VICCI: A Programmable Cloud-Computing Research Testbed

Larry Peterson, Andy Bavier, Sapan Bhatia
Princeton University

September 7, 2011

1 Overview

VICCI is a programmable cloud-computing research testbed that provides an experimental platform for developing next-generation Internet services. VICCI encompasses a distributed set of virtualized compute clusters and networking *hardware*, as well as the *software* that enables multiple researchers to innovate both at and above the infrastructure layer. VICCI is designed to support research into the design, provisioning, and management of a global, multi-datacenter infrastructure—the *cloud*—and also into the design and deployment of large-scale distributed services that use such an environment. VICCI is a distributed instrument, with a point-of-presence at Princeton, Georgia Tech, Stanford, and the University of Washington, along with international clusters in Europe and Japan.

VICCI enables a broad research agenda in the design of network systems that require both multiple points-of-presence and significant compute/storage capability at each site. It enables research in:

Building block services designed to be used by other cloud applications. As cloud computing becomes increasingly prevalent, the demands for new services will increase. VICCI supports the development of much-needed services that address issues of replication, consistency, fault-tolerance, scalable performance, object location and migration. Researchers can explore multiple—sometimes conflicting, sometimes complementary—approaches to these problems.

New cloud programming models designed for targeted application domains. These models—focused, for example, on constructing virtual worlds and managing personal data—are designed to give application writers a way to think about programming the cloud.

Cross-cutting issues at the foundation of the cloud's design. The VICCI testbed allows research teams to investigate fundamental issues related to cloud computing, including how the network is managed within data centers, between data centers, and between a multi-site cloud and end-users, and how to build a trusted cloud platform that ensures the confidentiality and integrity of hosted computations.

VICCI has been bootstrapped with working software from PlanetLab, a next-generation Internet testbed developed at Princeton that is used by thousands of researchers and has carried live traffic for millions of users worldwide. Our ultimate goal is to fold the results of VICCI research back into the testbed itself, allowing us to improve the system over time and making it an even more effective platform for research into scalable network systems.

VICCI is designed to provide a realistic environment to evaluate and deploy scalable network services. Our philosophy for VICCI is strongly influenced by our experience with PlanetLab, which has demonstrated the importance of deploying experimental network services on platforms that are realistic enough to attract a real-user community. It is only by observing how experimental systems are used in practice that researchers

learn the right set of problems to solve, and these experiences also provide crucial training for future generations of computer scientists. Toward these ends, VICCI is designed to support *deployment studies* of the prototype systems put forward by the research community. In doing so, our goal is to accelerate the research and teaching process by supporting the seamless migration of scalable services and applications from early prototype, through multiple design iterations, to a continuously running experiment that generates meaningful results by carrying real traffic on behalf of a large end-client community.

2 Research Activities Enabled

VICCI is a programmable cloud-computing research testbed, encompassing both a distributed set of virtualized compute clusters and networking *hardware*, and the *software* that enables multiple researchers to innovate both at and above the infrastructure layer. VICCI is designed to support research both into the design, provisioning, and management of a global, multi-datacenter infrastructure—*the cloud*—as well as the design and deployment of large-scale distributed services that use such an environment.

VICCI enables a broad research agenda in the design of network systems that require both multiple points-of-presence and significant processing/storage capabilities at each of those sites. This section gives several example projects that illustrate the types of research supported by VICCI. The examples are drawn from the initial set of faculty and senior researchers that have been instrumental in creating—and are already using—VICCI:

- **Princeton University:** Larry Peterson, Jennifer Rexford, Vivek Pai, Michael Freedman.
- **University of Washington:** Tom Anderson, Ed Lazowska, Hank Levy, Steve Gribble, Arvind Krishnamurthy.
- **Stanford University:** Nick McKeown, Philip Levis, Mendel Rosenblum, John Ousterhout, Guru Parulkar.
- **Georgia Tech:** Nick Feamster, Cristian Lumezanu, Wenke Lee.
- **ETH Zurich:** Timothy Roscoe.
- **Max Planck Institute for Software Systems:** Peter Druschel, Rodrigo Rodrigues, Paul Francis, and Krishna Gummadi.
- **University of Tokyo:** Aki Nakao.

As a point of reference, the research described in this section is organized into three subsections, loosely corresponding to three common models of cloud services [17]: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). We avoid a rigid interpretation of the XaaS model, however, as our goal is to permit full exploration of clouds without unnecessarily constraining research with a particular architecture.

2.1 Building Block Services

These are cloud services designed to be used either directly by end-users or as building blocks of other cloud applications. They address issues of data consistency, replication, scalable performance, object location, and migration—but sliced-and-diced in different ways. That is, the services described below reflect both

the variety of building blocks required and the diversity of design principles. This diversity is important, especially since it seems unlikely that one can design "one-size fits all" services for cloud computing.

