C-MART: Benchmarking the Cloud

Andrew Turner, Andrew Fox, John Payne, and Hyong S. Kim, *Member*, *IEEE*

Abstract—Cloud computing environments provide on-demand resource provisioning, allowing applications to elastically scale. However, application benchmarks currently being used to test cloud management systems are not designed for this purpose. This results in resource underprovisioning and quality-of-service (QoS) violations when systems tested using these benchmarks are deployed in production environments. We present C-MART, a benchmark designed to emulate a modern web application running in a cloud computing environment. It is designed using the cloud computing paradigm of elastic scalability at every application tier and utilizes modern web-based technologies such as HTML5, AJAX, jQuery, and SQLite. C-MART consists of a web application, client emulator, deployment server, and scaling API. The deployment server automatically deploys and configures the test environment in orders of magnitude less time than current benchmarks. The scaling API allows users to define and provision their own customized datacenter. The client emulator generates the web workload for the application by emulating complex and varied client behaviors, including decisions based on page content and prior history. We show that C-MART can detect problems in management systems that previous benchmarks fail to identify, such as an increase from 4.4 to 50 percent error in predicting server CPU utilization and resource underprovisioning in 22 percent of QoS measurements.

Index Terms—Client/server and multitier systems, distributed/Internet-based software, performance measures, testing tools

1 Introduction

Benchmark applications are essential for performance testing and validating systems before deployment to production environments. They emulate a typical production system and provide measurements of a secondary system under test (SUT) such as the processing power of a host, the maximum number of serviceable clients, or the efficiency of a resource allocation algorithm [1]. They allow various configurations, settings, parameters, and so on, to be compared so that the best values for a given scenario can be identified. Benchmark web applications typically consist of an example website, such as an online store or a social networking website, and a workload generator that sends traffic to the website in a manner emulating web-browsing clients. However, existing benchmarks were not designed for cloud computing environments [1] and therefore produce misleading results when benchmarking cloud management systems [2]. This can result in poor performance, resource underprovisioning and quality-of-service (QoS) violations.

Benchmark applications are useful only if they accurately emulate the expected behavior of production environments. Existing benchmarks were designed for traditional dedicated hardware data centers, not cloud computing environments. Their workload generators are simplistic and do not accurately represent user behaviors. The websites are therefore not excited with the variability of traffic patterns that would be observed in production environments. This produces overly optimistic results when used for systems testing. The benchmarks therefore fail to accurately validate the performance of the SUT.

We present C-MART, a web application benchmark designed to evaluate systems running in cloud computing environments such as infrastructure as a service or virtualized data centers. We address four main concerns with existing benchmarks that make systems testing difficult, limited, and inaccurate. These concerns are:

- *Scalability:* Cloud computing environments allow on-demand resource provisioning [1]. As such, C-MART is designed to elastically scale at each tier of the application. It also includes a deployment server and scaling API to emulate a cloud computing environment in a local data center. Existing benchmarks are difficult or impossible to elastically scale.
- *Modern technologies:* Current web applications utilize technologies such as AJAX, CSS, SQLite, HTML5, and have multimedia-rich interfaces. Existing benchmarks do not utilize all of these features, reducing the variability in their resource utilizations, response times, and simplifying their management. We include these technologies so that C-MART emulates a modern application.
- *Flexibility:* C-MART is designed with multiple implementations of each tier. This allows a SUT to be validated under different architectures. C-MART can run as a two-tier system up to a six-tier system. Existing benchmarks have only a single design structure, thus limiting their testing scope.
- *Client realism:* Clients accessing real-world applications exhibit variable behaviors and access patterns. C-MART's client generator uses variable typing speeds, think times, browsing behaviors, QoS expectations, and page transition probabilities for each client. The distributions used are derived from user

• *The authors are with the Electrical and Computer Engineering Department, Carnegie Mellon University, 5000 Forbes Avenue, Hamerschlag Hall, D-level, Pittsburgh, PA 15213.*
E-mail: {andrewtu, afox1, johnpayn, kim}@ece.cmu.edu.

Manuscript received 1 Mar. 2012; revised 31 Aug. 2012; accepted 22 Nov. 2012; published online 12 Dec. 2012.
Recommended for acceptance by V.B. Misic, R. Buyya, D. Milojicic, and Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-03-0241.
Digital Object Identifier no. 10.1109/TPDS.2012.335.

TABLE 1
Overview of How C-MART Features Can Identify Problems in Management Systems Missed by Current Benchmarks

Feature	Current	C-MART	Use case	Impact
Client Caching	Low/ None	High/SQLite	Load balance by request URL	Current benchmark: Low variability in response time 109 ± 138ms C-MART: High variability in response time 1720 ± 6100ms
Page access frequency	Static	Variable	Linear regression scheme to predict CPU Utilization based on request rate	Current benchmark: Regression scheme has 4.4% error C-MART: Regression scheme has 50% error Resources are underprovisioned since linear regression on request rate is insufficient for determining CPU utilization
Session based QoS	No	Yes	Determine profit based on clients completing browsing sessions	Current benchmark: Clients do not use QoS in decision making process and management is based on aggregation across clients C-MART: Individual QoS, causes client levels to change in open loop Can be used to relate resource provisioning to QoS
Page Content Variability	Low	High	Consolidating VMs based on average CPU utilization	Current benchmark: Violates SLA 0% of time C-MART: Violates SLA 22% of time Consolidation scheme would perform poorly in production system

studies and real-world websites to ensure realistic client emulation. Previous benchmarks typically use only a single think time distribution for each page and a static Markov chain for determining page transition probabilities.

Table 1 provides a summary of how C-MART's features allow it to identify problems in SUTs that are overlooked when using current benchmarks. We present examples where applications will be improperly allocated resources, causing either QoS violations or low resource utilization. Further details are discussed in Section 4.

The remainder of the paper is organized as follows: Section 2 gives an overview of the related works and existing benchmarks. Section 3 outlines the architecture of C-MART. Section 4 provides experimental results comparing C-MART to existing benchmarks. Lastly, conclusions are given in Section 5.

2 Related Works

The purpose of a cloud benchmark is to help developers determine the right architecture, services, and settings for their applications by providing a testing platform that delivers relevant and comparable measurements [2]. This section summarizes existing benchmarks and highlights some of the deficiencies that make them unsuitable for use as benchmarks for cloud management systems.

