This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Estimating the Performance of Hypothetical Cloud Service Deployments: A Measurement-Based Approach

Y. Angela Wang
Polytechnic Institute of New York University
Brooklyn, NY 11201

Cheng Huang
Microsoft Research
Redmond, WA 98052

Jin Li
Microsoft Research
Redmond, WA 98052

Keith W. Ross
Polytechnic Institute of New York University
Brooklyn, NY 11201

Abstract

To optimize network performance, cloud service providers have a number of options available to them, including co-locating production servers in well-connected Internet eXchange (IX) points, deploying data centers in additional locations, or contracting with external Content Distribution Networks (CDNs). Some of these options can be very costly, and some may or may not improve performance significantly. Cloud service providers would clearly like to be able to estimate a *priori* performance gain of the various options before sinking significant capital expenditures into major infrastructure changes.

In this paper we take a measurement-oriented approach and develop methodologies that accurately predict the performance improvement for making major infrastructure changes. Our methodologies leverage active web content, existing large-scale CDN infrastructures, and the SpeedTest network. We then apply our methodologies and a *CloudBeacon* tool to the problem of locating satellite data centers throughout the world. The results show that for North America, a deployment limited to 11 locations will be sufficient. However, in order to provide good latency and throughput performance on a global scale, somewhere between a total of 36 and 72 cloud-service locations with good peering connections is most likely needed.

1. INTRODUCTION

Today cloud service providers (such as Amazon, Google and Microsoft) host a broad range of services, including e-commerce, search, portal services, web mail, shared calendars, social networking, remote computation, word processing, spreadsheet applications, and so on. Most of these cloud applications involve dynamic content. Mega data centers (DCs) hosting such services are often only in a hand full of remote locations and can involve non-negligible latency for access.

End-users, on the other hand, are extremely sensitive to the responsiveness of the services. Large-scale A/B experiments, for instance, have shown that 100 ms extra delay can cost 1% drop in sales [10].

Therefore, an important goal of cloud service providers is, to the greatest extent possible, hide the network from the user – that is, to give the user the illusion that between the user and the cloud the delay is zero and the bandwidth is infinite. To that end, cloud service providers are increasingly rely on *edge networks* to expand the reach of the mega data centers. Edge networks consist of many (tens to a few hundreds) small computing and storage clusters, which we call *satellite data centers*, distributed at locations with abundant network connectivity. The satellite DCs can cache and deliver static content, similar to traditional Content Distribution Networks (CDNs). More importantly, they can proxy requests and responses between clients and the mega data centers to accelerate dynamic cloud services. *Even though the services are ultimately offered from the mega data centers, simply by maintaining persistent connections over the long-haul links to the mega DCs, the satellite DCs are able to avoid multiple round trips on those links and reduce the 95-percentile web search response time by as much as 43% [16].* This technique is sometimes referred to as TCP splitting with persistent long-haul connections. Therefore, the satellite data centers provide TCP splitting as well as caching.

Edge networks can be private – Google has built a in-house network to accelerate its services [19]. Alternatively, edge networks can be shared infrastructures – Akamai's network is actively being used by Microsoft and many other providers to accelerate their services. We emphasize that these edge networks are used to accelerate the delivery of dynamic content (such as search and email) as well as static one.

The important questions to be answered are *1) how many satellite data centers are needed? 2) where should they be deployed?* and *3) what kind of peering connectivity should they have?* The options to cloud service providers include co-locating production servers in well-connected Internet eXchange (IX) points, deploying satellite data centers in additional locations (beyond their mega data centers), or contracting with external CDNs. Some of these options can be very costly, and some may or may not improve performance significantly. Cloud service providers would clearly like to be able to estimate *a priori* performance gain of the various options before sinking significant capital expenditures into major infrastructure changes.

The goal of this paper is to answer these questions. Analytical and queuing methods will not be sufficient, since they don't accurately capture the complex traffic patterns that arise in cloud services. Hence, in this study, we take a measurement-oriented approach. There are two key challenges. First, how do we design the experiments so that the measurements capture the perceived performance of real end-users? Second, how can we accurately measure the performance of a hypothetical deployment *before* physically deploying the satellite data centers?

To address the first challenge, we leverage the ubiquity of web browsers and develop a JavaScript-based measurement tool – *CloudBeacon*. CloudBeacon can be embedded in any web page served by the cloud providers or by partners of the cloud provider. It is launched and executed within the end-users' browser and thus takes measurements from behind the "last mile". With our CloudBeacon tool, we can measure delay and throughput from users in the wild, dispersed over the entire globe, to any set of target hosts. To address the second challenge, we need to identify candidate target hosts that are not only broadly geographically dispersed but also have ISP peering arrangements that are similar to those that satellite data centers would have. Our key idea here is to leverage widely deployed commercial infrastructures. To this end, we develop a novel latency measurement methodology, called *Reflection Ping*. We show that by combining several large scale infrastructures together, cloud providers can compare virtually all the potential locations for satellite data centers; evaluate different performance metrics (such as latency vs. throughput); and examine peering connectivity options at the candidate locations.