2.1.1 A Consistent DHT for Cloud Applications

Large-scale distributed systems increasingly rely on scalable storage and lookup services such as Distributed Hash Tables (DHTs). Researchers have leveraged scalable lookup to design distributed filesystems, content distribution networks, location sensing systems, and rendezvous-based communication infrastructures. DHTs are also used in various deployment settings ranging from data center applications to peer-to-peer systems. Most existing DHTs, however, support only weak consistency semantics, often with negative consequences for the client application programmer [22, 30, 23, 14, 4, 24, 3, 5]. For example, with Kademlia [14], the DHT used to coordinate swarm membership within BitTorrent, a read of a key's value after a write to the key is not guaranteed to return the new value. Similarly, Amazon's Dynamo[5]—used as a distributed store for various e-commerce applications within the data center—sacrifices consistency under certain failure scenarios. In such cases, it is the responsibility of the application to manage its state replication. Furthermore, even if data is not lost, different nodes might maintain different values for the same key and the application would have to reconcile conflicting replica values. These issues undermine the utility of DHTs by increasing the complexity of developing applications on top of them.

To address this, researchers at UW have designed Harmony [10], a DHT aimed at very large scale distributed applications, but one in which "you get what you put". The distinguishing feature of Harmony is that it organizes nodes into self-managing groups, merging and splitting with neighboring groups as necessary to deal with churn. The DHT itself manages its heterogeneous resources to provide high availability, high performance, and serializable consistency. Strict consistency is conceptually easier for the application programmer and often necessary for complex distributed applications. The hypothesis is that consistency can be provided without a prohibitive performance penalty.

An initial implementation of Harmony has been built in Python. The prototype implements the mechanisms required for consistent and robust management of groups of nodes. The researchers plan to use VICCI to identify and solve issues relating to performance, adaptability, and large-scale deployment. For instance, performance bottlenecks arising out of high geographical dispersal of nodes could be overcome by intelligently selecting group members and the leader during operations that change the group's composition. As the expected lifetime of nodes might vary based on the nature of the deployment (with data centers having stable nodesets while P2P settings having more rapid churn), Harmony could adapt to workload characteristics to improve robustness and load balance.

A primary goal of the project is to build and deploy a production version of Harmony, and consequently spur the development of a new generation of distributed systems that can assume robust, consistent lookup and storage as a fundamental building block. Today's distributed testbeds such as PlanetLab are valuable in evaluating and optimizing for wide-area peer-to-peer settings, but they are insufficient in providing a realistic environment for emulating data center settings and providing workloads that would be typical of the cloud. VICCI provides a testbed where Harmony can be evaluated and customized for applications that are deployed across multiple data centers—a particularly challenging setting given the combination of geographical dispersion across sites and large scale within a single site.

2.1.2 A Global, Content-Oriented Filesystem

Cloud computing provides an opportunity for converging three previously-disparate systems: content distribution networks, network accelerators, and filesystems. By combining these systems, the cloud can provide

a location-independent, high-performance, bandwidth-efficient system to many classes of users in a relatively transparent manner.

Content distribution networks are typically used to help scale Web sites by moving copies of files to geographically-distributed replication servers, which can address many of the scale, congestion, and latency issues that arise in centralized infrastructure [35, 19]. Beyond being Web-centric, these systems typically tend to be read-only, and require advance preparation. These requirements are well-suited to Web sites needing scalability, but are not compatible with traditional filesystem behavior.

Network accelerators, also known as wide area network (WAN) accelerators, are systems that provide point-to-point compression to overcome bandwidth limitations when sending redundant content. These systems operate transparently and can operate at a granularity smaller than files, such as network packets or pieces of packets. Unlike content distribution networks, which assume an open-ended user population, WAN accelerators typically serve closed populations within multiple locations of a single organization.

Extending traditional filesystems to wide-area environments has historically proven challenging. The earliest example of this approach is AFS [12], which allows distributed groups to share a single namespace through the use of aggressive local caching. However, data is provided only by authoritative servers and only to local users in each location, limiting location-independence. Research systems, such as Ivy [15], have used DHTs to provide greater wide-area coverage, but at a loss of performance due to the use of a non-overwriting log to record all file modifications.

To address these issues, Princeton researchers are building Syndicate [31], a cloud-based wide-area filesystem that provides scalability, performance, and efficient bandwidth usage. In order to achieve these goals, the filesystem is content-oriented—the data exchanges within the system are decoupled from the control transfers, allowing the data to be served efficiently while the control plane provides the traditional filesystem behavior and semantics. The data transfers in the filesystem are chunked using content-based fingerprinting, as is done for WAN accelerators. All data transfers of chunks are performed over a content distribution network, further decoupling the location of content access from the location of the provider and consumer.

A prototype of Syndicate is currently running on VICCI. Syndicate's "filesystem in the cloud" requires an infrastructure that, like VICCI, decouples the location of the producer and the consumer, and provides considerable storage and transformation capabilities in the wide area. The researchers also plan to use VICCI to explore hybrid approaches that combine a cloud-based filesystem with dedicated local servers or peer-to-peer networks.

2.1.3 Scalable Services with Strong Robustness Guarantees

With modern datacenters reaching tens of thousands of servers, large-scale services initially embraced weakened consistency models to achieve scalability and performance goals. This direction proved highly successful for a variety of applications, including web crawling, search, and content distribution. Still, the recent trend to move ever-more dynamic applications to the "cloud" portends a shift in service requirements—in which losing data or applying operations out-of-order may not be acceptable—as does far-flung demand for concurrent access to data and services, spread across datacenters world-wide.

