Data Caching under Number Constraint

Himanshu Gupta and Bin Tang

***Abstract*— Caching can significantly improve the efficiency of information access in networks by reducing the access latency and bandwidth usage. However, excessive caching can lead to prohibitive system cost and performance degradation. In this article, we consider the problem of caching a data item in a network wherein the data item is read as well as updated by other nodes and there is a limit on the number of cache nodes allowed. More formally, given a network graph, the read/write frequencies to the data item by each node, and the cost of caching the data item at each node, the problem addressed in this article is to selected a set of P nodes to cache the data item such that the sum of the reading, writing (using an optimal Steiner tree), and storage cost is minimized. For networks with a tree topology, we design an optimal dynamic programming algorithm that runs in $O(|V|^3P^2)$, where $|V|$ is the size of the network and P is the allowed number of caches. For the general graph topology, where the problem is NP-complete, we present a centralized heuristic and its distributed implementation. Through extensive simulations in general graphs, we show that the centralized heuristic performs very close to the exponential optimal algorithm for small networks, and for larger networks, the distributed implementation and the dynamic programming algorithm on an appropriately extracted tree perform quite close to the centralized heuristic.**

I. Introduction

In recent years, with the advent of wireless technology and file sharing applications, the traditional client-server model has begun to lose its prominence. Instead, information sharing by spontaneously connected nodes has emerged as a new framework. In such networks, all network nodes are equal in terms of capacity and functionality. Moreover, the ownership of the files is not critical, and a file (data item) does not belong to a specific node and hence, is read and written by multiple nodes in the network. Caching an object at various network nodes can play an important role in improving overall system performance by drastically reducing the time to read an object.

In this article, we address the data caching problem in above described multi-hop networks wherein the given data item may be read and written by multiple other network nodes, and the objective is to minimize the total reading, writing, and storage cost by placing a limited number of caches. Here, the cost of reading the data item by a node is defined as the distance to the closest cache node times the read frequency, the cost of writing is defined as the cost of the minimum Steiner tree over the writing node and all the cache nodes times the write frequency, and the storage cost is the given cost of caching the data item at the node.

The rest of the paper is organized as follows. In Section II, we present our network model, formulate the data caching problem addressed in this article, and present an overview of the related work. Section III presents the optimal dynamic programming algorithm for tree topology networks. In Section IV, we design centralized and distributed heuristics for general graph networks. Simulation results are presented in Section V, and concluding remarks in Section VI.

Department of Computer Science, Stony Brook University. Email: hgupta,bintang@cs.sunysb.edu

II. Data Caching Problem Formulation

In this section, we present our model of the network, give a formal definition of the problem, and present a discussion on related work. We use the term *cache node* to refer to a network node that caches the data item.

Network Model and Notations. We model the network as a connected general graph, $G(V, E)$, where V is the set of nodes/vertices, and E is the set of edges. There is a single data item in the network, which is to be cached at selected network nodes. For each node $i \in V$, the frequency of reading the data item is r_i, the frequency of writing the data item is w_i, and the cost of caching (i.e., storing) the data item at node i is s_i. Let d_{ij} denote the shortest distance (in number of hops) between any two nodes i, j, and let $d(i, M) = \min_{j \in M} d_{ij}$ be the shortest distance from i to some node in a set of nodes M. Also, let $S(X)$ be the optimal cost of a Steiner tree over the set of nodes X. Given a set of cache nodes M where the data item is cached, the total cost of reading the data item by a node i is $r_i d(i, M)$, while the cost of writing by node i is $w_i S(M \cup \{i\})$. Note that we do not assume a server for the data item in the network, since in our model, a server can be looked upon as a predetermined cache node.

Data Caching Problem. The *data caching problem* in the above network model can be defined as follows. Given a network graph $G(V, E)$ and a number $P (1 \leq P \leq |V|)$, select at most P cache nodes such that the total (reading, writing, and storage) cost is minimized. For a given network graph G and a set of cache nodes M, the total cost is denoted by $\tau(G, M)$ and is defined as:

$$\tau(G, M) = \sum_{i \in V} r_i d(i, M) + \sum_{i \in V} w_i S(\{i\} \cup M) + \sum_{i \in M} s_i \quad (1)$$

In the above equation, the terms on the right hand side represent total read cost, total write cost, and total storage cost respectively. Essentially, the data caching problem is to select a set of cache nodes M ($|M| \leq P$) such that the total cost $\tau(G, M)$ is minimized.