As a proof of concept, we deploy CloudBeacon on a number of popular websites. When users in the wild download pages from these websites, CloudBeacon makes measurements between the users and the servers at three existing large-scale commercial infrastructures, including two Content Distribution Networks [8] and the SpeedTest network [2]. Using the results of these measurements, we evaluate how cloud service providers should design their edge networks (that is, locate their satellite data centers).

We conclude that edge network performance is *not* only determined by geographic presence but also critically by the extent of ISP peering arrangements. When considering the same subset of common locations, we find that two edge networks yield greatly different latency performance for the users in Asia, due to different ISP peering connectivity arrangements. The performance, on the other hand, is almost identical in North America and Europe. In addition, we conduct evaluation of hypothetical deployments of edge networks, for latency sensitive and throughput sensitive applications. The results show that it is not difficult to achieve good performance in North America – a deployment limited to 11 locations will be sufficient. However, in order to provide good latency and throughput performance on a global scale, somewhere between a total of 36 and 72 cloud-service locations with good peering connectivity is most likely needed. As a result, excessive deployments in the order of several hundred locations is clearly unnecessary.

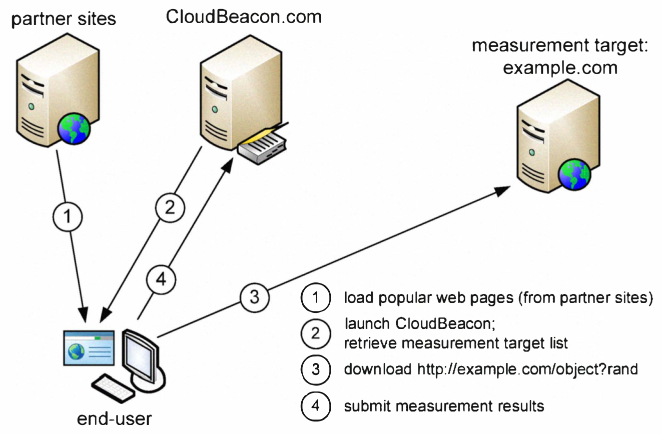

partner sites
CloudBeacon.com
measurement target:
example.com
end-user
1
2
3
4
1 load popular web pages (from partner sites)
2 launch CloudBeacon; retrieve measurement target list
3 download http://example.com/object?rand
4 submit measurement results

Figure 1: The CloudBeacon Measurement Architecture

2. MEASUREMENT ARCHITECTURE

Normally, measuring end-to-end performance requires measurement applications be installed and executed in thousands of representative clients geographically distributed around the world. However, today end-users tend to avoid downloading and installing executables whenever possible. For example, Joost [1] has publicly acknowledged that it was a strategic mistake to require users to install a client before they are able to watch any video. Client side installation is a *huge* deployment barrier, even for compelling services such as Joost. Thus, it is critical that the cloud-service measurement tool be distributed and executed in end systems without asking the end-user to download and install programs or even making minor changes to default system configurations.

Our measurement platform leverages the ubiquity of web browsers and JavaScript support. As shown in Figure 1, it consists of three parts: a CloudBeacon web gadget, partner web sites (such as the web servers of Polytechnic Institute of NYU and Microsoft Research), and the CloudBeacon workload server. The CloudBeacon web gadget is written in JavaScript. It is hosted by multiple partners on their (preferably popular) web sites. When a client (in the wild) retrieves a web page from a partner's website, the CloudBeacon gadget is loaded into the client at the end of the web page (so as not to affect user-perceived page load time). The CloudBeacon gadget then retrieves a measurement target list from the CloudBeacon server. After obtaining the target list, the CloudBeacon gadget performs Internet measurements to hosts in the target list and submits the results back to the CloudBeacon server. With this architecture, the target list can be dynamically modified by the CloudBeacon server based on the client's origin. For example, the CloudBeacon server can return, from a large set of potential measurement targets, the target that is the closest to the client. To minimize the visual impact on the end-users, the CloudBeacon is embedded in a partner's web site through an 1-pixel `iframe` (thus "invisible" to the end-users) as follows.

```
<iframe src='http://cloudbeacon.com/measure.htm'
        scrolling=no frameborder=0
        marginwidth=0 marginheight=0
        width=1 height=1>
</iframe>
```

The CloudBeacon gadget is implemented in the `measure-.htm` page. If we update the gadget, we simply modify this HTML page and the updated gadget will be automatically loaded by clients.

Next, we describe actual measurement conducted by the CloudBeacon gadget. Each measurement task instructs the client to fetch a particular object from a target web server (say `example.com`). A download time is calculated as the time elapsed from when the client initiates the request until the response arrives. To do this, an "innerHTML" can be embedded into a dynamic `div` tag as follows:

```
div.innerHTML =
 "<img src='http://example.com/object?rand'
    onload='endtime()'></img>"
```

When clients visit the partner's page, the measurement is triggered through an image event in HTML. (It does *not* matter even if the object is not an image.) The time difference between creating the dynamic tag and the "endtime" being invoked is the download time. The measurement is repeated several times and the minimum value is reported through a separate web request. Here, a random sequence is generated on-the-fly for each dynamic `div` tag to avoid browser side caching of the object.

