Distributed, Self-Stabilizing Placement of Replicated Resources in Emerging Networks *

Bong-Jun Ko Dan Rubenstein
Department of Electrical Engineering, Columbia University
New York, NY
{kobj, danr}@ee.columbia.edu

Abstract

Emerging large scale distributed networking systems, such as P2P file sharing systems, sensor networks, and ad hoc wireless networks, require replication of content, functionality, or configuration to enact or optimize communication tasks. The placement of these replicated resources can significantly impact performance. We present a novel self-stabilizing, fully distributed, asynchronous, scalable protocol that can be used to place replicated resources such that each node is "close" to some copy of any object. We describe our protocol in the context of a graph with colored nodes, where a node's color indicates the replica/task that it is assigned. Our combination of theoretical results and simulation prove stabilization of the protocol, and evaluate its performance in the context of convergence time, message transmissions, and color distance. Our results show that the protocol generates colorings that are close to the optimal under a set of metrics, making such a protocol ideal for emerging networking systems.

1. Introduction

Emerging large scale distributed networking systems, such as P2P file sharing systems, wireless ad-hoc and sensor networks, and utility computing systems utilize replication of resources, information, and jobs at participating network nodes to increase the overall performance of the networked system. For example, P2P file sharing applications [6] replicate content at nodes throughout the network, reducing the expected retrieval time. In utility computing applications such as Grid Computing [5], a computing task replicated among multiple computing nodes not only further extends the distribution of the computation load, but also reduces the time spent transporting tasks to nodes. In multi-channel wireless networks, the capacity of the network is often increased by assigning channels to nodes such that the distance between nodes assigned to the same channel is maximized. In all these systems, the decision of where to replicate particular items, whether the items take the form of information, tasks, or resources, has a tremendous impact on the performance of the entire system.

While the use of replication in the different environments described above addresses a vastly different set of performance issues, we make the observation that at an abstract level, the placement objectives are remarkably similar. In particular, it is important either to place different items in the vicinity of each node or to place the identical items as far away from each other as possible. We note that a placement that achieves one of these objectives also does it well for the other, i.e., if the distance between two identical resources is increased, it gives room for different resources to be placed in nearby points, and vice versa.

There are several properties of emerging networks that complicate the design of a protocol to place or configure replicated resources:

- The number of participating nodes will often be in the thousands or millions.
- Participation will be dynamic, with nodes joining and leaving the network at unpredictable times, making it difficult to elect "leader" nodes that make decisions for large portions of the topology.
- Clocks will not be perfectly coordinated, making it difficult or impossible to perform synchronized actions among large collections of nodes. Unless carefully constructed, in such an environment, a protocol can enter a livelocked or deadlocked state.
- Transmissions across large distances will incur high bandwidth costs or will increase the time needed to decide where to place the replicas.

For these reasons, these networks cannot utilize protocols that implement centralized (leader-oriented) computation, require synchronized messaging, or require communications between distant nodes.

* This material was supported in part by the National Science Foundation under Grant No. ANI-0117738 and CAREER Award No. ANI-0133829, and by a gift from Lucent Technologies. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

This paper describes a distributed, hierarchy-free, self-stabilizing protocol that performs placement of replicated resources. We address the fundamental problem of placing multiple copies of resources within the network, in which each node must hold some resource, such that an arbitrarily chosen resource is reachable over a short path from any starting point in the network, and that there are large distances between identical copies. We develop a protocol that exhibits several desirable properties:

- Self-configuring: no third party is used to coordinate placement.
- Fully distributed: all nodes are assigned an identical set of tasks to aid in the decision of the placement.
- Scalable: information exchange is performed within localized areas.
- Asynchrony: no clocks are required to sequence or coordinate the operations at nodes.

We explore this problem in the context of an arbitrary graph, where each node is colored with one of k colors, where each color represents a resource class. The protocol operates by having each node continually change its color in a greedy manner to maximize its own distance to a node of the same color. To this, we add a novel mechanism that avoids race conditions to ensure that the protocol stabilizes in asynchronous communication environments indicative of these emerging networks.

We use both theoretical analysis and simulation to evaluate the performance of our protocol. Our theoretical analysis proves that the protocol always converges to a fixed coloring. This analysis also formally bounds the distance that must be traveled from a node to an arbitrarily chosen color within a factor of three of what can be achieved in an optimal coloring. The results of discrete-event simulation demonstrate that for randomly generated graphs, our protocol is scalable in that the convergence time and the number of protocol messages grow slowly with the number of nodes in the network, and that the coloring is on average much closer to the optimal allocation than the theoretical upper bound.

The rest of the paper is organized as follows. In Section 2, we present the network model and formally state the coloring problem. Section 3 describes the distributed protocol, and Section 4 discusses the issues involved in its extension and some potential applications. The convergence to a fixed coloring and theoretical bounds on performance are presented in Section 5. Section 6 presents performance evaluation results of the protocol observed through simulation. Section 7 briefly reviews related work, and we conclude the paper in Section 8.

2. Model and Problem Formulation

In this section, we introduce the network model and the optimization goals in the context of that model. Throughout the description of the model and the protocol, we assume the protocol messages exchanged between a pair of nodes are reliably delivered in the order they were transmitted, using some existing reliable transfer mechanism relevant to the particular network system, for example, hop-by-hop TCP connections.

We view a distributed networking system as an undirected, connected graph, $G = (N, E)$, where $N = \{1, 2, \cdots, n = |N|\}$ is the set of (numbered) nodes of the graph and E is the set of edges. Each node n is assigned one of k **colors**, $c_1, \cdots, c_k$; each color represents a specific set of resources being replicated, where a node's color can change with time. We assume that the value of k is determined in advance and remains fixed for the duration of the execution of the protocol. Also, each edge $e \in E$ is assigned an arbitrary, non-negative weight. The **distance** between two nodes x and y, $d(x, y)$, is the sum of the weights of edges along a shortest path between x and y. The **color** of node x at time t is denoted $C_t(x)$.