Related Work. When there are no writers and $P = |V|$, the data caching problem is exactly the same as well-known facility-location problem. When there are only read costs, the data caching problem is the well-known P-median problem. Both the problems are NP-hard, and a number of constant-factor approximation algorithms have been developed for each of the problems [1], [6], [2], under the assumption that

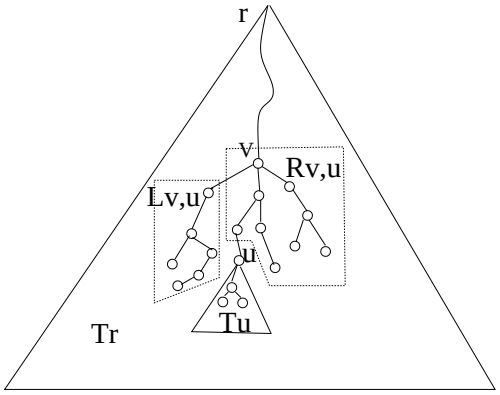

Fig. 1. Subtree Notations.

the edge costs in the graph satisfy the triangular inequality. Without the triangular inequality assumption, either problem is as hard as approximating the set cover [5], [10], and therefore cannot be approximated better than $O(\log |V|)$ unless $\mathbf{NP} \subseteq \tilde{\mathbf{P}}$. Several papers in the literature circumvent the hardness of the facility-location and P-median problems by assuming that the network has a tree topology [9], [8], [11].

The related optimal residence set problem has been studied extensively [3]. Wolfson and Milo [13] show that the optimal residence set problem without storage cost is NP-hard for general topologies. They provide efficient optimal algorithms for complete, tree, and ring topologies. However, their write policy uses the minimum spanning tree of the distance graph of the replica nodes. In comparison, our problem formulation considers storage cost and uses a write policy based on the optimal Steiner tree over the writer and the set of cache nodes.

The work that is most closely related with ours is that by Kalpakis et al. [7]. They considered the problem of finding a Steiner-optimal P-replica set in a tree topology in order to minimize the sum of reading, writing, and storing costs. They developed a very complicated (more than 20 pages of case analysis) optimal dynamic programming algorithm that runs in $O(|V|^6P^2)$ time and finds a Steiner-optimal replica set of size *exactly* P in tree topologies. In our understanding, their work gives a $O(|V|^6P^3)$-time algorithm for finding a Steiner-optimal replica set of size *at most* P in trees. In this article, we essentially address the same problem and design a much simpler dynamic programming optimal algorithm that runs in $O(|V|^3P^2)$ time and finds an optimal set of caches of size at most P. In addition, we design centralized and distributed heuristics to solve the problem in general graph topologies, and show through extensive simulations that our proposed algorithms perform well in general graph topologies.

III. Data Caching in Tree Topology

In this section, we address the data caching problem in the special case of a tree topology, and present an optimal dynamic programming algorithm. We start with some subtree notations (as in [9]) that are needed to describe our dynamic programming algorithm.

Subtree Notations. Let $G(V, E)$ be a given network tree, and let r be a network node (i.e., $r \in V$). Let T_r denote the entire network tree rooted at r. In general, we use T_u to denote the subtree rooted at u in the tree T_r. We use T_u to also represent the set of nodes in the subtree T_u. Now, consider two nodes v and u in the network tree, such that v in an ancestor of u in T_r. Let $\pi(v, u)$ denote the unique path from node u to node v in T_r. As shown in Figure 1, let $L_{v,u}$ be the subtree in T_v consisting of nodes on the left of the path $\pi(v, u)$, but excluding the nodes on the path $\pi(v, u)$. Also, let $R_{v,u}$ be the subtree consisting of nodes on the right of the path $\pi(v, u)$, but including the nodes on the path $\pi(v, u)$ except for u. Thus, the tree T_v is partitioned into three disjoint subtrees, viz., $L_{v,u}$, T_u, and $R_{v,u}$.

Dynamic Programming Algorithm. Our dynamic programming approach changes the write frequencies of the subtree nodes, but keeps the read and storage frequencies unchanged. Thus, for presentation of our dynamic programming solution, we need to represent nodes' write frequencies in a given subtree as a separate parameter.

Notation Γ. Let us use $\Gamma(T_v, \mathcal{W}_v, p)$ to denote the optimal (minimum) total cost for the subtree T_v using at most p caches *including* v, wherein the parameter $\mathcal{W}_v = \{(i, \bar{w}_i) | i \in T_v\}$ represents the write frequencies of nodes in T_v and $p > 0$. Note that the write frequencies $\bar{w}_i$ may be different from the original write frequencies w_i given for the network graph G, while the nodes' read frequencies and storage costs are the same as original given values and are implicit in the notation T_v. Also, since we have assumed that the root v is necessarily a cache node, we can only place at most $p - 1$ additional caches.

Below, we present a recursive dynamic programming equation for computing $\Gamma(T_v, \mathcal{W}_v, p)$ in terms of optimal cost Γ over smaller subtrees contained in T_v. Note that the original problem of determining the optimal cost due to placement of at most P caches in the given network can be solved by evaluating $\min_{r \in V} \Gamma(T_r, \mathcal{W}_G, P)$, where V is the set of all nodes in the network and $\mathcal{W}_G = \{(i, w_i) | i \in V\}$ is the complete set of original write frequencies. Our given recursive equation can be easily modified to compute the actual set of optimal cache nodes, but we present the equation for computing the optimal cost for sake of simplicity of presentation.