2.1 System Deployment and Data Collection

The measurement system was deployed for more than 2 months. CloudBeacon is currently hosted on multiple popular partner websites, including the front page for Microsoft Research, the front page for Polytechnic Institute of NYU, all pages of the Packet Video workshop, the front pages of three small online gaming websites, as well as a number of personal home pages belonging to the authors and their colleagues. The data collection is summarized in Table 1.

3. MEASUREMENT METHODOLOGIES

Given a target deployment location, the fundamental performance metrics are latency and throughput. In this section, we present methodologies for accurate measurement of both metrics for clients in the wild to the target location.

3.1 Latency Methodology

The servers in Content Distribution Networks are ideal target nodes for our study. CDN production servers are deployed in hundreds of locations. Many of these locations are densely populated with servers. In addition, CDNs often establish extensive peering connectivity with a large number of ISPs. Peering connectivity establishes a direct network connection between two lower-tier ISPs. That way, traffic between the two ISPs does *not* have to route through higher-tier ISPs, which helps to improve performance and reduce bandwidth costs. It is a common practice for CDNs to peer aggressively and extensively, especially to so-called "eyeball ISPs", that is, with ISPs that service end users. Cloud service providers are expected to follow suit. In short, the existing locations of CDN servers are excellent candidates for cloud service providers' satellite data centers. Thus, we choose the servers of two large-scale CDNs as targets for our latency measurements.

3.1.1 The Wrong Way to Do Things

Intuitively, latency can be simply measured by downloading a small object from a CDN server. However, practical challenges render accurate latency measurement far more complicated. Here, we first present a seemingly correct, but misleading, methodology.[1] To measure the latency from a client to CDN Z, we instrument CloudBeacon to retrieve a small object from CDN Z. For example, if we know that `customer.com` uses CDN Z to deliver content, we can instruct CloudBeacon to request `http://customer.com/tiny.gif?rand`, for the file tiny.gif stored in CDN Z's servers. Note that it is necessary to append a random string at the end of each request to avoid browser caching.

Unfortunately, when we instrumented CloudBeacon to conduct such measurements against two popular large-scale CDNs [8] (anonymized and simply referred to as CDN A and CDN B), results were surprisingly counter-intuitive. CDN A has a much broader geographic presence than CDN B, so we expected CDN A's performance to be better. However, the results were com-

[1]We hope reporting such methodology can help other researchers avoid the same pitfall.

	unique IPs	cities	countries	ASes
CloudBeacon clients in the wild	18,418	2,064	134	1,100+

Table 1: Data Collection Summary

pletely the opposite. Eventually, we discovered that, in the above measurement for CDN A, the obtained latency was *not* between the client and the CDN, but rather between the client and the original server (`customer.com` here) *through a CDN A server*. The random string appended to each request not only prevented caching in the client browser, but also in the CDN server.[2] This experiment clearly shows that, if not careful, a seemingly-correct measurement methodology can lead to completely false conclusions!

The methodology is also severely limited because it can only measure those servers returned by a CDN's DNS-based redirection logic, rather than arbitrary CDN servers.

3.1.2 Reflection Ping

Now, we describe a novel new methodology – *Reflection Ping*, which allows us to measure the latency between a client and an *arbitrary* CDN server.

We construct an HTTP request using the server's IP address, instead of its hostname. For example, suppose `customer.com` maps to `192.168.0.1`, the HTTP request, instead of `http://customer.com/tiny.gif?rand`, is now constructed as

`http://192.168.0.1/tiny.gif?rand.`

Because the CDN server uses the hostname to associate a request with its customers (by examining the "Host" field in an HTTP request header), it cannot map `http://192.168.0.1/tiny.gif?rand` to any particular customer in this case. As a result, it denies such a request and sends back `HTTP/1.x 400 Bad Request`. In addition, the CDN server closes the connection after such a reply. Hence, each request completes in *exactly* 2 RTTs (one for TCP establishment and the other for the request/reply). The final RTT latency is calculated as half of measured elapsed time.

One subtlety worth pointing out is, to ensure accuracy, the client should avoid using a CDN server with which it currently has a persistent connection (for cases when the client has recently – say, within the a normal persistent connection timeout of 5 minutes – requested content from `customer.com`). We now describe a randomization technique to minimize the probability of this occurring. The basic observation is that for a given CDN provider, each of of its locations typically has many CDN servers. We will thus have the client contact different CDN servers when repeatedly visiting the same CDN location.

[2]CDN B, on the contrary, ignored the random string and delivered the object directly from its caching locations.

