2014 IEEE International Conference on Cloud Computing

Use of Network Latency Profiling and Redundancy for Cloud Server Selection

Minseok Kwon*, Zuochao Dou†, Wendi Heinzelman†,Tolga Soyata†, He Ba†, Jiye Shi‡
*Dept. of Computer Science
Rochester Institute of Technology
Rochester, NY 14623, USA
jmk@cs.rit.edu
†Dept. of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627, USA
{zdou2,wheinzel,soyata,ba}@ece.rochester.edu
‡UCB Pharma
Slough, UK
Jiye.Shi@ucb.com

***Abstract*—As servers are placed in diverse locations in networked services today, it becomes vital to direct a client's request to the best server(s) to achieve both high performance and reliability. In this distributed setting, non-negligible latency and server availability become two major concerns, especially for highly-interactive applications. Profiling latencies and sending redundant data have been investigated as solutions to these issues. The notion of a cloudlet in mobile-cloud computing is also relevant in this context, as the cloudlet can supply these solution approaches on behalf of the mobile. In this paper, we investigate the effects of profiling and redundancy on latency when a client has a choice of multiple servers to connect to, using measurements from real experiments and simulations. We devise and test different server selection and data partitioning strategies in terms of profiling and redundancy. Our key findings are summarized as follows. First, intelligent server selection algorithms help find the optimal group of servers that minimize latency with profiling. Second, we can achieve good performance with relatively simple approaches using redundancy. Our analysis of profiling and redundancy provides insight to help designers determine how many servers and which servers to select to reduce latency.**

***Keywords*-Cloud computing; server selection; latency profiling; data redundancy; measurement study**

I. Introduction

Today, servers (or replicas) in the Internet are increasingly placed in diverse locations to offer efficient and reliable services that handle a huge amount of data. These networked services include online shopping (e.g., Amazon), social networking (e.g., Facebook), video streaming (e.g., YouTube), and file hosting (e.g., Dropbox). This trend has only accelerated with the emergence and success of cloud computing that allows users to remotely access elastic servers with ample resources in a pay-as-you-go manner over the network. In this distributed setting, a client's request is ideally directed to a server (or a group of servers for parallelized processing) that minimizes response time seamlessly without user intervention.

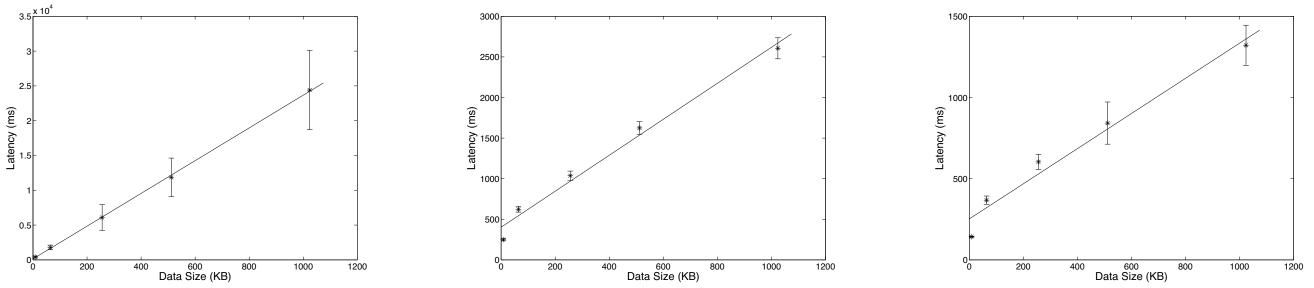
In providing networked services, non-negligible latency for remote access is considered a technical hurdle with respect to performance, and server availability is another concern for reliability. These concerns become more critical for highly-interactive applications that require frequent communications but relatively lightweight computation such as distributed database queries and updates, stock quotes and trading, and content distribution services, e.g., GFS [1] and Akamai [2]. One approach to deal with these concerns is profiling latencies to servers based on network measurements and solving server selection as an optimization problem given profiled latencies [3]. Another approach is sending data redundantly to servers, as this increases the probability to receive responses even in case of server failures and reduces response time by extending the pool of servers accessed [1].