Recursive Equation. Given a set of selected cache nodes (including v) in T_v, let u be the leftmost deepest cache node (other than v) in T_v. More formally, let u be the cache node in T_v such that there are no cache nodes in $L_{v,u}$ or on $\pi(u, v) - \{u, v\}$. Recall that $\pi(u, v)$ is the path connecting u and v in the given network tree. To optimally place upto p caches in T_v, we try to optimally place upto q caches in T_u and $p-q$ caches in $R_{v,u}$, where $q \leq p-1$. The recursive equation below defines $\Gamma(T_v, \mathcal{W}_v, p)$ in terms of $\Gamma(T_u, \mathcal{W}_u, q)$ and $\Gamma(R_{v,u}, \mathcal{W}_{v,u}, p-q)$ for appropriately defined $\mathcal{W}_u$ and $\mathcal{W}_{v,u}$. The equation is further explained in the following paragraph,

and its correctness is formally proved in Theorem 1. Let

$$\begin{aligned} \mathcal{W}_v &= \{(i, \bar{w}_i) | i \in T_v\}, \quad \text{and} \\ C_1 &= \sum_{i \in T_v} (r_i d_{iv} + \bar{w}_i d_{iv}) + s_v. \end{aligned}$$

Then, the recursive equation for computing $\Gamma(T_v, \mathcal{W}_v, p)$ is given by:

$$\Gamma(T_v, \mathcal{W}_v, p) = \begin{cases} C_1 & \text{if } p = 1 \\ \min\Big(C_1, \min_{u \in (T_v - \{v\})} \min_{1 \le q \le p-1} \Big(& \\ \quad \sum_{i \in L_{v,u}} (r_i d_{iv} + \bar{w}_i d_{iv}) & \\ \quad + \Gamma(T_u, \mathcal{W}_u, q) & \\ \quad + \Gamma(R_{v,u}, \mathcal{W}_{v,u}, p-q) & \\ \quad + \sum_{i \in T_v} \bar{w}_i d_{vu} \Big)\Big) & \text{if } p > 1 \end{cases} \quad (2)$$

Above,

$$\mathcal{W}_u = \{(i, \bar{w}_i) | i \in (T_u - \{u\})\} \cup \{(u, \sum_{i \in (T_v - T_u) \cup \{u\}} \bar{w}_i)\} \text{ and}$$

$$\mathcal{W}_{v,u} = \{(i, \bar{w}_i) | i \in (R_{v,u} - \{v\})\} \cup \{(v, \sum_{i \in (T_v - R_{v,u}) \cup \{v\}} \bar{w}_i)\}.$$

Note that above recursive equation is for an arbitrary subtree in the original graph, and hence, also applies to a subtree of the kind $R_{v,u}$.

Explanation. We now explain the above recursive equation (Equation 2) in more detail. When $p = 1$, no additional caches can be placed since v is already a cache node. Thus, the optimal total cost $\Gamma(T_v, \mathcal{W}_v, p)$ is C_1, the total cost incurred by the nodes in T_v when the only cache node is v. For the general case ($p > 1$), either v is still the only cache node (in which case the total optimal cost is still C_1) or there are additional cache nodes in T_v. In the latter case, the deepest leftmost cache node u in T_v exists. Then, the optimal cost $\Gamma(T_v, \mathcal{W}_v, p)$ is computed by iterating over all values of q ($1 \le q \le p-1$) and nodes $u \in (T_v - \{v\})$. The total read and storage costs of nodes in T_v are fully embedded in the first three terms for nodes in $L_{v,u}$, T_u, and $R_{v,u}$ respectively. Since there are no cache nodes in $L_{v,u}$, the storage cost of nodes in $L_{v,u}$ is zero and the read requests are satisfied by v. The storage and read cost of nodes in T_u are subsumed in $\Gamma(T_u, \mathcal{W}_u, q)$, since the storage cost of each node is independent of other caches and the read requests of nodes in T_u are satisfied by cache nodes (including u) in T_u only. Similarly, the storage and read cost of nodes in $R_{v,u}$ are subsumed in $\Gamma(R_{v,u}, \mathcal{W}_{v,u}, p-q)$.

The cost of the Steiner tree spanning over a writer node i and the cache nodes M of T_v can be divided into the cost of Steiner tree spanning over cache nodes in T_u, cost of Steiner tree spanning over cache nodes in $R_{v,u}$, the cost of the path $\pi(v, u)$, and the cost of the shortest path connecting i to the closest cache node. Thus, the total write cost over cache nodes in T_v is embedded in multiple terms of Equation 2, viz., $\bar{w}_i d_{iv}, \Gamma(T_u, \mathcal{W}_u, q), \Gamma(R_{v,u}, \mathcal{W}_{v,u}, p-q)$, and $\sum_{i \in T_v} \bar{w}_i d_{vu}$. The above claim of write cost division forms the core of the proof of the following theorem.

Theorem 1: Equation 2 correctly represents the optimal total cost $\Gamma(T_v, \mathcal{W}_v, p)$ for a given T_v, $\mathcal{W}_v$ and p.

Proof: It is easy to see that the Equation 2 is correct for the case of $p = 1$. Below, we assume that $p > 1$.