2.1 RUBiS [3]

RUBiS is an auction website modeled after eBay [4] that is used to evaluate the performance of various application design patterns. However, there are a number of flaws in RUBiS that make it unsuitable for use as a modern application benchmark [8]. For example, TCP connections are shared between multiple clients, which would not occur amongst real clients. Additionally, the think times for all page requests are determined by a single exponential distribution with a mean of 7 seconds regardless of page content.

Cecchet et al. [9] analyzed the number of CSS, JavaScript, and multimedia objects contained on the homepage of RUBiS compared to the real application it emulates. RUBiS does not contain any CSS or JavaScript objects, and the number of multimedia objects is negligible compared to the real counterpart application. These objects significantly affect the response times of the website, therefore greatly reducing RUBiS's effectiveness.

RUBiS uses a closed-loop client generator, in which the number of concurrent emulated clients is constant in each experiment. This static workload pattern would not likely exist as client arrivals are not dependent on departures. The static behavior can produce overly optimistic results as knowing client data for one time interval unrealistically gives detailed knowledge for subsequent intervals. Additionally, the RUBiS data tier is configured as a nonscalable SQL database which is not representative of the distributed storage technologies used in cloud computing environments. Pugh et al. [5] conclude that "the amount of effort required to get RUBiS up and running outweighs the benchmark's usefulness at this point."

2.2 TPC-W [6]

TPC-W is a transactional web benchmark that emulates an online bookstore. Its specification states, "TPC-W [represents] any industry that must market and sell a product or service over the internet [...] TPC-W does not attempt to be a model of how to build an actual application." This design mantra results in TPC-W being a comparative benchmark between different sets of hardware. It is not designed to test application-level QoS in a cloud computing environment. It is noted in [8] that TPC-W's workload generator shares a similar implementation to RUBiS's workload generator and suffers from similar issues.

Binning et al. [2] concluded that the TPC benchmarks are not sufficient for analyzing novel cloud services. TPC-W does not represent modern web applications, as it lacks a significant multimedia presence, and client generated and AJAX content. TPC-W compensates for this by disallowing caching, thereby increasing network traffic [6]. This ignores the crucial caching factors that affect real applications. TPC-W uses a SQL database and requires that ACID properties are enforced. Cloud computing systems do not always offer such strong transactional guarantees. TPC-W's performance metric, web interactions processed per second (WIPS), is not of primary importance for cloud applications

that are more concerned with scalability characteristics and QoS guarantees.

2.3 Cloudstone Olio [7]

Cloudstone's Olio is an open-source social-event calendar web application with web 2.0 features. It utilizes some modern technologies such as AJAX and Memcache [8]. However, the application only has a limited number of functions, uses a single think time distribution, a static probability matrix for page transitions, and does not capture or emulate SQLite. The database tier uses MySQL or PostgreSQL, and is scalable through read-only replicas, rather than the NoSQL approach that is increasingly popular in cloud environments [9]. A second workload generator, Rain [10], has been used with Olio. Rain allows the user to specify different workload mixes in different time intervals; however, there is no deviation from these as the generator does not consider the content from the page responses in determining future page transitions.

2.4 YCSB 2010 [11]

YCSB is not an application emulation benchmark, but a framework for benchmarking different databases for cloud data storage. YCSB can be configured with various distributions of database operations (e.g., reads, inserts, deletes, and so on.). It then benchmarks a database for throughput and response time performance. However, as a database-only benchmark, it does not account for the interactions between the various tiers of applications. In production applications, it is difficult to predict the number of database transactions required per incoming client request. Additionally, this request number changes depending on clients' access patterns and the caching techniques used. Extrapolating application-level performance from only database performance is difficult, even if all other application tiers have overprovisioned resources.

2.5 SPECweb2009 [12]

SPECweb2009 is designed to measure webserver performance, specifically how it relates to power efficiency. It reports transactions as a function of power usage as its primary metric. It simulates a lightweight and efficient back end instead of using a traditional database, and uses a closed-loop workload generator. They use a static Markov chain for page transition probabilities and all think times are based on a single exponential distribution.

2.6 CloudSim [13]

CloudSim is an extensible simulation toolkit that enables modeling and simulation of cloud computing environments. However, its application models are simplistic and do not account for modern technologies. The simulated workloads do not produce the variability that would be observed in real-world applications. There is also no SQL application model. As databases are an important part of any real application, this is a significant component missing in CloudSim which limits its testing abilities.

3 C-MART

C-MART is designed as an online auction and shopping website. It is a multitier application written using the model-view-controller design principle. We use Java servlets for the application logic due to Java's prevalence in production applications. C-MART includes a custom built workload generator that emulates clients accessing the website. A deployment and scaling API is provided to simplify the management of C-MART.

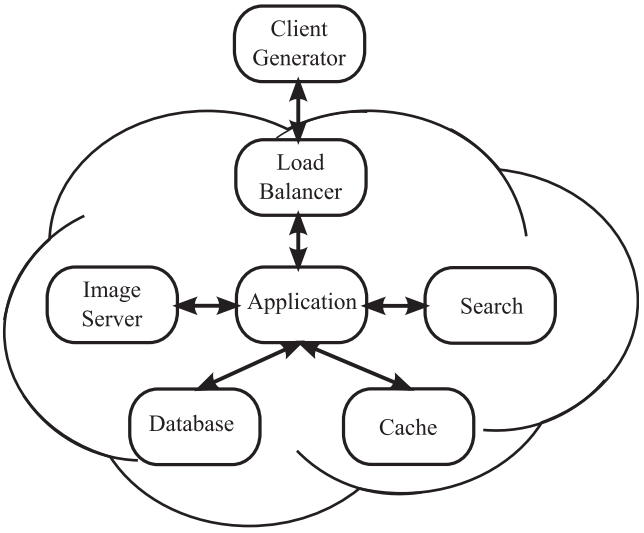

Fig. 1. C-MART architecture. The client sends requests to the (up-to) six-tier application running in the cloud.

C-MART was created with four main design principles to ensure its suitability for use as a cloud computing benchmark application: *scalability*, *modern technologies*, *flexibility*, and *client realism*. These principles ensure that the designs and technologies used mimic the behavior of real-world applications.