3.1.3 Charting CDN Servers

In order to use a CDN's servers in our measurement experiments, we need to know their IP addresses. We now describe our methodology for charting a CDN's servers. Our methodology explores the same "DNS magic" used by all modern CDNs to connect end-users to their servers. Let's illustrate this with an example. Assume `BestBuy.com` contracts with CDN Z to accelerate its website. An end-user visiting `http://www.-BestBuy.com` resolves the hostname to an IP address by querying its *local* DNS (LDNS) server. The LDNS then contacts the *authoritative* DNS server of `BestBuy.com`, which returns a CNAME `a1105.cdnZ.net` to the query. Here, the CNAME belongs to the domain of CDN Z and `a1105` represents the customer. The LDNS again resolves `a1105.cdnZ.net` and eventually obtains the IP address of a CDN Z server hosting BestBuy's content.

Depending on the query (representing the CDN customer) and the origin of the LDNS (representing the end-user), CDN Z chooses a (CDN Z) server that hosts the customer's content and is nearby the end-user (and returns the IP address thereof). In addition, due to load balancing, for the same query and LDNS, CDN Z returns different servers over time. Therefore, to chart the complete list of CDN Z's servers, conceptually, we can take three steps: 1) finding all the customers that are using CDN Z; 2) querying a large number of *open* LDNSes all over the world (as if the queries were from geographically dispersed end-users); 3) repeating the queries over time.

Applying the above technique, we have conducted a detailed study of CDN A and CDN B, and eventually compiled a list of more than $27,000$ and $4,100$ servers, respectively. Using a commercial IP to geolocation database, the servers of CDN A map to several hundred locations (i.e., cities) over six continents and those of CDN B map to more than two dozen locations over four continents, as summarized in Table 2 (please refer to [8] for more details).

3.1.4 Reflection Ping against CDN Servers

Once we have charted out the complete list of servers for both CDN A and CDN B, we group all the servers by their geographic locations. Again, CDN A's deployment covers several hundred locations, while CDN B covers more than two dozen. The CloudBeacon server maintains a list of up to 32 active servers for each loca-

(a) CDN A

Country	# of IP	Percentage(%)
United States	16,843	61.09
United Kingdom	1,690	6.13
Japan	1,622	5.88
Germany	1,103	4.00
Netherlands	857	3.11
France	722	2.62
Australia	514	1.86
Canada	438	1.59
Sweden	396	1.44
Hong Kong SAR	370	1.34
Others	3018	10.95
Total	27,573	100.00

(b) CDN B

City	Country	# of servers
All Cities	United States	2830
Frankfurt	Germany	314
London	United Kingdom	300
Amsterdam	Netherlands	199
Tokyo	Japan	126
Toronto	Canada	121
Paris	France	120
Hong Kong	Hong Kong SAR	83
Changi	Singapore	53
Sydney	Australis	1
Total	-	4147

Table 2: Geographic Distributions of CDN.

tion (the activeness of a CDN server is tested by having the CloudBeacon server attempt a TCP connection with the CDN server at port 80). The list is randomly generated from all the servers in each location, and refreshed every hour.

When a client loads CloudBeacon and requests a workload item, the CloudBeacon server will first choose a location for the measurement target, and then randomly select a server from the list of CDN servers at that location. In this manner, with high probability the client contacts different CDN servers when repeatedly visiting the same CDN location (and therefore doesn't use persistent TCP connections). CloudBeacon then conducts reflection pings several times to the randomly chosen server; we then estimate the RTT as the minimum of the observed latencies between the client and this location.

3.2 Throughput Measurement Methodology

Evaluating throughput performance requires downloading large objects from the servers at *targeted* locations. Unfortunately, CDN servers make this task very difficult – as explained earlier, a HTTP request will be rejected by the CDN servers, if the request URL contains an IP address (so as to specify a target), instead of a hostname. Therefore, we explore another existing infrastructure to measure throughput – the SpeedTest network (STN) [2], which allows CloudBeacon to download large objects from its servers.

The SpeedTest network is the infrastructure behind a broadband testing service provided at `www.speedtest.net`. It consists of web servers located in 240+ cities in over 80+ countries. Broadband testing works as follows: a user visiting `www.speedtest.net` is provided a world map, showing all available SpeedTest servers by their locations. The user clicks any server to trigger a broadband test (most users will choose a highlighted one, which is the closest in geographic distance to the user). Through a series of HTTP requests/responses, SpeedTest provides the user with the estimated latency, as well as with the estimated download and upload bandwidths to the selected server. To measure latency, a

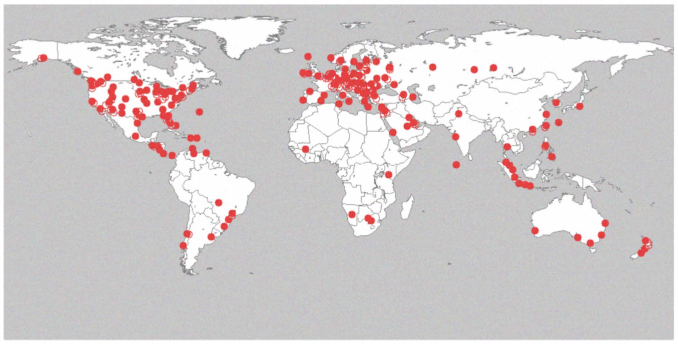


Figure 2: Server Presence of the SpeedTest Network

small text file is requested multiple times. To measure bandwidth, a large image file is requested multiple times. (More specifically, a relatively small image file is requested first to roughly estimate the bandwidth; then an image file of appropriate size is chosen to ensure that the download time won't be too short.)

