Latency-Driven Replica Placement

Michal Szymaniak Guillaume Pierre Maarten van Steen
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081HV Amsterdam, The Netherlands
{*michal,gpierre,steen*}*@cs.vu.nl*

Abstract

This paper presents HotZone, an algorithm to place replicas in a wide-area network such that the client-to-replica latency is minimized. Similar to the previously proposed HotSpot algorithm, HotZone places replicas on nodes that along with their neighboring nodes generate the highest load. In contrast to HotSpot, however, HotZone provides nearly-optimal results by considering overlapping neighborhoods. HotZone relies on a geometric model of Internet latencies, which effectively reduces the cost of placing K replicas among N potential replica locations from $O(N^2)$ to $O(N \cdot max(logN, K))$.

1 Introduction

Replication is commonly employed by modern distributed systems to improve the communication delay experienced by their clients. Such systems typically deploy several replicas of their service in different parts of the Internet and automatically redirect each client to its proximal replica, which results in a better client experience [2].

An important issue that must be addressed before a service can be replicated is where to place replicas. Different replica placements are likely to result in different client-experienced communication delay, hence this decision is crucial for the performance of the entire system.

Several algorithms have been proposed to address the replica placement problem [4]. The *Greedy* algorithm, for example, places replicas one-by-one, each time exhaustively evaluating all the possible replica locations. It has been shown to produce very good placements, yet its computational cost is quite large. Another algorithm, called *HotSpot*, places replicas on nodes that along with their neighbors generate the greatest load. It has a slightly lower computational cost, but its produced placements are not nearly as good as those of Greedy.

We propose to produce high quality placements while reducing the required computation time by taking a two-step approach. The first step is to select *network regions* where replicas should be placed. A network region is a group of nodes whose latencies to each other are relatively low. Other properties of nodes, such as the available storage space and network connection bandwidth, are irrelevant at this stage. This simplification accelerates the region-selection algorithm, especially if the latencies are modeled efficiently.

Once the network regions have been identified, the second step is to choose individual replica-holding nodes in different regions. This time, however, it is possible to consider all the factors ignored during the first step, as the number of nodes in a region is much smaller.

This paper proposes HotZone, an algorithm that addresses the first step, that is the region identification. For evaluation purposes, we assume that the second-step algorithm simply selects the node with minimal average distance to all other nodes in the region.

HotZone relies on the fact that Internet latencies can be modeled in an M-dimensional geometric space [9]. In this model, nodes are assigned M-dimensional coordinates. The latencies between any two nodes are modeled as the distance between their corresponding coordinates. HotZone identifies network regions as groups of nodes with proximal coordinates and places replicas in the most active regions. In this way, it avoids the costly pairwise latency estimations between all potential replica locations. This reduces the computational cost of HotZone to $O(N \cdot max(logN, K))$, which is significantly lower than those of the previously proposed algorithms.

We compare HotZone with three other algorithms, including Greedy and HotSpot. We show that the placement quality offered by HotZone is on average off by 5% from that of Greedy. We also demonstrate that the lower complexity of HotZone leads to significant gains in placement computation times, up to 3 orders of magnitude when placing 20 replicas[1].

The rest of this paper is structured as follows. Section 2 discusses relevant research efforts. Section 3 describes the details of our replica placement algorithm, and analyzes its computational complexity. Section 4 evaluates the algorithm performance. Finally, Section 5 concludes.

[1]This paper has been shrunk to fit 7 pages. The full 10-pages version can be found at http://www.globule.org/publi/LDRP_saint2005.html

2 Related Work

2.1 Replica Placement

Many replica placement algorithms have been proposed in the literature [4]. In general, their goal is to either optimize the client performance given an existing infrastructure, or minimize the infrastructure cost while achieving given client performance at the same time. This performance goal can be expressed by means of many different metrics, but placement algorithms typically optimize only client-to-replica latency, as it corresponds to the actual communication delay experienced by the client.

Since the replica placement problem is NP-complete, placement algorithms optimize their chosen metrics by means of heuristics [3]. An example of such a heuristic is the Greedy algorithm, which determines replica locations one-by-one [8]. It starts with exhaustively evaluating all possible locations for the first replica, and then chooses the location yielding the best performance. The subsequent replicas are placed in the same manner, except that starting from the second replica, all the previously placed replicas are considered to be fixed during metric calculation. Greedy can be used to optimize any metric. When optimizing the hop count metric, the Greedy algorithm produces placements that have been shown to be within a factor of 1.5 of those yielding optimal client performance.