To meet these changing needs, researchers at Princeton are reconsidering the challenge of building storage and replicated systems with strong robustness guarantees and at scale. Their work leverages certain trends common to many of these datacenter services, including simple data models, effective object-based interfaces, and read-heavy object workloads. Since large-scale, complex systems are typically built-up from smaller groups—which either act as different subsystems to provide varying functionality, or are simply used to partition some larger state-space into smaller, more manageable parts—their research is guided by a *group*

compositional approach to the problem. First, they are designing novel protocols for smaller groups of nodes that offer strong properties at minimal overhead, considering both fail-stop and Byzantine models of failures. Respectively, these have been realized in the CRAQ object-storage system [32] and the Prophecy system for Byzantine-fault-tolerant (BFT) replicated state machines [28]. Second, they are designing a coordination service and a suite of management algorithms that adaptively organizes these groups and composes them together, with the goal of building larger-scale systems that preserve their base guarantees. These algorithms particularly explore problems of dynamic load balancing, topological control, and security that arise both within and between datacenters.

To test the usefulness and efficacy of these strongly robust infrastructure services, the researchers plan to deploy prototypes on VICCI. They already have significant experience with VICCI since they used it to evaluate the COPS key-value store [13]. VICCI provides a realistic deployment environment since it resembles commercial datacenter environments in terms of hierarchical topology, fate sharing, and realistic network latencies and throughputs. Once the prototypes are solid, they plan to operate them as publicly-available services on VICCI. Several other distributed systems they are building—including a scalable virtual world [11], a peer-assisted content distribution network [33] and an anycasted name resolution service [27]—will rely on these core services.

2.1.4 Object Location and Distribution

Many services are replicated across multiple data centers, or migrate from one server or data center to another in response to failures, changes in demands, or fluctuations in energy prices. In this context, we explore how to handle the inherent churn in the mapping of services to particular server machines and locations. Princeton researchers are creating a new architecture called Serval [6] that redefines the relationship between naming and routing to meet the needs of these wide-area replicated services. Serval routes traffic to data centers (or "Service Systems") based on object names, and supports both successive refinement (to hide churn within the data center) and reactive packet redirection (to support seamless migration of a service from one data center to another). We believe our architecture will be scalable in the face of considerable churn, while also simplifying the design and operation of wide-area services.

The researchers intend to use VICCI to evaluate several key research issues with Serval. First, communication in Serval takes place between two "objects" with unique identifiers. The routing system directs the packets to an appropriate host, which may change over time due to failures, load balancing, planned maintenance, or migration. For scalable routing, packets are directed to a particular Service System (SS), based on a name-resolution process that is closely tied with packet routing. When an object moves, packets in flight are redirected from one SS to another. Designing, implementing, and deploying the protocol for mapping objects to the appropriate SS and host is a major part of the Serval project. Experimental evaluation in the wide area is critical to determine how the protocol scales in the face of churn in the association of objects to hosts.

Second, compared to today's Internet, Serval places more functionality on the edge nodes that connect a Service System to the rest of the Internet. These nodes are responsible for directing packets toward internal hosts, and deflecting packets to other SSes when necessary. The researchers plan to prototype and deploy an edge node that leverages the OpenFlow switches in each VICCI site. The control plane for the edge node will run on a NOX server that installs packet-forwarding rules in the OpenFlow switch. They intend to build and compare two variants of the Serval protocol, one that places more state in the edge node, and another that piggybacks state on data traffic. This will allow them to evaluate trade-offs in scalability of the edge node and the end hosts, and how quickly the system can respond to unexpected churn.

Third, the real test of Serval is whether it simplifies the design, deployment, and operation of wide-

area services. To evaluate the effectiveness of Serval, they plan to build several example services on top of a Serval deployment on VICCI. The example services include a reliable DNS hosting service to provide name-to-address mapping for geo-replicated services, a port of the CoralCDN [7] content distribution network, and a next-generation virtual-world application called Meru [11]. These applications can greatly benefit from direct support for naming and routing in terms of objects, rather than hosts.

2.2 Cloud Programming Models

These projects represent new programming models designed for targeted application domains. They give application writers a way to think about programming the cloud.

2.2.1 A Cloud for Federated Virtual Worlds

The possibilities and capabilities of virtual worlds have long been a facet of fiction. A visual, three-dimensional metaphor could provide an intuitive and simple interface for navigating information and communicating. Virtual worlds today, however, are far from these visions. Worlds today exhibit properties that prevent success similar to applications such as the web: they scale poorly, have centralized control, or cannot be easily extended.

Researchers at Stanford and Princeton are attempting to overcome these limitations by focusing on five important properties to scalably support large numbers of interacting virtual worlds [8].

- **Federation**: Many service providers may contribute computational, networking, and storage resources to run the virtual worlds. These physical administrative domains must be able to cooperate and interoperate, yet they must also have freedom in managing the internal workings of their networks.
- **Expansibility**: The design must be able to support virtual worlds with very different uses and logical administrative properties.
- **Scalability**: The ecosystem should be able to support large numbers of worlds, worlds with great expanses of virtual regions, and large numbers of objects and activity within small regions of virtual space.
- **Migration**: Providers need to physically migrate objects and environments across hosts for load balancing, fail-overs, and optimization; users and objects need to logically migrate objects across worlds for seamless integration.
- **Security**: All parties (users, application developers, world administrators, service providers) need to be able to protect themselves from malicious entities.