Let M_v be a set of cache nodes in T_v, where $v \in M_v$. Let $|M_v| > 1$; we will incorporate the case when $|M_v| = 1$ in the end. Let u be the leftmost deepest (as defined before) cache in T_v; since $|M_v| > 1$, the node u exists. Consider the $L_{v,u}$, T_u, and $R_{v,u}$ subtrees, as defined before. Let $M_v = M_u \cup M_{v,u}$, where M_u and $M_{v,u}$ denote the cache nodes in T_u and $R_{v,u}$ respectively.

Given a subtree T_x, set of nodes' write frequencies $\mathcal{W}$, and a set of cache node M. Let $w(T_x, \mathcal{W}, M)$ denote the total write cost incurred by nodes in T_x for writing to caches in M (not necessarily contained in T_x) at write frequencies given in $\mathcal{W}$. Let $r(T_x, M)$ denote the total reading cost incurred by the nodes in T_x using the caches in M, and $s(T_x, M)$ denote the total storage cost of the caches nodes contained in T_x.

Now, the total cost $\tau(T_v, \mathcal{W}_v, M_v)$ in T_v for a given set of write frequencies $\mathcal{W}_v = \{(i, \bar{w}_i) | i \in T_v\}$ and a set of cache nodes M_v is given by

$$\tau(T_v, \mathcal{W}_v, M_v) = w(T_v, \mathcal{W}_v, M_v) + r(T_v, M_v) + s(T_v, M_v),$$

and can be further manipulated as below. Recall that $S(M)$ denotes the cost of the optimal Steiner tree spanning over a set of nodes M, and $d(i, M)$ denotes the shortest distance between i and a node in M.

$$\begin{aligned} &\tau(T_v, \mathcal{W}_v, M_v) \\ &= w(T_v, \mathcal{W}_v, M_v) + r(T_v, M_v) + s(T_v, M_v) \\ &= w(T_u, \mathcal{W}_v, M_v) + r(T_u, M_u) + s(T_u, M_u) \\ &\quad + w(R_{v,u}, \mathcal{W}_v, M_v) + r(R_{v,u}, M_{v,u}) + s(R_{v,u}, M_{v,u}) \\ &\quad + w(L_{v,u}, \mathcal{W}_v, M_v) + r(L_{v,u}, \{v\}) \\ &= \sum_{i \in T_u} \bar{w}_i (d(i, M_u) + S(M_v)) + r(T_u, M_u) + s(T_u, M_u) \\ &\quad + \sum_{i \in R_{v,u}} \bar{w}_i (d(i, M_{v,u}) + S(M_v)) + r(R_{v,u}, M_{v,u}) \\ &\qquad + s(R_{v,u}, M_{v,u}) \\ &\quad + \sum_{i \in L_{v,u}} \bar{w}_i (d_{iv} + S(M_v)) + \sum_{i \in L_{v,u}} r_i d_{iv} \end{aligned}$$

Substituting $S(M_v) = S(M_u) + S(M_{v,u}) + d_{uv}$ (since, u is the leftmost deepest cache node of M_v) into the above equation and grouping terms of $\bar{w}_i$ together, we get:

$$\begin{aligned} &\tau(T_v, W_v, M_v) \\ &= \sum_{i \in T_u} \bar{w}_i d(i, M_u) + \sum_{i \in T_v} \bar{w}_i S(M_u) + r(T_u, M_u) + s(T_u, M_u) \\ &\quad + \sum_{i \in R_{v,u}} \bar{w}_i d(i, M_{v,u}) + \sum_{i \in T_v} \bar{w}_i S(M_{v,u}) + r(R_{v,u}, M_{v,u}) \\ &\qquad + s(R_{v,u}, M_{v,u}) \\ &\quad + \sum_{i \in L_{v,u}} (r_i d_{iv} + \bar{w}_i d_{iv}) + \sum_{i \in T_v} \bar{w}_i d_{vu} \end{aligned} \quad (3)$$

In the above Equation 3, let us manipulate the first four terms.

$$\begin{aligned}
&\sum_{i\in T_u} \bar{w}_i d(i, M_u) + \sum_{i\in T_v} \bar{w}_i S(M_u) + r(T_u, M_u) + s(T_u, M_u) \\
&= \sum_{i\in T_u} \bar{w}_i d(i, M_u) + \sum_{i\in T_u} \bar{w}_i S(M_u) + \sum_{i\notin T_u} \bar{w}_i S(M_u) \\
&\quad + r(T_u, M_u) + s(T_u, M_u) \\
&= \tau(T_u, \mathcal{W}_v, M_u) + \sum_{i\notin T_u} \bar{w}_i S(M_u) \\
&= \tau(T_u, \mathcal{W}_u, M_u), \qquad (4)
\end{aligned}$$

where

$$\mathcal{W}_u = \{(i, \bar{w}_i) | i \in (T_u - \{u\})\} \cup \{(u, \sum_{i\in(T_v - T_u)\cup\{u\})} \bar{w}_i)\}.$$