The computational complexity of the Greedy algorithm is $O(K \cdot N^2)$, where K is the number of replicas to be placed and N is the number of potential replica locations. In a large-scale distributed system, N can be very large, leading to long-lasting computations. In such cases, the value of N can be reduced by clustering [5]. However, even relatively coarse-grained clustering may be unable to reduce N to a value for which Greedy can be run efficiently.

Another heuristic, called HotSpot, places replicas on nodes that generate the greatest load [8]. It first orders the nodes according to the amount of traffic generated jointly by each node and its neighboring nodes. Then, it places replicas on the first K nodes with the most active neighbors. Being a neighbor node is defined in terms of some network distance metric. For example, a node can consider all nodes whose latency is less than X milliseconds to be its neighbors. The placements produced by the HotSpot algorithm have been shown to be within a factor of 1.6 – 2.0 of the optimal one when optimizing on the hop count metric. Compared to the Greedy algorithm, the drop in efficiency is traded for only a slightly more attractive complexity, which is $O(N^2 + min(N \cdot logN + N \cdot K))$. This can still be too much for globally-distributed systems.

2.2 Latency Estimation

Latency-driven replica placement algorithms require that latencies between each pair of nodes can be estimated. A naive approach could be to let each node measure its latencies to all the other nodes. However, this may be infeasible if the number of nodes in the system is large, or if some of them remain out of the system's control. In a content delivery network, for example, most of the nodes are Web clients remaining beyond the control of the CDN operator.

A promising technique for latency estimation has been proposed in GNP, which models the Internet as an M-dimensional space [6]. GNP estimates the latency between any pair of nodes as the Euclidean distance between their corresponding M-dimensional coordinates. The coordinates of node X are calculated based on the measured latencies between X and Z designated "landmark" nodes, where Z is slightly larger than M. Consequently, estimating pairwise latencies between N nodes requires much fewer measurements ($N \cdot Z$) than in the naive approach ($\frac{1}{2}N^2$). It has been shown that estimation accuracy increases with the value of M up until $M = 6$, and only slightly afterwards [9]. Also, the GNP authors show that, in 90% of cases, the latency estimations derived from their system satisfy the following constraint:

$$\frac{2}{3}L_{observed} \leq L_{estimated} \leq \frac{3}{2}L_{observed}$$

where $L_{observed}$ and $L_{estimated}$ denote the observed and estimated latency, respectively. This relatively large error margin limits the applicability of GNP to systems that can tolerate such estimation inaccuracies. Since our algorithms operates on groups of nodes, it does not require high estimation precision. Consequently, we can rely on GNP to provide latency estimations at low cost.

Apart from facilitating latency estimation, the coordinates produced by GNP can also be used for node clustering [1]. Node clustering reduces the number of nodes that an algorithm must process as it treats each group as a single node. Since nodes with similar GNP coordinates are supposed to have minimal latency between them, coordinate-based clustering is natural in latency-optimizing systems.

In our earlier work, we have demonstrated that a system of distributed cooperating Web servers can employ GNP to locate Web clients such that the positioning application is completely transparent to the clients [9]. In the next sections, we show how such a system can efficiently calculate nearly-optimal replica placements.

3 Algorithm

The goal of a replica placement algorithm is to determine which nodes should host replicas. To solve this problem, we take a two-step approach. The first step is to select *network regions* where replicas should be placed. A network region is a group of nodes that access the replicated content and whose latencies to each other are relatively low. Since other node properties are irrelevant for region selection, only the relative latencies between nodes must be analyzed at this stage.

Once the network regions have been identified, the second step is to choose individual nodes that shall act as replica servers in different regions. During node selection, we consider various node-specific factors ignored during the region selection. However, because node selection takes place inside a region, the number of nodes that must be considered is much smaller, which makes the problem easier to solve.

We discuss in detail the region-selection algorithm, called HotZone, in the following sections. The complementary algorithm for node selection within a region is currently under development. For evaluation purposes, we assume that it simply chooses the node with minimal average distance to all other nodes in the region.