3.1 Scalability

A major benefit of cloud computing is the ability to provision resources on demand. This allows applications to quickly scale up and scale down as workloads vary. To emulate a cloud computing application, we ensure that C-MART can horizontally-scale at every application tier.

3.1.1 Tier Scalability

Fig. 1 shows the architecture of C-MART when utilizing all of its possible tiers (flexibility is discussed in Section 3.3). Each tier of the application is scalable due to the design of the application and the underlying software, as shown in Table 2. The client can send requests to different front-end load balancers using a system similar to round-robin DNS load balancing. The application tier is scalable as it is stateless, does not use user sessions, and does not acquire locks to shared resources on other tiers; all user state is stored at the data tier. This also allows the application tier to be quickly scaled down if the workload is reduced, as the tier retains no client data. The search, image, and cache tiers scale due to the underlying software, Solr [14], MongoDB [15], and Memcache, respectively. Lastly, the database tier scales as C-MART has a NoSQL implementation using Cassandra [16], again utilizing the scaling ability of the underlying software. Connection caching and limits can be configured via a web API.

TABLE 2
Scalability Methods at Each Tier

Tier	Description
Client	Run on multiple hosts, consolidates statistics
Load Balancer	Clients can be directed to multiple load balancers, similar to DNS balancing system
App	Stateless, does not lock shared resources
Cache	Memcache Distributed Hash Table
Search	Solr Multiple read-only copies
Image	MongoDB replicas
Database	Cassandra replicas

```
<Host hostOS="Fedora" port="4444"
IP="10.66.1.3">
   <VirtualMachine IP="10.1.4.3">
     <OSType>Fedora</OSType>
     <VMType Type="Application" Backup="Hot"/>
     <UserName>root</UserName>
     <Password>cmart</Password>
   </VirtualMachine>
   <VirtualMachine IP="10.1.1.7">
     <OSType>Fedora</OSType>
     <VMType Type="SQL" RAMSIZE="512"/>
   </VirtualMachine>
</Host>
```

Fig. 2. Example C-MART deployment file.

3.1.2 Dynamic Scalability and Deployment

To reduce the complexity of scaling C-MART, we have created a deployment and scaling API. Using preconfigured virtual machines (VMs) provided on the C-MART website [17], the number of servers to be deployed at each tier can be defined using an XML description, as shown in Fig. 2. The deployment descriptor can define hot and cold backup VMs, CPU and RAM allocations, and additional application specific information. During experiments, the scaling API can be used to add additional tiers to the application programmatically. This allows benchmark users to create custom data center management algorithms and dynamically provision resources. The APIs are provided for a KVM platform; however, we also provide installation scripts for use with any Linux operating system.

The API simplifies the deployment and configuration of C-MART compared to other benchmarks. In C-MART, only the XML definition files need to be configured. C-MART tiers are then automatically deployed to each physical host and configuration data, such as IP addresses in the load balancer or my.cnf files in the MySQL servers are automatically generated. This reduces the deployment time from hours with previous benchmarks, where IP addresses, network interfaces, and application-specific data need to be set manually, to only minutes required to define an XML document. Also, defining additional experiments is possible via the API, instead of editing the source code as required for example with RUBiS.

3.2 Modern Technologies

To ensure that C-MART's results accurately represent those of a modern application, it is important to design it to accurately emulate a real-world website using contemporary design methods and technologies.

3.2.1 HTML5, AJAX, CSS, Multimedia, and SQLite

Modern web applications utilize technologies such as HTML5, AJAX, CSS, rich multimedia, and SQLite. These technologies can have a significant impact on resource utilization levels and request access patterns, which ultimately affect the clients' browsing experience. For example, SQLite is used to locally cache dynamic data. This allows the remote application to only return data that have been updated because it was last locally cached. Without SQLite, the entire page must be rendered by the server for each request, consuming significantly more resources.

Multimedia presence on websites represents a large component of pages. A significant number of connections, round trip delays, network traffic, and disk I/O need to be consumed when the multimedia content is high, leading to a higher potential for QoS degradations.

HTML5 and AJAX automate or simplify processes that were previously created by a user-initiated request for an entire page to the server. HTML5 does advanced validation on forms reducing the number of pages regenerated due to input errors. AJAX requests can update specific page elements by requesting only certain information from the server; resulting in bursty database accesses depending on how many elements need to be updated and the time period since the client previously accessed the data. This prevents the entire page from being recreated for only partial page updates. AJAX requests may also be sent periodically, not initiated by a user page click. C-MART takes advantage of these design technologies. C-MART pages are formatted using CSS, JavaScript, and jQuery UI. On the item page, prices are updated using periodic AJAX requests and item pictures scroll in a timed gallery. AJAX requests return JSON or XML objects from which JavaScript inserts the returned data into the existing page. Each C-MART page also provides statistics on how long different elements of the webpage took to generate. By comparison, the RUBiS item page is entirely HTML4 and always contains the same two images.

3.2.2 Real-World Distributions

To allow clients to make content-based decisions that are representative of real-world clients, the data used to populate C-MART need to emulate a production website. We sampled 100,000 eBay auctions to create empirical distributions of various website content including:

- number and frequency of words in products' titles and descriptions,
- number of items in each product category,
- number of images on each product page,
- product bid values and buy now prices,
- seller ratings.

TABLE 3
A Sample of C-MART's Configuration Flags for Different C-MART Implementations

Flag	Effect
Solr	On/Off to use Solr as the sites search engine
Cache	0, 1, 2 use Memcache to cache either none, database heavy query results or all results
Database	MyISAM, InnoDB for MySQL storage, Cassandra for NoSQL
Image	'img' for local storage, 'netimg' for NFS, 'mongoDB' for GridFS/MongoDB
Web	Web 1.0 or Web 2.0 enable HTML5, SQLite, AJAX, JavaScript

These distributions cause data hotspots to naturally form within the website as emulated clients use the page content and item data to differentiate between items and rate their appeal. The RUBiS item title and descriptions are instead generic and repeated across items making them essentially identical.

3.3 Flexibility

Current benchmarks are designed with a single architecture and typically allow only minor configuration details to be altered. C-MART is instead designed with a highly customizable architecture, determined by simply setting flags in its configuration file. This allows a relative comparison of how different application architectures and technologies will perform with the SUT.