The SpeedTest network servers are not as ideal as CDN servers, because they are less widely deployed than CDN A and likely not as well-connected as CDNs either. Nevertheless, the SpeedTest network still offers important insights. As illustrated in Figure 2, it is a viable alternative infrastructure for our throughput evaluation purpose, because 1) it is widely deployed; 2) its servers have good performance and are well-provisioned (as providing broadband testing is its core business).

4. EVALUATION OF CLOUD SERVICE DEPLOYMENT

The performance of a cloud service is, in principle, closely tied to the extent of its geographic coverage, i.e., how close it is to the end-users. On the other hand, extensive deployment requires high capital costs. It is thus desirable to investigate the trade-offs between deployment scale and performance. In this section, by applying the methodologies described in the previous section, we first compare two existing deployments. Then, we answer the following question – how many satellite data centers should the cloud service provide and where should the data centers be located.

One way to answer these questions is to deploy servers in hundreds of potential locations, and then use CloudBeacon to measure latency and throughput performance from end-users to these servers. However, this is exorbitantly expensive, especially for prediction purposes. In what follows, we will leverage the existing Internet infrastructures – CDN A, CDN B and the SpeedTest network – which already have servers deployed in many locations throughout the world, to provide insights to performance and cost tradeoffs.

4.1 Comparing Existing Deployments

We instrument CloudBeacon to conduct two experiments: 1) compare CDN A and CDN B with their full deployments; that is, the client (in the wild) is instructed to measure the performance to the closest location for each of the two CDNs (i.e., the closest among several hundred locations for CDN A and 18 locations for CDN B); 2) compare CDN A and CDN B under the same geographic deployment; that is, for CDN A, we choose only those locations where CDN B also has presence (denoted as CDN A*).

To determine the closest CDN location, the CloudBeacon server maps the client's IP address into a geographic location (longitude and latitude), computes the great circle distance to all CDN locations and selects the minimum one. Note that this method of determining the closest CDN location is only approximate, as it does not take into account network topology and dynamic network conditions. As server redirection is not the focus of this paper, it suffices to use this rough method in our studies here.[3] We use a commercial GeoLocation database to convert IP addresses into geographic locations. Table 3 summarizes the client data set.[4]

	# of clients
All	6,577
North America	3,678
Europe	1,382
Asia	1,232
South America	185
Africa	38
Oceania	31
Unknown	31

Table 3: Geographical Distribution of Clients

Figure 3 shows results for both experiments. Not surprisingly, CDN A achieves much lower latency than CDN B (87ms vs 406ms at the 95^{th} percentile), with both CDNs evaluated using the closest (geographical) location. On the other hand, when CDN A and CDN B are evaluated under the same geographic deployment, their performance (that is, of CDN B and CDN A*) is very close at low percentiles (e.g., 96ms vs 110ms at the 80^{th} percentile). However, at high percentiles, CDN A* shows much lower latency than CDN B, but why? To further examine this issue, we divide the clients by continents. From the results shown in Figure 4, we make the following observations:

[3]Please refer to a companion study [7] for the server redirection problem.

[4]Note that since the web sites hosting our CloudBeacon gadget are all in Seattle and New York City, to avoid bias, we only take those clients that are more than 100 miles away from Seattle/NYC into account.

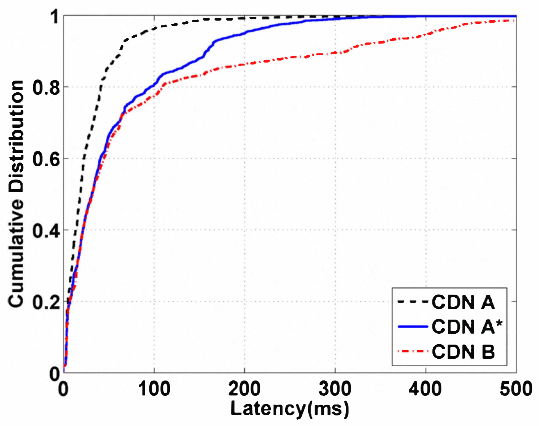


Figure 3: Comparison of Existing Deployments (CDN A* is a subset of CDN A, choosing only the same locations as CDN B.)

- For the clients in North America, all three deployments have comparable performance. This suggests that the scale of CDN B (or CDN A*) is sufficient in North America. Additional locations for CDN B in North America beyond the current 11 locations do *not* appear to reduce latency.