3.1 Region Selection

HotZone places replicas in network regions based on the relative latencies between nodes. Essentially, it works similar to the HotSpot algorithm: it first identifies network regions, then orders the regions according to the load they generate, and finally places replicas one-by-one in subsequent regions starting from the most active region. Placing a replica inside a heavily-loaded network region seems to be attractive, as it should improve the access latency for a large number of requests.

Identifying network regions means determining groups of nodes whose latencies to each other are relatively low. In general, it would require analyzing all pair-wise latencies between nodes, which is likely to be computationally expensive in large-scale systems.

To reduce the computational overhead, we identify network regions based on node coordinates produced by GNP. Recall that GNP approximates the latency between two nodes with the distance between their corresponding coordinates in an M-dimensional Euclidean space. The main observation we make at this point is that node coordinates are not uniformly distributed over the Euclidean space. Moreover, each cluster of node coordinates denotes a highly concentrated group of nodes whose latencies between each other are very low. It is therefore natural to identify network regions by determining clusters of node coordinates in the Euclidean space.

A question remains how to identify and measure coordinate clusters. To this end, we split the entire M-dimensional space into *cells* of identical size. A cell is an M-dimensional hypercube whose edge length is equal to some fixed value C. Each cell is uniquely defined by its *center point*. Because of the geometric properties of our space partition, the coordinates of each center point are

$$(C.k_1 + \tfrac{1}{2}C, .., C.k_M + \tfrac{1}{2}C)$$

for some values of k_i, where i ranges from 1 to M. We identify each cell by means of the integer-valued vector $(k_1, .., k_M)$ yielding the center point of that cell. The *density* of a cell is defined as the number of nodes whose coordinates fall within that cell. Note that this definition can easily be extended to support different node weights depending on the load generated by each node. To calculate the densities of all cells, the coordinates of each node $(x_1, .., x_M)$ are mapped to their corresponding cells $(k_1, .., k_M)$ according to the formula

$$k_i = \lfloor x_i/C \rfloor \quad i = 1, .., M$$

In this way, N nodes can be mapped into their corresponding cells in $O(N)$ time.

In a straightforward approach to the cluster identification problem, we could treat each cell as a potential cluster, and place replicas in the most dense cells only. However, clusters may span multiple cells. This happens when cell boundaries divide a coordinate cluster into several parts with each part falling into a different cell. A split cluster is less likely to be given a replica, since each of its parts can be too little to overweigh smaller, but undivided clusters.

To alleviate the problem of split clusters, we introduce the notion of a *zone*. Each zone is uniquely defined by a cell, and consists of that cell plus all the immediate neighboring cells. Each zone contains therefore 3^M cells in total. In other words, a zone is a group of adjacent cells, which together form a hypercube with edge length $3C$. Each zone shares its center point with the cell that defines that zone. We define the density of a given zone as the sum of the densities of all its member cells. Since zones do overlap, the density of a single cell contributes to the densities of several zones.

Operating on zones instead of cells does not completely solve the problem of split coordinate clusters, because coordinate clusters can still be large enough to be scattered over several disjoint zones. We return to this issue below, when discussing how to choose the cell edge length C.

When expressed in terms of zones, HotZone works similar to the Greedy algorithm. In every iteration, it identifies the most dense zone, marks this zone as a "replica holder," and removes all the node coordinates in this zone so that they are not considered in the remaining iterations. In this way, we implicitly assign the removed nodes to the replica that is currently being placed. More importantly, however, we reduce the possibility that replica-holding zones overlap, as empty cells are unlikely to fall within a zone considered to be dense. This ensures that all the replica-holding zones together cover as many node coordinates as possible, and that the replicas are ultimately placed in different parts of the system.

3.2 Cell Size Choice

The notion of zone allows us to reduce the problem of identifying coordinate clusters to that of identifying cells that yield dense zones. Note that there is no one-to-one correspondence between zones and coordinate clusters. Depending on the cell edge length C, a zone can contain several coordinate clusters, or a coordinate cluster can be scat-

tered over multiple disjoint zones. Therefore, selecting the value of C plays a key role for the efficiency of HotZone.

There are two factors on which the cell edge length C should intuitively depend. The first factor that influences it is the number of replicas K. Recall that HotZone effectively splits the entire space into K parts, and assigns each part to a different replica. The larger the value of K, the smaller the parts should be that are produced by the algorithm. Given how HotZone works, each such part should ideally be identified as a separate zone. The cell edge length C should therefore be inversely proportional to the number of replicas K, which is given to HotZone as a parameter.