3.3.1 Tier Configuration

Not all application tiers are required when running C-MART. It can function as a two-tier application and database server configuration, up to the six-tier architecture shown in Fig. 1. In addition, some tiers have multiple options for the underlying technology as C-MART has been implemented using multiple architectures. For example, the database tier has both a SQL and NoSQL implementation which can be chosen using a single flag. This allows C-MART to emulate both a traditional application using a relational data store and a modern application using a cloud storage engine. Any additional storage engine could be used by implementing a provided abstract class. Populators are provided to prepopulate the databases with a configurable amount of data before each experimental run. Table 3 provides an overview of the architecture options that can be configured for the different implementations of C-MART.

3.3.2 Web Design Technologies

To analyze the effects of different web design technologies, C-MART includes two web implementations with identical functionality but using different design architectures. The first implementation is a traditional client-server model website where each page request is entirely processed and rendered at the server. The second implementation renders the pages locally and more heavily uses JavaScript, CSS, and AJAX, and increasingly relies on SQLite, as described in Section 3.2.1.

3.3.3 Client Flexibility

The workload generator has a number of different operating modes to test different client behaviors and access patterns. The workload generator can be run in an open-loop or closed-loop mode with either a static or time varied number of clients. Users can also enable an option to create random bursts of clients, known as flash crowds or "the Slashdot effect" [18]. Clients can operate as individuals with complex decision-making processes (see Section 3.4) use a simple Markov chain to determine page transitions. There are also preconfigured read-heavy and write-heavy biases that can be enabled in conjunction with any of the other options. Users can define client satisfaction levels that allow clients to behave differently if they receive poor response times from the website.

3.3.4 Experiment Repeatability

C-MART experiments are logged for repeatability so management algorithms can be directly compared. Clients' actions and think times from experimental runs are output to XML files which can be read by the workload generator and reproduced in subsequent runs. Bid values and exact items chosen may be modified on the repeated runs to avoid concurrency problems on the server.

3.4 Client Realism

Real clients exhibit unique behaviors and have different expectations of performance when interacting with a website. The variability of clients creates variability in the resource utilization of the application, and ultimately changes how applications need to be managed. We highlight factors that influence client behaviors and their effects on server utilizations and management systems.

3.4.1 Content and History-Based User Decisions

The content of a specific page and to an extent the entire website influences client behavior. For example, an item with a good review, thorough description, and multiple high-resolution photographs is a more attractive item than one with a short description sold by a user with negative reviews. A client is more likely to bid on the former over the latter. Also, a client's prior actions influence their current decision-making process. For example, a client is more likely to bid on an item they have been outbid on than another random item. It is therefore inappropriate to model page transition probabilities as a simple Markov model according to page type as current benchmark workload generators do. Clients in C-MART analyze page content and track their previous decisions when determining page transition probabilities. Knowing only a client's current page request type is therefore insufficient knowledge to use in predicting their next action. Some content-based decisions can create bursts in traffic and data hotspots. Clients are much more likely to bid on an item as auction time nears expiry, creating bidding wars which produce a large traffic burst to a single page.

As stated previously, empirical distributions are used from eBay to create the content for each item. The clients are

set to make decisions by slightly different criteria. For example, some clients may only buy from sellers with an extremely high rating while others consider the quality of an item's description to be of the greatest importance. Clients also have different pricing criteria. Two clients may find the same item equally enticing but only one may bid as the item may be out of the other's price range. As different webpages cause different amounts of server resource utilization, the variable client behavior increases the difficulty of managing the application.

3.4.2 QoS-Based User Decisions

Amazon loses 1 percent of revenue for every 100 ms of additional delay on response time [19]. When clients view more items and pages load faster, they are more likely to find and buy a desired product. Conversely, clients leave slow websites. Keeping clients active also generates ad revenue with increased page clicks. Clients' traffic patterns therefore change depending on the response time. The faster a website responds, the more pages users will attempt to access. For example, a user may compare a different number of related items, which changes the amount of information a client has access to. This ultimately weighs on the final bidding/purchasing decision.

Each C-MART client has their own expectations of the website's performance; some are more patient than others. Clients also have different performance expectations of different pages. Clients know that processing credit card information typically takes longer than bringing up an item page and are therefore more patient. These behaviors are particularly important when using the open-loop client generator. In the open-loop generator, client arrivals and departures are independent of each other. Therefore, QoS-related decisions are extremely important to maintain high client levels. Current benchmarks use closed-loop client generators where a client departing is instantly replaced by another arriving. This fails to account for QoS effects as poor website service does not cause a decrease in the number of active clients. A management scheme would therefore ignore that many clients leave due to dissatisfaction with the website experience.

3.4.3 Modern Browsers

Modern Internet browsers support tabbed browsing which can change clients' behaviors. Clients may open many pages at a time, for example from a search page to compare multiple items. This causes large bursts in traffic followed by a long idle period as the client examines each opened tab. It also means the search page does not need to be reloaded between items. Browsers also allow clients to automatically fill out online forms. Typing speed and typing error rate are two factors used to create variable clients. However, autocomplete form fillers allow clients to enter data extremely quickly. This creates multimodal think time distributions on form pages which make for highly variable access patterns.

Clients initially accessing a website do not necessarily arrive with an empty data slate. Clients may have previously accessed the site and would therefore already have a prepopulated cache when they arrive. This is extremely important for the initial access when most of the JavaScript, CSS, and common image files are initially downloaded. The number of clients who arrive with prepopulated caches is a tunable parameter in C-MART.

3.5 Performance Metrics

It is important to use proper metrics when evaluating the performance of a benchmark. Metrics such as WIPS which may have been useful for benchmarking hardware are inadequate for evaluating the performance of an application in the cloud.

We provide a number of different metrics to evaluate the benchmark performance. We provide distributions of the resulting response times for each page. For each client, we also provide the length of the browsing session, how many pages were visited, and whether or not the client left due to dissatisfaction with the QoS. We report the server utilization levels for each tier and the distribution of how long it took to build each component of the pages, for example, the times to process the parameters, access the database, process the page data, and render the page.

4 Experimental Results

To identify where current benchmarks may produce inaccurate results for existing management schemes, we run C-MART against current benchmarks for a number of common datacenter scenarios. By comparing the results, we identify where the SUT fails to correctly react in a production environment if they are tested using current benchmarks rather than C-MART. We begin by outlining a number of common management techniques and algorithms and how they are applied to cloud applications.