As mobile devices like smartphones proliferate, these approaches also become important from the mobile-cloud computing perspective. Mobile devices use the cloud as the backend servers transparently, offloading heavy computing parts to the cloud [4], [5], [6], [7], [8], or creating an ad-hoc cloud of computing resources available at the distributed mobile devices [9]. In this case, an application running on the mobile may suffer from long latency and lack of informed server selection. The notion of a cloudlet was introduced as the agent that provides sufficient resources to the mobile and coordinates the connections between the mobile and the cloud [7], [10], [8], [4]. Mobile-cloud computing can benefit from having an intelligent function of server selection using profiling or redundancy in a cloudlet.

We study the effects of profiling and redundancy on latency when a client has a choice of multiple servers to connect to based on network measurements. Network profilers in existing server selection approaches [3] and mobile-cloud computing systems [6], [5] characterize the quality of network connections by measuring their performance such as average latencies and throughput, and their standard deviations. Beyond these performance metrics, we measure latencies to servers as samples when different sizes of data are sent, and create models to estimate latencies for an arbitrary data size. Using the estimated latencies at a given data size, we compare the latencies of server selection strategies with and without profiling or redundancy as the number of servers changes and data is partitioned into multiple chunks being sent to the servers. Our contribution is to characterize latency behaviors when profiling or redundant data transfer is adopted for server selection using network measurements. Such characterization helps gain insight in determining the number of servers and which servers should be selected.

Our key findings are summarized as follows. First, if data

978-1-4799-5063-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CLOUD.2014.114



(a) Instituto Tecnologico Buenos Aires (b) Zuse Institute Berlin (c) University of Oregon

Figure 1. Estimated latencies as a linear model using latency measurements for different data sizes.

are evenly distributed across servers, latencies decrease initially as more servers take the burden, but eventually increase as more and more servers are involved. Second, if data are distributed greedily–use the server with short response time first (see Section III-A for details)–utilizing the profile of average latencies to servers, latencies decrease and stabilize at the lowest point. Third, when data are evenly distributed to random servers without profiling, latencies do not change much even when more servers participate. Fourth, if data are distributed to random servers, this time redundantly but without profiling, latencies indeed decrease as more servers join, and moreover, latencies further decrease with higher redundancy.

The rest of the paper is organized as follows. Section II discusses our latency measurements. Section III discusses server selection and data partitioning algorithms using our measured latencies and their analysis. We summarize related work in Section IV and conclude in Section V.

II. Measurement of Latencies

We discuss our approach for obtaining latency measurements and latency estimation, and then present the results.

A. Configurations and Metrics

We conduct experiments on PlanetLab in which 1,168 computers (or nodes) are available at 550 sites around the world [11]. Also, the geographically diverse nodes in PlanetLab allow us to test different parts of the network with more comprehensive observations. We record our measurement data in February 2013. Five nodes operate as clients, and a total of 150 nodes are randomly selected as servers: 36 in North America, 6 in South America, 27 in Asia, 58 in Europe, and 3 in Oceania. Each client sends data to 30 servers (again randomly selected) over TCP, and we are able to obtain valid measurement data from 130 nodes out of the 150 servers. A client sends five different data sizes, namely 8 KB, 64 KB, 256 KB, 512 KB and 1 MB, giving 400 measurements for each data size totally, with 200 over the weekends and the other 200 on weekdays. This provides us with 2,000 measurements per server and totally 300,000 measurements.

We use latency as the primary performance metric. We measure round-trip time as latency, which is defined as the elapsed time between sending the first packet by the client and receiving an acknowledgment message for the last packet (at the application layer) from the destination server via the same connection. Due to clock drifts (clocks at the sender and the receiver are not synchronized), it is nearly impossible to measure one-way latency from the sender to the receiver. We analyze the latency distributions and basic statistics like average and standard deviation when various file sizes are transferred to see their characteristics. Throughput can also be estimated indirectly using the inverse of our latency measurements as needed.

B. Results

Figure 1 depicts the latencies measured at three servers (Argentina, Germany, and the West Coast of the United States) as an example out of the 130 servers. The dots denote the average latencies measured when the five different sizes of data are sent. The error bars show the standard deviation of the measured latencies. Both of the latencies and standard deviations increase as data size increases for all of the servers. The latency increase fits well a linear model, and thus we estimate latencies for other data sizes using linear regression. The resulting estimated latencies are represented as a line in the figure, and the values for a and b in the linear model, $y = ax + b$, are shown in Table I where x is data size and y is latency.

Table I
Linear model $y = ax + b$ (in ms) for the three servers in Figure 1.