Similarly, we get

$$\begin{aligned}
&\sum_{i\in R_{v,u}} \bar{w}_i d(i, M_{v,u}) + \sum_{i\in T_v} \bar{w}_i S(M_{v,u}) + r(R_{v,u}, M_{v,u}) \\
&\quad + s(R_{v,u}, M_{v,u}) \\
&= \tau(R_{v,u}, \mathcal{W}_{v,u}, M_{v,u}), \qquad (5)
\end{aligned}$$

where

$$\mathcal{W}_{v,u} = \{(i, \bar{w}_i) | i \in (R_{v,u} - \{v\})\} \cup \{(v, \sum_{i\in(T_v - R_{v,u})\cup\{v\}} \bar{w}_i)\}.$$

Now, substituting Equation 4 and Equation 5 into Equation 3, we get

$$\begin{aligned}
&\tau(T_v, \mathcal{W}_v, M_v) \\
&\qquad = \tau(T_u, \mathcal{W}_u, M_u) \\
&\qquad + \tau(R_{v,u}, \mathcal{W}_{v,u}, M_{v,u}) \\
&\qquad + \sum_{i\in L_{v,u}} (r_i d_{iv} + \bar{w}_i d_{iv}) + \sum_{i\in T_v} \bar{w}_i d_{vu} \qquad (6)
\end{aligned}$$

Now, for a given v and u, the last two terms in Equation 6 are independent of M_v. Thus, the optimal set M_v (where $1 < |M_v| \leq p$) that minimizes $\tau(T_v, \mathcal{W}_v, M_v)$ can be chosen by considering all possible values of u and $q = |M_u|$, and for each such pair of values, selecting optimal sets of cache nodes M_u and $M_{v,u}$ that minimize $\tau(T_u, \mathcal{W}_u, M_u)$ and $\tau(R_{v,u}, \mathcal{W}_{v,u}, M_{v,u})$ in T_u and $R_{v,u}$ respectively. Finally, pick the pair of values u and q that minimizes the total cost $\tau(T_v, \mathcal{W}_v, M_v)$ using Equation 6 and optimal cache sets M_u and $M_{v,u}$. The recursive equation (Equation 2) does exactly the above to compute the optimal cost, except it also considers the case $|M_v| = 1$ when the total cost incurred in C_1. ■

Time Complexity. As mentioned before, to compute the minimum total cost of placement of P caches in the original given graph G, we need to compute $\min_{r\in V} \Gamma(T_r, \mathcal{W}_G, P)$ where V is the set of all vertices and $W_G = \{(i, w_i) | i \in V\}$ represents the original given write frequencies. To compute $\Gamma(T_r, \mathcal{W}_G, P)$ for all r, we precompute all $\Gamma(T_x, \mathcal{W}_x, p)$ and $\Gamma(R_{x,y}, \mathcal{W}_{x,y}, p)$ as defined below.

Each edge (x, x') of the network tree divides the graph into two subtrees. Consider the subtree T_x that contains x, and define $\mathcal{W}_x$ as

$$\mathcal{W}_x = \{(i, w_i) | i \in (T_x - \{x\})\} \cup \{(x, \sum_{i\in(V - T_x)\cup\{x\})} w_i)\}.$$

We precompute $\Gamma(T_x, \mathcal{W}_x, p)$ for all values of p and edges (x, x') in the network graph. In addition, we also precompute $\Gamma(R_{x,y}, \mathcal{W}_{x,y}, p)$ for all values of p and $y \in T_x$, where

$$\mathcal{W}_{x,y} = \{(i, w_i) | i \in (R_{x,y} - \{x\})\} \cup \{(x, \sum_{i\in(V - R_{x,y})\cup\{x\})} w_i)\}.$$

It can be shown that the above set of Γ values can all be computed in the order of subtree sizes in a dynamic programming manner using Equation 2. Once the above values have been computed, $\Gamma(T_r, \mathcal{W}_G, P)$ for each $r \in V$ can then be computed using the same Equation 2. Thus, we need to compute $P|V|^2$ values, where computation of each value takes $O(|V|P)$ time. Here, we assume that the first and last terms of Equation 2 are already precomputed (using $O(|V|^2)$ preprocessing time). Thus, the total time complexity of our dynamic programming algorithm is $O(P^2|V|^3)$ where $|V|$ is the size of network and P is the number of cache nodes allowed.

IV. General Graph Topology

In this section, we address the data caching problem in a general graph topology. In a general graph, the data caching problem is NP-hard, since it reduces to the facility-location problem when the write frequencies are zero. Here, we first design a centralized greedy algorithm, and then present a distributed implementation of the centralized algorithm. We have used similar techniques in our recent work [12] on a related problem of data caching under update cost constraint. We will show through simulations that the centralized heuristic developed in this section perform close to the optimal solution in small general graph networks.

A. Centralized Greedy Algorithm

We now present a polynomial-time Centralized Greedy Algorithm for the data caching problem. We start with defining the concept of a benefit of a set of nodes.

Definition 1: (Benefit of Node) Let M be the set of nodes that have been already selected as cache nodes by the Centralized Greedy Algorithm at some stage. The *benefit* of an arbitrary node A, denoted as $\beta(A, M)$, is the reduction in total cost due to selection of A as a cache node. More formally, $\beta(A, M) = \tau(G, M) - \tau(G, M \cup \{A\})$, where $\tau(G, M)$ is the total cost of selecting a set of cache nodes M in graph G, as defined in Equation 1. □

Note that since the minimum-cost Steiner tree problem is NP-hard, we adopt the 2-approximation Steiner tree algorithm [4] to compute writing costs.