We use a custom datacenter with a flat local area network using commodity hardware. The physical hosts' OS is Red Hat 6.32 and runs the KVM hypervisor. Each server has a dual-core 3.2-GHz CPU with 4 GB of RAM. The hosts are connected via a Gigabit switch. Each VM has one virtual CPU and is allocated 1 GB of memory.

4.1 Management Systems

Cloud management systems involve procedures for VM placement, VM migration, resource provisioning, cost optimization, energy optimization, and achieving SLAs and QoS targets. We detail the techniques and algorithms used by these management systems and how their performance is changed when tested with C-MART instead of the existing benchmarks.

Controllers have been developed to model application response time as functions of resource utilizations (CPU, memory, I/O, network). These functions are used to adjust applications' resource allocation levels to drive response times to target values [20], [21]. These models typically assume that response time is a deterministic function of resource utilizations, which we show is not a valid assumption for dynamic webpages. Additionally, response time is often modeled using the total resource utilization during a time interval. This method is only suitable if the ratios of requests for each page type are static over time. However, real websites have highly varied and nonstationary page access patterns which makes these deterministic models impractical to use [22].

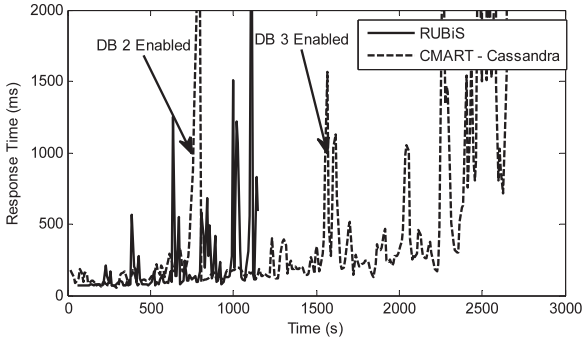

Fig. 3. C-MART scalability. Additional database instances are activated when response time degrades.

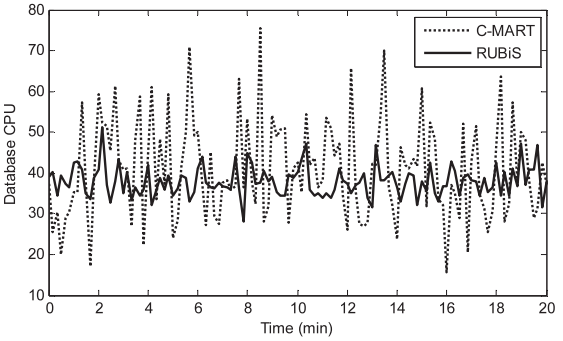

Fig. 4. C-MART and RUBiS database CPU for a static client level at the same CPU average.

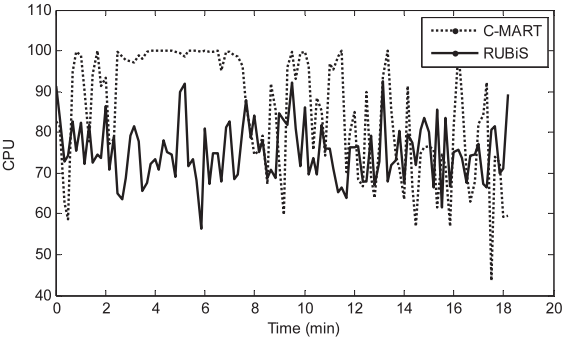

Fig. 5. CPU of two consolidated instances of C-MART and RUBiS.

Some management systems model QoS metrics as a function of request type. Works such as [23] model the response time of each webpage as a linear function of the page request rate. This is a reasonable assumption when each request consumes the same amount of resources as is common in existing benchmarks. However, due to effects from multimedia, caching, and SQLite, the amount of resources consumed for two identical page requests at different times or from different clients can be drastically different. Therefore, calculating response time as a deterministic function of request rate does not produce accurate results when used with a real application.

Autocorrelation of response time and resource utilizations [24] is used in management systems to predict future performance. This is effective when clients have predictable access patterns, as is the case with benchmarks that use static page transition probability distributions. However, it is much less effective when clients' behaviors are less predictable. Larger variance in the resource consumption of each page also makes the use of autocorrelation techniques more difficult. It is therefore important to test such management systems with a realistic benchmark to fully validate their potential performances.

4.2 Application Scaling

A major benefit of cloud computing environments is the ability to elastically scale applications to react to changes in workload levels. A benchmark application must be able to scale to utilize a large number of servers. C-MART is designed to horizontally scale at every tier. This prevents any one tier from becoming a performance bottleneck. A major component of this is the ability to run either SQL or NoSQL database storage. Current web benchmarks utilize SQL databases that can be difficult to scale. NoSQL storage allows users to dynamically add resources to their storage tier, redistributing storage keys and data replicas to evenly redistribute load. NoSQL is commonly found in cloud computing environments such as Google's Big Table or Amazon's Dynamo.

Fig. 3 shows the 95th-percentile response time of both C-MART and RUBiS as their workloads are increased. RUBiS is configured with one load balancer, six App VMs, and one database VM. C-MART is configured with one load balancer, one Solr VM, six App VMs, and three Cassandra VMs. RUBiS only receives one VM on its data tier as this is its default configuration. We modified the RUBiS client generator to linearly increase the number of emulated clients over time. We run C-MART initially with only one Cassandra VM. As application response time increases, we enable additional Cassandra VMs. As expected, having the flexibility to add additional processing and storage power to the data tier allows C-MART to scale to three times as many database instances. In addition, C-MART sends on average 8 kB of data per request compared to only 0.8 kB for RUBiS.

4.3 VM Consolidation

Cloud computing environments rely on VM consolidation to achieve high resource utilization. This consolidation reduces the amount of hardware required to run a set of applications and reduces energy consumption. To satisfactorily consolidate VMs, one must ensure that all VMs receive a sufficient amount of a host's resources. Otherwise, applications' performances will degrade resulting in QoS violations and unsatisfied clients.