Server	a	b
Buenos Aires	23.535531	13.3258792
Berlin	22.162589	40.1769235
Oregon	10.825825	25.1895639

The measured latencies will be used to estimate and profile latency to a target server for an arbitrary data size when original data can be split into multiple chunks and be sent to multiple servers. For example, if 1 MB data are evenly distributed and sent to two servers, the linear model for the first server is used to estimate the latency for the first chunk of 512 KB and the linear model for the second server is used to compute the latency for the other half chunk. More details are discussed in the following section.

III. Use of Estimated Latencies

Using the estimated latencies, we compare different server selection strategies in detail in this section. Our results provide insight and tradeoffs to be considered when profiling latencies for server selection. We consider data redundancy as well in this comparison.

A. Server Selection Algorithms

Latencies to servers vary widely depending on the locations of the servers and on the network conditions, as shown in the previous section. In some instances, a client has the option to divide data into smaller chunks and send these data chunks to a set of servers (possibly a different number). An interesting question is whether the latency linear model computed for each server can be utilized for the client to select server(s) to minimize the latency of its data transfer. Motivated by this question, we devise three data partitioning and server selection algorithms, namely *random*, *fixed*, and *greedy*, and conduct simulations, based on the estimated latencies as well as real experiments. Note that these algorithms allow requests to be sent to multiple servers concurrently and be processed in parallel.

In the random algorithm, a client sends m bytes data to n randomly selected servers located in different places for $n \geq 1$. The data are evenly divided among servers, i.e., $\frac{m}{n}$ bytes to each server. Moreover, the client may send copies of data chunks to more servers redundantly. In this case, it is possible for the client to receive multiple responses for a data chunk due to this redundancy. Latency in this case is defined as the elapsed time from when the client starts sending the first data chunk until when the first responses for all of the data chunks are received. The redundant data transfer thus exploits more servers and networks, and hence reduces latency by increasing the probability that a response returns earlier.

The fixed algorithm is the same as the random algorithm except 1) there is no redundancy, and 2) there is server selection. In this algorithm, the servers are sorted in non-decreasing order in terms of the average estimated latencies, and then the first n severs from the lowest latency upward are selected when sending the data to n servers. Latency is the time that it takes for the last response to be returned (note that there is no redundant response).

In the greedy algorithm, we first order the servers by their latencies like the fixed algorithm, and give the first chunk to the server that can respond in the minimum amount of time. The size of the first chunk is the smallest possible that the data can be divided into. We then give the second chunk to the server that can complete in the minimum amount of time. Note that this may be the same server as given the first data chunk if the time for the first server to complete both of the first and second data chunks is less than the time for the second server to complete just data chunk two. We continue in this way, greedily selecting the server for each chunk in turn. This latency is indeed the lower bound of latency. Since finding a set of such data chunks is inherently a complex optimization problem, we use the approximation algorithm described in Algorithm 1 for our simulations and experiments. The main idea is to move a portion of data ($a(i)u$ in line 8) from $server(i)$ when $i = 1, \ldots, k-1$ to a newly added $server(k)$, and iterate the process until finding the minimum latency.

Algorithm 1 Approximated Greedy Algorithm

```
1:  for $k = 2 \rightarrow n$ do
2:      $flag \leftarrow true$
3:      while $flag$ do
4:          for all $server(i)$ as $i = 1 \rightarrow k-1$ do
5:              $temp(i) \leftarrow data(i)$
6:              $data(i) \leftarrow data(i) - a(i)u$
7:          end for
8:          $data(k) = totalData - \sum_{i=1}^{k-1} data(i)$
9:          $curLat \leftarrow \infty$
10:         if $max(lat(0), lat(1), \ldots, lat(k)) < curLat$ then
11:             $flag \leftarrow true$
12:             $curLat \leftarrow max(lat(0), lat(1), \ldots, lat(k))$
13:         else
14:             $flag \leftarrow false$
15:             for all $server(i)$ as $i = 1 \rightarrow k-1$ do
16:                 $data(i) \leftarrow temp(i)$
17:             end for
18:         end if
19:     end while
20: end for
    {$data(i)$: data size sent to $server(i)$
    $a(i)$: rate that latency increases as $data(i)$ increases
    $lat(i)$: latency of $server(i)$
    $totalData$: total data size}
```

