2011 IEEE International Conference on High Performance Computing and Communications

SOLARE: Self-Organizing Latency-Aware Resource Ensemble

Heungsik Eom*, David Isaac Wolinsky**, and Renato J. Figueiredo*

*Advanced Computing and Information Systems Laboratory
Department of Electrical and Computer Engineering
University of Florida, Gainesville, USA
E-mail: eomhes@gmail.com, renato@acis.ufl.edu

**Department of Computer Science
Yale University
New Haven, USA
E-mail: isaac.wolinsky@gmail.com

Abstract— **This paper proposes and evaluates Self-Organizing Latency-Aware Resource Ensemble (SOLARE), a peer-to-peer self-organizing and self-managing cluster system based upon network coordinates and utility functions. In contrast to previous works, SOLARE is a fully decentralized clustering algorithm without any central units such as servers, super peers, cluster heads or landmarks. Furthermore, SOLARE allows for adaptability to dynamic network changes by monitoring the utility of a cluster and migrating nodes to other higher-utility clusters when the utility of an existing cluster is low. Quantitative, simulation-driven evaluations show that SOLARE is able to satisfy user demands expressed by utility functions that integrate system parameters in terms of intra cluster latencies and the number of cluster members. Also, we verify the ability of SOLARE to adapt to dynamic network changes through simulation based experiments that consider the number of nodes which migrate into another cluster and average utility value as nodes join SOLARE.**

Keywords-Clustering, utility functions, latency-aware, structured P2P networks.

I. INTRODUCTION

In recent years, the structures of large-scale network have been extended from client-server architectures for traditional web services to the distributed systems such as Peer-to-Peer (P2P) networks and Distributed Hash Tables (DHTs) [28-30]. Particularly, distributed systems have been increasingly applied to applications including online gaming, content distribution networks, and storage systems. One of the key challenges posed by these systems is locating or connecting a subset of nodes that satisfy user-demands with respect to network proximity. In distributed online gaming services, for example, players seek to organize clusters so that those in the same cluster can experience low latency to each other and enjoy the game without delay perceived by users. In addition, the supply of powerful commodity computers and the availability of high speed Internet have enabled grid computing environments capabilities. In such environments, latency-sensitive applications such as parallel processing tasks and workflows benefit from proximity among workers, and the ability to discover resources which satisfy job demands and are bound by pair-wise latency constraints is important. For such reasons, the inherent support for large-scale systems to self-organize into clusters that guarantee a certain degree of network proximity allows middleware and applications to seamlessly discover resources that are close in a network latency sense. With the approach described in this paper, proximity-aware queries can be efficiently performed through scalable P2P queries within the self-organized cluster(s) that a node belongs to.

In this paper, we propose SOLARE, a peer-to-peer, utility function based self-managing system that self-organizes clusters in a proximity-aware structured overlay topology. To do this, we rely on a structured P2P network as underlying global overlay network and a self-organizing, decentralized network coordinate system. As reviewed in the next section, there have been a number of related approaches whereby nodes are clustered based on certain criteria, but to the best of our knowledge, this paper proposes the first implementation which applies utility functions and network coordinates for the clustering process. The main idea of SOLARE is that each node searches and joins the highest utility valued cluster. On the other hand, if there is no cluster whose utility is greater than the user-defined threshold, the node creates a new cluster declaring its own coordinate as the virtual center of cluster. Furthermore, a node periodically monitors the status of the cluster that it is currently a member of in order to migrate to another cluster whenever the utility is dropped below a threshold. In our system, migrating clusters means changing the cluster membership rather than physically moving from one cluster to another cluster. In order to calculate the utility of clusters, we select the distance to the virtual center of the cluster and the size of the cluster (i.e., the number of cluster members) as utility properties. All participants are located in a 2-dimensional space such that each node has the ability to calculate the Euclidean distance to any points of 2-dimensional space or other nodes using a Vivaldi network coordinate system [2]. It is important to note that SOLARE is independent of the network coordinate system. In fact, even though we adopt the Vivaldi network coordinate system, other related approaches such as GNP [12] and PIC [13] can be used for the core of network latency prediction of our cluster algorithm as well. While this paper focuses on latency-based network coordinates, approaches that consider a bandwidth-based coordinate systems, such as Sequoia [11][31] can be also considered.

The rest of the paper is structured as follows. In Section II, we overview previous works on clustering approaches for distributed systems and utility functions in autonomic systems. Section III presents the system model and basic elements of SOLARE. Section IV describes the procedure of

978-0-7695-4538-7/11 $26.00 © 2011 IEEE
DOI 10.1109/HPCC.2011.38

the proposed clustering system and the performance evaluation of our system is presented in Section V. Lastly, Section VI concludes the paper.

