A Mathematical Model, Heuristic, and Simulation Study for a Basic Data Staging Problem in a Heterogeneous Networking Environment

Min Tan*, Mitchell D. Theys†, Howard Jay Siegel†, Noah B. Beck†, and Michael Jurczyk‡

*Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
mintan@cisco.com

†Parallel Processing Laboratory
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285, USA
{hj, theys, noah}@ecn.purdue.edu

‡Department of Computer Engineering and Computer Science
University of Missouri at Columbia
201 Engineering Building West
Columbia, MO 65211, USA
mjurczyk@cecs.missouri.edu

Abstract

Data staging is an important data management problem for a distributed heterogeneous networking environment, where each data storage location and intermediate node may have specific data available, storage limitations, and communication links. Sites in the network request data items and each item is associated with a specific deadline and priority. It is assumed that not all requests can be satisfied by their deadline. This work concentrates on solving a basic version of the data staging problem in which all parameter values for the communication system and the data request information represent the best known information collected so far and stay fixed throughout the scheduling process. A mathematical model for the basic data staging problem is introduced. Then, a multiple-source shortest-path algorithm based heuristic for finding a suboptimal schedule of the communication steps for data staging is presented. A simulation study is provided, which evaluates the performance of the proposed heuristic. The results show the advantages of the proposed heuristic over two random based scheduling techniques. This research, based on the simplified static model, serves as a necessary step toward solving the more realistic and complicated version of the data staging problem involving dynamic scheduling, fault tolerance, and determining where to stage data.

This research was supported by DARPA/ISO and the Office of Naval Research under ONR grant number N00014-97-1-0804, and by NRaD under contract number N66001-96-M-2277, M. D. Theys was also supported by a Purdue Benjamin Meisner Fellowship and an Intel Fellowship.

Keywords: BADD, data staging, data management, Dijkstra's multiple-source shortest-path algorithm, distributed heterogeneous networking environment, heterogeneous computing.

1. Introduction

The DARPA Battlefield Awareness and Data Dissemination (BADD) program [12] includes designing an information system for forwarding (staging) data to proxy servers prior to their usage by a local application, using satellite and other communication links. The network combines terrestrial cable and fiber with commercial VSAT (very small aperture terminal) internet and commercial broadcast. This provides a unique basis for information management. It will allow web-based information access and linkage as well as server-to-server information linkage. The focus is on providing the ability to operate in a server-server-client environment to optimize information currency for many critical classes of information.

Data staging is an important data management problem that needs to be addressed by the BADD program. An informal description of the data staging problem in a military application is as follows. A warfighter is in a remote location with a portable computer and needs data for planning troop movements. The data can include detailed terrain maps, enemy locations, troop movements, and current weather predictions. The data will be available from Washington D.C., foreign military bases, and other data storage locations. Each location may have specific data available, storage limitations, and communication links. Also, each data item is associated with a specific priority, where larger priority value implies

higher importance. It is assumed that not all requests can be satisfied by their deadline. Data staging involves positioning data prior to its use in decision making for facilitating a faster transfer time when it is requested.

Positioning the data before it is requested can be complicated by the dynamic nature of data requests and network congestion; the limited storage space at certain sites; the limited bandwidth of links; the changing availability of links and data; the time constraints of the needed data; the priority of the needed data; and the determination of where to stage the data [13]. Also, the associated garbage collection problem (i.e., determining which data will be deleted or reverse deployed to rear-sites from the forward-deployed units) arises when existing storage limitations become critical [12, 13]. The storage situation becomes even more difficult when copies of data items are allowed to reside on different machines in the network so that there are more available sources from which the requesting sites can obtain certain data [16], and so there is an increased level of fault tolerance, in cases of links or storage locations going off-line.