The second factor that should affect the cell edge length C is the distribution of node coordinates in the space. Since we are using zones to identify coordinate clusters, it is natural that the cell size depends on the typical cluster size. For example, if most node coordinates fall in a small number of dense clusters, then these clusters can be identified with a small cell edge length C. On the other hand, if node coordinates are more evenly dispersed over the entire space, then the cell edge length C must be longer.

We defined coordinate distribution as the *average* distance between node coordinates. Note that calculating the exact average distance would typically require $O(N^2)$ operations, which we strive to avoid. In our case, however, it is enough to compute a good estimate of the average distance, as HotZone is not very sensitive to this parameter (see Section 4).

To obtain an estimate of the average distance, we calculate the average distance between an incrementally-growing number of node coordinates until the resulting value stabilizes. We found that, if we take node coordinates in a random order, then the incrementally-computed values quickly converge to the actual average distance. We note the average distance between the first E nodes as D_E. We determine the stabilization point by checking whether the value of D_E has stabilized over the last 100 iterations. To do so, every 10 iterations, we calculate the difference between the maximum and minimum value of D_E that has occurred within the last 100 iterations, and verify whether it is less than some threshold value ϵ:

$$max(D_{E-99}, .., D_E) - min(D_{E-99}, .., D_E) < \epsilon$$

If the threshold is exceeded, then we increase E and proceed with the next 10 iterations. Otherwise, we treat D_E as our estimate of the average distance.

We verified this method on a sample data set that contained 5728 node coordinates calculated in a 6-dimensional space. The positioned nodes were the Web clients that accessed our departmental Web server between the 1st and 30th of November 2003. The node coordinates were produced by SCoLE, which essentially runs a GNP instance in cooperation with a number of other hosts, acting as landmarks [9]. We set $\epsilon = 10$.

In our experiments the value of D_E converges after $E = 1110$ iterations. As we show in Section 4, similar numbers of iterations are sufficient to calculate the average inter-node distance also for significantly larger sets of node coordinates.

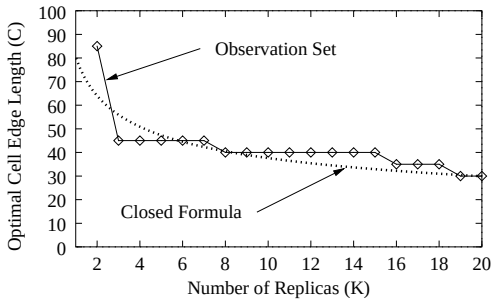

Figure 1. The (K,C) observation set and its approximation

In order to discover how the number of replicas and the average inter-node distance contribute to the cell edge length C, we empirically determined the optimal values of C in a wide range of scenarios. We then used non-linear regression to determine a function which outputs a good value of C for any combination of K and D. Considering that C is expected to be inversely proportional to the number of replicas K, and proportional to the average distance D, we decided to investigate the following family of functions:

$$C = \alpha \cdot \frac{D}{K^{\beta}}$$

where α and β are the coefficients that we need to determine.

To obtain the optimal values of C, we repetitively applied HotZone to the sample data set for each number of replicas $2 \leq K \leq 20$. We ignored the case when $K = 1$, as the best location is obviously the node whose average distance to all the other nodes is minimal. For each value of K, we repetitively ran HotZone with values of C ranging from 5 to D, taken in steps of 5. For each value of C, we evaluated the resulting placement by calculating the median distance between all nodes and their closest replicas. The C value yielding the shortest median distance was considered to be the best for the given number of replicas K. The outcome of this experiment was a set of observations, each being a pair of (K, best value of C). This set is depicted in Figure 1.

To obtain a closed function formula, we applied non-linear regression to our set of observations. The resulting values of α and β were 0.126 ($\approx \frac{1}{8}$) and 0.329 ($\approx \frac{1}{3}$), respectively, which gave us the following closed formula for the cell edge length C:

$$C \approx \frac{1}{8} \cdot \frac{D}{\sqrt[3]{K}}$$

where D is the average distance between nodes, and K is the number of replicas. The corresponding function is plotted in Figure 1. Note that since we computed the set of

observations based on a single data set, we used the set-specific value of the average inter-node distance D equal to 670. As we show below, however, the resulting closed function formula works also for other data sets, for which the value of D is completely different.