II. RELATED WORK

A. Clustering approaches on distributed systems

Similar studies to our work have been done in order to offer proximity-based routing protocols. In [19], the Plethora routing core employs a two-level overlay architecture with global overlay and local overlays. In particular, local overlays rely on the organization of the Internet as a collection of Autonomous Systems (ASs) so that they serve as caches to provide data locality. IP-based clustering for P2P networks has no need for active measurement of latencies, minimizing system maintenance cost [23]. Instead, the authors provide a simple method for the construction of clusters based on static, readily available information (i.e., IP address) proving the correlation among common IP prefix length of communication nodes and latency. The above approaches, however, use only static information without any measurements, which makes it difficult to handle dynamic network changes. Francois Cantin et al. [20] adopt the Quality Threshold (QT) algorithm to propose a self-organized clustering scheme which is built on an existing Internet coordinate system. Also, they present two variants of distributed clustering algorithm: one aims at reduction of the clustering construction time, the other tries to minimize the overhead of the clustering process. Nevertheless, each node needs to perform the computation for QT algorithm as well as calculating the distance between its first- and second-order neighbors, introducing additional overheads. The novelty which distinguishes our work from the above-mentioned approaches is that our work is a fully decentralized and autonomous approach, not relying upon any central units such as servers, super peers, cluster heads or landmarks but utilizing utility functions.

Clustering techniques have been also adopted in other domains such as network modeling and performance evaluation. In [22], the authors show that both geographical and interest-based clustering properties inherently exist and can be leveraged to improve the search mechanisms in file sharing systems using P2P networks. Tian Bu et al. [27] provide the Internet topology generator using the cluster coefficient. Also, wireless sensor networks achieve prolonging the lifespan and reducing the overhead via cluster-based protocols [24-26].

B. Utility functions in autonomous systems

Utility functions provide a natural and advantageous framework for achieving self-optimization in autonomic computing [1]. In fact, applying utility functions to autonomic systems has been done in various studies. In [14][16], the authors have proposed utility-based resource allocation system consisting of a number of logically separated application environments each having an independent utility function. Along with application environments, a global resource arbiter performs the allocation of resources optimizing the summation of utility functions from application environments. Similar work has been considered in [17]. In this work, however, utility is described with respect to low-level resources such as CPU, disk, and memory instead of system-level attributes. In [18], the authors propose an energy efficient cooling system for data centers applying utility functions. It formulates simple utility functions that describe a tradeoff between energy and temperature and shows how to optimize a setting of control parameters, fan speeds and on/off state of air conditioners, using utility functions. Paul de Grandis et al. [21] present the fundamental steps of non-analytic approach on leveraging utility functions at the application level. Also they apply their approach to two different case studies, FTP download and log explosion.

III. SYSTEM MODEL AND ARCHITECTURE

The main goal in SOLARE is to construct utility functions-based, self-organizing and self-managing cluster systems such that nodes join highest ranked cluster with respect to its own preference expressed by a utility function. The system assumes a global, large-scale structured P2P overlay, from which virtual clusters are self-organized as sub-overlays based on the SOLARE algorithm.

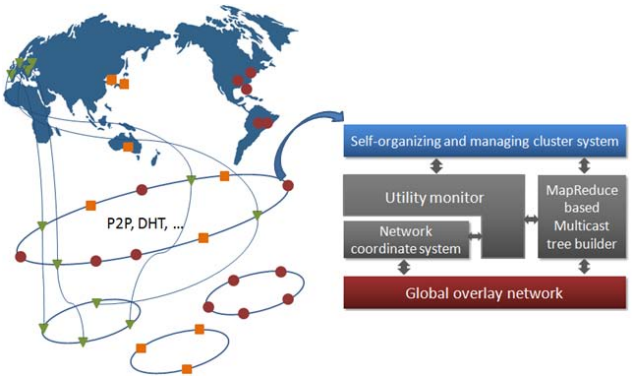

Figure 1. Self-Organizing Latency-Aware Resource Ensemble. Four nodes (the shape of triangle) in Europe may want to organize a cluster along with the others within a few tens of milliseconds. They participate in the self-organizing and managing cluster system on top of global overlay network and finally organize a cluster which satisfies their requirements.

Figure 1 illustrates the base concept of our system. In the figure, three smaller-scale clusters self-organize as virtual sub-overlays (triangle, circle, square) of the main global overlay based on their network coordinates and utility functions. Furthermore, each node monitors the cluster that it is currently involved in, so if its utility value drops below a threshold due to dynamic changes in network conditions, it migrates to another cluster to improve cluster utility. Consider, for example, nodes that are geographically distributed and connected to a large-scale overlay with millions of nodes. Assume a node wishes to connect to 100 nodes or more which are within 50 milliseconds of latency (i.e., round trip time). This node specifies its desirable cluster characteristics in terms of a utility function, and takes part in SOLARE on top of global overlay network, requesting

information from existing clusters through a bounded-multicast query mechanism which discovers which nodes belong to clusters that would maximize the node's utility. Finally, the node makes a decision on the best-suitable cluster based on its requirements. These steps involve the modules shown in Figure 1 (right). In this section, we describe the main constituents of SOLARE.

A. *Structured P2P network as global overlay*

The global overlay network provides an underlying substrate to place nodes in a network coordinate space and to propagate query messages. As soon as nodes join the global overlay network, they calculate their positions and send a query message for cluster information. We choose Symphony [7], a 1-dimensional Kleinberg small-world network as underlying global overlay network, and build upon the implementation of BruNet [6], a general P2P software framework. Symphony organizes nodes in a ring topology in which each node has a constant number of near connections and *log(N)* number of shortcut connections, where *N* is the number of nodes in the network.

