used in our prototype system [1] which is why we do not further elaborate on them. This is also because industry trends indicate that hardware-level support for virtualization will become ubiquitous on newer generation devices, including on embedded platforms for mobile devices and smartphones. In summary, the overheads experienced on current platforms are moderate, with further reductions expected on next-generation platforms.

Tradeoffs in data placement. There are definite tradeoffs in the costs of data access not only in the home, for on- vs. off-node resources, but even more so when using remote cloud resources. We use the eDonkey [16] peer to peer dataset to demonstrate these tradeoffs, the goal being to emulate representative local vs. remote access patterns. In order to use it, the dataset is modified as follows. The original dataset contains files of different formats and sizes, where each file is described with an identifier, size and tags that define its context, and each access is tagged with a client ID and time. The original dataset represents a large number of clients performing only a few repetitive file accesses. We modify it by combining clients into smaller sets (emulating 6 clients) that each access a large number of files (1300 in total), performing repeated accesses across these files. The percentage of store vs. fetch operations is set to 60% and 40%, respectively.

First, we evaluate the tradeoffs when using a remote cloud for data storage service. We classify all objects into small (1-10 MB), medium (10-20 MB), large (20-50 MB), and super-large(50-100 MB) buckets. In each experiment, we store the objects of a single bucket into the remote cloud, and we structure our experiments so that the total number of bytes in a bucket is kept constant (Method 1 in Figure 5), or so that the total number of files in a bucket is constant (Method 2). We next use the access traces to compute the average throughput for all remote cloud interactions.

The results in Figure 5 indicate that for both types of experiments, the throughput measurements show similar trends. Originally, as the size of individual file transfers to and from the remote cloud increases, the aggregate throughput actually increases. This is due to the use of TCP as an underlying transport. First, longer transfers are performed mostly in the congestion avoidance vs. slow-start phase, thereby utilizing more of the available bandwidth. Second, cloud providers such as S3 increase the TCP window size during a single transfer up to some maximum limit,

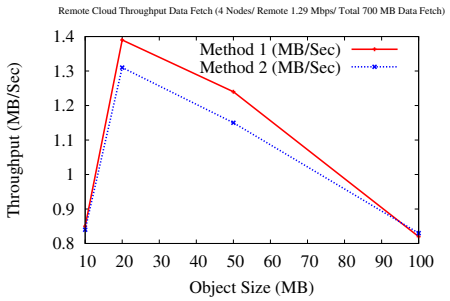

Figure 5. Remote Cloud - optimal object size

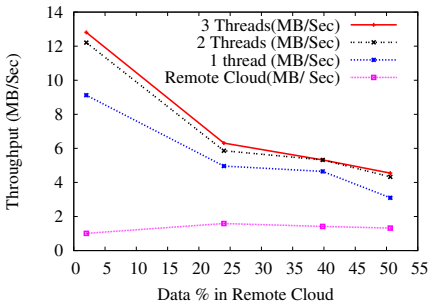

Figure 6. Fetch Throughput

approximately 1.6 MB in the case of S3. Therefore, longer transfers benefit from the use of this larger window size.

Beyond a certain point, throughput starts to deteriorate rapidly. The observed degradation is primarily due to traffic shaping and rate limiting policies enforced by ISP providers, which become visible for long bandwidth-hogging data transfers. This implies that when placing data in remote clouds, certain data sizes result in significantly better performance. The 'optimal' sizes vary depending on available upload/download bandwidth between the home and remote cloud, transport-level parameters, such as TCP window sizes used by the cloud provider, and traffic policing enforced by third-party ISP providers. In our experimental setup, the best aggregate throughput levels are achieved when using remote clouds for object sizes of approximately 20 MB.

These results indicate that, despite the significantly larger and more variable latency, remote cloud accesses can be improved by careful selection of the types of interactions for which they are being used (e.g. storing data of certain sizes).