The simplified data staging problem addressed in this paper requires a schedule for transmitting data between pairs of nodes in the corresponding communication system for satisfying as many of the data requests as possible, with the high priority requests given precedence. Each node in the system can be: (1) a source machine of initial data items, (2) an intermediate node for storing data temporarily (e.g., routers or switches), and/or (3) a final destination machine that requests a specific data item. This problem comes under the topic of distributed heterogeneous computing [15] because nodes may have different storage limitations, different communication links available, different data available, and different data to request. The actual data staging problem is dynamic in nature, because in reality the network configuration can change, certain communication links may become unavailable, new data requests can be submitted sporadically, priorities of existing requests can be modified, and certain nodes in the network may fail.

This paper concentrates on solving a simpler version of the data staging problem in which all parameter values for the communication system and the data request information (e.g., requesting machines and network configuration) represent the best known information collected so far and stay fixed throughout the scheduling process. It is assumed that not all of the requests can be satisfied due to storage capacity and communication constraints. Also, the fault tolerance issues mentioned above are not addressed. The model is designed to create a schedule for movement of data from the source of the data to a ‘‘staged’’ location for the data. It is assumed that the user can easily retrieve the data from this location. A heuristic is presented and evaluated that effectively satisfies this simplified data staging problem. This research, based on the simplified model presented here, serves as a necessary step toward solving the more realistic and complicated version of the data staging problem involving dynamic scheduling, fault tolerance, and determining where to stage data.

Section 2 provides overviews of work that is related to the data staging problem. In Section 3, a mathematical model for a basic data staging problem is introduced. Section 4 presents a multiple-source shortest-path algorithm based heuristic for finding a suboptimal schedule of the communication steps for data staging. This heuristic adopts the simplified view of the data staging problem described by the mathematical model. A simulation study is discussed in Section 5, which evaluates the performance of the proposed heuristic. A BADD-like network environment has been used in developing the parameters for conducting this simulation study. Section 6 summarizes this paper, giving the current status of this research on data staging, and plans for future work. A glossary of notation is included in Section 7 for reference purposes.

2. Related work

To the best of the authors’ knowledge, there is currently no other work presented in the open literature that addresses the data staging problem, designs a mathematical model to quantify it, or presents a heuristic for solving it. A problem that is, at a high level, remotely similar to data staging is the facility location problem [8] in management science and operations research. Under the context of the construction of several new production facilities, a manufacturing firm needs to arrange the locations of the facilities and plants effectively, such that the total cost of transporting individual components from the inventory facilities to the manufacturing plants for assembly is minimized. It is required that the firm makes several interrelated decisions: how large and where should the plants be, what production method should be used, and where should the facilities be located? If an analogy is made between (1) the plants and the destination nodes that make the data requests, (2) the individual manufacturing components and the requested data elements to be transferred, and (3) the facilities and the source locations of requested data, then at a high level the facility location problem has features similar to those of the data staging problem (e.g., use a graph-based method to reduce the facility location problem to a shortest-path or minimum spanning tree problem).

However, when examing the relationship between the facility location problem and the data staging problem carefully, there are significant differences. First, each component that a plant requests is usually not associated with a prioritizing scheme, while in the data staging problem each data request has a priority. Also, each component request from a plant commonly does not have a corresponding deadline related factor, while in the data staging problem each data request has a deadline. For the data staging problem, the priority and deadline associated with each data request are the two most important parameters for formulating the optimization criterion. For example, the minimization of the sum of the weighted priorities of satisfiable data requests (based on their deadlines) is used as the optimization criterion in the mathematical model of a basic data staging problem presented in Section 3. But for the facility location problem, in general, researchers adopt optimization criteria that are related to the physical distances between plants and facilities in either a continuous or discrete domain without any prioritizing schemes or deadline related factors (e.g., [4, 6, 9, 10, 14]). Thus, although lessons can be drawn from the design of algorithms for different versions of the facility location problem, there are no obvious direct correlations between either the formulations or the potential solutions of those two problems.