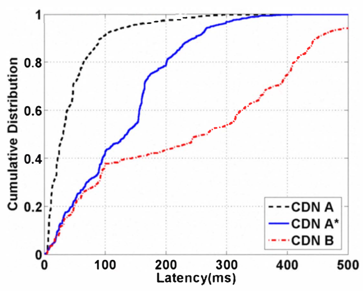
- For clients in Europe, CDN A* and CDN B have very similar performance under the same geographic deployment. However, when the scale of geographic deployment increases (i.e., from CDN A* to CDN A), latency is reduced. Thus, by improving the European coverage of CDN B, the latency can be significantly reduced.

- For clients in Asia, CDN A* outperforms CDN B significantly. These clients also largely contribute to the long tail for CDN B in Figure 3. This implies that geographic presence is not the only factor determining latency performance. Other factors, such as peering connectivity, also play a critical role. Quite likely, CDN B has *not* established extensive peering relationships with Asian ISPs.

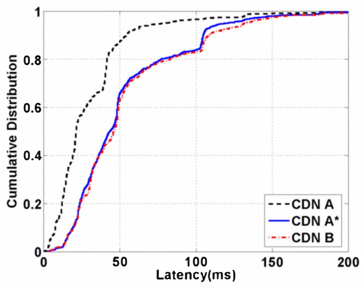
Therefore, when a cloud service provider evaluates the locations of a hypothetical deployment, it should aggregate the servers from various CDNs and use CloudBeacon to measure latency to the servers from each CDN in the location. The minimum value can then be considered the smallest possible latency (for when extensive ISP peering relationships are setup).

4.2 Evaluating Hypothetical Deployment

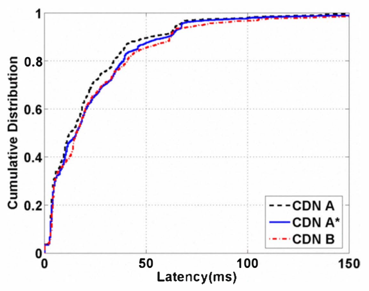
In this section, we describe a methodology to determine the number of locations a cloud service's edge network should contain and where the locations should be. Say the infrastructure we choose to leverage has M (sev-



(a) Asia



(b) Europe



(c) N. America

Figure 4: Comparison of Existing Deployment (Breakdown by Continent)

eral hundred) potential locations. Our task is to pick a subset (say N sites, with $N \ll M$) to form a smaller-scale deployment. Using CloudBeacon, we can evaluate the performance of this hypothetical deployment. By varying N, we can quantify the trade-off between deployment scale and performance.

Now, the interesting question is – how should we pick the N sites out of the total M potential locations? For that, we have developed the following heuristic method. We start with an initial deployment (could be empty), find the *best additional location* from the remaining locations, and add this location to the current deployment. During this process, we examine each additional location L by constructing a hypothetical deployment D' which consists of the current deployment D plus this location L. We then evaluate the performance from each client C to the hypothetical deployment D'. Aggregation across all clients yields a score for L. We then choose the location L that has the highest score. We continue to add locations one at a time until the deployment contains the desirable number of locations.

The following pseudo-code explains the process of finding the best next location.

```
// D: {locations of current deployment}
find_best_next_location(D)
   foreach L in remaining locations
      // D': hypothetical new deployment
      D' = D + {L}
      foreach C in all clients
         sum = sum + best_performance(C, D')
      if sum has reduced
         best_next_location = L
   return best_next_location
```

The above process requires measurements from each client to all the M locations. The measurement load can be prohibitively high. As an alternative, we use the following geographic distance-based method to choose deployment configurations. We first obtain the client population of a targeted cloud service and map each client into a *location atom* (a latitude-longitude tuple) by its IP address. Multiple clients can be mapped to the same location atom. The best performance between a client and a given deployment is approximated by the great circle distance between the client and the closest location in the deployment.

In this simplified method, finding the best next location becomes:

```
// D: {locations of current deployment}
find_best_next_location(D)
   for each L in remaining locations
      // D': hypothetical new deployment
      D' = D + {L}
      foreach A in all location atoms
        d = min_geographic_distance(A, D')
        sum = d * (total clients in A)
      if sum has reduced
         best_next_location = L
   return best_next_location
```

4.3 Deployment of Latency Sensitive Services

Now we have developed a method to choose a hypothetical cloud deployment of size N, we are ready to evaluate and compare deployments of different sizes. To factor in the distribution of real-world end-users, we first obtain the client population (5 million randomly selected IP addresses) from one popular Windows Live service. These IP addresses are grouped into 13,576 location atoms. We then select locations from CDN A's deployment to form our hypothetical deployment. In this paper, we compare deployments of size 18, 36, 72, as well as all the several hundred locations of CDN A (denoted as CDN18, CDN36, and so on). In addition, we also include a deployment with only 3 locations: east Washington state, Bay area and west Virginia (denoted as CDN3), which are popular locations where many commercial companies build their large data centers.