Node x's **distance to color** c_i at time t, $d_t(x, c_i)$ is defined as the distance at time t between x and the closest node to x of color c_i i.e., $d_t(x, c_i) = \min_{1 \le j \le n}\{d(x, j) : C_t(j) = c_i\}$. Note that $d_t(x, C_t(x)) = 0$, and also that if no node is assigned to a particular color c_i, then $d_t(x, c_i) = \infty$. We define $\delta_t(x)$ to be the distance at time t between x and the closest other node of the same color, i.e., $\delta_t(x) = \min_{1 \le j \le n}\{d_t(x, j) : C_t(j) = C_t(x), j \neq x\}$.

In the context of this model, there are several different objective functions whose minimization or maximization would yield desirable placements of replicated resources depending on the design principle. For example, one could argue that the objective function is to:

- minimize $\max_x \max_{i \neq C_t(x)} d_t(x, c_i)$,
- minimize $\sum_x \sum_{i \neq C_t(x)} d_t(x, c_i)$, or
- maximize $\sum_x \delta_t(x)$.

The first two objective functions are used to place colors so that each node has a short distance through the graph (i.e., network) to reach any other color. Such placement is useful when it is desirable to have replicated contents or tasks (represented by colors) near to any arbitrarily chosen point in the network. With the last objective, we would like to place colors so that each node is far from nodes of the same color. Such placement is useful when nearby placement of identically-configured nodes causes a conflict, such as assigning two nodes to transmit upon the same wireless channels. Note that these two types of objectives fit naturally together: placing the same color far away makes room for other colors nearby.

Unfortunately, the above optimization problems are NP-hard[1]. Hence, we focus on developing a distributed protocol that computes approximations to these problems.

3. The Asynchronous Distributed Coloring Protocol

In this section, we describe our Asynchronous Distributed Coloring (ADC) protocol and the procedure that each node performs. Our focus in this section is the development of a decentralized protocol with which each node learns of its nearby nodes' colors and how it selects and changes its own color without causing race conditions.

Since up-to-date color information about a node y takes time to reach node x, it will be important to distinguish between the perception that node x has about the current coloring and the actual coloring of the graph. We use $\hat{d}_t(x, c_i)$, and $\hat{\delta}_t(x)$ to denote the distances to the closest node of color c_i and $C_t(x)$ respectively as **perceived by node** x at time t based on the most recent coloring information it has received. The **real** distances (i.e., as perceived by an oracle that knows every node's actual color at time t) remain denoted by $d_t(x, c_i)$ and $\delta_t(x)$.

We say that a node x is **locally stable** at time t if $\hat{d}_t(x, c_i) \leq \hat{\delta}_t(x)$ for all $1 \leq i \leq k$, i.e., x perceives that no color is further from it than the closest node whose color matches its own. Otherwise, the node is **locally unstable**. If $d_t(x, c_i) \leq \delta_t(x)$ for all $1 \leq i \leq k$, then at time t, x is said to be **stable**, and is otherwise **unstable**. A graph coloring is said to be **stable** when all nodes in the graph are stable.

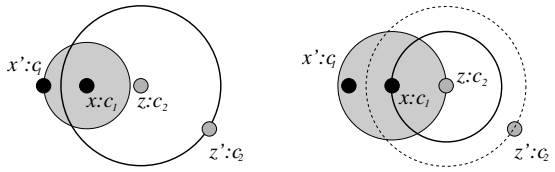
3.1. Coloring Rule

We begin by describing the process used to adjust the color of a node. Each node is responsible for selecting its own color. We assume for now that, with the color change propagation protocol described in the next subsection, each node continually receives updated information of the colors of nearby nodes. More specifically, at any time t, a node x keeps $\hat{d}_t(x, c_i)$ for all colors c_i, $1 \leq i \leq k$ and $\hat{\delta}_t(x)$ and applies the following simple color changing rule.

The color change rule : At any time t, a locally stable node keeps its color fixed. However, any locally unstable node will change its color to a color c_i that satisfies $\hat{d}_t(x, c_i) \geq \hat{d}_t(x, c_j)$ for all $1 \leq j \leq k$.

In other words, a node seeks to change its color to c_i if it perceives that the closest node of color c_i is further from it than the closest other node of its current color. Changing color in this manner is desirable in that it increases $\hat{\delta}_t(x)$, decreases $\sum_i \hat{d}_t(x, c_i)$ (i.e., average distance to a color), and does not increase but often decreases $\max_i \hat{d}_t(x, c_i)$ (i.e., distance to the furthest color).

Figure 1 provides an illustrative example of a node x that starts as color c_1 at time t and completes its change to color c_2 at time $t' > t$. Nodes x and x' are initially colored c_1 and nodes z and z' are initially colored c_2. In this example we use Euclidean distance to indicate the distance between nodes. In Figure 1(a), x' is the closest node to x of color c_1, and z is the closest node to x of color c_2. Since $\delta_t(x) < d_t(x, c_2)$, x can increase its distance to a node of the same color by changing its color from c_1 to c_2. The result is depicted in Figure 1(b). We see that the shaded region around x (i.e., the minimum distance to the same color) has grown. Note that the color change by x is not necessarily favorable for other nodes in the graph. For instance, the change by x to color c_2 substantially shrinks the distance from node z to a node of its same color, c_2. Because of this, it is not immediately obvious that repeated application of our coloring rule will lead to a stable coloring. This stability is proven in the next section.