Fig. 4 shows the CPU utilization for both C-MART and RUBiS for a static number of clients over a 20 minute time interval. The average CPU is equal for both benchmarks at 40 percent; however, the standard deviation is 340 percent greater with C-MART. This is due to the complex client request patterns and use of technologies such as AJAX and SQLite. This experiment was repeated with Olio where the C-MART CPU standard deviation was similarly 286 percent greater. This increased variability makes VM consolidation difficult. Fig. 5 shows the result of what happens when two C-MART or RUBiS VMs are colocated on a server. All VMs were run in the same mode as for Fig. 4. It can be seen that the RUBiS VMs consolidate satisfactorily as there is always CPU unutilized. However, because of the increased

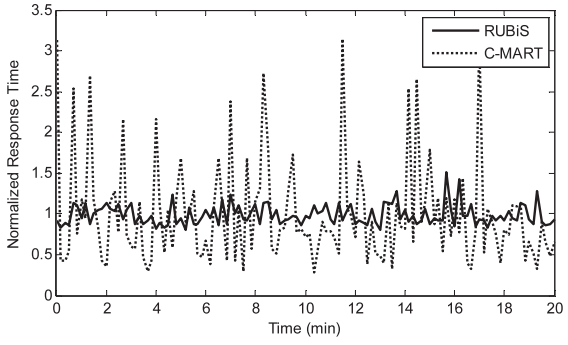

Fig. 6. C-MART and RUBiS response times for a static client level. For comparison purposes, each response time was normalized to the average response time from the respective experiments.

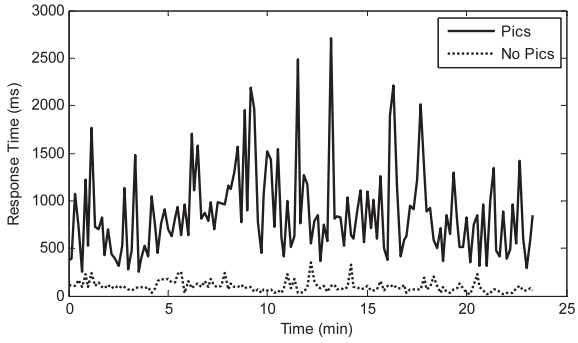

Fig. 7. C-MART response times when pictures, CSS, JavaScript are and are not downloaded.

variance, the CPU is exhausted in 22 percent of the measurement intervals when consolidating the C-MART instances. This results in poor response times for clients. This is also evident from the 90th percentile response times shown in Fig. 6 for the single VM experiment, where the standard deviation of response time is five times greater for C-MART than for RUBiS. These results illustrate the resource underprovisioning that may occur if the SUT is not properly validated. Using a current benchmark, average CPU utilization appears to be a sufficient metric for VM consolidation; however, C-MART identifies that this is not true.

Fig. 7 shows the significant impact that multimedia content can have on webpage response time. Here, C-MART is run with its images, CSS, and JavaScript first enabled for download and then disabled. It can be seen that the application's response time is not only greater due to the additional multimedia content, but the variance of the response time is also significantly increased. Testing systems with benchmarks that do not contain significant multimedia content will produce overly optimistic results. This results in poor estimates of application's resource requirements in production environments causing resource underprovisioning and QoS violations.

4.4 Performance Prediction

Performance prediction schemes typically rely on applications' historic and current workload levels to estimate the resources required to achieve a given QoS level and mitigate violations. For example, Huang et al. [23] estimate resource utilization caused by each incoming request using

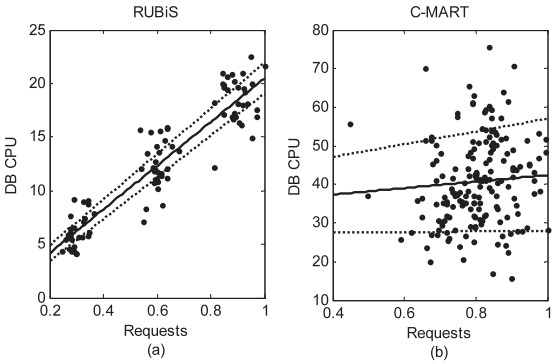

Fig. 8. CPU prediction based on workload for (a) RUBiS and (b) C-MART. Regression line shown with one standard deviation.

regression analysis, then determines current resource needs based on current request levels. However, such schemes are effective only if the ratio of each type of incoming request can be accurately predicted, as they can be in current benchmarks which use Markov chains to transition between pages. However, as noted in [22], this behavior is not observed in production applications.

Fig. 8 shows the regression analysis for incoming request rates and database CPU utilization. The request rates were normalized for comparison purposes. The graphs show the regression line bounded by fit lines of one standard deviation. When using RUBiS, the database's CPU utilization is highly correlated to the number of incoming requests. However, due to greater variation in page behaviors, the correlation between CPU utilization and the number of requests in C-MART is significantly reduced. The equations in terms of the workload, λ, are

$$CPU_{\mathrm{RUBiS}} = (20.4 \pm 0.9)\lambda + (0.1 \pm 0.6)$$
$$CPU_{\mathrm{C-MART}} = (18 \pm 9)\lambda + (26 \pm 8).$$

The RUBiS equation has little error on its slope and passes through the origin within error. The C-MART slope has a standard deviation half of its actual value, a 1,040 percent increase in error, and the equation greatly misses the origin. This is not nearly well enough defined to be used for analysis. Applying the estimation scheme used successfully in RUBiS to C-MART would result in significant resource underprovisioning and QoS violations.

To further illustrate the static nature of request patterns created by Markov chains, Fig. 9 shows the ratio of requests for the most popular and second most popular page for C-MART, RUBiS, and Olio during 10 second periods. The ratio of page requests in RUBiS and Olio is clustered around a single point with a low variance. However, the ratio of page requests in C-MART varies from roughly 10:1 to 2:3 of most popular to second most popular. This closely follows the observed behavior of a production website in [22].

4.5 Caching and SQLite

Caching allows for files such as images and CSS to be accessed locally after the first download. Similarly, SQLite can store dynamic data used to populate the webpages. Only small updates in the form of JSON or XML objects need to be sent from the application server to the client on subsequent requests as the client already has the previously

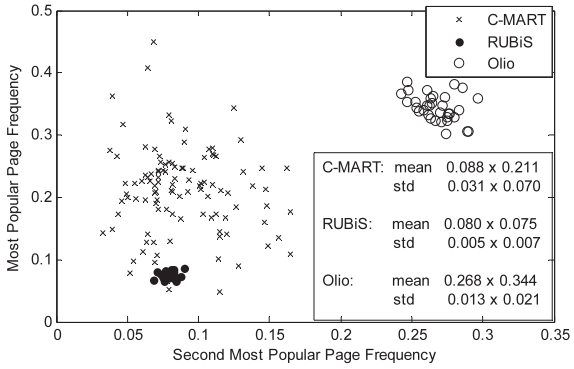

Fig. 9. Frequency of popular page accesses in different time intervals.