B. Results

We compare latencies estimated for the three algorithms using Monte-Carlo simulations and real experiments. For the simulations, we randomly generate latency L_i following a distribution with average latency E_i and standard deviation σ_i for a given data chunk size. We find that the latencies of the linear models (Figure 1) approximate a Gaussian distribution well with different mean and standard deviation values, and E_i and σ_i are estimated based on that. We use Gaussian distributions with interpolated mean and standard deviation values for the points for which measurement data are not available (e.g., when data size is 400 KB in Figure 1). We then run the random, fixed, and greedy algorithms, and compute the total latency, which is $max(L_1, L_2, \ldots, L_n)$ for n participating servers, if there is no redundancy. This process repeats a sufficiently large number of times. Note that in the figures, R1 represents the random algorithm with the client sending only a single copy of data (no redundancy), R3 is the random algorithm with the client sending three copies of the data (redundancy), R5 sends five copies, and R10 sends ten copies.

In Figure 2(a), we show the latencies estimated from the simulations when data are partitioned and sent using the random algorithm. Each random algorithm sends a different number of duplicate copies as defined above. The number of servers changes from 1 to 55 and 1 MB data are sent. For each algorithm, we run 100 simulations with different random number generator seeds for selecting servers and average the results. No server is selected for more than one copy of data chunk, i.e., each copy is sent to a different server. This algorithm increases reliability in case of server or network failures–responses return with higher probability with redundancy. The algorithm, however, reduces the total number of servers that a client can reach when copies of data chunks are sent as discussed. For example, the client in R3 can use only one third of the available servers since three copies of a data chunk are issued.

In addition, we run real experiments over the wide-area network to validate our simulation results. For the experiments, we are able to collect valid data from 41 servers out of 55 randomly chosen PlanetLab nodes (refer to Section II for details) providing 400 measurements for latency during weekdays as a node at Yale University sends 1 MB data to the chosen servers. Again, we repeat the experiments 100 times. As Figure 2(b) shows, the latencies from the real experiments behave similar to those from the simulations for all R1, R3, R5, and R10. In comparison to the simulation results (Figure 2(a)), the latencies are in general higher (20–30%) because the latencies from the experiments are measured during weekday daytime while the simulation latencies are profiled based on the latencies measured at different days and times including weekdays, weekends, daytime and night.

The estimated latencies for the fixed and greedy algorithms are shown in Figure 3. The latencies from both the simulation and the real experiments are presented and compared. The experimental setup is the same as that of the random algorithm as discussed above. As shown in the figure, the number of servers in the experiments are no more than ten due to the limited number of concurrent TCP sessions at the sender. The latencies from the experiments behave nearly identical to the latencies from the simulations for the fixed algorithm; the trends of the latencies from the experiments and simulations are similar for the greedy algorithm. Due to the same reason as discussed above for the random algorithm (Figure 2(b)), the latencies from the experiments are higher than those from the simulations. Figure 4 shows the latencies of all the three algorithms together computed from the simulations for comparison.

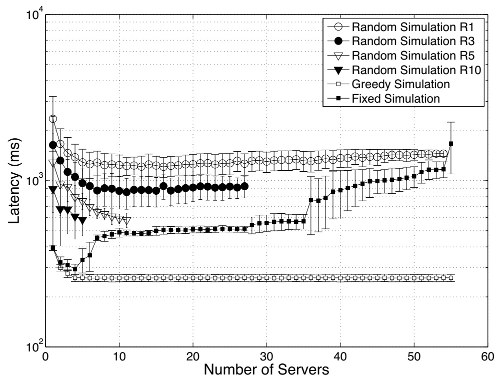
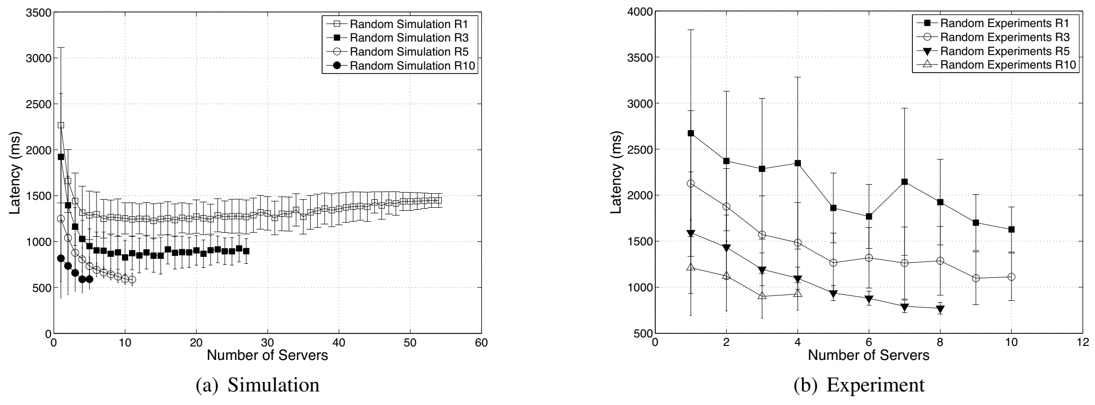