Data management problems similar to data staging for the BADD program are studied for other communication systems. With the increasing popularity of the World Wide Web (WWW), the National Science Foundation (NSF) recently projected that new techniques for organizing cache memories and other buffering schemes are necessary to alleviate memory and network latency and to increase bandwidth [3]. More advanced approaches of directory services, data replication, application-level naming, and multicasting are being studied to improve the speed and robustness of the WWW [2]. Evidence has been shown [7], that several file caches could reduce file transfer traffic, and hence the volume of traffic on the internet backbone. In addition, distributed environments are looking for ways to increase system performance with intelligent data placement [1]. The study of data staging can potentially draw lessons from and generate positive input for the active research in these related, but not directly comparable, areas of research.

Other research exploring heuristics for use in the BADD environment is being performed [10]. This work examines methods for scheduling the ATM-like channels of the BADD environment efficiently. The work does not develop a mathematical model and does not include several parameters considered here, such as deadlines and starting times. The work does show that ‘‘greedy’’ heuristics are effective tools for use in the BADD environment and uses a network simulator to corroborate this statement.

3. Mathematical model

A mathematical model for a basic data staging problem is presented in this section. This model serves as an initial version of a quantitative model for data staging. It allows the heuristic introduced in Section 4 to be given formally. As stated and discussed in Section 1, this paper concentrates on solving a simpler version of the data staging problem *statically*, where all parameter values for the communication system and the data request information stay fixed throughout the scheduling process. The values of all parameters in the following model may change temporally to reflect the dynamic nature of the underlying network system when the model is extended and used in a dynamic situation. In that case, the parameter values represent the best known information collected at the given point in time (e.g., all requests for data elements include only those known at any specific time instant). All necessary parameters for specifying the communication system and the data request information are introduced as follows. The model includes information about (1) the nodes in the network, (2) the links in the network, and (3) the data requests in the network. Each machine has parameters for the storage capacity and node number. A link has an availability starting time, availability ending time, bandwidth, latency, source node and destination node. Every request has an approximate data size, list of sources, and list of destinations. Each request source consists of a node number and a time after which the data is available on that node. Each request destination contains a node number, priority, and deadline for the data request. This description of the network and associated data requests are used to formulate the mathematical model to be used in solving the basic data staging problem. A glossary of notation is included in Section 7 for the readers convenience.

A communication system M consists of m machines $\{M[0], M[1], ..., M[m-1]\}$. Each machine can be a server that stores data elements and a client that makes data requests to the system. Each machine also can be an intermediate node for storing a copy of a specific data item temporarily. $Cap[i](t)$ represents the available memory storage capacity of machine $M[i]$ ($0 \leq i < m$) at time t.

A network topology graph G_{nt} specifies the connectivity of the communication system for the machines in M with the following notation. A set of m vertices V =

$\{V[0], V[1], ..., V[m-1]\}$ is generated that corresponds to the m machines in the communication system. In this model, if two machines are connected by the same transmission link during v non-overlapping and discontinuous time intervals, then v different *virtual* links corresponding to the appropriate available time intervals are used to represent this situation (e.g., the availability of a satellite link for fifteen minutes each hour). Also, each transmission link is uni-directional. A bi-directional link between two machines is represented as two different virtual uni-directional links that correspond to the transmission link in each direction.

Let $Nl[i,j]$ be the total number of direct virtual communication links from $M[i]$ to $M[j]$. $L[i,j][k]$ denotes the k-th direct virtual communication link from $M[i]$ to $M[j]$, where $0 \le i,j < m$, $i \ne j$, and $0 \le k < Nl[i,j]$. For each $L[i,j][k]$, a directed edge $E[i,j][k]$ from $V[i]$ to $V[j]$ is added to G_{nt}. All the added edges constitute the set of edges E of G_{nt}. Each $L[i,j][k]$ is associated with one unique time frame during which the corresponding link is available for communication. Let $Lst[i,j][k]$ denote the link starting time when $L[i,j][k]$ becomes available and $Let[i,j][k]$ denote the link ending time when $L[i,j][k]$'s availability terminates. With the above notation, link $L[i,j][k]$ is available between $Lst[i,j][k]$ (starting time) and $Let[i,j][k]$ (ending time).