Their approach is based on three key design principles. First, rather than being centralized or peer-to-peer, the system is based on federation: cooperative but not necessarily collaborative interaction between multiple parties. Second, application communication is grounded in three-dimensional coordinate spaces: objects can only communicate after being introduced through proximity. This geometric addressing decouples applications from their physical locations on hosts. Third, by using this communication model, the system can directly interface with the physical world. But it also governs how we can leverage physical constraints, such as constant flux per unit space, in order to bound communication and thus scale to large, complex environments [11].

The researchers are closely collaborating with the builders of an open, large, distributed virtual world system, called Meru. Their goal is to provide a framework for Meru to leverage, while it in turn provides an application driver for their design. They plan to build and deploy a prototype of their virtual world environment on VICCI, since it satisfies their requirements for a hosting infrastructure: low latency and high throughput between commonly-interacting objects hosted on nearby nodes; fault tolerance between different worlds or regions by limiting fate sharing; and content and computation hosted at administratively-specified locations for security and privacy.

2.2.2 A Cloud for Personal Applications

Large, online services are the dominant model today for personal applications that require more compute resources than are available on the device. Such applications provide short-term "lock-in" benefits to providers, but require large capital investment for each application, increasing the barrier-to-entry. Furthermore, the centralization of control and personal data that results from such a "mainframe model" are beginning to raise ethical, political, and legal concerns.

Researchers at ETH Zurich argue that a new class of compelling applications can be enabled by a different model: each user has a *personal* computing platform comprising a mixture of cloud resources and personal devices such as phones, PDAs, and PCs. A simple example application scenario is as follows: a user employs his phone's camera to capture location-tagged images of his surroundings, which are then processed by a rented cluster acquired on-demand, together with related images stored on the user's home server, to synthesize 3D content such as a scene reconstruction which may then be rendered on the phone a few seconds later. They are interested in the abstractions and mechanisms to support such applications that do not rely on a dedicated service infrastructure (as required with, for example, Microsoft PhotoSynth).

Such a system faces a number of challenges. One is in acquiring the right resources, in response to application load and external conditions (such as provider outages, user mobility, or changing pricing structures). A second challenge is heterogeneity—the system must capture and represent knowledge about the characteristics of diverse devices (phones, PCs, virtual machines, etc.) and allow the application to reason about their current (and future) device set. A third is routing—an integrated but heterogeneous platform for personal applications must maintain inter-device connectivity over a diverse and rapidly changing network topology.

The researchers are designing Rhizoma [36], a stable platform for deploying and accessing personal applications which ties together a dynamic collection of cloud resources with personal devices such as phones and home computers. Rhizoma consists of an *overlay network* providing uniform connectivity, resource discovery, and synchronization services, together with a *knowledge base* to handle the heterogeneity in the system and a *coordinator* which automatically acquires and releases resources (both network links and computational power) on demand. Rhizoma addresses the challenges identified above by applying constraint logic programming (CLP) and using cloud machines to run a CLP solver in realtime on behalf of the application. CLP is appealing since it combines the expressive representative power of formats like RDF with powerful reasoning capabilities based on constrained optimization. Furthermore, recent work on declaratively-specified routing protocols translates cleanly to the CLP environment, providing a powerful tool to tackle routing both within and between applications.

Current cloud computing providers are not yet a good fit for Rhizoma. They provide low-level resources in the form of virtual machines, but without control over the networking resources used nor, crucially, the locations of the virtual machines. For example, EC2 provides a choice of VMs in Europe or the US, but nothing more fine-grained. VICCI provides a stable computing environment, representative of real cloud providers, but with the additional control over VM placement and networking.

2.3 Cross-Cutting Issues

These are projects focused on broad, cross-cutting questions about how the underlying cloud infrastructure should be designed, with a particular focus on faults, networking, and security.

2.3.1 Tolerating and Detecting Faults in a Cloud

Cloud applications are often designed as a composition of multiple services. For instance, the customers of Amazon's EC2 can use the S3 storage service, which in turn uses Dynamo as a storage substrate (also used by Amazon's e-commerce platform). As another example, Google's indices are built using the MapReduce parallel processing framework, which in turn can use GFS for storage.

Faults are inevitable in such environments, and can not only have various sources, but, more importantly, various effects on the behavior of faulty components. To give a few recent examples, Amazon's S3 storage service suffered a multi-hour outage, caused by corruption in the internal state of a server that spread throughout the entire system [25]; an outage in Google's App Engine was triggered by a bug in datastore servers that caused some requests to return errors [9]; and a multi-day outage at the Netflix DVD mail-rental was caused by a faulty hardware component that triggered a database corruption event [16]. Dealing with faults that cause these complex failure modes is critical. Researchers at MPI-SWS are exploring two different vectors for handling them: masking their effects using replication, and detecting them using accountability.

Services replicate data and computation to mask the effects of machine crashes in data centers. Replicas commonly span multiple sites to withstand events that make an entire data center unreachable, such as network partitions, maintenance events, and physical disasters. However, it is commonly assumed in these environments that nodes fail cleanly by becoming completely inoperable; when such failure assumptions are not met, the results can be catastrophic as the above events suggest. Byzantine Fault Tolerance (BFT) can withstand complex faults, but current BFT protocols can become unavailable if a small fraction of their replicas are unreachable. This is because they favor strong safety guarantees (consistency) over liveness (availability). The researchers have proposed a new BFT replication protocol, called Zeno [29], that trades consistency for higher availability. Zeno replaces strong consistency (linearizability) with a weaker guarantee (eventual consistency): clients can temporarily miss each others' updates, but when the network is stable the states from the individual partitions are merged by having the replicas agree on a total order for all requests.