Figure 4. Comparison of the latencies of the random, fixed and greedy algorithms.

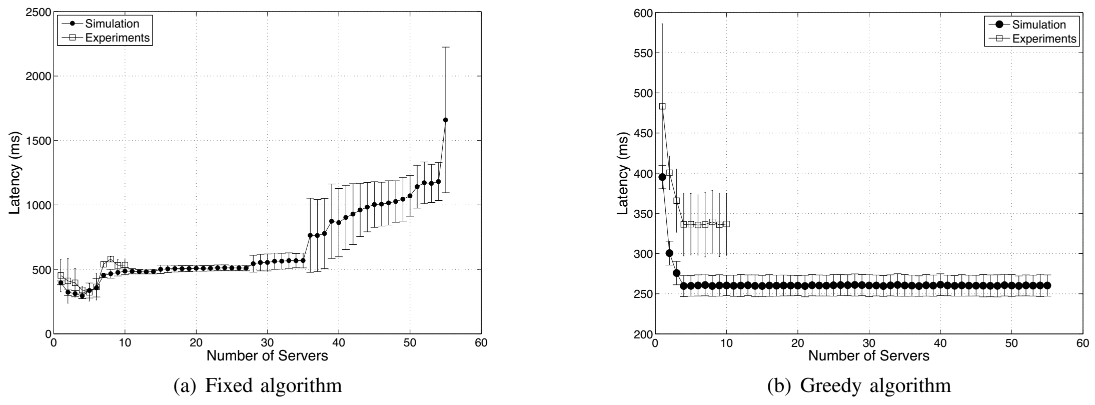
From these results, we observe several interesting phenomena. First, the latencies obtained from the random algorithm without redundancy (R1) do not change significantly as the number of servers changes except an extremely few servers (1–2), and are approximately five times higher than those of the greedy algorithm. The server that responds last determines the latency since the latency is $max(L_1, L_2, \ldots, L_n)$, and that server is selected at random regardless of the number of servers. Second, the latencies with redundancy (R3, R5, and R10) decrease as more servers are selected, and drop more sharply with higher redundancy (more copies), e.g., R10 drops faster than R3 and the latency of R10 is 2.5 times lower than the latency of R3 when the number of servers is 5. As more servers are involved for more copies of data chunks, the probability that a response for a data chunk returns earlier increases. The decrease of latencies, however, does not continue beyond a certain number of servers (e.g., seven for R3 in Figure 2) because there are not enough redundant copies to explore the increasingly diverse servers.

Third, the latencies in the fixed algorithm decrease and then increase beyond a certain number of servers–around five to six in Figure 3(a). Latency initially decreases as more servers take the burden of data, while latency increases later as the long latencies of servers that join later overshadow the benefits of burden sharing. Fourth, the latencies in the greedy algorithm decrease to the lowest point and stabilize similar to the random algorithm with high redundancy. The greedy algorithm seeks optimal server selection in order to minimize latency. The latencies do not continue to decrease since the data transfer requires fundamental overheads like propagation delay.



(a) Simulation (b) Experiment

Figure 2. Latencies as the number of servers changes when the random algorithm is used.



(a) Fixed algorithm (b) Greedy algorithm

Figure 3. Latencies as the number of servers changes when the fixed and greedy algorithms are used.

C. Profiling versus Redundancy

In a distributed computing setting like cloud computing, a client often relies on profiling based on network measurements to select the optimum group of servers and the way data are partitioned in order to meet its performance goals, e.g., minimal latency. A profiler probes the network, gathers measurements from the probe, analyzes the measurements, and creates the profiles of the measurements as functions of different parameters like days (e.g., weekdays or weekends), time (e.g., morning or night), and geographical locations. Of course, this probe and analysis needs to repeat periodically. Using the profiles, the user can run an intelligent algorithm to compute the best group of servers without much computational overhead. The results and analysis in the previous section re-iterate the importance of this intelligent algorithm in profiling. For example, the profiler can find out which six servers would provide the minimal latency when data size is 1 MB using the results in Figure 3.