Let a data item be a block of information that can be transferred between machines. For any data item d, $|d|$ represents the size of the associated data set. Let $D[i,j][k](|d|)$ denote the communication time for transferring data item d (of size $|d|$) from machine $M[i]$ to machine $M[j]$ through their k-th dedicated virtual link during time frame $[Lst[i,j][k], Let[i,j][k]]$. $D[i,j][k](|d|)$ includes all the various hardware and software related components of the inter-machine communication overhead (e.g., network latency and the time for data format conversion between $M[i]$ and $M[j]$ when necessary). Machines $M[i]$ and/or $M[j]$ may be intermediate nodes for transferring d rather than the original source or the final destination node of d.

Suppose n is the number of data items with distinctive names (identifiers) available in the corresponding communication system M. Let $\Delta = \{\delta[0], \delta[1], ..., \delta[n-1]\}$ be the set of these data items, where each $\delta[i]$ is unique. For example, a weather map of Europe generated at 2 p.m. would have a different name than a weather map of the same region generated at 6 p.m. A data location table that specifies the initial locations of the n available data items can be constructed with the following notation. Let $N\delta[i]$ be the number of different machines that the data item $\delta[i]$ is located at initially. $Source[i,j]$ denotes the j-th initial source location of the data item $\delta[i]$ (with no implied significance for the ordering of the sources), where $0 \le i < n$, $0 \le j < N\delta[i]$, and $0 \le Source[i,j] < m$. Also, $\delta st[i,j]$ denotes the starting time at which $\delta[i]$ is available at its j-th initial source location.

Suppose ρ is the number of the requested data items with distinctive names (identifiers) in the corresponding communication system M, where $0 \le \rho \le n$. Let $Rq = \{Rq[0], Rq[1], ..., Rq[\rho-1]\}$ be the set of the requested data items. Each $Rq[j]$ $(0 \le j < \rho)$ is the name of a data item and there must exist i $(0 \le i < n)$, such that $Rq[j] = \delta[i]$. Each $Rq[j]$ must be unique. A data request table that specifies the requests of data items can be constructed with the following notation. Let $Nrq[j]$ denote the number of different requests for $Rq[j]$. $Request[j,k]$ denotes the machine from which the k-th request for data item $Rq[j]$ originates (with no implied order among the requests), where $0 \le j < \rho$, $0 \le k < Nrq[j]$, and $0 \le Request[j,k] < m$. Also, $Rft[j,k]$ denotes the finishing time (or deadline) after which the data item $Rq[j]$ on its k-th requesting location is no longer useful (e.g., data items may be needed before a specific time when certain decisions must be made). Suppose the priority of each data request is between 0 and P, where P is the highest priority possible (i.e., a member of the class of most important requests). $Priority[j,k]$ denotes the priority for the data request of the data item $Rq[j]$ on its k-th requesting location.

Assume that the scheduling procedure of the communication steps starts at time 0. Let $S = \{S_0, S_1, ..., S_{\sigma-1}\}$ denote a set of σ distinct schedules for the communication steps of transmitting requested data items. Consider a specific schedule S_h, where $0 \le h < \sigma$. The k-th request for data item $Rq[j]$ is satisfiable with respect to S_h if $Rq[j]$ can be obtained by the requesting machine, $M[Request[j,k]]$, before the deadline, $Rft[j,k]$. Let $Srq[S_h]$ denote the set of two-tuples $\{(j,k) \mid k\text{-th request of the data item } Rq[j] \text{ is satisfiable}\}$. Suppose $W[i]$ $(0 \le i \le P)$ denotes the relative weight of the i-th priority. These weightings allow system administrators to specify the relative importance of priority α data request versus priority β data request, where $0 \le \alpha,\beta \le P$. The effect, $E[S_h]$, of the scheduling scheme S_h is defined as

$$E[S_h] = -\left(\sum_{(j,k)\in Srq[S_h]} W[Priority[j,k]]\right).$$

The global optimization criterion is defined as

$$\min_{0 \le h < \sigma} E[S_h].$$

Given this mathematical model, the objective of data

staging in this paper for a specific communication system is to find an S_h such that $E[S_h]$ is minimized (i.e., the total sum of the weighted priorities of all satisfiable data requests with respect to S_h is maximized). It should be noted that an exhaustive set of schedules is not created in this research.