B. *Query propagation system*

One of the essential features of a self-organizing cluster system is how nodes locate a candidate set of clusters in an efficient way, without missing high-quality clusters to join in terms of the user demand. With the purpose of improving the cluster searching quality with scalable query mechanism, we employ Map and Reduce functions on a self-organizing multicast tree [32]. This approach is based on a multicast tree builder consisting of two parts: Map and Reduce functions which process the propagation of query and merge the results, and a multicast tree builder that is the recursive algorithm to build a multicast tree. MapReduce is a software framework associated with information processing on large data sets [4]. The Map function works on a set of data to generate intermediate value while Reduce function merges all intermediate values associated with the identical intermediate key.

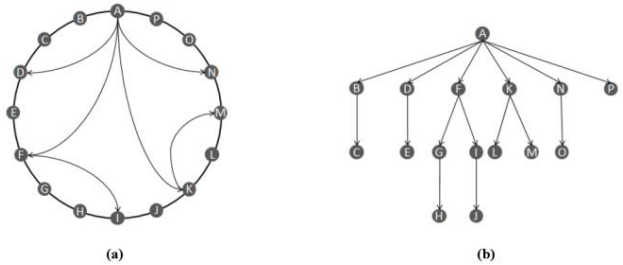

Figure 2. Map and Reduce functions based multicast tree builder (a) Ring topology of Symphony for 16 nodes (b) The multicast tree constructed by Map and Reduce functions based multicast tree builder

In conjunction with Map and Reduce functions, the multicast tree builder assists in propagating a message in a scalable manner (with *log(N)* depth) by building a multicast tree using structured shortcut and near connections [5]. This approach can be applied recursively to sub-overlays [33] such that the same mechanism used across the overlay to discover candidate clusters to be joined, can also be used at the application layer to discover nodes that satisfy requirements within a proximity-aware cluster.

Figure 2 shows how Map and Reduce functions based multicast tree builder creates the multicast tree and propagates a query on top of symphony. Let us assume that node A is an origin node which sends a message over the entire network (or alternatively a bounded region of the network, in this example the bounds are A and P). Node A sends a message to nodes B, D, F, K, N and P (which are referred as Node A's child nodes) through its near connections and shortcut connections by setting sub-multicast range as [B,D), [D, F), [F, K), [K, N) , [N, P) and [P, A-1) respectively. Nodes which receive the message send the message to their neighbors within sub-multicast range. Differently stated, node B has the responsibility of building the multicast tree in range [B, D). In such a way, nodes multicast the message until there is no connection and the multicast tree is built as shown in Figure 2(b). Each node in the tree computes a Map function, and results are back-propagated through the tree, with each node computing a Reduce function on the values they receive from its children. The strength of Map and Reduce functions based multicast tree builder lies in the ability to parallelize the map and reduce functions, providing the ability to select a subset of results based on utility functions, thereby bounding the bandwidth used in such queries. With a reduce function that bounds the number of results returned by a node by a constant, the average per-node bandwidth consumed by a query is constant [5][32].

C. *Network coordinate system*

A network coordinate system places nodes in some synthetic network coordinate space such that each node can predict the latency to other nodes. One example is Vivaldi, which achieves this by modeling a spring system [2]. Due to the triangle inequality of the network coordinate space, Vivaldi network coordinate system attempts to minimize the error between the predicted distance and actual latency instead of exploiting accurate coordinates. Each node involved in Vivaldi network coordinate system measures the RTT (Round Trip Time) to its neighbor n_i whose coordinate is x_i and computes the error e between measured RTT and Euclidean distance of its coordinate and x_i. Finally, it updates its new coordinate as follows:

$$Coordinate_{new} = Coordinate_{previous} + \delta \times e \times D. \qquad (1)$$

where δ is the timestep which quantifies the size of step toward new coordinate and D is the direction to new coordinate. An adaptive value of δ provides the control on the fraction of the step so that each node converges towards approximately accurate position quickly and precisely. The Vivaldi system has been implemented on top of Brunet and is used as a basis for the experiments in this paper. As soon as each node joins the global overlay, it runs the Vivaldi network coordinate system to obtain 2-dimensional coordinates so as to predict the latency to the candidate clusters as well as the neighbors.

IV. SOLARE MODULES AND ALGORITHMS

A. Utility functions

A self-organizing and self-managing system needs to serve diverse requests from a number of users who have different expectations on service quality. The challenge of this task is not only to approximately satisfy the demands from all the users but also to maximize the utilization of entire system. In this work, we make use of utility functions as a practical scheme of achieving self-organizing and self-managing cluster system in which each node describes its own preference on joining a cluster, and makes the best decision given a set of clusters. To apply utility functions to our system, first of all, we identify two attributes that nodes attempt to optimize.

- **Distance to the cluster (D_i)**: means the Euclidean distance between the coordinate of node and a virtual center of a particular cluster, *i*.
- **Size of cluster (*Si*)**: is the size of a cluster, that is the number of cluster members which involve in a particular cluster, *i*.

Next, we establish two utility functions, $U_{distance}$ and U_{size}, each expressing the preference on above two attributes. Figure 3 illustrates utility functions $U_{distance}$ and U_{size} we set in our system. Note that we have two key parameters to configure: maximum tolerance for distance to the virtual cluster center (D_{max}) and minimum preference for cluster size (S_{min}). The former is the maximum distance to the cluster which has positive utility value, while the latter means the minimum size of the cluster which has maximum utility, 1, regardless of the size of the cluster. Users can define their own utility functions by differently setting these two parameters.