Our results also imply that redundancy can contribute considerably to reducing latency in a reasonably simple way. As illustrated in Figure 4, a large amount of redundancy (R10) can reduce the latency compared with a random algorithm with no redundancy (R1) by 50%, approaching the latency achieved by the fixed and greedy algorithms. While redundancy injects many more packets into the network, it does not require the profiling overhead for network probes and estimation.

Additionally, these properties open the potential of hybrid dynamic profiling in which both sophisticated profiling and relatively simplistic redundancy are used together to select target servers. For frequent and long-lived network flows, profiling with periodic network probes helps, while short-lived flows can take advantage of redundancy without much overhead for network measurement and computing. It is also feasible that the latencies measured from on-going flows with redundancy are fed to the profiler and used as network measurements.

IV. Related Work

There have been attempts to measure latency accurately in the literature. IDMaps [12] deploys end hosts at strategic locations, gathers latencies, and then synthesizes them to estimate the latency between end hosts. GNP [13] estimates latency based on coordinates that are computed using landmark hosts. King [14] is a tool that estimates the latency between arbitrary end hosts by using recursive DNS queries without requiring extra infrastructure. Vivaldi [15] helps an end host compute its Internet coordinate for localization using only a few latency measurements in a fully distributed way. Madhyastha *et al.* [16] predict latency based on measurements of the Internet routing topology, connectivity, and policy. Netvigator [17] is a network proximity and latency estimation tool using landmark hosts and intermediate routers. Zhang *et al.* analyze and characterize Internet delays using these tools [18].

The measured latencies are used for server selection in routing, content distribution networks, and cloud computing. DONAR [3] is a distributed system that determines the best replica server considering both client and server loads, cost, and locations. Related to DONAR, several studies were conducted for content retrieval and server selection with the cloud. Khosla *et al.* investigate the performance degradation problem in cloud-based DNS [19]. Xu and Li propose a practical framework for cloud datacenter selection with special attention to fairness [20], and study the problem of joint request mapping and response routing [21]. CloudGPS [22] presents a server selection algorithm in the cloud that enables ISPs to configure a global performance function for low latency or cost in a scalable and ISP-friendly way. Given the high variability in performance of cloud services, Dealer [23] seeks to minimize response time for geo-distributed, interactive and multi-tier applications.

Recent advances in mobile, cloud and networking technologies enable mobile devices to offload heavy computation seamlessly, transparently, and cost-effectively to the cloud, in which rich resources are available [4], [6], [5], [24]. These mobile-cloud computing architectures have a profiler that provides estimated duration times of input executables based on measurements for devices, programs, or networks. Our results suggest that cloudlets can be used to support mobile-cloud computing by enabling network latency profiling in conjunction with intelligent server selection algorithms to connect to the best-performing server(s) in the cloud. Alternatively, the mobile, which cannot easily profile the network latencies, can achieve good latency performance through the use of redundancy as discussed in [25].

V. Conclusions and Future Work

We have studied latency profiling for server selection using network measurements in distributed and cloud computing. The objective of this study is to understand the behaviors of latencies as a client has options to choose among different groups of servers, and to design efficient and practical algorithms that select servers with minimal latency. Using models based on measured latencies to servers, we have tested the random, fixed and greedy data partitioning and server selection algorithms. Our results indicate that latencies can be reduced using the fixed algorithm compared to the random strategy, and can be nearly optimized as data are partitioned and sent to servers greedily. Moreover, the random algorithm can reduce latency significantly if data are sent redundantly. This implies the potential of dynamic profiling in which the greedy strategy is used for long-lived regular traffic while the random strategy with redundancy is used for short-lived traffic.

We have developed a mobile-cloud architecture that provides a face recognition platform on a mobile device. We will extend this architecture to incorporate our dynamic profiling and redundancy ideas and results. Highly interactive applications including face recognition in the distributed setting would benefit significantly from such profiling and redundancy. We are also considering an intermediate device called a cloudlet for accelerating the execution of the face recognition application. The computation for profiling and redundancy can be performed at the cloudlet for faster processing.