4. Data staging heuristic

4.1. Overview

The heuristic for solving the data staging problem introduced in this section is based on Dijkstra's algorithm for solving the multiple-source shortest-path problem on a weighted and directed graph [5]. In Subsection 4.2, background information about Dijkstra's algorithm is provided. It is summarized from the material in [5]. Only the relevant part with respect to the data staging heuristic is discussed in detail. Subsection 4.3 presents the heuristic that is used to schedule the communication steps. The definition of the shortest-path estimate for Dijkstra's algorithm and the cost function of a communication step for local optimization in the heuristic are defined. The complexity analysis of the heuristic is provided in Subsection 4.3 as well.

4.2. Dijkstra's algorithm

The multiple-source shortest-path problem is defined as follows. Let $\tilde{G} = (\tilde{V}, \tilde{E})$, be a weighted and directed graph, where $\tilde{V}$ is a set of vertices and $\tilde{E}$ is a set of edges. For a single data item, suppose $V_S \subset \tilde{V}$ denotes a set of source vertices and $V_D \subset \tilde{V}$ denotes a set of destination vertices. The goal is to find a shortest path from any source vertex $v_s \in V_S$ to every destination vertex $v_d \in V_D$. The length of a path is the sum of the weights of its constituent edges. For an established path, an immediate predecessor vertex $\pi[v]$ of each vertex $v \in \tilde{V}$ is either another vertex or NIL. Any multiple-source shortest-path algorithm needs to set $\pi[v]$ properly so that the chain of predecessors originating at a vertex $v_d \in V_D$ corresponds to a shortest path from any $v_s \in V_S$ to v_d.

The main technique used in this multiple-source shortest-path algorithm is relaxation. For each vertex $v \in \tilde{V}$, an attribute $d[v]$, which is referred to as a shortest–path estimate, is maintained. This $d[v]$ is an upper bound on the length of a shortest path from any source vertex $v_s \in V_S$ to $v_d \in V_D$. The relaxation procedure with respect to a directed edge from vertex u to vertex v consists of testing whether the length of the shortest path to v found so far can be decreased by going through u (with known $d[u]$) via that edge from u to v. If the above testing results in a positive answer, a relaxation step decreases the value of the shortest-path estimate $d[v]$ (e.g., set $d[v]$ as $d[u]$ plus the weight of the edge from u to v) and updates v's immediate predecessor vertex as u. Relaxation is the only way by which the shortest-path estimate $d[v]$ and the predecessor vertex $\pi[v]$ can change.

Algorithms for solving a multiple-source shortest-path problem may differ in the way by which the edges are relaxed. In Dijkstra's algorithm, a set of vertices V_F (set to be V_S initially), whose final shortest paths from any $v_s \in V_S$ have already been determined, is maintained. The algorithm repeatedly selects the vertex $u \in \tilde{V} - V_F$ with the minimum shortest-path estimate, inserts u into V_F, and relaxes any edge from u to v by updating $d[v]$ and $\pi[v]$ properly, for all $v \in \tilde{V} - V_F$. The algorithm terminates when $V_F = \tilde{V}$.

4.3. Heuristic and complexity analysis

All necessary communication steps are scheduled by the data staging heuristic presented in this subsection. The heuristic utilizes the following three strategies collectively:

(i) Choose an order of data transfers that results in a set of data items being available on intermediate nodes earlier. This set of data items will be chosen based on some criteria that will cause the ordering to satisfy more data requests.