For each hypothetical deployment, we instruct CloudBeacon to measure latency to the closest CDN server. To see how deployment scale affects the performance in different regions, we break down the data by continents. Results are only shown for North America, Europe, and Asia (as Figure 5), for which we have sufficient samples.

We make the following observations:

- Clients in Asia and Europe experience high latency under the 3-location deployment. This is intuitive as all the three locations are in US. But surprisingly, the 3-location deployment appears to be good enough for the clients in North America! The latency is not very large (less than 80ms) even at high percentiles, and increasing the deployment size to 18 shows only marginal improvement.

- There are noticeable improvements from 36 locations to 72 locations for the clients in Europe and Asia. But 72 locations are certainly sufficient.

- As expected, the clients in Asia receive worse performance than those in Europe and North America. With only 34 locations, this is most likely due to the limited presence of CDN A itself (and thus our location choices) in Asia.

4.4 Deployment of Throughput Sensitive Services

To evaluate hypothetical deployments for throughput performance, we instrument CloudBeacon to retrieve large objects from the SpeedTest network (following the same request format as if a real client is doing broadband testing). Similar to the latency evaluation, we evaluate the throughput performance of a hypothetical deployment with only 3 locations (denoted as STN3). We also evaluate the deployments with 18 locations (the same locations as CDN B, which is assumed to have made good choices), 36, 72 and 200+ (denoted as STN18, STN36, and so on). The data set we collected is summarized in Table 4.

	# of data points
All	5,880
North America	3,348
Europe	1,291
Asia	992
South America	176
Oceania	26
Africa	23
Unknown	24

Table 4: Geographical Distribution of Clients

Again, we break down the results by continents, as shown in Figure 6. We have the following observations:

- The throughput gain from the deployment with 3 locations to 18 locations is most significant in North America and Europe, but not in Asia. This is most likely due to the choice of our initial deployment, as CDN B is deployed sparsely in Asia.

- The deployment with 18 has 11 locations in North America. It appears that, in terms of throughput performance, there is little gain further deploying in this region.

- Again, the clients from Asia receive worse performance than those in Europe and North America.

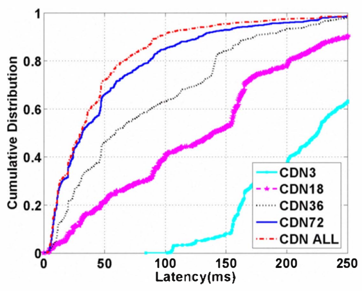
- Finally, in every continent examined, a significant portion of end-users cannot download faster than 1 Mbps. This explains why current streaming video services, such as Netflix, keep the lowest bitrate at only 375 Kbps.

5. RELATED WORK

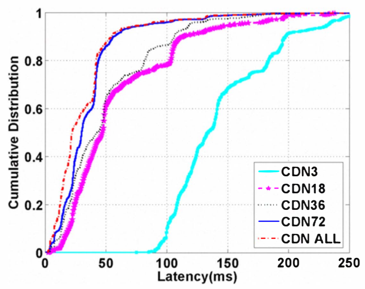
Cloud service providers often need to answer "what-if" configuration and deployment questions. Configuration questions include how service response time will be affected by changing the mapping of clients to the servers in different existing data centers. Tariq et al. [19] applied machine learning techniques and proposed novel algorithms to learn causal factors from user traces through existing deployments. They are able to predict response-time distributions for configuration changes. The WISE study [19], however, cannot evaluate "what-if" questions such as how the end-user perceived performance will be affected if a new data center is deployed and where should the location of the new data centers be so as to maximize the performance improvement. On the other hand, CloudBeacon provides methodologies and techniques to evaluate hypothetical new deployments.

There have been many measurement-based studies of web service performance. Researchers have developed systems and tools [4, 11, 13, 17, 18] to determine network characteristics and measure web service performance. Besides these efforts, many network monitoring systems [5,6,14,15,20,21] have been introduced as well. These systems applied various techniques, such as live packet trace collection, server-side log analysis, etc., to determine service performance. In our study, we not only provide a tool to measure service performance accurately, but also propose new methodologies for cloud service providers to evaluate hypothetical new deployments.

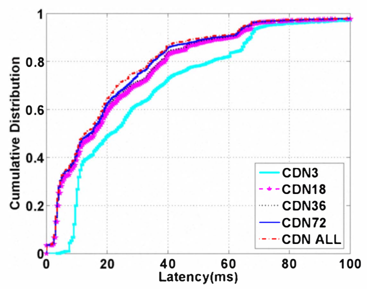
To benchmark the performance of Internet services, JavaScript is often embedded inside web pages. As an example, HP's Open View Web Transaction Observer uses JavaScript to implement many measurement functionality. A number of proposals, such as [3,4,9,12,17], also adopted JavaScript to evaluate the performance from end hosts. Different from those studies, CloudBeacon addresses the unique challenges in measuring existing infrastructure and evaluating hypothetical deployments for cloud services.