VI. Acknowledgments

This research was funded in part by UCB Pharma and by CEIS, an Empire State Development-designated Center for Advanced Technology.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proc. of ACM SOSP*, October 2003, pp. 29–43.

[2] Akamai Technologies, "Akamai," 2010, http://www.akamai.com.

[3] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford, "DONAR: Decentralized Server Selection for Cloud Services," in *Proc. of ACM SIGCOMM*, Aug.–Sep. 2010, pp. 231–242.

[4] M. Satyanarayanan and P. Bahl and R. Caceres and N. Davies, "The Case for VM-based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, 2009.

[5] B. Chun and S. Ihm and P. Maniatis and M. Naik and A. Patti, "CloneCloud: Elastic Execution Between Mobile Device and Cloud," in *Proc. of EuroSys*, Apr. 2011, pp. 301–314.

[6] E. Cuervo and A. Balasubramanian and D. Cho and A. Wolman and S. Saroiu and R. Chandra and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proc. of ACM MobiSys*, Apr. 2010, pp. 49–62.

[7] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision: Real-Time Face Recognition Using a Mobile-Cloudlet-Cloud Acceleration Architecture," in *Proc. of IEEE ISCC*, Jul. 2012.

[8] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating Mobile Cloud Computing: A Survey," in *Communication Infrastructures for Cloud Computing*, H. T. Mouftah and B. Kantarci, Eds. Hershey, PA, USA: IGI Global, Sep. 2013, ch. 8, pp. 175–197.

[9] H. Ba, W. Heinzelman, C. Janssen, and J. Shi, "Mobile Computing – A Green Computing Resource," in *Proc. of IEEE WCNC*, Apr. 2013.

[10] T. Soyata, R. Muraleedharan, S. Ames, J. Langdon, C. Funai, M. Kwon, and W. Heinzelman, "COMBAT: mobile-Cloud-based cOmpute/coMmunications infrastructure for BATtle-field applications," *Modeling and Simulation for Defense Systems and Applications VII, SPIE*, vol. 8403-20, Apr. 2012.

[11] PlanetLab, "PlanetLab," http://www.planet-lab.org.

[12] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 525–540, Oct. 2001.

[13] E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinate-based Approaches," in *Proc. of IEEE INFOCOM*, Apr. 2001, pp. 170–179.

[14] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts," in *Proc. of ACM IMW*, Nov. 2002, pp. 5–18.

[15] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," in *Proc. of ACM SIGCOMM*, Aug.–Sep. 2004, pp. 15–26.

[16] H. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A Structural Approach to Latency Prediction," in *Proc. of ACM IMC*, Oct. 2006, pp. 99–104.

[17] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, "Estimating Network Proximity and Latency," *ACM SIGCOMM CCR*, vol. 36, pp. 39–50, Jul. 2006.

[18] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space," in *Proc. of ACM IMC*, Oct. 2006, pp. 85–98.

[19] R. Khosla, S. Fahmy, and Y. C. Hu, "Content Retrieval Using Cloud-based DNS," in *Proc. of IEEE Global Internet Symposium*, March 2012, pp. 1–6.

[20] H. Xu and B. Li, "A General and Practical Datacenter Selection Framework for Cloud Services," in *Proc. of IEEE CLOUD*, June 2012, pp. 9–16.

[21] ——, "Joint Request Mapping and Response Routing for Geo-distributed Cloud Services," in *Proc. of IEEE INFOCOM*, Apr. 2013, to appear.

[22] C. Ding, Y. Chen, T. Xu, and X. Fu, "CloudGPS: A Scalable and ISP-friendly Server Selection Scheme in Cloud Computing Environments," in *Proc. of IEEE IWQoS*, June 2010, pp. 1–9.

[23] M. Hajjat, S. P. N, D. Maltz, S. Rao, and K. Sripanidkulchai, "Dealer: Application-aware Request Splitting for Interactive Cloud Applications," in *Proc. of ACM CoNEXT*, Dec. 2012, pp. 157–168.

[24] T. Verbelen and P. Simoens and F. De Turck and B. Dhoedt, "Cloudlets: Bringing the Cloud to the Mobile User," in *Proc. of ACM Workshop on Mobile Cloud Computing and Services*, 2012, pp. 29–36.

[25] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More is Less: Reducing Latency via Redundancy," in *Proc. of ACM HotNets*, Oct. 2012, pp. 13–18.