An alternative approach is fault detection: instead of attempting to mask the effects of faults, they are detected after the fact and their root causes traced. For instance, customers of a cloud computing service expect the service to correctly execute the code provided by the customer. If a malfunction occurs, it may be difficult to establish which node is responsible for the problem, and even more challenging to produce evidence that proves a party's innocence or guilt. In the case of a customer's bug, the provider would like to be able to prove its innocence, and in the case of the faulty provider, the customer would like to obtain proof of that fact.

To address these issues the researchers are designing a framework for running accountable web services, where the entire system being outsourced runs inside an *accountable virtual machine*. By applying the principles and protocols of accountability at the level of the virtual machine monitor, they can detect deviations from the expected behavior of an entire operating system and respective applications. VICCI will enable them to deploy their replication protocols and better understand the trade-off between consistency and availability when replicas are located in different data centers. For the fault detection component of the project, they plan to use VICCI to study the overheads of adding accountability in a virtualized environment.

2.3.2 Networking Issues

The proposed experimental platform enables a wide range of research in network support for clouds—within and between data centers, and between the cloud and users. Princeton researchers plan to use VICCI to design, prototype, and deploy new architectures that make clouds more efficient, more flexible, and easier to manage.

Simple data-center networks with static multipath routing. The routers and switches in data-center networks are a relatively large fraction of the cost and complexity in data centers. The researchers believe data-center networks could be much simpler, and more reliable, if the network elements provided a simple abstraction of "static" multipath routing. In their architecture, the network-management system, or fabric controller, computes multiple paths between pairs of nodes and installs the corresponding forwarding tables in the network elements. At the edge of the network, edge switches or even end hosts split traffic over these paths. When links fail, simple end-to-end path-failure detection allows the edge nodes to avoid directing traffic over the failed paths, without triggering any routing convergence inside the network. The network elements simply forward packets at the behest of the fabric controller, and do not need to run any control-plane protocols. They believe this simple architecture will be effective at reducing the complexity and cost of the network, while achieving efficient load balancing and fast failure recovery.

Multi-path routing to improve reliability and load balance. Existing data-center topologies rely on multi-path routing schemes for recovery from failure, but existing failover protocols (e.g., MPLS fast re-route) do not scale and are not sensitive to congestion. In data-center networks, variations in the offered demand put a heavy strain on the network, and the networks themselves can be subject to switch and port failures. Fortunately, low propagation delays enable quick feedback about network conditions, and the end-host servers are a natural place to adapt the sending rates. The researchers are exploring extensions to their previous work in scalable multipath routing in the wide-area for improving reliability and traffic load balance in data centers. The new protocols rely on having links in the network export simple statistics, such as link utilization, to the fabric controller. The fabric controller can use this information to either compute new topologies or instruct the edge nodes to change their sending rates and splitting percentages. The controller can run optimization algorithms to either derive the appropriate response for each edge node to optimize network-wide metrics, or recompute the underlying topology itself.

Peering On-Demand. Services that are hosted in a cloud may require a virtual networking environment to retain real-time, fine-grain control over how traffic enters and exits the cloud. For example, given the global nature of many Internet applications, cloud data centers can be in many diverse geographical locations; a cloud service could expose some of these connectivity options and let the service provider decide how the traffic is routed into, out of, and across the cloud for that service. Currently, this type of flexible connectivity is difficult to achieve. Cloud infrastructure providers try to balance performance and cost for Internet connectivity, and customizing upstream connectivity for each service or virtual network in the cloud may be challenging. Virtual networks, on the other hand, face multiple hurdles if they want to connect directly with the service providers on the Internet, including negotiating direct contracts with ISPs and obtaining routable IP addresses for proper peering, and connecting the virtual network to the external Internet. These challenges become even more significant when the virtual networks are short-lived (e.g., a virtual network is provisioned for a specific event). In these cases, negotiating upstream connectivity from ISPs, who prefer stable, predictable BGP sessions, may be inconvenient. In the same way that cloud computing infrastructure allows services to adapt quickly to growth and changes in demands, the network resources and connectivity (e.g., BGP peering, capacity, hosting), should also be *elastic*. The researchers are investigating the ability of these Elastic Networks to support various types of applications, particularly those that require rapid adaptation of network resources, control over inbound and outbound traffic, or both.

2.3.3 Trusted Cloud Computing

Recent studies have found that, despite the potential benefits, chief executives and IT managers are reluctant to deploy internal systems into the cloud due to fears about security threats and loss of control of data and systems. One of the most serious concerns is the possibility of confidentiality violations: either maliciously or accidentally, the employees of cloud providers can tamper with or leak a customer's confidential data.

In order to prevent confidentiality violations, customers of cloud services might resort to encryption. While encryption is effective in securing data before it is stored at the provider site, it cannot be applied in services where data is to be computed, since the unencrypted data must reside in the memory of the host running the computation. In Infrastructure as a Service (IaaS) cloud services such as Amazon's EC2, the provider hosts virtual machines (VMs) on behalf of its customers, who can do arbitrary computations. In these systems, anyone with privileged access to the host can read or manipulate a customer's data. Consequently, customers cannot protect their VMs on their own. There is a clear need for a technical solution that guarantees the confidentiality and integrity of cloud computation in a way that is verifiable by the customers of the service.