(a) Before: time t (b) After: time t'

Figure 1. x changes color from c_1 to c_2

An important characteristic of this coloring rule is that a node only needs the knowledge of the distance to the *closest* node of each color, i.e., the required information exchange is often available within a local region.

We now describe the messages exchanged by neighboring nodes to implement the ADC protocol. The messages exchanged between neighbors perform two essential functions. First, nearby nodes exchange information that describes the color configuration of nodes within a local area. Second, nodes seek approval from nearby neighbors before changing their colors to guard against a small set of changes which, if performed by nodes simultaneously, could livelock or deadlock the protocol before it reaches a stable configuration.

3.2. Color Propagation

A node needs to be continually updated about any changes in coloring at other nodes that affect its minimum distance to a color, since its own "desired" color is based on these values. We have devised a distributed method we call *Colored Bellman-Ford* or CBF for short, which exchanges the necessary information among nodes. Our description here assumes that the reader is familiar with the Bellman-Ford algorithm[4].

As in the Bellman-Ford algorithm, each node in CBF learns the distances to other nodes by exchanging a distance **update** message with each of its neighbor nodes. The difference is, while Bellman-Ford updates each node with the

distances to all other nodes in the network, CBF updates information only about the distances to the closest nodes of all other *colors*, localizing the scope of the propagation of **update** messages. In CBF, a node maintains a distance vector table, storing information about the closest node of each color. When a node running CBF receives an *update* from a neighboring node, like Bellman-Ford, it updates its (color) distance table in the respective entry and forwards *update* messages to all of its neighbors containing shortest path information (length and node ID) for any color to which its shortest distance has changed.

The one limitation of the above method is that a node x will often not learn about the closest other node of the same color, $C_t(x)$, when its neighbors all point to x as their closest node of that color. This problem is resolved by having each node store information pertaining to the **two** closest nodes of each color that can be reached through each neighbor. The information about these two closest nodes is also forwarded to neighbors when an update changes either member of this closest pair, and hence a node's table entry for the same color must contain at least one node other than itself.

Clearly, CBF will continue to send updates as long as nodes continue to change their colors, since color changes affect the distances stored in the distance tables. We have proven the following result which will be used later to prove that our distributed protocol stabilizes to a coloring. The proof is not included due to the lack of space, and can be found in a technical report version of this paper[11].

Claim 1 *When nodes cease to change their colors, i.e.,* $\exists t, \forall x, \forall \tau > t, C_\tau(x) = C_t(x)$*, CBF will terminate transmitting messages, and each node* x *will have accurate information about the distance to and the identity of the two closest nodes of each color* $c_i \neq C(x)$*, and of the closest node other than itself of color* $C(x)$.

CBF, coupled with the color-changing rule, allows nodes to change their colors and deliver their new color information to those nodes whose own color decisions are affected by that information. However, when nodes are actively changing their colors, the lag in time it takes for a node to learn about another node's color change can lead to anomalous situations in which the procedure does not stabilize. A simple illustration of this phenomenon involves a graph to be colored with two colors, where all nodes are initially colored c_1, and propagation delays between all pairs of nodes takes one unit of time. Since each node's distance to c_2 is infinite, all nodes simultaneously change to c_2 and propagate their change information around the graph. Upon learning in the next time unit that their nearest neighbors are now colored c_2, all nodes return to color c_1, and the process repeats indefinitely.

Our goal here is to prevent this anomalous behavior, but we would like to permit as many locally unstable nodes to change their colors simultaneously when these simultaneous changes cause no such conflict.

3.3. Asynchronous Color Change Procedure

Coping with the asynchrony and handling possible anomalous behaviors is done by a 3-way handshake exchange using four types of protocol messages: *request*, *accept*, *reject*, and *decision*. These messages are used to ensure that the graph will converge to a stable coloring in which all nodes are stable. Before describing how these messages are used, we begin by looking at a monotonicity property, one of the critical properties that our protocol will satisfy to prove convergence.

3.3.1. Monotonicity Property Later in the paper, we will show that the protocol produces a stable coloring as long as $\delta_t(x)$ increases whenever node x changes its color. For $\delta_t(x)$ to increase, it is sufficient to change node x from c_1 at time t to c_2 at time t' when the following two conditions hold:

- There exists a node (other than x) *at* distance $\hat{\delta}_t(x)$ from x that is color c_1 at time t', ensuring that $\delta_t(x) \leq \hat{\delta}_t(x)$.
- There is no node of color c_2 within distance $\hat{\delta}_t(x)$ from x at time t', ensuring that $d_t(x, c_2) > \hat{\delta}_t(x)$.

Clearly, if these two conditions are satisfied, then so is $\delta_t(x) < d_t(x, c_2)$. Hence, by changing to color c_2 at time t', we have $\delta_{t'}(x) > \delta_t(x)$. Note that x's decision uses only perceived distance information since this is the only information that is available to the node.

Now let us present our distributed protocol, beginning by describing at a high level how the messages are passed between nodes .

3.3.2. Protocol Messages When x of color c_1 "perceives" (by CBF) that it can change its color to c_2 by the coloring rule, i.e., when $\hat{\delta}_t(x) < \hat{d}_t(x, c_2)$, it broadcasts a **request** message to all nodes whose distance lies within $\hat{\delta}_t(x)$ of x. We refer to this set of nodes as x's *time t disk*. [1] The request message states x's desire to change from color c_1 to color c_2, and requests that one of the nodes in x's time t disk maintain color c_1 and none of these nodes change to color c_2.

A node y that receives a **request** message from x responds either with an **accept** or **reject** message as follows. Node y immediately sends x a **reject** message if y satisfies either of the following properties:

- y is of color c_2.
- y is the node that x "thinks" is of color c_1, but y is in fact no longer of color c_1.