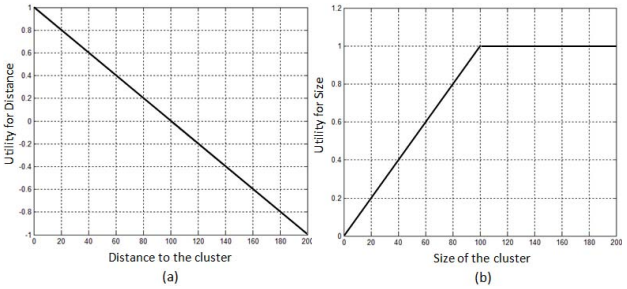

Figure 3. The type of utility functions for (a)distance to the cluster (D_{max} = 100) (b)size of the cluster (S_{min} = 100)

Finally, after adding the coefficients for each utility function, we merge two utility functions into total utility function as Eq(2) in which each node computes total utility value for a cluster.

$$U_{total} = \alpha_{size} \times U_{size} + \alpha_{distance} \times U_{distance} \qquad (2)$$

where α_{size} and $\alpha_{distance}$ are coefficients for U_{size} and $U_{distance}$, respectively, such that the sum of two values is 1. Similar to D_{max} and S_{min}, coefficients for size and distance are user-definable parameters so that each user presents the priority among the attributes.

B. Self-organizing and managing clsuter process

As described in previous section, utility functions represent user preference on selecting a cluster to join. In this section, we describe how nodes can self-organize the cluster architecture using utility functions. The main idea of our clustering algorithm is that each node searches and joins the highest utility valued cluster. On the other hand, if there is no cluster whose utility is greater than the user defined threshold, node creates new cluster declaring its own coordinate as the virtual center of cluster. Furthermore, a node periodically monitors the status of cluster that it is currently involved in order to migrate to another cluster whenever the utility is dropped below the threshold. The clustering process is presented in Algorithm 1 and described in detail as follows.

Upon joining the global network, a node computes its coordinates in 2-dimentional Euclidean space using Vivaldi network coordinate system. In order to avoid too frequent cluster migrations on the way toward the accurate position, the cluster process starts after the deviation of coordinate update becomes relatively small. As soon as node initiates the clustering process, it sends a request message including its coordinates to find the highest utility valued one of existing clusters through the multicast tree constructed by Map and Reduce functions based multicast tree builder. As the response for the request message, all the nodes calculate the utility of its cluster using the coordinate of the query origin node with Map function and compare the utility of its cluster to the utilities from its child nodes using Reduce function. Then, nodes send only the information of the highest utility valued cluster to parent node of the multicast tree. As a result, the query origin node receives only one response from a child node, and it can find the highest utility valued cluster with a lightweight comparison. It is worth noticing that Map and Reduce functions at the response phase help in reducing the number and the size of response messages. After joining the highest utility valued cluster, the node informs cluster members of its intent to join by multicasting a join message. By counting this type of message, cluster members keep the size of cluster up to date; they can also periodically query the cluster for membership count using a Map and Reduce functions multicast query within the cluster sub-overlay. If utility values of all the existing clusters do not satisfy the demand of the node (which means that utility values are less than the user-defined threshold), the node creates a new cluster.

Creating a new cluster is straightforward. A node generates a random clusterID ranging from 0 to 2^{160}-1 which is identical to the node address space of Symphony. Also, the node declares its current coordinates as the virtual center of cluster. With doing this, the node can subsequently reply to cluster requests from other nodes. After first joining the cluster, the node starts a utility monitoring task. Network

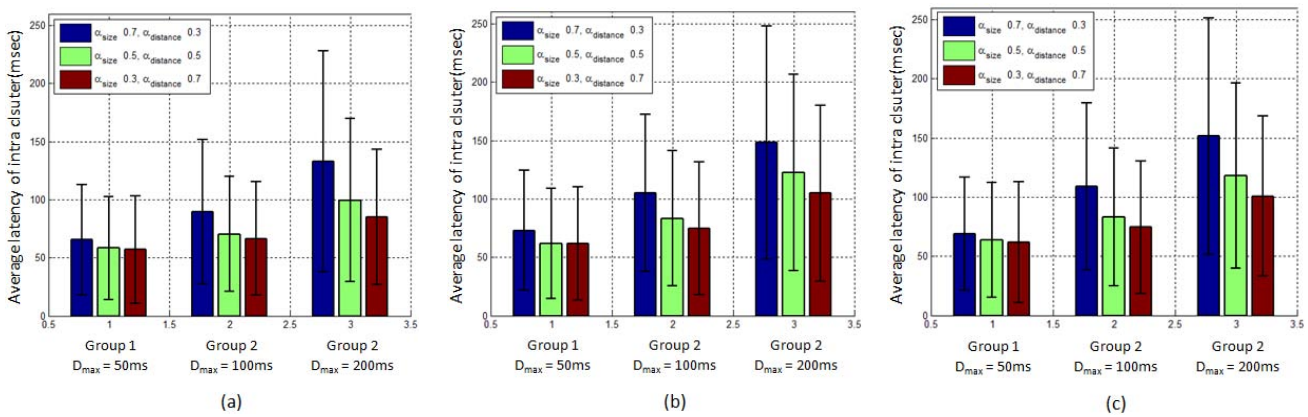

Figure 4. Average latency of intra cluster (a)S_{min} = 50 (b)S_{min} = 100 (c)S_{min} = 200

status has the possibility to be changed due to the churn and dynamic changes in traffic in the physical network. Such changes in the network result in the fluctuation of the utility value and nodes can take the potential advantage from the ability to migrate from the current cluster to another cluster. The self-managing cluster system needs to deal with this problem. The utility arbiter checks whether utility value is dropped below the threshold or not. To do this, nodes periodically calculate the distance to the virtual center of cluster and count the number of cluster members. If the utility value is dropped below the threshold, nodes repeat above cluster searching, joining or creating process.