3.3 Complexity Analysis

In this section, we analyze the computational complexity of HotZone. Similar to the notation used in the previous sections, we use N to denote the number of nodes, K to denote the number of replicas, and M to denote the GNP space dimension.

The computational cost of HotZone consists of three parts, each corresponding to a single step. The first step is to determine the average distance between nodes. As we show in Section 4, HotZone obtains a good estimate of this distance by calculating the distance between a fixed number of randomly-selected nodes. The cost of this operation is constant.

The second step is to construct the set of zones. HotZone first assigns nodes to their corresponding cells, which costs $O(N)$ operations. Then, the set of non-empty cells is translated to the set of zones. To do that, HotZone identifies the neighboring cells of each cell, and sums their densities. Given that each zone contains a constant (3^M) number of cells, and that the number of cells cannot exceed N, this operation costs $O(N)$ cell-accesses. In our implementation, we sort all the cells according to their center points using Radix sort, which costs $O(N \cdot M) = O(N)$ operations. Then, we access individual cells using binary search, which yields the cost of $O(logN)$ per a cell access. The total cost of the second step is therefore $O(N \cdot logN)$.

The third step is to iteratively place replicas. For each replica, we identify the most dense zone, which requires inspecting all the zones. The identification of the most dense zone costs $O(N)$ operations, as we access all the cells in our cell table directly, and without using binary search this time. Given that the same operations are performed for each replica, the total cost of the third step is $O(K \cdot N)$.

The total computational cost of HotZone is the sum of the costs of the above three steps:

$$O(1)+O(N \cdot logN)+O(K \cdot N) = O(N \cdot max(logN, K))$$

This cost is significantly lower than that of the previously proposed algorithms, such as Greedy ($O(K \cdot N^2)$) and HotSpot ($O(N^2 + min(N \cdot logN + N \cdot K)$).

4 Evaluation

We evaluate HotZone based on three data sets produced by SCoLE, a GNP-like system that positions Web clients in a 6-dimensional space based on their latencies to 12 landmarks deployed worldwide [9]. We deployed SCoLE on three independent Web sites participating in our node-positioning experiment, and positioned all the clients that visited these sites between the 1st of February and the 31st of March, 2004. Each data set correspond to one of the sites, and consists of the positions of the clients visiting that site (see Table 1).

Dataset	Unique Clients	Client Profile
Andy	5,758	Universities worldwide
Cartoon	14,682	US schools and DSL users
MP3	64,041	European DSL users

Table 1. Dataset statistics

Dataset	Sample	Andy	Cartoon	MP3
α value	0.126	0.154	0.363	0.130
β value	0.329	0.310	0.651	0.373

Table 2. The values of the α and β coefficients

4.1 Average Distance Calculation

Determining the right cell size is crucial to good behavior of HotZone. Since the cell size depends on the average distance between nodes, we first investigated how the incremental calculation method performs on our data sets.

As it turns out, a similar number of nodes is necessary to calculate a good estimate of the average distance: 740, 650, and 820, for the Andy, Cartoon, and MP3 dataset, respectively. This observation confirms that this cost should be treated as a constant in the evaluation of the computational complexity of HotZone. Furthermore, this constant time is very small compared to the overall time of replica placement computation (about 100 milliseconds vs. several seconds).

4.2 Closed Formula Verification

HotZone calculates the cell size using a formula that depends on two parameters, α and β. Recall that in Section 3, we determined α and β by applying non-linear regression to a set of optimal cell sizes, which we computed based on a single sample data set. In this experiment, we verify how general these α and β values are by computing them for each of the other three data sets. The results are presented in Table 2.

As can be observed, the values of α and β computed for the Andy and MP3 data sets are very similar to their counterparts derived from the sample data set. However, it is clearly not so for the Cartoon data set, where the values of α and β are significantly different. We believe that this is because of the fact that nodes in the Cartoon data set are

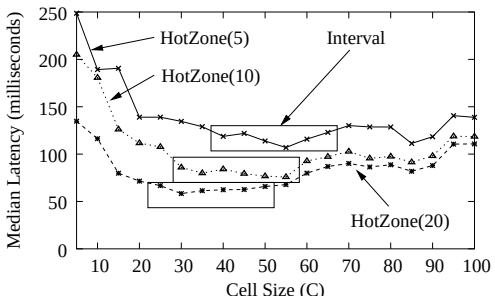

Figure 2. Placement quality vs. C values

more evenly distributed over the space. In that case, clients barely form any clusters, which leads to an optimal cell size significantly larger than expected. However, as we show in the next section, even sub-optimal values of C lead to placements almost as good as those produced by Greedy.