1 There are several ways to scalably implement scoped broadcasting of messages within a node's time t disk and the corresponding feedback mechanism. We do not cover this issue here, but more discussion can be found in [11].

Note that the former of the above two conditions, under which y rejects x's request, means that x's perception of having no nodes of color c_2 in its disk was incorrect, such that color c_2 is at least as close as the closest other node of c_1 that is known by x. The latter condition means that the node that x depended on as the node of color c_1 is no longer of that color, hence c_1 may be further away from x than x had anticipated.

If neither of these properties hold, y sends x an **accept** message. If y accepts x's request to change from c_1 to c_2, then y is prohibited from changing from c_1 or changing to c_2 until it receives x's **decision** message.

Now, if any node rejects x's request, x aborts the color change. If all nodes accept x's request, then x changes its color. Whether x changes its color or aborts, it sends a **decision** message to the same set of nodes to which it had sent a request, removing the color-change restriction that was placed on these other nodes.

3.3.3. Handling Deadlock/Livelock Network propagation delays are problematic when trying to coordinate message requests within the network. In particular, we must address the possibility that two nodes, x and y, each wishing to change color, request one another to hold their colors fixed until further notice. Nodes cannot simply accept the other's change requests when such conflict arises, as this could lead to a livelocking phenomenon, causing both to keep changing their colors. Both cannot simply reject the other - this leads to a livelock, preventing one another from changing to a "better" color. If both hold each other's request until they receive **accept** or **reject** to their own requests, a deadlock situation will arise. Also, it will often be the case that the distances over which x's and y's messages travel will differ. In particular, it could be the case that $\hat{\delta}_t(x) < d(x,y) < \hat{\delta}_t(y)$. Hence, even applying a simple tie-breaking rule between x and y is difficult, because one of the nodes may not even be aware of the conflict.

We address this dilemma by having each node transition between two states: **STALLED** and **MOVING**. Each node also maintains three sets that hold requests that it receives from other nodes: **stalling**, **master**, and **slave**. These messages are removed from a set only after the outcome of the request is known (i.e., the request was rejected, or the response corresponding to the request was received). Furthermore, we assign arbitrary, unique identifiers that can be used to impose a well-defined ordering among nodes. We write $i > j$ to indicate that node i is ordered before node j. Figure 2 presents the state diagram relevant to these procedures, which are described as follows.

Initially, a node is in the STALLED state. Upon sending a request, a node transitions to the MOVING state, and transitions back to the STALLED state immediately after sending a decision message corresponding to the request that caused it to enter the MOVING state.

When in the STALLED state, a node x places in its stalling set any request that it accepts. x cannot transition

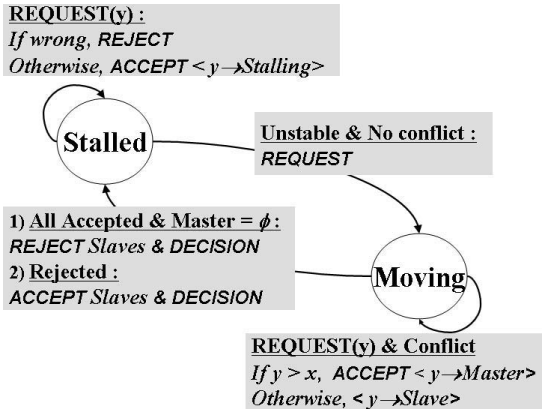


Figure 2. State diagram for handling protocol messages

to the MOVING state (i.e., cannot issue a change request) if its stalling set has an entry that conflicts with x's desired color change. In particular, x cannot change from color c_1 to color c_2 if there is a request in its stalling set asking to change to color c_2, or there is at least one node y that depends on x to be color c_1.

When in the MOVING state (x has issued a request), x places a node y's request in its stalling set if the request does not conflict with its own intended color change. However, if y's request does conflict, it differentiates the request based on the order of x and y as follows. If $x > y$, then x places y's request in its slave set without accepting or rejecting the request. On the other hand, if $x < y$, then x places y's request in its master set and immediately accepts the request. x will not change its color while its master set is nonempty. A message in x's master set is deleted when x receives the decision regarding the message.

If x's request is accepted by all nodes and its master set is empty, it rejects (and removes) all messages in the slave set, sends a decision message, changes its color, and transitions to the STALLED state. If x's request is rejected, it accepts and removes all messages in its slave set, transfers all messages in the master set into the stalling set, and transitions to the STALLED state without changing its color.

This prioritized STALLED/MOVING procedure prevents race conditions by having requests from lower-priority nodes "frozen" until nodes of higher order complete their decision of whether or not to change. The protocol does not deadlock because the node with the highest order that is in the MOVING state will never be "frozen" by another node. A formal proof of this claim is presented in Section 5.1.

To summarize the ADC protocol, nodes use the distributed CBF method to exchange updates with neighbor nodes of nearest colorings. When a node wishes to

change its color, it broadcasts and receives messages about this color change that respectively travel no further than the distance to the closest node perceived to be the same color. Since these procedures occur within localized regions and have a small set of restrictions that prevent two nodes from simultaneously changing color, many of the color changes can occur simultaneously. This massive parallelism significantly reduces the time to reach a stable coloring in comparison to a protocol that permits only a single color change at a time.

4. Discussion

In this section, we present some variations and extensions of the graph coloring problem that may be of use in practice. For each extension we can show that the ADC protocol will still converge. Next, we discuss practical applications, for which our protocol can offer useful solutions either directly or via one of the forementioned extensions.