Traditional trusted computing platforms like Terra [2] take a compelling approach to this problem. Terra is able to prevent the owner of a physical host from inspecting and interfering with a computation, and also provides a remote attestation capability that enables a remote party to determine up-front whether the host can securely run the computation. Such platforms can effectively secure a VM running in a single host. However, many providers run data centers comprising several hundreds of machines, and a customer's VM can be dynamically scheduled to run on any one of them. This complexity and the opaqueness of the provider backend creates vulnerabilities that traditional trusted platforms cannot address.

Researchers at MPI-SWS have proposed *trusted cloud computing platform* (TCCP) [26], a system that ensures the confidentiality and integrity of computations that are outsourced to IaaS services. The TCCP provides the abstraction of a closed box execution environment for a customer's VM, guaranteeing that no privileged administrator can inspect or tamper with its content. Moreover, before requesting the service to launch a VM, the TCCP allows a customer to reliably and remotely determine whether the service backend is running a trusted TCCP implementation. This capability extends the notion of attestation to the entire service, and thus allows a customer to verify if its computation will run securely.

The researchers plan to implement a working prototype of TCCP. Once the prototype is built, through thorough experimentation in realistic deployment environments such as VICCI, they hope to demonstrate that TCCP can significantly improve the security of outsourced computation, and create new incentives for a more widespread adoption of cloud computing services. VICCI will also allow them to evaluate the TCCP attestation protocols in challenging yet realistic scenarios where VMs might be migrated from one data center to a different one, e.g. for the purpose of locating services closer to the clients that access them.

3 VICCI Design and Development

VICCI is designed to provide a realistic environment to evaluate and deploy scalable network services. Our philosophy for VICCI is strongly influenced by our experience with PlanetLab, which has demonstrated the importance of deploying experimental network services on platforms that are realistic enough to attract a real user community. It is only by observing how research prototypes are used in practice that researchers learn the right set of problems to solve [20]. To this end, VICCI is designed to support *deployment studies* of the research prototypes described in the previous section, as well as similar systems being proposed by the research community.

VICCI is currently under construction (for up-to-date status, see [34]). When completed, VICCI will consist of hardware and software components, as follows:

- At the hardware level, VICCI will consist of seven geographically dispersed compute clusters: four in the US (at Princeton, Stanford, Georgia Tech, and the University of Washington), two in Europe (at the Max Plank Institute for Software Systems and ETHZ) and one in Asia (University of Tokyo). Each cluster will consist of $70 \times$ 12-core servers (840 cores) connected by programmable OpenFlow switches [18]. Each cluster will be connected to the Internet (and hence, each other) by 100Mbps-1Gbps of network connectivity. The US sites are also connected by VINI [1]—a virtualized and programmable backbone network leveraging Internet2 and the National Lambda Rail. Each site will also support 10Mbps connectivity to the commodity Internet, and hence, to end users.
- At the software level, VICCI will consist of two primary components: (1) lightweight virtualization software that runs on each server, giving each researcher an isolated virtual machine (VM) in which they can run their experimental service; and (2) remote management software that sets up, provisions, and controls distributed VMs on behalf of research projects.

3.1 Rationale

The goal is to support research into system support for geographically distributed and widely used applications. These applications are currently not served by either the large central cluster model (one site, one MapReduce), or by the PlanetLab model (many poorly provisioned sites) available today. Most widely used applications fall between these two extremes—services we want to offer on a planetary scale, and so can't be hosted at just one site, but still computation and data intensive enough to not be feasible on PlanetLab's limited per-site resources.

We are building seven geographically distributed sites for the same reason commercial services provided by companies like Google and Yahoo run at multiple sites today: to locate services within 10-20ms of users. We settle on approximately 1000 cores per site as a compromise—10,000 or 100,000 cores per site would be prohibitively expensive, but O(1000) cores per site will be enough to expose the important issues such a system must face.

For example, one general class of system architecture is to build up richer services by loosely coupling a number of smaller functional groups. These groups may differ by function (e.g., the spell-checking service for web search) or by partitioning some larger dataset into individual components (e.g., using consistent hashing as in Harmony, or through a directory service as in GFS/HDFS). At O(1000) cores, service developers will have sufficient scale to deploy rich services that are composed of a number of smaller functional groups. This will naturally support research into a management framework for organizing and composing such cloud services even across researchers. As a secondary effect, the size of the clusters implies that deployments will need to stretch across multiple racks within each collocation site. Such a deployment requires that system developers grapple with varying bisectional bandwidth and latencies between cluster nodes.

Viewed another way, we believe we are provisioning VICCI with sufficient horse-power to run experimental services that support 100k to 1M end users. This target seems well justified given our experience with services this same research community has deployed on PlanetLab. In fact, we envision VICCI and PlanetLab complementing each other in their support of realistic network services, with VICCI providing a small number of resource-rich data centers and PlanetLab providing an modest compute and storage capacity distributed to locations near the end users. In other words, PlanetLab will serve as VICCI's access network.