(ii) Maximize the sum of the priorities of the potentially satisfiable data requests.

(iii) Consider the urgency of a request as its deadline approaches.

A data staging heuristic that has a well-balanced set of local optimization criteria using the above three strategies should intuitively perform well. The multiple-source shortest-path algorithm based heuristic presented in this subsection is built upon Dijkstra's algorithm and utilizes all of the above three strategies. The heuristic iteratively picks which data item to transfer next constrained by a cost function. Each iteration of the heuristic involves: (a) running Dijkstra's algorithm for each data request individually, (b) determining the ''cost'' to transfer a data item to its successor in the shortest path, (c) picking the lowest cost data request and transferring that data item, (d) updating system parameters to reflect resources used in (c), and (e) repeating (a) through (d) until there are no more satisfiable requests in the system.

1. For all k ($0 \le k < Nl[s,r]$), do the following steps.
2. if ($A_T[i,s] > Lst[s,r][k]$) {
   /* if $Rq[i]$ is obtained by $M[s]$ after $L[s,r][k]$ is available */
3. if ($[A_T[i,s] + D[s,r][k](|Rq[i]|)] \le Let[s,r][k]$)
   /* if the available time interval is long enough to transfer $Rq[i]$ via $L[s,r][k]$ */
4. if ($Cap[r](A_T[i,s]) \ge |Rq[i]|$)
   /* if $M[r]$ has enough storage capacity for $Rq[i]$ */
   $A_L[s,r][i][k] = A_T[i,s] + D[s,r][k](|Rq[i]|)$
   /* find ‘‘available time’’ using this link*/
5. } else { /* if $Rq[i]$ is obtained by $M[s]$ before $L[s,r][k]$ is available */
6. if ($[Lst[s,r][k] + D[s,r][k](|Rq[i]|)] \le Let[s,r][k]$)
   /* if the available time interval is long enough to transfer $Rq[i]$ via $L[s,r][k]$ */
7. if ($Cap[r](Lst[s,r][k]) \ge |Rq[i]|$)
   /* if $M[r]$ has enough storage capacity for $Rq[i]$ */
   $A_L[s,r][i][k] = Lst[s,r][k] + D[s,r][k](|Rq[i]|)$ }
   /* find ‘‘available time’’ using this link */
8. if ($A_T[i,r] > \min_{0 \le k < Nl[s,r]} \{A_L[s,r][i][k]\}$) {
   /* if smaller shortest-path estimate is found */
9. $A_T[i,r] = \min_{0 \le k < Nl[s,r]} \{A_L[s,r][i][k]\}$
   /* update the shortest-path estimate for $V[r]$ */
10. $k_l = k$ giving minimum in step 9 /* record the virtual link used */
    } /* $\underline{k_l}$ is the argument k that minimizes $A_L[s,r][i][k]$ */

Figure 1: Pseudocode for implementing the relaxation step.