4.1. Extensions and Variations

4.1.1. Hardwired nodes We can relax the assumption that all nodes participate in the ADC protocol and allow some nodes to fix their colors during the execution of the protocol. This is useful, for instance, in networks where there is an original source of information from which replicas are copied, and the point of origin of the information retains the original copy.

4.1.2. Assigning multiple colors to a node Each node can be assigned an arbitrary number of colors that it can store. This can be done by mapping the graph to another graph where a node that can store ℓ colors in the original graph is mapped to an ℓ-clique of nodes that can each store one color, where the edges between nodes in the ℓ-clique have length zero. Our coloring protocol can be immediately used without modification if each node with ℓ colors in the original graph performs the procedure for ℓ independent nodes in the converted graph. The proof and all supporting results follow trivially.

This can be of use in practice, when some node has larger capacity than the others, for example, nodes with big storage, a wireless node supporting multiple channels, etc.

4.1.3. Variable Density Coloring Each color c_i can be assigned a density, β_i such that a node y of color c_1 is considered to be "closer" to x than a node z of color c_2 whenever $\beta_1 d_t(y,x) < \beta_2 d_t(z,x)$. Nodes of colors with larger density will more densely populate the graph. For instance, consider coloring a given graph with two colors c_1 of density ϵ and c_2 of density $1/\epsilon$ as $\epsilon \to 0$. Having a single node x of color c_1 in the graph is sufficient, since no matter how physically far this node is from another node y in the graph, $\beta_1 d(x,y)$ is infinitesimally small.

4.2. Applications

4.2.1. Wireless Channel Allocation In multi-channel wireless networks, transmissions can still conflict under conventional channel allocation mechanisms that assign different channels to nodes that share a common communication neighbor. This is because the interference range of a node is typically greater than its transmission range. Moreover, nodes' mobility in such networks gives rise to a need to re-configure the channel assignment as the topology changes.

Our graph coloring protocol creates long distances between nodes utilizing the same channel (represented by color) given a fixed number of channels. This reduces both the interference level and the frequency of channel reconfiguration. Even when the allocation by our coloring needs to be re-configured, this can be done quickly as the simulation results in Section 6 indicate. However, a more thorough evaluation in highly dynamic environments is yet to be done.

4.2.2. Distributed Leader Election Leader election algorithms are useful building blocks in network protocols that impose a hierarchical structure, e.g., in hierarchical routing in large scale ad-hoc networks. One challenge in those protocols is to distribute the burden of being leader. Using the method for assigning multiple colors to a node, our coloring protocol can be applied in this environment such that each node can find a node of every other color in some of its neighbors. Once the protocol stabilizes, the leader re-election problem simplifies to having each node take its turn for being the leader.

4.2.3. Distributed Resource Directory Service In large scale distributed networks that require decentralized directory services, the cost of using the directory is reduced if the directory information is accessible at nearby nodes. Our protocol can be used to map each index of the resource (or content) to a specific color using a predetermined method such as a hash function, allowing the index to be stored at a nearby node. An additional benefit of our protocol is that, while determining the location of resources, the direction that a search should take to reach the closest copy of the index is computed at each hop (by CBF). This makes it trivial to design a search that heads directly toward the closest copy of the index.

5. Analysis

In this section, we prove the convergence of our ADC protocol, and provide formal results about its performance in comparison to "optimal" colorings.

5.1. Convergence

We begin by proving that the ADC protocol always converges to a stable coloring, i.e., nodes all reach a point

where they cease changing their color and therefore cease sending messages. Proofs of several stated claims are omitted here, but can be found in [11].

Claim 2 *If x is in the STALLED state and y has an accurate view of node colors (in particular, $\hat{\delta}_t(y) = \delta_t(y)$ and $\hat{d}_t(y, c_i) = d_t(y, c_i)$ for all $1 \leq i \leq k$), then x will not prevent y from changing color (i.e., x does not "freeze" y's messages).*

Claim 3 *If the graph is unstable, then there exists a node that is or will be in the MOVING state.*

Lemma 1 (Liveness) *If the graph is unstable, then some node will eventually successfully change its color.*

Proof: Our proof is by contradiction. Assume the graph is unstable but no node ever changes color. By Claim 1, all nodes eventually have correct view of graph coloring. Since the graph is unstable, by Claim 3, there exists a node that is or will be in the MOVING state. Let v be the node of highest order among those in the MOVING state. Since v's order is higher than any other nodes in the MOVING state, v is not in the slave set of any other node (only nodes in the MOVING state may have a non-empty slave set, and all such nodes have lower order than v). Furthermore, since v has the correct view of graph coloring, v's request cannot be rejected. Hence nothing can stop v from changing its color, which contradicts the assumption that no node ever changes color. ■

Our final lemma, before proving our first main result, is similar in flavor to a lemma in [10] that proves the convergence of the coloring rule on a graph whose edges have identical length. The proof in this lemma requires that we consider a vector $V(t) = \langle \delta_t(n_1), \delta_t(n_2), \cdots, \delta_t(n_{|N|}) \rangle$, where $\{n_1, n_2, \cdots, n_{|N|}\}$ are all the nodes of the graph, ordered such that $\delta_t(n_i) \leq \delta_t(n_{i+1})$ for all $1 \leq i < |N|$, i.e., $V(t)$ is an ordered $|N|$-component vector whose components are the $\delta_t(x)$ for all nodes x. We say vector $V(t) < V(t')$ when $V(t)$ is *lexicographically ordered* before $V(t')$, i.e., if $V(t) = \langle v_1, v_2, \cdots, v_{|N|} \rangle$ and $V(t') = \langle w_1, w_2, \cdots, w_{|N|} \rangle$ then there is some $0 < i \leq |N|$ where $v_j = w_j$ for all $j < i$ and $v_i < w_i$.