Based on the above definition of benefit, our proposed Greedy Algorithm can be described as follows. Let M be the set of cache nodes selected at any given stage. Initially, M is empty. At each stage of the Greedy Algorithm, we add to M the node A that has the highest benefit with respect to M at that stage. The process continues until P caches nodes have been selected or there is no node with positive benefit. The running time of the above described algorithm is $O(P|V|^5)$, since the time to compute a 2-approximation Steiner tree over a set of s nodes is $O(s|V|^2)$.

B. **Distributed Greedy Algorithm**

In this subsection, we present a distributed localized implementation of the Centralized Greedy Algorithm.

To facilitate communication between nodes, we assume presence of a *coordinator* in the network. Our Distributed Greedy Algorithm consists of rounds. During each round, each non-cache node A estimates the benefit (as described in the next paragraph) of caching the data item at A. If the benefit estimate at a node A is positive and is the maximum among all its non-cache neighbors, then A decides to cache the data item. At the end of a round, the coordinator node gathers information about the cache nodes newly added. The number of cache nodes that can be further added is then broadcast by the coordinator to the entire network. The algorithm terminates, when either more than P cache nodes have already been added or no more cache nodes were added in a round.

Estimation of $\beta(A, M)$. A non-cache node A considers only its "local" traffic and estimation of distance to the nearest cache node, to estimate $\beta(A, M)$, the benefit with respect to an already selected set of cache nodes M. In particular, a node A observes its local traffic, i.e., the data access requests that A forwards to other cache nodes. Of course, the local traffic of a node includes its own data requests. We estimate the benefit of caching the data item at A as

$$\beta(A, M) = fd - s_a - d\sum_{i \in V} w_i,$$

where f is the frequency of the local data access traffic observed at A, d is the distance to the nearest cache from A (which is computed as shown in the next paragraph), s_a is the storage cost at A, and w_i is the write frequency at a node i in the network. In the above equation, we have estimated the increase in total writing cost due to caching at A as $d\sum_{i \in V} w_i$. The local traffic f can be computed if we let the normal network traffic (using only the already selected cache nodes in previous rounds) run for some time between successive rounds.

Estimation of d – the distance to the nearest cache from A. Let A be a non-cache node, and T_A be the shortest path tree from the coordinator to the set of communication neighbors of A. Let $C \in M$ be the cache node in T_A that is closest to A. In the above Distributed Greedy Algorithm, we estimate d to be $d(A, C)$, the distance from A to C. The value $d(A, C)$ can be computed in a distributed manner at the start of each round as follows. As mentioned before, the coordinator initiates a new round by broadcasting a packet containing the remaining number constraint to the entire network. If we append to this packet all the cache nodes encountered on the way, then each node should get the set of cache nodes on the shortest path from the server to itself. Now, to compute $d(A, C)$, each node only needs to exchange the above information with all its immediate neighbors.

V. **Performance Results**

In this section, we evaluate the relative performances of the various cache placement algorithms proposed in our article.

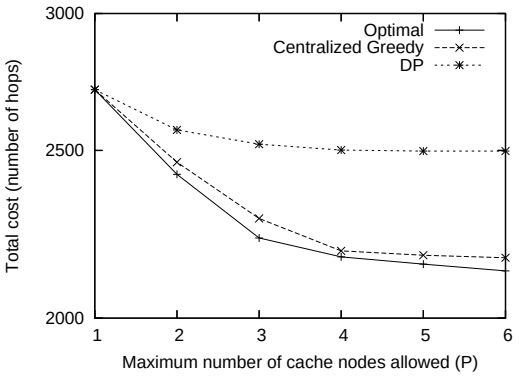

Fig. 2. Comparison of Centralized Greedy Algorithms with the optimal algorithm. Here, the network size is 50, R (the ratio of average write to average read frequency) as 0.1, and percentage of readers and writers is 50%.

Experiment Setup. We use a network of 50 to 400 nodes placed randomly in a square region of size 30×30. We consider unit-disk graphs wherein two nodes can communicate with each other if the distance between them is less than a given number (called the *transmission radius*). For our simulations, we use a transmission radius of 9, which is the minimum to keep even small networks of size 50 connected. We vary various parameters such as network size, the maximum number of cache nodes P, percentage of readers and writers in the network, and the *ratio* R of average write frequency to average read frequency. Note that in practical settings we expect R to be low. The read frequency of a reader node is chosen to be a random number between 0 and 100, the write frequency of a writer node is chosen to be a random number between 0 and $100R$, and the storage cost at a node is chosen to be a random number between 0 and 100.

In our simulations, we compare the performance of various data caching placement algorithms, viz., Centralized Greedy Algorithm, Distributed Greedy Algorithm, and Dynamic Programming Algorithm (DP) on the shortest path tree rooted at v that results in the minimum total cost $\tau(G, \{v\})$. Each data point in the graph plots is an average over five different random graph topologies. We start with comparing our Centralized Greedy Algorithm with the optimal algorithm in small size networks.