4.3 Sensitivity to C values

Although the performance of HotZone depends on the choice of a a good value for C, we show here that the quality of placements is quite resilient to suboptimal values of C. Figure 2 shows the performance of placements obtained with varying values for C betwen 5 and 100ms. For sake of clarity, we only show the results for placing 5, 10, and 20 replicas based on the Andy dataset. Other datasets exhibit similar behavior. Framed boxes show the interval of C values that yield performance results within 10% of the optimal.

As can be observed, the value of C can vary by a large amount without having very high impact on the quality of placements. This explains why HotZone produces nearly-optimal placements even though the C values themselves may be imperfect.

4.4 Placement Quality Comparison

A popular method of evaluating replica placements is calculation of the *average* latency between nodes and their closest replicas. However, the results produced by this method very much depend on outliers, which are nodes whose latencies to any other nodes are very high. Outliers typically use slow network connections, such as modems, and do not tend to form clusters. Because of these two factors, it is hard to improve the replica-access latencies of outliers without placing replicas specifically on them. We therefore decided to concentrate on the remaining nodes by evaluating placements with the *median* latency between nodes and their closest replicas.

We evaluated HotZone against the Greedy and HotSpot algorithms described in Section 2. Recall that HotSpot needs to know how to determine whether two given nodes are neighbors or not. We configured HotSpot to consider nodes to be neighbors only if the distance between them is less than some threshold value. To ensure the fairness of comparison, we tried various threshold values and report only the best result.

We also considered a variant of HotSpot, which we call "HotClear." After placing a replica on a given node, HotClear removes all the nodes from the neighborhood of this node so that they are not considered in the subsequent iterations. Also in this case, we tried several threshold values and report only the best result.

We apply the four evaluated algorithms to each of the three data sets. We iteratively place K replicas, for K between 1 and 20, and calculate the median node-to-replica latency for each value of K. The results are presented in Figure 3.

As can be observed, the original HotSpot algorithm performs significantly worse than all the others. This is not surprising: as it turns out, HotSpot places replicas on nodes whose coordinates are close to the centers of a few very active neighborhoods. Although these neighborhoods may change depending on the threshold value, the replicas are ultimately placed close to each other, which results in poor performance. This is exactly the effect that HotZone tries to avoid by using overlapping network regions.

The remaining three algorithms produce comparable results. Compared with Greedy, HotZone offers median latency that remains within 13%, 14% and 9% of that offered by Greedy for the Andy, Cartoon, and MP3 data set, respectively. The average difference between latencies offered by these two algorithms, however, is between 3% and 5% in all the data sets. Note that HotZone sometimes slightly outperforms Greedy, which can be seen in the graph depicting the replica placement based on the MP3 data set.

HotZone usually performs better than HotClear. In this case, however, the average difference in latencies is very small (between 5% and 7%) for the two smaller data sets, but it increases to 16% for the biggest data set. The case of HotClear shows that the original HotSpot algorithm can easily be improved to work effectively on node coordinate sets, but even then it still cannot outperform HotZone.

4.5 Placement Computation Times

The main advantage of HotZone over other algorithms is its low computational cost. In this experiment, we show how the differences in computational complexities of different algorithms translate into placement computation times. We measured the execution times of the four evaluated algorithms for each number of replicas K between 1 and 20, based on all the three data sets. The tests were performed on an idle PC equipped with a Pentium III 1GHz. The results are depicted in Figure 4 (note the logarithmic time scale).

As can be observed, the time needed by HotZone to compute its placements up to 3 orders of magnitude lower than the time needed by Greedy. In comparison with HotSpot