Claim 4 *If x is the only node that changes color between time t and t', then $\delta_{t'}(x) > \delta_t(x)$.*

Proof: Let x changes from color c_1 to color c_2 at time $\tau, t < \tau < t'$.[2] By the 3-way handshaking protocol, x can change its color only when its request to change from c_1 to c_2 has been accepted by all nodes within distance $\hat{\delta}_t(x)$, which means that at time t, there is at least one node of color c_1 at distance $\hat{\delta}_t(x)$ and no node of color c_2 in x's time t disk. Hence x changes color after ensuring that $\hat{\delta}_t(x) = \delta_t(x)$ and $d_t(x, c_2) > \hat{\delta}_t(x)$. Since no other node in x's time t disk changes color between times t and t', $\delta_{t'}(x) > \hat{\delta}_t(x)$, and hence $\delta_{t'}(x) > \delta_t(x)$. ■

Lemma 2 (Monotonicity) *When one node changes color between times t and t', then $V(t) < V(t')$.*

Proof: Let x be the one node that changes from color c_1 to color c_2. We have $\delta_{t'}(x) > \delta_t(x)$ from Claim 4. Consider any node y where $\delta_{t'}(y) < \delta_t(y)$. For this decrease to occur, some node that at time t was not colored $C_t(y) = C_{t'}(y)$ must have changed to $C_{t'}(y)$ by time t'. Since only x changed its color during this time interval, the node must be x and $C_{t'}(y) = c_2$. Since x is no closer than $\delta_{t'}(x)$ to a node of the same color at time t' and x is now the closest node of color c_2 to y, we have that $\delta_{t'}(y) = d(y, x) \geq \delta_{t'}(x)$. Thus, $\delta_t(x) < \delta_{t'}(x) \leq \delta_{t'}(y)$, i.e., some component of $V(t)$ increased to yield $V(t')$ (in this case, the component contributed by node x), and any decreasing components remain larger than the final value of the increased component (for instance, y's component). This is clearly a lexicographic increase, hence $V(t) < V(t')$. ■

Theorem 1 (Convergence) *ADC Protocol converges to a stable coloring.*

Proof: If the graph is unstable, by Lemma 1, there will always be at least one node which changes its color, and whenever a node changes its color, by Lemma 2, there is a lexicographic increase in $V(t)$. Since each component is either bounded by the maximum diameter of the graph or else equals ∞, we must reach a time T where $V(t)$ is fixed for all $T > t$. This means that nodes cease changing their colors, and therefore the coloring is stable. ■

5.2. Formal Bounds

Now we prove some theoretical bounds on color distances generated by our distributed protocol. For the purpose of evaluating the performance of a stable coloring, we define the super-optimal distance of a node as follows.

Definition 1 *The* **super-optimal distance** *of a node x in a graph that is colored with k colors, $d_{opt}(x)$, is the distance to the k-th closest node to x including x.*

The super-optimal distance is the minimum distance for which there exists a coloring in which a node can reach all k colors (including its own color). We coin the term as "super-optimal" because for a large number of graphs, there is no single coloring in which each node can reach all colors within its super-optimal distance.

Theorem 2 *If the graph is stable, then for each node x, $d_t(x, c_i) \leq 3d_{opt}(x)$ for all $i = 1, \cdots, k$. This is a tight bound.*

Proof: Consider an arbitrary coloring of a graph in which there is a node x and a color c_i where $d(x, c_i) > 3d_{opt}(x)$. Let y be the closest node to x of color c_i. Since there are at least k nodes within distance $d_{opt}(x)$ from x, none of which

2 If two nodes change at exactly the same time, clearly the result is the same as in a system where the difference in time is positive but bounded below any ϵ. We can then choose t and t' such that their difference is less than ϵ.

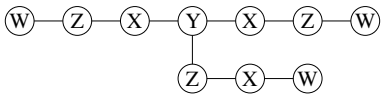


Figure 3. An example of stable coloring : $\max d(x, C_i) = 3\ d_{opt}(x)$

is color c_i, there are two nodes w and z within this distance that are both the same color, c_j. Since both of these nodes are within distance $d_{opt}(x)$ from x, they are at most distance $2d_{opt}(x)$ from one another, and more than distance $2d_{opt}(x)$ from any node of color c_i. It follows that neither w nor z is stable, hence the graph is not stable. ■

We note that this bound is tight, i.e., that there can exist a stable coloring in which $d(x, c_i) = 3d_{opt}(x)$ for some color c_i. Figure 3 demonstrates this, where all edges in the figure have weight 1 and nodes are labeled by their colors. The node colored Y has super-optimal distance of 1, the coloring of the graph is stable, and a distance of 3 is needed to reach color W.

Our next result extends the k-hop bound Lemma for unit-length edge graphs in [10] to graphs whose edge lengths are arbitrary.

Lemma 3 *Let $D_k(x)$ be the minimum distance from node x to a node that is k hops from x. Then for any color $c_i \neq C_t(x)$, $d_t(x, c_i) \leq D_k(x)$ in a stable coloring.*

The above bound is tight when applied to the general class of connected graphs, since equality holds for the case of a chain containing k nodes and $k-1$ unit-length edges. Last, we present a Lemma that lower-bounds the distance between nodes of the same color.

Lemma 4 $\delta_t(x) \geq d_{opt}(x)$ *in a stable coloring.*

Proofs of these lemmas are also available in [11].