Algorithm1 Self-organizing and managing cluster system

```
1: define  Parameters Dmax, Smin, αdistance, αsize, Uthreshold
2: if  Deviation of coordinate update < Dmax *1%
3:     Requests cluster information sending a query message
       via a multicast tree.
4:     Computes the utility with parameters Dmax, Smin, αdistance,
       αsize.
5:     if  There are clusters that the utility > Uthreshold
6:         Joins the highest utility valued cluster.
7:         Starts utility monitor task.
8:         if  The utility of the cluster that it currently joins <
               Uthreshold
9:             Migrates to another cluster repeating the process
               from line 3
10:        else
11:            Stays in current cluster.
12:    else
13:        Creates new cluster.
14: else
15:    Keeps updating the network coordinates
```

The main contribution of this work is that the clustering algorithm has a fully decentralized feature which does not rely upon any central units such as servers, super peers, cluster heads or landmarks; hence it is robust against the single point of failure. Because each node holds the information of its own cluster locally, it is unlikely that node failures or leave affects the performance of the whole system. Also, the destruction of cluster which does not have any cluster members at all is done without additional process. The departure of last cluster member means that the cluster does not exist in the network.

V. PERFORMANCE EVALUATION

In this section, we present results from simulation based analyses for SOLARE. In this evaluation, we use BruNet in event-driven simulation mode [15] and configure simulated latencies using King data set [3] with 1740 nodes. BruNet simulator uses simulated virtual time based upon an event-driven scheduler instead of real time. While existing simulators such as p2psim [8], NS2 [9], and netmodeler [10] are algorithm-oriented simulators which aim to evaluate algorithm validation, BruNet simulator has the ability to simulate the deployed system stack as well as algorithm using a specialized transport layer to avoid the overhead of using TCP or UDP on the host system. The specialized transport uses datagrams to pass messages between nodes, thus from an individual node's perspective, it is very similar to a UDP transport and can simulate both latency and packet dropping.

A. Intra-cluster latencies

Since D_{max} is the maximum distance to the virtual cluster center that a node expects when it searches a cluster to join, we can refer D_{max} as the radius of cluster. Therefore, upon joining a particular cluster, nodes are able to expect at most $2\times D_{max}$ of the latency between cluster members. To evaluate the performance of our clustering algorithm, we measure intra-cluster latencies – the latency between cluster members in the same cluster. We set D_{max} to 50ms, 100ms, and 200ms, and S_{min} to 50, 100, and 200, respectively. Also, $\alpha_{distance}$ is varied with 0.3, 0.5, and 0.7 (i.e., α_{size} is 0.7, 0.5, and 0.3). Figure 4(a), (b), and (c) show the average and standard deviation of intra cluster latency with various parameters setup. First of all, we observe that all the combinations of the parameters satisfy $2\times D_{max}$ in terms of the average latency of intra cluster. Specially, even in all cases that D_{max} is 50ms, the average latency is less than $2\times D_{max}$, 100ms. From this result, we note that even though nodes only measure the distance to the virtual center of cluster, it does not prevent nodes from getting neighbors who are mostly within D_{max}.

Second, by comparing three bars which have different colors in each group, we observe that the larger coefficient

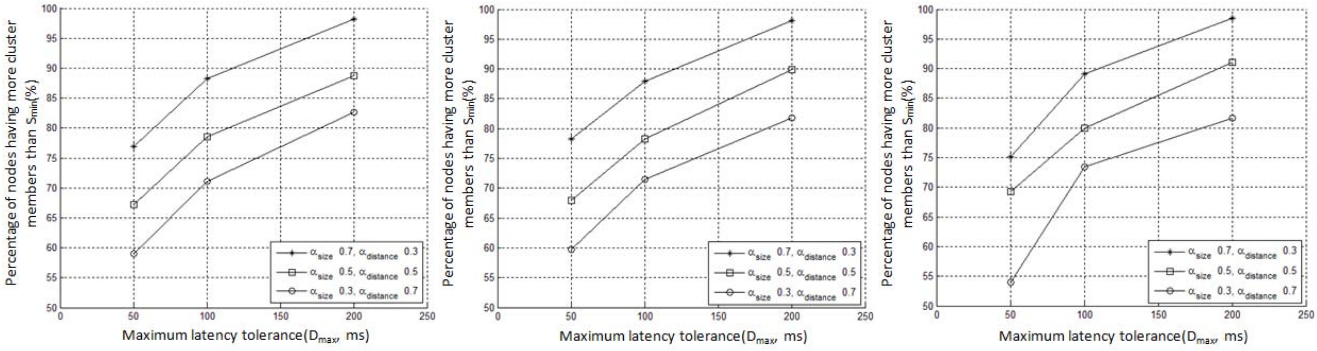

Figure 4. Percentage of nodes having more cluster member than S_{min} (a)S_{min} = 50 (b)S_{min} = 100 (c)S_{min} = 200