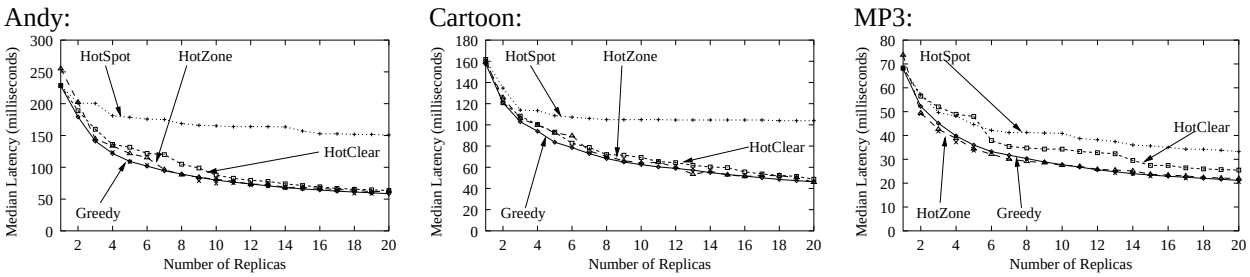

Figure 3. Placement quality comparison

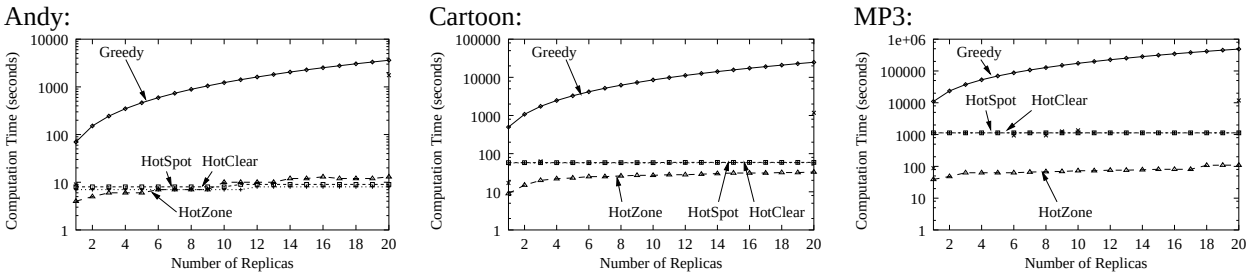

Figure 4. Placement computation times

and HotClear, however, the computation time of HotZone is better only for the two larger data sets, and the advantage of HotZone over these two algorithms increases with the number of nodes in a data set. On the other hand, for the smallest data set, the computation time of HotZone is comparable to these of HotSpot and HotClear. This indicates that HotZone is particularly suitable for large-scale systems, where the number of nodes is very high.

5 Conclusion

We have presented HotZone, a replica placement algorithm that optimizes node-to-replica latency based on node coordinates produced by SCoLE. Similar to HotSpot, HotZone places replicas on nodes that along with their neighboring nodes generate the highest load. However, the computational cost of placing K replicas using HotZone is $O(N \cdot max(logN, K))$, which is significantly lower than that of any previously proposed algorithm. This makes HotZone attractive for use in large-scale distributed systems.

HotZone produces results of comparable quality to those of Greedy. Furthermore, for large data sets, the computation time of HotZone is significantly lower than those of all other evaluated algorithms. In particular, it is up to 3 orders of magnitude lower than the computation time of Greedy.

We plan to use HotZone in Globule, a peer-to-peer content delivery network that our group is developing [7]. We believe that it will help us efficiently place replicas among a large number of Globule nodes.

References

[1] L. Amini and H. Schulzrinne. Client Clustering for Traffi c and Location Estimation. In *24th International Conference on Distributed Computing Systems*, Mar. 2004.

[2] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, 6(5), Sept. 2002.

[3] M. Karlsson and C. Karamanolis. Choosing Replica Placement Heuristics for Wide-Area Systems. In *24th International Conference on Distributed Computing Systems*, Mar. 2004.

[4] M. Karlsson, C. Karamanolis, and M. Mahalingam. A Framework for Evaluating Replica Placement Algorithms. Technical report, HP Laboratories, Palo Alto, CA, 2002.

[5] B. Krishnamurthy and J. Wang. On Network-Aware Clustering of Web Clients. In *ACM SIGCOMM*, Aug. 2000.

[6] T. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *21st IEEE INFOCOM Conference*, June 2002.

[7] G. Pierre and M. van Steen. Design and Implementation of a User-Centered Content Delivery Network. In *The 3rd IEEE Workshop on Internet Applications*, June 2003.

[8] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the Placement of Web Server Replicas. In *20th IEEE INFOCOM Conference*, Apr. 2001.

[9] M. Szymaniak, G. Pierre, and M. van Steen. Scalable Cooperative Latency Estimation. In *10th International Conference on Parallel and Distributed Systems*, July 2004.