B. *Utility of joint usage of home and remote resources.*

We modified the synthetic dataset to consist only of objects with the 'optimal' data size determined in the above experiments – 10-25 MB – and distributed it across the home and remote resources using a policy that stores private data (in our case all .mp3 files) locally and shareable data (i.e., all other types of files) remotely. The lowest curve in Figure 6 shows the aggregate throughput when client applications executing on 3 of the 6 devices in our home cloud perform VStore++ file accesses, one at a time. We avoid using all 6 home devices so as to limit the contention for the scarce bandwidth resource between home and remote cloud devices. As the percentage of data stored in the remote cloud increased, the aggregate throughput decreased when only a single thread performs sequential object accesses.

To observe the effect of increased number of concurrent access, we modified the client application to use multiple threads. Threads fetch objects from the home or remote cloud, transparent to the application. When content in present mostly in the home cloud, as the number of concurrent requests made by different threads increase, the overall throughput of system increases by factor of 45%. In this case, most access are performed at home, making effective utilization of the locally available bandwidth, with less contention for Internet bandwidth. As the number of remote accesses increases (i.e., more content present in the remote cloud), we continue to observe benefits due to increase of concurrency, but those benefits are reduced due to the fact that they all contend for the aggregate bandwidth available to the remote cloud.

The utility of joint usage of home and remote resources is also illustrated with the following example. Consider an application where a sequence of images is to be compared against an existing image dataset, for instance using a face recognition algorithm. We compare three different scenarios: (i) the image sequence is processed at home, using a 60 MB dataset stored across home devices, (ii) the processing is performed on EC2 instances in the Amazon cloud, using 190 MB dataset, consisting of our original 60 MB of images, plus additional public images available in the remote cloud only, and (iii) the sequence processing is split between the home and remote cloud. In general, the actual decision how processing should be split must also consider the state present in the home vs. remote cloud and its impact on the quality of information provided by the specific service. In this example, we use a simplistic policy which splits the image sequence roughly proportional to the amount of home vs. remote resources, to illustrate the feasibility and utility of our approach. The resulting processing times for each of these scenarios are 162 sec, 127 sec, and 98 sec, respectively, demonstrating significant importance and performance gains due to joint usage of home and remote cloud resources.

Object Processing and Storage. We next evaluate the flexibility offered by VStore++ in combining object manipulation functions with storage. Consider a use case derived from a home surveillance application. Captured images are first processed by a CPU-intensive face detection step (FDet), followed by memory-intensive face recognition (FRec). Both algorithms are deployed on two home nodes: *S1* – in 512 MB VM with one VCPU on a 1.3 GHZ dual-core Atom platform, and *S2* – in a 128 MB multi-VCPU VM on a 1.8 GHz quad-core processor. In addition, both algorithms are supported in an extra large EC2 para-virtualized instance

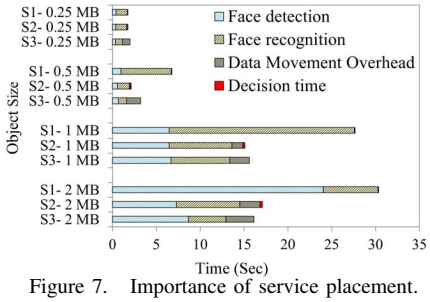

Figure 7. Importance of service placement.

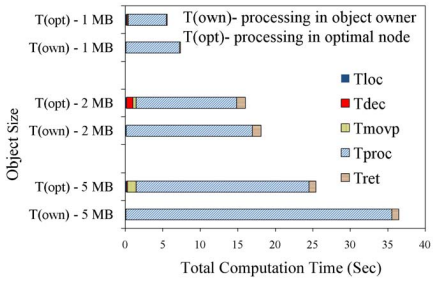

Figure 8. Feasibility of dynamic request routing.

with five 2.9 GHZ CPUs with 14 GB memory, labeled *S3*. We use images of size 0.25, 0.5, 1 and 2 MB. For each size, we use different resolution of the same image. Since the training data for FRec is usually very large, we make the assumption that it is available on any of the processing locations, to avoid considering the costs of moving such large content in and out of a remote cloud.