for distance, $\alpha_{distance}$ is, the smaller average intra-cluster latency is. In the case that D_{max} is 50ms in figure 4(a), the average intra-cluster latency with $\alpha_{distance}$ of 0.7 is reduced by 11% and 2% compared with $\alpha_{distance}$ of 0.3 and 0.5 respectively. Such this difference becomes much clearer as D_{max} increases showing 35.9% and 14.4% of decrease compared the case of $\alpha_{distance}$ of 0.7 with the cases of 0.3 and 0.5 and D_{max} of 200ms. The standard deviation when $\alpha_{distance}$ is set to 0.7 is also decreased by 38.7% and 17.2% compared to $\alpha_{distance}$ of 0.3 and 0.5 in the group1 of figure 4(a). Furthermore, we observe that the minimum preference for cluster size, S_{min} does not affect the average intra-cluster latency by observing the same pattern of average latency regardless of S_{min} in figure 4(b) and (c) even though there is a slight trend upward of average latency as S_{min} increases.

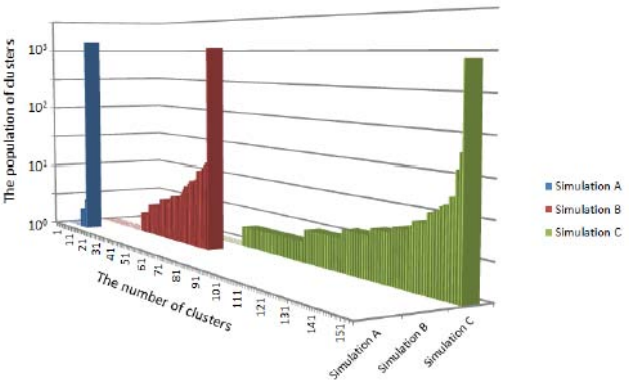

Figure 5. The number of created clusters and the population of clusters
Simulation A: Dmax 200ms, Smin 200, αdistance 0.3, αsize 0.7
Simulation B: Dmax 100ms, Smin 100, αdistance 0.5, αsize 0.5
Simulation C: Dmax 50ms, Smin 50, αdistance 0.7, αsize 0.3

B. *The number of cluster members*

In addition to intra-cluster latencies, we measure the cluster size which takes the other part of total utility function. To measure the cluster size, we took snapshots of the number of created clusters and cluster members for each node when all nodes join a cluster. Due to the limitation of space, the results from only three simulations of all the simulations we performed are represented in Figure 5 depicting the number of created clusters and the population of clusters. Firstly, in simulation A, we observe that only 16 clusters are created and 98.5% of nodes are included in only one cluster. In fact, for this simulation set, D_{max} and α_{size} are set to 200ms and 0.7 which is most likely to organize the largest cluster of all the simulations. However, as D_{max} and α_{size} decrease, the number of created cluster increases and the population is also distributed throughout the created clusters. Particularly, for the case of simulation C in which nodes require relatively small cluster size but much closer cluster members, 155 clusters are created and nodes join clusters more evenly than simulation A or B. Furthermore, each result from three simulations shows one big cluster including at least 58% of nodes which is caused by the type of utility function for cluster size we used. From Section IV-A, recall the baseline of utility function for cluster size in such that nodes prefer larger cluster size without an upper limit. If we consider another type of utility function which defines upper limit of cluster size, one major cluster may disappear and the population of cluster should become more even.

To summarize the evaluation of quantitative performance, as shown in Figure 6(a), (b) and(c), we consider the percentage of nodes which have more cluster member than S_{min}. Figure 6(a) shows that as D_{max} and α_{size} become larger, more nodes can obtain same or more cluster members than S_{min}. Although, indeed, only 54% of nodes have more cluster members than S_{min} with 50ms of D_{max} and 0.3 of α_{size}, in the case with 200ms of D_{max} and 0.7 of α_{size}, almost 98% of nodes satisfy the minimum preference for cluster size, S_{min}. Regardless of setting of S_{min}, the same trend is observed in Figure 6(b) and (c) where S_{min} is set to 100ms and 200ms with more than 97% and 98% of nodes satisfied with S_{min}. Thus, we can confirm the correctness of our clustering system with the result of the number of cluster members.

C. *Adaptability to dynamic network conditions*

Finally, we discuss the adaptability of SOLARE. Network status can be changed dynamically due to the node joining, leaving, and the change of network latencies. Therefore, nodes should adapt to dynamic network status for the ability to self-manage the cluster system. After a node joins the cluster, it periodically updates the utility for its cluster, so if utility is dropped below the threshold, a node repeats the cluster joining procedure to migrate into another high utility-valued cluster. To evaluate the adaptability of

this system, we observe the percentage of the number of nodes which need the cluster migration and average utility value over the entire network. Instead of directly modifying network latency at the simulation environment, we assume that new joining in global network can cause the change of nodes' position in the network coordinate space. Figure 7(a) shows the percentage of the number of nodes which need to migrate from current cluster into another cluster due to the negative utility value. Because 1740 nodes sequentially join in global network with 500 milliseconds of interval in our simulation scenario, all nodes complete joining the global network around 15 minutes after starting the simulation. In the case of simulation A where D_{max} and S_{min} are set to 50ms and 50 respective, cluster migrations happen most frequently. In fact, because parameters for simulation C imply the smallest cluster, it is most likely that cluster migrations occur by slight change of the coordinate. However, after all nodes join the global network and step into correct position, the occurrence of migrations decreases and the percentage of nodes which migrate to another cluster is dropped below 5% after 30 minutes from starting the simulation.