Finally, we note that there are possible alternatives to VICCI, but they have limitations that make them less effective platforms. One is Amazon's EC2, which gives users the ability to create and configure VMs. Like VICCI, EC2 gives users the option of placing their VMs at roughly half a dozen geographically dispersed sites. Unfortunately, EC2 gives researchers no visibility into, or control over, the internal workings of the cloud, and as a consequence, is it is not suitable for experimenting with the design of the cloud itself. Moreover, our analysis of the types of workloads VICCI researchers want to run—i.e., continuously running network services that transmit terabytes of data to/from Internet hosts every day—shows that the cost of using EC2 would be significantly higher than VICCI, approaching $1M per experimental service per year.

Open Cirrus (OC) is another a multi-site cloud initiative. It involves HP, Yahoo, and Intel, plus a handful of Universities across the US, Europe and Asia. At a high level, OC has many of the same objectives as VICCI. The reality, however, is quite different. OC does not support a common set of abstractions, with each site offering a different programming model and control framework (e.g., the Yahoo testbed offers Hadoop running in Java VMs, the HP testbed offers Xen domains, and the Intel testbed is based on the Tashi VM management system).[1] Moreover, OC adopts a model of research projects being allocated large fractions of each testbed for limited periods of time (i.e., to run large, data-intensive MapReduce jobs), whereas VICCI is being designed to support continuously running services and applications that support a real user community.

Finally, Microsoft's Windows Azure offers researchers access to cloud facilities, but it also focuses on enabling applications (e.g., "exploring rich and diverse multidisciplinary data on a large scale") rather than the mechanisms, subsystems, and programming models underlying the cloud's design.

3.2 Development Strategy

PlanetLab software provides a starting point for VICCI—allowing us to rapidly deploy VICCI and make it available for early use—but it needs to be extended and specialized in several ways to support the full VICCI research agenda. The following outlines the the five key extensions we plan to make to the PlanetLab software stack.

- **Node Virtualization:** PlanetLab currently supports container-based virtualization, which means all VMs share a common (Linux) kernel. We will extend the implementation to also support other types of VMs, including Xen, OpenVZ, and Linux Containers. This will involve leveraging an open-source project (called *libvirt*) that aims to provide a unified interface for configuring many types of virtual machines. We will also ensure that these various types of VMs are properly configured for use by researchers, with the right subsystems installed.

- **Network Virtualization:** PlanetLab currently does not manage the local interconnect. We will leverage the OpenFlow switches that interconnect the cluster nodes to manage intra-cluster traffic on a per service/application basis. This will involve installing packet-forwarding rules that direct incoming requests to the appropriate nodes. We will also explore the use of OpenFlow to balance server load, while respecting the need for successive packets to reach the same server.

- **Bandwidth Management:** PlanetLab currently limits outgoing network bandwidth from each node, so as to not overwhelm the hosting site, but we will need to extend this bandwidth throttle to work

[1] Of the individual clusters participating in Open Cirrus, the HP effort has the most in common with VICCI. There already exists a close collaboration between HP and the PIs, working to wrap the HP cluster with the VICCI programmatic interface, thereby allowing VICCI users to include the HP cluster in their experiments.

across all the nodes in a cluster. Our plan is to integrate the *distributed rate limiter* mechanism developed at UCSD [21]. The mechanism enforces a global limit, but also ensures that congestion-responsive transport-layer flows behave as if they traversed a single, shared limiter. We will also explore the use of DRL across VICCI as a whole.

- **Resource Allocation:** PlanetLab currently supports a best-effort model of resource allocation. We will extend the implementation to support resource guarantees, for example, allowing VMs to be allocated entire processor cores. As the necessarily schedulers and allocators already exist, this will primarily involving extending the management software to account for the resources that have been allocated and the resources that are still available for allocation. We will also implement an admission control mechanism and a complementary set of incentives to encourage the research community to contribute more resources to VICCI.

- **Cluster Support:** The current PlanetLab deployment assumes a small number of servers at each site (generally two or three), each of which runs a *Node Manager* that responds to requests to create, provision, and destroy VMs on the node. To support clusters, we will build a new software component, called a *Site Manager*, to configure and manage a cluster of nodes as a unit, thereby eliminating the overhead of remotely managing the individual nodes at a given site. In addition, we will extend the Site Manager to be responsible for mapping VMs onto the available physical nodes. This will make it possible for users to request N VMs at a site without having to know about the availability of individual nodes.

Making these extensions will result in an instrument capable of supporting the research outlined in the previous section, but folding the results of that research back into VICCI will make it an even more effective platform for research into scalable network systems. Supporting such an incremental evolution of VICCI is a fundamental strategy of this proposal.

Sometimes called spiral design, the strategy is to build incrementally, taking experience and user feedback into account. It is a well known result of computer science research that in software or hardware construction efforts, errors are cheapest to fix when they are caught early. The best way to do that is to put the system into active use at the earliest possible moment, gain experience with the live system, and incrementally evolve the system based on what you learn. This is an approach we have used with success building PlanetLab.

We take this strategy a step further by planning for the results of the research done **on** VICCI eventually being folded back **into** VICCI. The key to doing this is to define interfaces that make it possible to support multiple versions of any given feature—both competing approaches and multiple versions (old/limited and new/improved) of various system functions. In other words, one can view the research outlined in the previous section as the "advanced development plan" for VICCI, the instrument.