The results in Figure 7 represents processing time required to execute the home surveillance application from low-end Atom node, *S1*.For each data size, we consider the costs of individual steps in the processing pipeline when performed on *S1*, vs. on *S2* or *S3*. For small image sizes, the resources available on *S1* are sufficient to efficiently execute the entire processing pipeline, particularly since this eliminates the need for data movement. As image sizes increase, benefits from the availability of additional computational resources on *S2* outweigh the data movement costs. Finally, for the largest image size considered, the limited amount of memory on the *S2* VMs starts delaying the execution of the FRec step. In this case, the most efficient deployment of the processing service is the one that uses the remote cloud resources, *S3*, despite the even greater data movement costs. These tradeoffs demonstrate the need for flexible and dynamic mapping of object access and manipulation codes in a manner that takes into account runtime resource availability and cost.

We use another example, based on a media conversion service that downgrades files from the '.avi' video format to a mobile compatible '.mp4' format, using the x264 CPU-intensive library. A low-end Atom-based device 'owns' a video file, which is being accessed by another mobile device. (i) The format conversion may happen at the 'owner' node (T_{own} in Figure 8), or (ii) VStore++'s mechanisms for dynamic resource discovery may determine that a third, desktop node, is most suitable for the execution of this service. The observation for T_{opt} in Figure 8 show that the latter(ii) case results in substantial performance gains, despite the additional costs for moving data from owner to the desktop node and executing the VStore++ decision algorithm. These results demonstrate the importance and feasibility of the Cloud4Home approach and its VStore++ realization.

VI. Related Work

Cloud4Home provides transparent virtualized object storage and processing for applications executing in a VM. The object nature of its VStore++ realization borrows from the SWALLOW project [17], which was among the first system to implement object stores in file systems, and a number of other object-based file systems [18–20]. Cloud4Home also borrows from extensive prior work on key-value stores (e.g., Dynamo developed for large-scale data centers [21]), by implementing a simple a DHT-based [2] key-value store built on top of the Chimera overlay mechanism.

VStore++'s approach to virtualization stems from our previous work on a virtualized object store termed O2S2 [22]. Contributions of O2S2 includes efficiency includes efficiency in implementation and exploration of role based object access controls for trusted vs. untrusted VMs. We do not currently use those access control methods, instead focus on the distributed storage and processing capabilities for VMs.

Gibson et al. [23] was one of the first to associate processing with storage, and recent work generalizes the storage-centric methods proposed in earlier work (i.e, quFiles [24]) to associate dynamic data processing with file accesses. In comparison with VStore++, qufiles retains a strictly file-based interface, which would make it more difficult to interface with object-based APIs like Amazon's S3. The semantic information available for objects also makes it easier to associate various operations with the entities being accessed, and make decisions whether to move data where operations are running and/or apply operations where data is currently located. This can be exploited to better deal with the relatively larger degrees of heterogeneity 'at the edge' vs. in server systems.

VII. Conclusions and Future Work

Cloud4Home is an approach to realizing end-user services that can leverage both home and datacenter resources. A specific implementation of the concept for data services, termed VStore++, is implemented in a virtualized system so that both the locations of data objects used by such services and selective processing on those objects are easily changeable as well as adapted to application needs

and current resource availabilities. The project is called Cloud4Home because it is not about reproducing datacenter level cloud infrastructures in the home environment. Instead, it is to make it easy to use both home and remote cloud facilities to provide better services for the home. Current use cases target the home environment, but Cloud4Home could easily be generalized to operate in office or larger scale environments like hospitals. Performance advantages derived from flexible home and datacenter operation include reduced latency of service provision, reduced data rates and bandwidth needs to/from end systems, while still retaining the potential benefits of using large datacenter storage and processing capabilities.