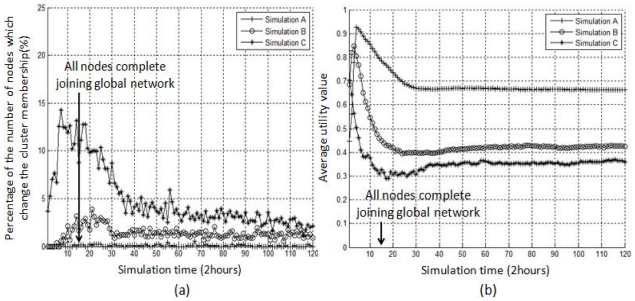

Figure 6. (a) Percentage of the number of nodes which need to migrate into another cluster (b) Average utility value. Setting parameters for simulation A, B, and C are identical to Figure 5.

Such a pattern of the occurrence of migrations is observed in the case of simulation B with D_{max} = 100ms and S_{min} = 100, but the percentage of nodes which migrate into another cluster is less than simulation C because of the parameters set which means bigger cluster than the case of simulation C. Especially, in simulation A where the biggest cluster is set, nodes barely migrate, because the change of nodes' coordinates occur within a cluster. Also, we can observe from Figure 7(b) that after the percentage of nodes which migrate is saturated or dropped drastically, the average utility value over the entire network is also saturated which means that the majority of nodes place at correct position and stay in current cluster. Note that the difference of the utility value saturated can be inferred from Figure 5 and 6. Because most of nodes with simulation A, about 98.5%, are in the same cluster, it is easy to gain higher utility value for cluster size than other cases. On the other hand, in the case of simulation C, nodes are distribute among many clusters, and only 59% of nodes have more cluster members than S_{min}. Consequently, it results in the lowest utility value for cluster size and total average utility value.

VI. CONCLUSIONS

In this paper, we introduced SOLARE to provide a proximity-aware topology by showing the ability to self-organize and self-manage the clustering system. All participants compute themselves the utility for existing clusters and join highest utility valued cluster. On the other hand, if there is no cluster whose utility is greater than the user defined threshold, nodes create a new cluster declaring their own coordinate as the virtual center of cluster. To establish utility functions, we select the distance to the cluster and the size of cluster as utility properties. Furthermore, our cluster system provides the adaptability for dynamic network status while each node monitors the cluster that it is currently joining. After describing the self-organizing and managing clustering procedure, we evaluated its performance in terms of latencies of intra cluster and the number of cluster members. Based on our evaluation, we confirmed that setting of parameters, which form utility functions, can control the properties of clusters such as latencies of intra cluster and the population of clusters. Additionally, by measuring the percentage of the number of nodes which need to migrate into another cluster and average utility value, we presented the adaptability of our system.

As a future work, we can extend this work by considering different Internet measures like bandwidth as mentioned in the introduction. Unlike using latency to express the network proximity, higher bandwidth is considered as closer distance and the distance for two end hosts is the lowest bandwidth link, we should adopt a different coordinate space rather than the Euclidean distance space. Especially, our future work can be inspired by Sequoia which uses the tree metric to predict the end-to-end bandwidth.

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation under Grant No.0855123. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Jeffrey O. Kephart, and Rajarshi Das, "Achieving Self-Management via Utility Functions," IEEE Internet Computing, Vol. 11, No. 1, pp. 40-48, Jan. 2007.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," In Proc. Of ACM SIGCOMM Conference, Portland, Oregon, USA, Aug. 2004.

[3] K.P. Gummadi, S. Saroiu, and S.D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," In Proc. Of SIGCOMM IMW2002, Marseille, France, Nov. 2002.

[4] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Proc. Of the 6th Conference on Symposium on Operating Systems Design & Implementation, San Francisco, CA, USA, Dec. 2004.

[5] Tae Woong Choi and P. Oscar Boykin, "Deetoo: Scalable unstructured search built on a structured overlay," In Proc. Of IPDPS Workshops, Atlanta, Georgia, USA, Apr. 2010.

[6] P.O. Boykin and et al., "A symphony conducted by brunet," http://arxiv.org/abs/0709.4048. 2007.

[7] G. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," In Proc. Of the 4th USENIX symposium on Internet Technologies and Systems, Seattle, Washington, USA, Mar. 2003.

[8] A simulator for peer-to-peer protocols, http://www.pdos.lcs.mit.edu/p2psim/index.html.

[9] The Network Simulator, NS2, http://www.isi.edu/nsnam/ns/.

[10] Netmodeler: a c++ package for analyzing complex networks, http://boykin.acis.ufl.edu/wiki/index.php/Netmodeler.