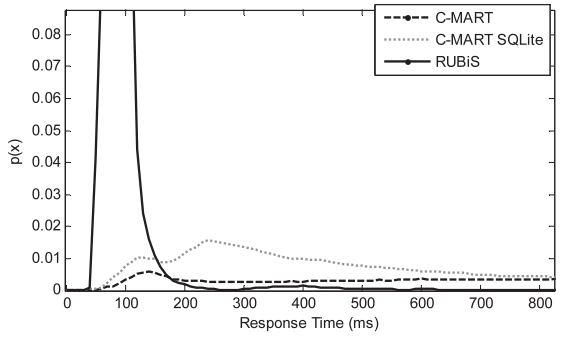

Fig. 10. Item page response time distributions for C-MART with and without SQLite, and RUBiS.

accessed data and performs the page rendering itself. Database access and response time becomes more bursty and varied as the amount of data required to be read is not constant for a given page.

To demonstrate this, we examine the download sequence of the browse page for RUBiS and C-MART, with and without SQLite, when loading the pages in Google Chrome. When first loading the C-MART Browse page with nothing cached the response time is 3.76 s, as JavaScript, CSS, images, and a prepopulation file for the SQLite database are downloaded. The second load of the browse page for the same client takes only 309 ms as images can be obtained from the cache and only data which has been modified needs to be sent from the application. For example, a JSON response containing the items listed on the page reduces from 34 kB to 598 B, and a 1.69 MB response that is used to populate the SQLite database on the first access is not required on the second. The first page load contains 51 different requests while the second contains only 47. By comparison, the RUBiS browse page contains the same three requests for every access of the same page. Olio contains more requests than RUBiS (32 on the homepage) including images and scripts; however, the request sequence is identical for each load of the page.

The use of caches makes profiling based on request type difficult. The increased variability and complexity of page response time distributions result in too much error when profiling using only the distributions' average or median. The distributions of response times for the item pages are shown in Fig. 10. RUBiS has an extremely large peak (that goes off the scale but is not shown for readability purposes) as all item pages are essentially identical. The non-SQLite version of C-MART has one peak but is more spread out due

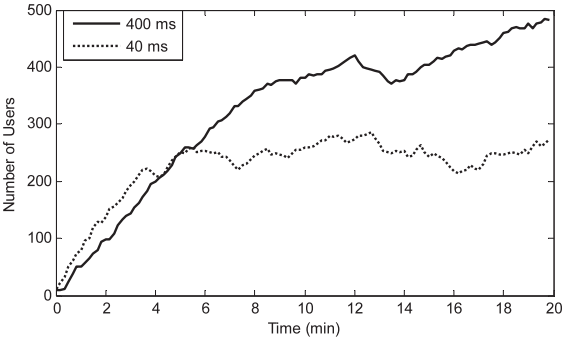

Fig. 11. User load for different response time expectations with an open-loop client generator.

to the larger differences between pages. The C-MART SQLite distribution is interesting as its peak at 130 ms is larger than the non-SQLite distribution. This is a result of the item data already being stored in the local client database. If there are no updates, all that needs to be obtained from the server are the new images. The second peak in the C-MART SQLite version is for when the item is not already in the client database and extra processing is required.

These wider and more complex distributions make it more difficult to determine response time as a function of request type. For the given RUBiS distribution, the response time is 109 ± 138 ms. The average and standard deviation of the response times for the C-MART non-SQLite and SQLite versions are $4{,}000 \pm 7{,}800$ ms and $1{,}720 \pm 6{,}100$ ms, respectively. These variations are far too great to be used for a linear regression analysis to relate the response time to the page type.

4.6 QoS Measurement

It is common for transaction-based applications to measure QoS on an aggregate level over all clients using the average or percentiles of response times. Clients in C-MART however make decisions based only on their individual QoS, not that received by other clients.

When determining how QoS affects clients, it is important to run the workload generator in an open-loop mode. A closed-loop generator automatically replaces an angry client leaving with a new client, holding the total user level static. This does not accurately represent the loss in profit that would occur from clients leaving the site prematurely. In real applications, the client arrival and departure rate are not directly dependent on one another. Clients leaving due to poor QoS would be reflected in a decline in workload. Using a closed-loop generator may give the impression that a resource provisioning scheme is performing well, when it would perform poorly in a production environment by causing many clients to leave the site prematurely.

We run C-MART twice in open-loop mode with the same arrival rate for both experiments. In C-MART, the probability that a client leaves the website increases as the response time increases above a threshold value. The thresholds are set to 400 and 40 ms in the two respective experiments. The resulting client levels over the duration of the experiments are shown in Fig. 11. The clients in the 40 ms experiment have much higher QoS expectations and

TABLE 4
Results of User Satisfaction with C-MART QoS

Situation	Percent of Clients that leave due to poor QoS	Average Client Session Length (s)
400 ms Response Time Threshold, Per-client QoS, Open Loop	19.5%	333
40 ms Response Time Threshold, Per-client QoS, Open Loop	48.4%	201
40 ms Response Time Threshold, Aggregate QoS, Open Loop	19.8%	239
40 ms Response Time Threshold, Per-client QoS, Closed Loop	57.7%	171

are therefore more likely to leave when service is slow. This is reflected in the lower client level.

In Table 4, we show clients' average session lengths and the percent of clients that leave unsatisfied due to poor QoS. It also shows how measuring QoS with aggregate response time can suggest much better results than when considering each client individually. When using aggregate response time, clients do not leave as often as the clients receiving bad service are balanced out by those receiving good service. Using this metric would overestimate the effectiveness of the SUT as real-world clients are not satisfied by other clients receiving good service; they want the good service themselves. Results for closed-loop clients are also included. Since the unsatisfied clients leaving are automatically replaced by new clients, the client load is not reduced which keeps the response time high. The closed-loop mode is not an accurate representation of the natural feedback which would occur in such a system.