Comparison with Optimal Algorithm in Small Networks. An optimal solution for the data caching problem can be computed by looking at all $O(|V|^P)$ subsets of nodes of size at most P, and picking the subset of nodes that gives the minimum total cost as the solution. Due to the high time complexity of the above algorithm, we choose the network size $|V| = 50$ and vary P from 1 to upto 6. We pick R (the ratio of average write frequency to the average read frequency) as 0.1, since it was just small enough to result in maximum number of cache nodes being selected. We observe in Figure 2 that the Centralized Greedy Algorithm performs very close to the optimal cost. Thus, in the following experiments, we use the Centralized Greedy Algorithm as a benchmark of comparison. We also observe that the DP algorithm performs only about 15% worse than the optimal algorithm.

Varying R. In this experiment, we vary R (the ratio of average

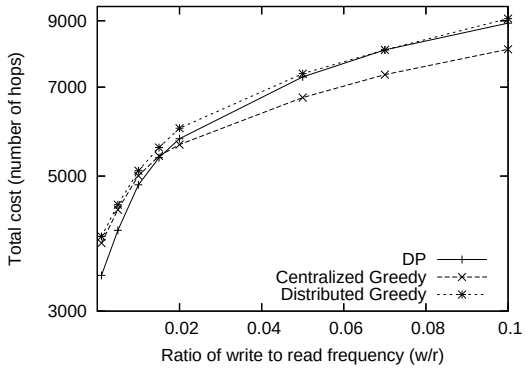

Fig. 3. Varying R, the ratio of average write to average read frequency. Here, the network size is 200, $P = 25$, percentage of readers and writers is 50.

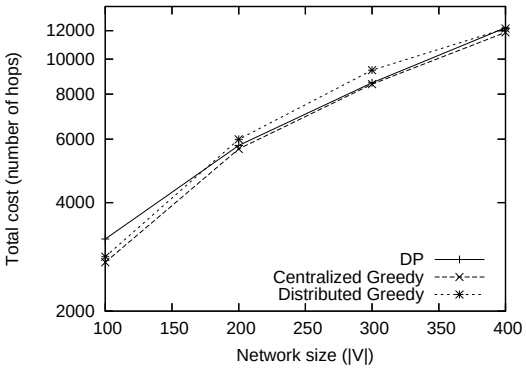

Fig. 4. Varying network size. Here, $P = 25, R$ (the ratio of average write to average read frequency) is 0.02, and percentage of readers and writers is 50.

read frequency to the average write frequency) from 0.001 to 0.1 in a network of size 200 with P (the maximum number of cache nodes allowed) as 25. We keep the percentage of readers and writers in the network at 50%. Figure 3 plots the total cost $\tau(G, M)$ corresponding to the set M of cache nodes delivered by various algorithms for given parameters. We see that the Centralized Greedy outperforms the Distributed Greedy Algorithm only by about 15%. However, when R is small, the centralized and distributed greedy algorithms perform very closely, but their relative performance becomes almost constant after $R = 0.02$. This implies that the estimation of writing costs done by the Distributed Greedy Algorithm is not as accurate as the estimation of reading costs. In contrast, we see that the DP algorithm actually outperforms the Centralized Greedy for very low values of R. For higher values of R, the DP algorithm performs close to the Distributed Greedy. Thus, the strategy of extracting the shortest path tree rooted at an appropriate node seems very effective when the writing cost is relatively very low. For $R = 0.1$, we observed that the number of cache nodes selected by any algorithm was very low (1 or 2). Thus, we did not increase the value of R beyond 0.1. Based on Figure 3, we fix R as 0.02 for all the remaining experiments, since for $R = 0.02$ the number of cache nodes is large enough (around 10) and the relative performance observed at $R = 0.02$ is representative of the general trend.

Varying Network Size. In Figure 4, we vary the network size from 100 to 400 and plot $\tau(G, M)$ corresponding to the solution M delivered by various algorithms. As suggested before, we fix $P = 25$ and $R = 0.02$. Also, the percentage of readers and writers in the network is kept as 50%. In Figure 4, we can that the Centralized Greedy Algorithm outperforms the Distributed Greedy Algorithm and DP algorithms only narrowly. More importantly, we observe that the relative performance of the various algorithms remains relatively stable, and hence, in all other simulations, we fix the network size to be 200.

Varying Percentage of Readers and Writers. In Figure 5 and Figure 6, we vary the percentage of reader and writer nodes respectively in the network and plot the values of $\tau(G, M)$ for the solution delivered by various algorithms. As suggested

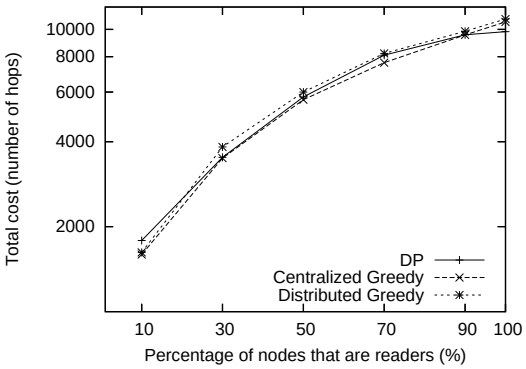

Fig. 5. Varying percentage of reader nodes in the network. Here, the network size is 200, $P = 25$, $R = 0.02$, and the percentage of writer nodes is 50%.