(a) Asia

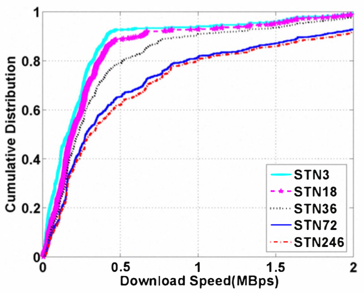


(b) Europe

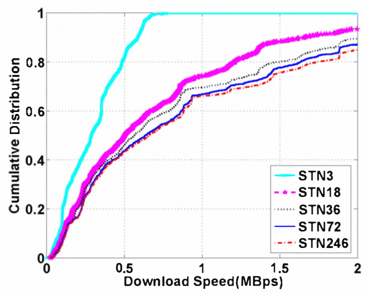


(c) N. America

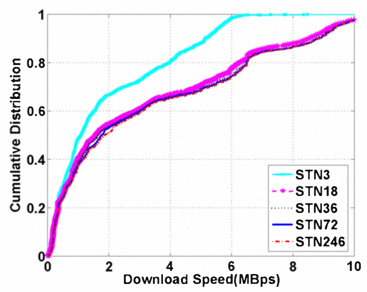
Figure 5: Evaluating Deployment for Latency Sensitive Services (Breakdown by Continent)



(a) Asia



(b) Europe



(c) N. America

Figure 6: Evaluating Deployment for Throughput Sensitive Services (Breakdown by Continent)

6. CONCLUSION

In this paper, we take a measurement-oriented approach and develop new methodologies for latency and throughput measurement for cloud service deployment. We develop CloudBeacon and, through illustrative large-scale measurements, show how such tool can help cloud services plan for expensive infrastructure changes.

7. REFERENCES

[1] Joost. http://www.joost.com.

[2] SpeedTest. http://www.speedtest.net.

[3] Atterer, R., Wnuk, M., and Schmidt, A. Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. *WWW* (2006).

[4] Casado, M., and Freedman, M. J. Peering through the shroud: the effect of edge opacity on ip-based client identification. *USENIX NSDI* (2007).

[5] Cherkasova, L., Fu, Y., Tang, W., and Vahdat, A. Measuring and characterizing end-to-end internet service performance. *ACM Trans. Interet Technol.* (2003).

[6] Fu, Y., Vahdat, A., Cherkasova, L., and Tang, W. Ete: Passive end-to-end internet service performance monitoring. In *Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference* (2002).

[7] Huang, C., Holt, N., Wang, Y. A., Greenberg, A., Li, J., and Ross, K. W. A dns reflection method for global traffic management. *USENIX ATC* (2010).

[8] Huang, C., Wang, Y. A., Li, J., and Ross, K. W. Measuring and evaluating large-scale cdns. *Tech Report MSR-TR-2008-106* (2008).

[9] Kiciman, E., and Livshits, B. Ajaxscope: a platform for remotely monitoring the client-side behavior of web 2.0 applications. *ACM SIGOPS Operating Systems Review* (2007).

[10] Kohavi, R., Henne, R. M., and Sommerfield, D. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. *KDD* (2007).

[11] Mao, Z. M., Cranor, C. D., Douglis, F., Rabinovich, M., Spatscheck, O., , and Wang, J. A precise and efficient evaluation of the proximity between web clients and their local dns servers. *USENIX* (2002).

[12] Marshak, M., and Levy, H. Evaluating web user perceived latency using server side measurements. *Computer Communications* (2003).

[13] Mosberger, D., and Jin, T. httperf: A tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.* (1998).

[14] Olshefski, D., Nieh, J., and Agrawal, D. Using certes to infer client response time at the web server. *ACM Trans. Comput. Syst.* (2004).

[15] Olshefski, D. P., Nieh, J., and Nahum, E. ksniffer: Determining the remote client perceived response time from live packet streams. *USENIX OSDI* (2004).

[16] Pathak, A., Wang, Y. A., Huang, C., Greenberg, A., Hu, C., Kern, R., Li, J., and Ross, K. W. Measuring and evaluating tcp splitting for cloud services. *PAM* (2010).

[17] Rajamony, R., and Elnozahy, M. Measuring client-perceived response times on the www. *USENIX Symposium on Internet Techonologies and Systems (USITS)* (2001).

[18] Seshan, S., Stemm, M., and Katz, R. H. Spand: shared passive network performance discovery. *USENIX Symposium on Internet Techonologies and Systems (USITS)* (1997).

[19] Tariq, M., Zeitoun, A., Valancius, V., Feamster, N., and Ammar, M. Answering what-if deployment and configuration questions with wise. *SIGCOMM* (2008).

[20] Wei, J., and Xu, C.-Z. eqos: Provisioning of client-perceived end-to-end qos guarantees in web servers. *IEEE Trans. Comput.* (2006).

[21] Wei, J., and Xu, C.-Z. smonitor: A non-intrusive client-perceived end-to-end performance monitor of secured internet services. *USENIX* (2006).