[11] V. Ramasubramanian, D. Malkhi, F. Kuhn, I. Abraham, M. Balakrishnan, A. Gupta, A. Akella, "A Unified Network Coordinate System for Bandwidth and Latency," Technical Report MSR-TR-2008-124, Microsoft Research, Sep. 2008.

[12] E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," In Proc. Of INFOCOM, New York, NY, USA, Jun. 2002.

[13] N. Hu, L.E. Li, Z.M. Mao, P. Steenkiste, andJ. Wang, "A Measurement Study of Internet Bottlenecks," In Proc. Of INFOCOM, Miami, FL, USA, Mar. 2005.

[14] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das, "Utility Functions in Autonomic Systems," In Proc. Of 1st International Conference on Autonomic Computing, New York, NY, USA, May. 2004.

[15] David I. Wolinsky, K.Y. Lee, P. Oscar Boykin, and Renato Figueiredo, "On the Design of Autonomic, Decentralized VPNs," In Proc. Of 6th International ICST Conference on Collaborative Computing: Networking, Applications and Worksharing, Chicago, Illinois, USA, Oct. 2010.

[16] David M. Chess, Alla Segal, Ian Whalley, and Steve R. White, "Unity: Experiences with a Prototype Autonomic Computing System," In Proc. Of 1st International Conference on Autonomic Computing, New York, NY, USA, May. 2004.

[17] Terence Kelly, "Utility-Directed Allocation," In Proc. Of 1st Workshop on Algorithms and Architectures for Self-Managing Systems, San Diego, CA, USA, Jun. 2003.

[18] Rajarshi Das, Jeffrey O. Kephart, Jonathan Lenchner, and Hendrik Hamann, "Utility-Function-Driven Evergy-Efficient Cooling in Data Centers," In Proc. Of 7th International Conference On Autonomic Computing, Washington, DC, USA, Jun. 2010.

[19] R.A. Ferreira, S. Jagannathan, and A. Grama, "Locality in structured peer to peer networks," Journal of Parallel and Distributed Computing, Vol. 66(2), pp. 257-273, 2007.

[20] Francois Cantin, Bamba Gueye, Mohamed Ali Kaafar, and Guy Leduc, "A Self-Organized clustering Scheme for overlay networks," In Proc. Of 3rd International Workshop on Self-Organizing Systems, Vienna, Austria, Dec. 2008.

[21] Paul de Grandis, and Giuseppe Valetto, "Elicitation and utilization of application-level utility functions," In Proc. Of 6th International Conference On Autonomic Computing, Barcelona, Spain Jun. 2009.

[22] F. Le Fessant, S. Handurukande, A.M. Kermarrec, and L. Massoulie, "Clustering in Peer-to-Peer File Sharing Workloads," In Proc. Of 3rd International Workshop on Peer-to-Peer Systems, San Diego, CA, USA, Feb. 2004.

[23] P. Karwaczynski, "IP-based clustering for peer-to-peer overlays," Journal of Software, Vol. 2, No. 2, pp 30-37, 2007

[24] Sajid Hussain, and Abdul W. Matin, "Hierarchical Cluster-based Routing in Wireless Sensor Networks," In Proc. Of 5th International Conference on Information Processing In Sensor Networks, Nashville, TN, USA, Jul. 2006.

[25] Adeel Akhtar, Abid Ali Minhas, and Sohail Jabbar, "Energy Aware Intra Cluster Routing for Wireless Sensor Networks," International Journal of Hybrid Information Technology, Vol. 3, No. 1, Jan. 2010.

[26] Qing Cao, Tarek Abdelzaher, Tian He, and Robin Kravets, "Cluster-Based Forwarding for Reliable End-to-End Delivery in Wireless Sensor Networks," In Proc. Of INFOCOM, Anchorage, AK, USA, Mar. 2007.

[27] Tian Bu, and Don Towsley, "On Distinguishing between Internet Power Law Topology Generators," In Proc. Of INFOCOM, New York, NY, USA, Jun. 2002.

[28] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, andH. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Trans. Network, Vol. 11, No. 1, pp. 17-32, 2003.

[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," In Proc. Of ACM SIGCOMM 2001, San Diego, CA, USA, Aug. 2001.

[30] B.Y. Zhao, L. Huang, S.C. Rhea, J. Stribling, A.D. Joseph, and J.D. Kubiatowicx, "Tapestry: A global-scale overlay for rapid service deployment," IEEE J-SAC, Vol. 22, No. 1, pp. 41-53, Jan. 2004.

[31] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella, "On the Treeness of Internet Latency and Bandwidth," In Proc. Of 11th International Joint Conference on Measurement and Modeling of Computer Systems, Seattle, WA, USA, Jun. 2009.

[32] Kyungyong Lee, Tae Woong Choi, Arijit Ganguly, David I. Wolinsky, P. Oscar Boykin, and Renato J. Figueiredo, "Parallel Processing Framework on a P2P System Using Map and Reduce Primitives," In Proc. Of 8th International Workshop on Hot Topics in Peer-to-Peer Systems, Anchorage, AK, USA, May. 2011.

[33] David I. Wolinsky, Pierre St. Juste, P. Oscar Boykin, and Renato J. Figueiredo, "Addressing the P2P Bootstrap Problem for Small Overlay Networks," In Proc. Of 10th IEEE International Conference on Peer-to-Peer Computing, Delft, Netherlands, Aug. 2010.