There remain many open issues with Cloud4Home, the most notable ones being (i) to deal with data privacy – i.e., to implement and experiment with richer access control methods and policies, (ii) to adapt and expoit the advantages offered by heterogeneity, (iii) to understand how to scale to larger numbers of @home and then in the cloud participants, (iv) to design and evaluate mechanisms that adapt to the changing network conditions, and (v) to evaluate use cases in which multiple Cloud4Home infrastructures collaborate. A concrete example of the latter(v) would be a 'neighborhood security' system in which multiple Cloud4Home systems interact to provide effective security services for entire neighborhoods. Other examples would be media or gaming services in which multiple home systems interact in social networks or joint games. Technical issues outstanding with Cloud4Home include better optimizations of large-scale object transfers across machines, by using better object transfer protocols, additional automation for selecting the locations at which certain operations should be run or where objects should be placed (i.e., policies), and aricher infrastructure for easily formulating and running diverse policies.

ACKNOWLEDGMENT

The authors would like to thank the ICDCS program committee reviewers as well Alex Merritt, Hobin Yoon and CERCS Kernel group members for their helpful comments. We gratefully acknowledge the technical support and guidance provided by researchers at Motorola Labs in Schaumburg, and Intel's Embedded and Communications Group.

REFERENCES

[1] S. Kannan, K. Babu, A. Gavrilovska, and K. Schwan, "Vstore++: Virtual storage services for mobile devices," in *MobiCloud '10 In Proceedings of International Workshop on Mobile Computing and Clouds*, Oct. 2010.

[2] "Chimera," current.cs.ucsb.edu/projects/chimera/.

[3] A. Pai, B. Seshasayee, and K. Schwan, "Customizable multimedia devices in virtual environments," in *MODUS '08*, 2008.

[4] "Opencv," http://opencv.willowgarage.com/wiki/.

[5] S. Hastings, S. Oster, S. Langella, D. Ervin, T. M. Kurç, and J. H. Saltz, "Introduce: An open source toolkit for rapid development of strongly typed grid services," *J. Grid Comput.*, vol. 5, no. 4, pp. 407–427, 2007.

[6] A. Azagury, V. Dreizin *et al.*, "Towards an object store," in *11th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2003.

[7] "Lustre," http://wiki.lustre.org.

[8] B. Zhao, L. Huang *et al.*, "Tapestry: a resilient global-scale overlay for service deployment," *Selected Areas in Communications, IEEE Journal on*, vol. 22, no. 1, Jan. 2004.

[9] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware*, 2001.

[10] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *SOSP*, 2001.

[11] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1)lookup performance for power-law query distributions in peer-to-peer overlays," in *NSDI*, 2004.

[12] S. Zoels, S. Schubert, W. Kellerer, and Z. Despotovic, "Hybrid dht design for mobile environments," in *AP2PC*, 2006.

[13] X. Zhang, S. McIntosh *et al.*, "Xensocket: a high-throughput interdomain transport for virtual machines," in *Middleware*, 2007.

[14] "S3 tools," http://s3tools.org/s3tools.

[15] "x264," http://www.videolan.org/developers/x264.html.

[16] S. Blond, F. Fessant, and E. Merrer, "Finding good partners in availability-aware p2p networks," in *SSS*, 2009.

[17] B. M. Oki, B. H. Liskov, and R. W. Scheifler, "Reliable object storage to support atomic actions," *SIGOPS Oper. Syst. Rev.*, vol. 19, December 1985.

[18] S. A. Weil, S. A. Brandt *et al.*, "Ceph: A scalable, high-performance distributed file system," in *OSDI*, 2006.

[19] F. Wang, S. A. Brandt *et al.*, "OBFS: A File System for Object-Based Storage Devices," in *Conf on Mass Storage Systems and Technologies*, 2004.

[20] M. Mesnier, G. R. Ganger, E. Riedel, and C. Mellon, "Object-based storage," 2003.

[21] G. DeCandia, D. Hastorun *et al.*, "Dynamo: amazon's highly available key-value store," in *SOSP*, 2007.

[22] H. Raj and K. Schwan, "O2s2: enhanced object-based virtualized storage," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, 2008.

[23] E. Riedel and G. Gibson, "Active disks - remote execution for network-attached storage," Tech. Rep., 1997.

[24] K. Veeraraghavan, J. Flinn, E. B. Nightingale, and B. Noble, "qufiles: The right file at the right time," *Trans. Storage*, vol. 6, September 2010.