4 Conclusion

We believe that VICCI will have a significant impact on the research community. VICCI will build on the successful model established by PlanetLab, which has become an essential tool for network and distributed systems research: researchers that make claims about network protocols and services use PlanetLab to demonstrate how their designs hold up in a real network. We are constructing VICCI to allow researchers to evaluate the next generation of scalable distributed systems research under real-world conditions.

We also expect VICCI, like PlanetLab, to serve as a training ground for the next generation of graduate students, giving them the opportunity to learn the art of building cutting-edge distributed systems. VICCI will expose students to the challenges of creating distributed systems that run at a global scale, giving them experience coping with transient failures, differences in connectivity cliques, variations in latency and bandwidth, abuses inflicted by real users (some of which are malicious), and more. VICCI will expose a new generation of students to the challenges of creating distributed services that support, comprise, and run within "the Cloud".

References

[1] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In vini veritas: realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14, 2006.

[2] T. G. Ben, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *19th ACM Symposium on Operating System Principles (SOSP'03)*, pages 193–206. ACM Press, 2003.

[3] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 97–108, New York, NY, USA, 2005. ACM.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 205–220, New York, NY, USA, 2007. ACM.

[6] Erik Nordström and David Schue and Prem Gopalan and Matvey Arye and Steven Ko and Jennifer Rexford and Michael J. Freedman. Serval: An End-Host Stack for Service-Centric Networking. http://www.serval-arch.org/docs/serval-tr11.pdf.

[7] M. J. Freedman, E. Freudenthal, and Mazières. Democratizing Content Publication with Coral. In *Proc. 1st Symposium on Network Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004.

[8] M. J. Freedman, T. Funkhouser, P. Hanrahan, V. Koltun, and P. Levis. A network architecture for federated virtual/physical worlds. In *NSF NeTS-ANET Grant (#0831374–Princeton, #0831163–Stanford)*, 2008.

[9] GAE outage. http://groups.google.com/group/google-appengine/browse_thread/thread/f7ce559b3b8b303b.

[10] Harmony wiki. http://hyperion.cs.washington.edu/projects/harmony.

[11] D. Horn, E. Cheslack-Postava, T. Azim, M. J. Freedman, and P. Levis. Scaling virtual worlds with a physical metaphor. *IEEE Pervasive Computing*, 8(3):50–54, July 2009.

[12] J. H. Howard, M. L. Kazar, S. G. Menees, A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6:51–81, 1988.

[13] W. Lloyd, M. J. Freedman, M. Kaminsky, , and D. G. Andersen. Dont Settle for Eventual: Stronger Consistency for Wide-Area Storage with COPS. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, Cascais, Portugal, October 2011.

[14] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[15] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI'02)*, pages 31–44, 2002.

[16] NetFlix outage. http://blog.netflix.com/2008_08_01_archive.html.

[17] NIST Definition of Cloud Computing v15. http://csrc.nist.gov/groups/SNS/cloud-computing/index.html.

[18] OpenFlow Switch Consortium. http://www.openflowswitch.org.

[19] K. Park and V. S. Pai. Scale and Performance in the CoBlitz Large-File Distribution Service. In *Proc. 3rd Symposium on Network Design and Implementation (NSDI)*, San Jose, California, May 2006.

[20] L. Peterson and V. S. Pai. Experience-Driven Experimental Systems Research. *Communications of the ACM*, 50(11):38–44, Nov. 2007.

[21] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud control with distributed rate limiting. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 337–348, New York, NY, USA, 2007. ACM.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[24] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, New York, NY, USA, 2001. ACM.

[25] S3 outage. http://status.aws.amazon.com/s3-20080720.html.

[26] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Workshop on Hot Topics in Cloud Computing (HotCloud'09)*, June 2009.

[27] A. Schran, J. Rexford, and M. J. Freedman. Namecast: A reliable, flexible, scalable DNS hosting system. Technical Report TR-850-09, Computer Science Department, Princeton University, Apr. 2009.

[28] S. Sen, W. Lloyd, and M. J. Freedman. Prophecy: Using history for high-throughput fault tolerance. In Submission, 2009.

[29] A. Singh, P. Fonseca, P. Kuznetsov, R. Rodrigues, and P. Maniatis. Zeno: Eventually Consistent Byzantine-Fault Tolerance. In *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI 2009)*, Apr. 2009.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[31] Syndicate wiki. https://trac.princeton.edu/Syndicate/.

[32] J. Terrace and M. J. Freedman. Object storage on CRAQ: High-throughput chain replication for read-mostly workloads. In *Proc. USENIX Annual Technical Conference*, June 2009.

[33] J. Terrace, H. Laidlaw, H. E. Liu, S. Stern, and M. J. Freedman. Bringing P2P to the Web: Security and privacy in the Firecoral network. In *Proc. International Workshop on Peer-to-Peer Systems*, Apr. 2009.

[34] VICCI web site. http://www.vicci.org.

[35] L. Wang, K. Park, R. Pang, V. S. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proceedings of the USENIX 2004 Annual Technical Conference*, Boston, MA, June 2004.

[36] Q. Yin, J. Cappos, A. Baumann, and T. Roscoe. Dependable self-hosting distributed systems using constraints. In *Proceedings of the 4th Usenix Workshop on Hot Topics in System Dependability (HotDep)*, Dec. 2008.