5 Conclusion

Existing benchmark applications do not represent modern websites and are inappropriate for benchmarking cloud systems. We therefore present C-MART, a new benchmark application designed to emulate the behavior of modern cloud computing applications. C-MART can dynamically scale to support a large number of clients and has a flexible application design, allowing it to emulate multiple different application architectures. It uses modern web technologies such as HTML5, CSS, AJAX, and SQLite and includes a workload generator that emulates clients accessing the website. It creates unique clients that change their behavior according to page content, history, and QoS. These factors make the client behavior more realistic and increase the variability of the server utilization and response time. C-MART's deployment server allows automatic, simple, and fast datacenter configuration and resource provisioning.

The results show that existing benchmarks are overly optimistic in testing cloud management systems for production environments as they are unable to identify many deficiencies in the SUT. We show that existing workload prediction models could have 1,040 percent greater error than what was previously validated. We also demonstrate how existing management schemes could underprovision applications' resources in 22 percent of time intervals.

C-MART is available for download at [17]. We supply KVM images for every tier and install scripts for use on platforms such as Amazon EC2. We also provide operation instructions and detailed architecture descriptions.

References

[1] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," 2009.
[2] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How Is the Weather Tomorrow? Towards a Benchmark for the Cloud," *Proc. Second Int'l Workshop Testing Database Systems (DBTest '09)*, 2009.
[3] "RUBiS," http://rubis.ow2.org/, 2013.
[4] "eBay," http://www.ebay.com, 2013.
[5] B. Pugh and J. Spacco, "RUBiS Revisited," *Proc. 19th Ann. ACM SIGPLAN Conf. Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '04)*, pp. 204-205, 2004.
[6] "TPC Benchmark W (Web Commerce) Specification," *San Jose, CA, USA*, 2002.
[7] "Olio," http://incubator.apache.org/olio/, 2013.
[8] E. Cecchet, V. Udayabhanu, T. Wood, and P. Shenoy, "BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications," *Proc. Second USENIX Conf. Web Application Development (WebApps '11)*, 2011.
[9] D.J. Abadi, M. Carey, S. Chaudhuri, H. Garcia-Molina, J.M. Patel, and R. Ramakrishnan, "Cloud Databases: What's New?" *Proc. VLDB Endowment*, vol. 3, nos. 1/2, pp. 1657-1657, Sept. 2010.
[10] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D.A. Patterson, "Rain: A Workload Generation Toolkit for Cloud Computing Applications," technical report, EECS Dept., Univ. of California at Berkeley, 2010.
[11] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," *Proc. First ACM Symp. Cloud Computing (SoCC '10)*, pp. 143-154, 2010.
[12] "Standard Performance Evaluation Corporation: SPECweb2009," http://www.spec.org/, 2013.
[13] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50, Jan. 2011.
[14] "Apache Lucene - Apache SOLR," http://lucene.apache.org/solr/, 2013.
[15] "MongoDB," http://www.mongodb.org/, 2013.
[16] "The Apache Cassandra Project," http://cassandra.apache.org/, 2013.
[17] "C-MART," http://theone.ece.cmu.edu/cmart/, 2013.
[18] P. Bodik, A. Fox, M.J. Franklin, M.I. Jordan, and D.A. Patterson, "Characterizing, Modeling, and Generating Workload Spikes for Stateful Services," *Proc. First ACM Symp. Cloud Computing (SoCC '10)*, pp. 241-252, 2010.
[19] R. Kohavi and R. Longbotham, "Online Experiments: Lessons Learned," *Computer*, vol. 40, no. 9, pp. 103-105, Sept. 2007.
[20] P. Padala et al., "Automated Control of Multiple Virtualized Resources," *Proc. Fourth ACM European Conf. Computer Systems (EuroSys '09)*, 2009.
[21] X. Zhu, P. Padala, and Z. Wang, "Memory Overbooking and Dynamic Control of Xen Virtual Machines in Consolidated Environments," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management*, pp. 630-637, 2009.
[22] C. Stewart, T. Kelly, and A. Zhang, "Exploiting Nonstationarity for Performance Prediction," *ACM SIGOPS Operating Systems Rev.*, vol. 41, pp. 31-44, 2007.
[23] X. Huang, W. Wang, W. Zhang, J. Wei, and T. Huang, "An Adaptive Performance Modeling Approach to Performance Profiling of Multi-Service Web Applications," *Proc. IEEE 35th Ann. Computer Software and Applications Conf.*, 2011.
[24] A. Kochut and K. Beaty, "On Strategies for Dynamic Resource Management in Virtualized Server Environments," *Proc. 15th Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems*, pp. 193-200, 2007.

Andrew Turner received the bachelor's of computing and management (honors) from Loughborough University, United Kingdom, in 2005. He received the MSc degree in computer science from Oxford University, United Kingdom, in 2006. Since 2006, he has been working toward the PhD degree in the Department of Electrical and Computer Engineering, Carnegie Mellon University. His research interests include datacenter and network management and analysis.

Andrew Fox received the bachelor's in engineering physics (honors) from Queen's University, Kingston, ON, Canada, in 2010. He received the MS degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2012, where he is currently working toward the PhD degree. His research interests include cloud computing and datacenter management.

John Payne received the BS degree in mathematics and computer science and the MS degree in applied mathematics from Rensselaer Polytechnic Institute, Troy, NY, in 2007. He is currently working toward the PhD degree in the Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA. His primary research interests include network traffic management and datacenter management.

Hyong S. Kim received the BEng (honors) degree in electrical engineering from McGill University in 1984, and the MASc and PhD degrees in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 1987 and 1990, respectively. Since 1990, he has been with Carnegie Mellon University (CMU), Pittsburgh, PA, where he is currently the Drew D. Perkins Chaired Professor of electrical and computer engineering. His primary research areas are advanced switching architectures, fault-tolerant, reliable, secure network architectures, and cloud computing. His Tera ATM switch architecture, developed at CMU, has been licensed for commercialization. In 1995, he founded Scalable Networks, a Gigabit-Ethernet switching startup. Scalable Networks was acquired by FORE Systems in 1996. In 2000, he founded AcceLight Networks, an optical switching startup, and was CEO until 2002. He is an author of more than 100 published papers and holds more than 10 patents in networking technologies. He was an editor of *IEEE/ACM Transactions on Networking* from 1995 to 2000. He received the National Science Foundation Young Investigator Award in 1995. He is a member of the IEEE.