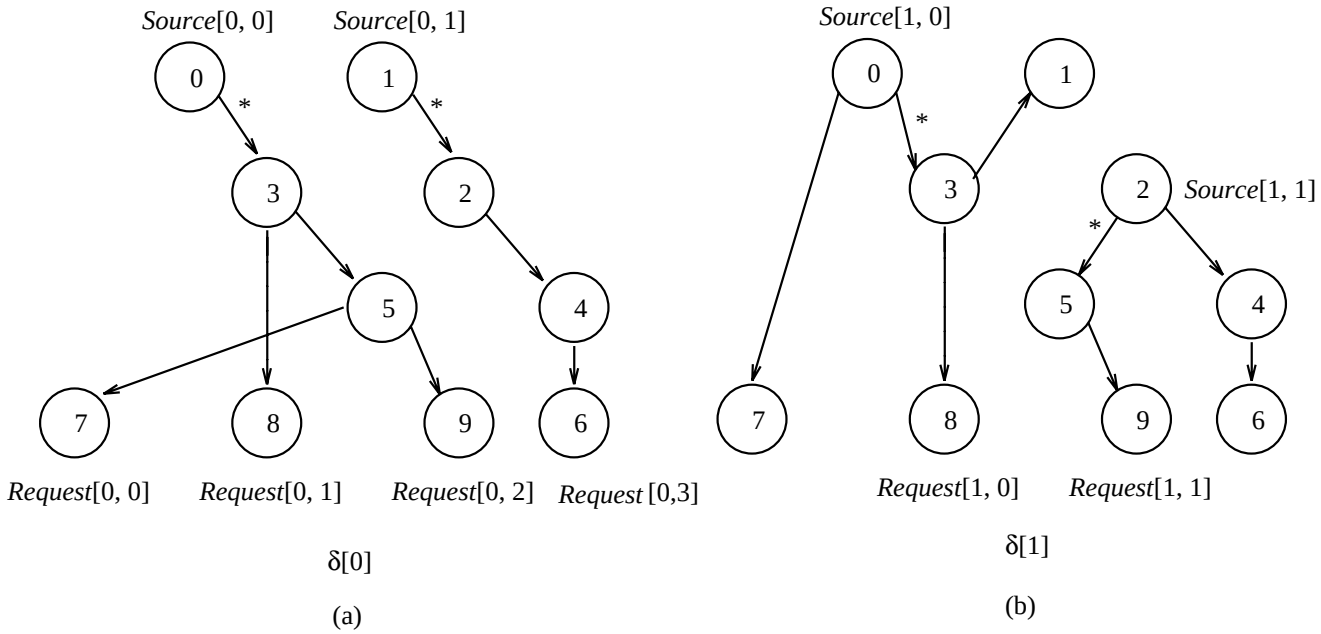

Figure 2: An example communication system that requests (a) δ[0] and (b) δ[1].

In some cases, Dijkstra's algorithm would not need to be executed each iteration for a particular data request, i.e. if the links that the request uses are not affected by updating the system parameters. Future versions of the heuristic are planned to take advantage of not having to recalculate all the shortest paths during each iteration.

For each requested data item $Rq[i_1]$ ($0 \le i_1 < \rho$), as stated in Section 3, there exists i_2 ($0 \le i_2 < n$), such that $Rq[i_1] = \delta[i_2]$. Suppose $V_S[i_1]$ is the set of source vertices corresponding to $Rq[i_1]$ in G_{nt}. Then $V[k] \in V_S[i_1]$ ($0 \le k < m$) if and only if there exists j ($0 \le j < N\delta[i_2]$), such that $Source[i_2, j] = k$. This means that $V_S[i_1]$ contains all the vertices of G_{nt} that correspond to the machines that are the initial locations of $Rq[i_1]$ (i.e., $\delta[i_2]$). Suppose $V_D[i_1]$ is the set of destination vertices corresponding to $Rq[i_1]$ in G_{nt}. Then $V[k] \in V_D[i_1]$ ($0 \le k < m$) if and only if there exists j ($0 \le j < Nrq[i_1]$), such that $Request[i_1,j] = k$. This means that $V_D[i_1]$ contains all the vertices of G_{nt} that correspond to the machines that are making data requests for $Rq[i_1]$. Let the length of a path from a source vertex $v_s \in V_S[i_1]$ to a destination vertex $v_d \in V_D[i_1]$ be defined as the difference between the earliest possible time when data item $Rq[i_1]$ can be available on the machine corresponding to v_d via the machines and the virtual communication links along the path and the time the data item is available on $v_s \in V_S[i_1]$ (this time can be calculated using the various parameters defined in Section 3). With the above defined $G_{nt} = (V,E)$, $V_S[i]$, and $V_D[i]$, where $0 \le i < \rho$, a separate multiple-source shortest-path problem is well defined for each of the ρ different requested data items in the context of the data staging problem.