in previous paragraphs, we fix R as 0.02 and the network size as 200. In addition, we use P as 25. In Figure 5, we vary the percentage of reader nodes from 10 to 100, while keeping the percentage of writer nodes fixed at 50%. Similarly, in Figure 6, we vary the percentage of writer nodes from 0 to 100%, while keeping the percentage of reader nodes fixed at 50%. We observe that the relative performance of the various algorithms remains largely unchanged with the change in percentages of readers or writers. In generally, we see the performance gap between various algorithm to be limited by 10-15%.

Varying P. In Figure 7, we vary P, the maximum number of cache nodes allowed, and plot $\tau(G, M)$ for various algorithms. We see that with the increase in P, the relative performance gap between the Centralized and Distributed Greedy Algorithms reduces. After $P = 10$, the performance of the various algorithms remains unchanged since for the given parameter values all algorithms place at most 10 caches. Again, we see the performance gap between various algorithm to be limited by 10-15%.

VI. Conclusions

In this paper, we addressed the problem of selection on nodes to cache a data item in a network, wherein multiple nodes can read or update the data items, individual nodes have storage limitations, and there is a limit on the number of nodes

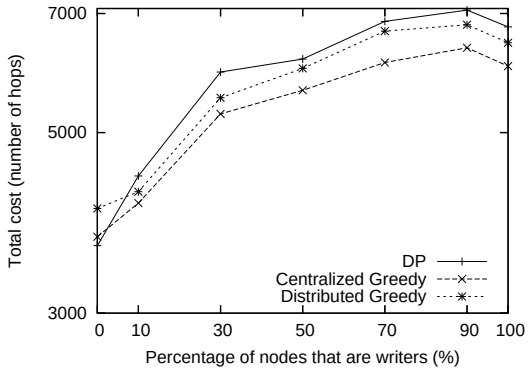

Fig. 6. Varying percentage of writer nodes in the network. Here, the network size is 200, $P = 25$, $R = 0.02$, and percentage of reader nodes is 50%.

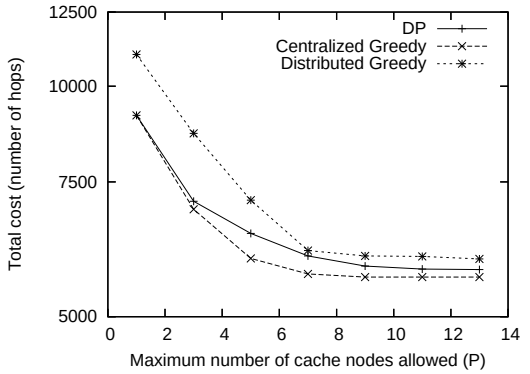

Fig. 7. Varying P. Here, the network size is 200, $R = 0.02$, and percentage of readers and writers is 50%.

that can be selected to cache the data item. The objective of our problem was to minimize the sum of appropriately defined total reading cost, writing cost, and storage cost. For the above data caching problem, we designed an optimal dynamic programming algorithm for tree networks. In addition, for general network graphs, we proposed Centralized Greedy and Distributed Greedy heuristics, and evaluated the performance of our proposed algorithms through extensive simulations. We observe that the Centralized Greedy performs very close to the optimal algorithm for small networks, and for larger networks, the Distributed Greedy and the dynamic programming algorithm on an appropriately extracted tree perform very close to the Centralized Greedy.

REFERENCES

[1] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. of IEEE Conference on Foundations of Computer Sciences*, pages 378–388, 1999.

[2] F.A. Chudak and D. Shmoys. Improved approximation algorithms for a capacitated facility location problem. *Lecture Notes in Computer Science*, 1610:99–131, 1999.

[3] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Survey*, 14(2):287–313, 1982.

[4] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM J. Appl. Math*, 16:1–29, 1968.

[5] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2), 2001.

[6] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[7] K. Kalpakis, K. Dasgupta, and O. Wolfson. Steiner-optimal data replication in tree networks with storage costs. In *Proc. of IDEAS*, pages 285–293, 2001.

[8] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Trans.on Networking*, 8:568–582, 2000.

[9] B. Li, M. J. Golin, G. F. Italiano, and X. Deng. On the optimal placement of web proxies in the internet. In *Proc. of INFOCOM*, volume 3, pages 1282–1290, 1999.

[10] J.-H. Lin and J. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5), 1992.

[11] Arie Tamir. An $o(pn^2)$ algorithm for p-median and related problems on tree graphs. *Operations Research Letters*, 19, 1996.

[12] B. Tang, S. Das, and Himanshu Gupta. Cache placement in sensor networks under update cost constraint. In *Proc. of AdHoc-Now*, 2001.

[13] O. Wolfson and A. Milo. The multicast policy and its relationship to replictated data placement. *ACM Transactions on Data Base Systems*, 16(1):181–205, 1991.