6. Performance Evaluation

In this section, we evaluate the performance of our protocol via discrete event-driven simulations, measuring its transient behavior (time to converge to a stable coloring, number of messages per node) as well as the quality of the colorings produced in the context of the distances to colors. Each simulation run is performed on a connected, undirected graph, which is generated randomly as follows. Given a set of nodes, edges are added to the graph between pairs of nodes chosen uniformly, and edge weights are selected uniformly at random within the interval $[1, 10]$. We vary the number of edges that are added and only select graphs that are connected (we add additional edges at random when the graph is not connected). We then classify graphs by the average degree of the nodes. Results presented here, unless explicitly stated otherwise, are based on graphs whose average degree is rounded to 5, i.e., between 4.5 and 5.5.

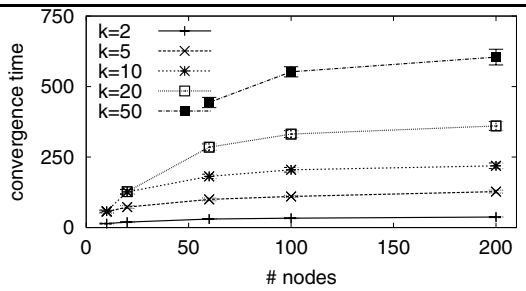


Figure 4. Convergence time

We vary the number of nodes, n, that make up the graph and generate 100 different random graphs for each value of n considered. For each graph, we vary k, the number of colors that are used to color nodes in the graph, and run a separate simulation for each value of k and each graph. At the start of the simulation, a node is assigned a color (from the k) at random and all nodes transmit CBF update messages. The simulation terminates when there is no outstanding message from any node, meaning the graph is stable. The message propagation delays along the edges are proportional to the weight of the edge. We choose proportional delays as a starting point for the simulations since often the edge weights are simply a measure of delay. We reiterate that the protocol will converge regardless of what drives the temporal or spatial dependence of the edge delays.

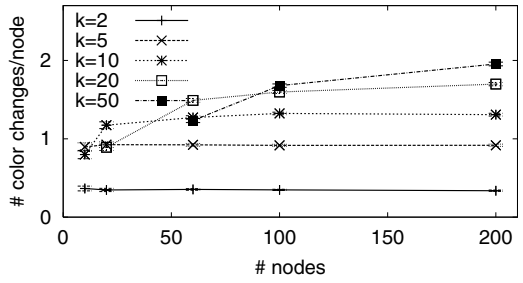
6.1. Transient Behavior

We evaluate the transient behavior of the protocol by measuring convergence time, the number of color changes and messages sent or forwarded per node. The convergence time is measured as the time elapsed until the graph enters a stable state, starting from the instance that every node is assigned a randomly chosen color.

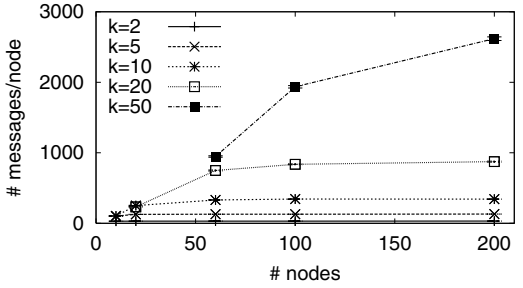
Figure 4 depicts the average convergence time with 95% confidence intervals from the simulations. We vary the number of nodes along the x-axis, with each curve fixing the number of colors used in that set of simulations. We see that as n grows large, the increase in time grows in a logarithmic manner. Clearly the time is more sensitive to the number of colors than it is to the number of nodes.

Next, we turn our attention to the number of times a node changes its color. Figures 5(a) plots the average number of color changes per node as a function of the number of nodes in the graph and of the number of colors used to color the graph. We see the average number of changes per node is between 0 and 2. These preliminary results suggest that the number of color changes, while somewhat sensitive to the number of colors, appears to grow slowly in both the number of nodes and colors.

Figure 5(b) plots the average number of messages transmitted over all types of messages, including those used in CBF update messages, which account for roughly $2/3$ of the total number of messages. While the total number of

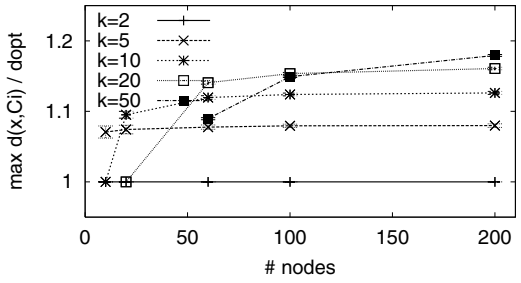


(a) Number of color change per node

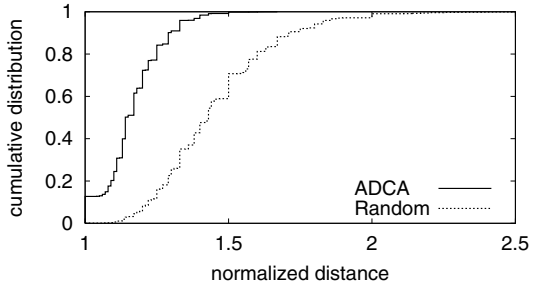


(b) Number of messages per node

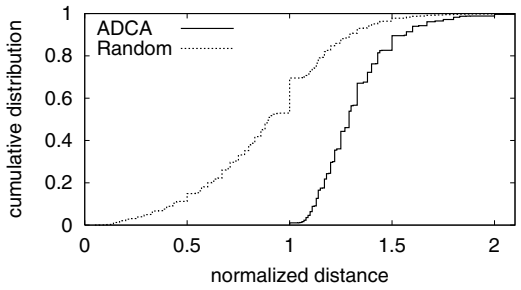
Figure 5. Number of color changes and messages per node



(a) Max distance to super optimal distance



(b) $\frac{\max d(x,c_i)}{d_{opt}(x)}$



(c) $\frac{\delta(x)}{d_{opt}(x)}$

Figure 6. Color distance comparison: Distributed Coloring Algorithm vs Random coloring

messages at first glance seems high for large values of k, we suspect the number is negligible in comparison to the number of messages that these same environments will require as part of the routing protocol, or in comparison to the amounts of data that will be transmitted. In addition, we have made no attempts to optimize the number of messages and suspect that aggregation and randomized suppression techniques can significantly lower the number of messages transmitted (for example, we could "fuse" some *update* messages with *decision* messages).

6.2. Coloring Distance

We evaluate the performance of our protocol by comparing the color distances in stable graphs generated by the protocol to the super-optimal distances and to distances in graphs where each node chooses its color uniformly at random.

Figure 6(a) plots the node's distance to its furthest color normalized by its super-optimal distance in the graph, i.e., $\frac{\max_{1<i<k} d(x,c_i)}{d_{opt}(x)}$. While we have shown that this value can be as high but no higher than 3, in practice we find the value no higher than 1.2 on average. This is quite impressive when one considers the fact that this value cannot be less than 1 and often there is no coloring for which this value is 1 simultaneously for all nodes.

Finally, we compare the coloring distance of the protocol to that of random colorings in which nodes select their colors uniformly at random. Figure 6(b) plots the fraction of nodes over all simulation runs whose distance to the closest node of the furthest color is less than x times the super-optimal distance for both colorings generated by our protocol and those generated at random, where x is varied along the x-axis. The results are taken from all simulations with

$n = 200$ and $k = 20$. The gain obtained by using our mechanism is clear. For example, almost 80% of nodes in colorings generated by our protocol have minimum distances to their furthest color within a factor of 1.25 of their super-optimal distances, while only 10% of nodes are within this factor in a random coloring. Figure 6(c) plots the fraction of nodes whose distance to the closest node of the same color is within a factor of x of the super-optimal distance. As proven in Lemma 4, this factor never falls below one, whereas in a random coloring, this factor is less than one for more than 50% of the nodes. This significant improvement on color distance is observed in all the other sets of simulations with different parameters.

7. Related Work

In [10], we presented a very preliminary centralized version of the greedy coloring protocol used here. The work makes multiple simplifying assumptions that preclude its use in practical networking environments. First, it was assumed that only one node could change its color at a time - a difficult property to ensure in a distributed, asynchronous medium. Second, it was assumed that each node could obtain from an oracle the current coloring of all other nodes within the rest of the graph. Last, all graph edges were assumed to be unit length. The solution in this paper relaxes all of these assumptions.

Our goal of optimizing distance to the nearby copy of resource shares many similarities with the optimal facility location problems. There have been numerous results in this area, such as polynomial-time, constant-factor approximation algorithms [13, 3, 7], and greedy methods [8] that further improve upon previous approximations. [2] is probably the most closely related to our work in the sense that it solves the problem of placing replicated objects in arbitrary nodes in the network, but it uses an off-line, centralized algorithm, and to our best knowledge, there has been no exploration of simple distributed approaches that address the problem in our context.

A large body of work has also looked at the problem of placing replicas within peer-to-peer overlays [9, 12]. However, these works differ from ours in that the focus is on identifying content that is popular and ensuring that this content is placed nearby. Our goal, in contrast, is to find a placement strategy that minimizes the distance that needs to be searched to find an arbitrarily chosen piece of content.

8. Conclusion

We have presented a decentralized, fully distributed, scalable protocol that places replicated resources in a network of arbitrary topology such that the furthest distance one must travel to find a particular copy of a resource is only slightly larger than optimal, and the distance between identical copies is large. Simulation results suggest that running time, messages, and the number of color changes for a node scale logarithmically in the number of nodes in the graph and colors used to color the graph. These properties make the protocol ideal for assigning locations of replicated resources that are needed in emerging networking environments such as ad-hoc, sensor, and overlay networks.

There are a number of interesting problems of both a theoretical and practical nature. First, it remains an open question as to whether the convergence time of the distributed protocol is polynomial in the number of colors and nodes of the graph. Simulation results suggest this to be the case, but we have not yet been successful in demonstrating this formally. Another challenge is dealing with frequent changes in topology, either due to node movement, node failure, or joining and leaving of participants, which is a common characteristic of these emerging networking environments. Since our protocol converges quickly starting from a random coloring, we suspect it will also quickly adapt to such changes. However, shifting of replicated resources may be expensive and so the practical details of such shifts must be considered.

References

[1] M. Adler. *personal communication*.

[2] I. D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington, DC, January 2001.

[3] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k -median problem (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 1–10, Atlanta, GA, May 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[6] Gnutella. `http://www.gnutella.co.uk/`.

[7] A. Goel, P. Indyk, and K. R. Varadarajan. Reductions among high dimensional proximity problems. In *Symposium on Discrete Algorithms (SODA)*, pages 769–778, Washington DC, 2001.

[8] Guha and Khuller. Greedy strikes back: Improved facility location algorithms. In *Symposium on Discrete Algorithms (SODA)*, San Francisco, CA, January 1998.

[9] J. Kangasharju, K. W. Ross, and D. A. Turner. Optimal Content Replication in P2P Communities. in submission, 2002.

[10] B.-J. Ko and D. Rubenstein. A greedy approach to replicated content placement using graph coloring. In *SPIE ITCom Conference on Scalability and Traffic Control in IP Networks II*, July 2002.

[11] B.-J. Ko and D. Rubenstein. Replica placement for emerging networks via an asynchronous, decentralized, graph-coloring algorithm. Technical report, Columbia University, May 2003.

[12] L. Qiu, V. Padmanabham, and G. Voelker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *Proceedings of the ACM SIGCOMM '01*, August 2002.

[13] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 265–274, El Paso, TX, May 1997.