Readers should notice that it may be impossible to use the individually shortest paths to all $v_d \in V_D[i]$ for each data item $\delta[i]$ ($0 \le i < \rho$) due to possible communication link contention in the network for transferring different data items. Also, a multiple-source shortest-path algorithm for G_{nt} only attempts to minimize the time when a given requested data item is obtained by its corresponding requesting locations. But as clearly stated in Section 1 and Section 3, other criteria like request deadlines and the priorities of the satisfiable data requests must be taken into account as well.

As reviewed in Subsection 4.2, the way of defining $d[v]$ (i.e., the shortest-path estimate) based on the known $d[u]$ and information about all edges from u to v (e.g., their weights) is essential for applying Dijkstra's algorithm to derive the data staging heuristic. For $G_{nt} = (V, E)$, $V_S[i]$, and $V_D[i]$ ($0 \le i < \rho$), let the shortest-path estimate of $V[j]$ (corresponding to machine $M[j]$) for requested data item $Rq[i]$ be defined as $A_T[i,j]$. This is the earliest possible available time found so far when the requested data item $Rq[i]$ is obtained by machine $M[j]$ with the consideration of the availability of the virtual links and the available memory capacity of machine $M[j]$. Suppose $\delta[k] = Rq[i]$ ($0 \le k < n$). Initially, $A_T[i,Source[k,q]] = \delta st[k,q]$ for all $0 \le q < N\delta[k]$. That is, for all initial source locations $v_s \in V_S[i]$, their shortest-path estimates are the starting times when $\delta[k]$ is available at those nodes.

It is assumed that each machine can send different data items (each via a different link) to its neighboring machines in the network simultaneously. Future work will relax this assumption. Suppose $A_T[i,s]$ and $A_T[i,r]$ are known and there are virtual links $L[s,r][k]$ ($0 \le k < Nl[s,r]$) from $M[s]$ to $M[r]$. Let $A_L[s,r][i][k]$ denote the time when the requested data item $Rq[i]$ can be available on machine $M[r]$ via fetching the copy from $M[s]$ through the virtual link $L[s,r][k]$. The relaxation step with respect to the edges from $V[s]$ to $V[r]$ based on the known $A_T[i,s]$ and $A_T[i,r]$ is implemented by the C-style pseudocode in Figure 1.

As illustrated by Step 10 in the above pseudocode, the exact virtual link $L[s,r][k_l]$ used for updating the shortest-path estimate of $V[r]$ needs to be recorded, due to the existence of multiple virtual links between $M[s]$ and $M[r]$. Thus, the predecessor vertex $\pi[r]$ in the usual description of the Dijkstra's algorithm is extended as a predecessor field and is defined as a two-tuple (s,k_l) in this data staging heuristic, where s records the source machine and k_l records the virtual link used. At this stage, the information about the availability of link $L[s,r][k_l]$ does not need to be updated because each execution of Dijkstra's algorithm is for a single given data item and the transfer of any specific data item only needs to use that link once.

For each $Rq[i]$ ($0 \le i < \rho$) individually, based on the above defined shortest-path estimate, the shortest paths for all $v_d \in V_D[i]$ can be generated with respect to G_{nt} defined in Section 3. Consider an example shown by Figure 2, with a communication system consisting of ten machines. Also, to simplify the presentation, suppose that there is at most one virtual link between any pair of machines. There are two data items $\delta[0]$ and $\delta[1]$ being requested. Machines 0 and 1 are the sources for $\delta[0]$ ($Source[0,0] = 0$, $Source[0,1] = 1$), and machines 0 and 2 are the sources for $\delta[1]$ ($Source[1,0] = 0$, $Source[1,1] = 2$). The destinations for $\delta[0]$ are machines 6, 7, 8, and 9 ($Request[0,0] = 7$, $Request[0,1] = 8$, $Request[0,2] = 9$, $Request[0,3] = 6$), and the destinations for $\delta[1]$ are machines 8 and 9 ($Request[1,0] = 8$, $Request[1,1] = 9$). Suppose that the shortest paths generated corresponding to $\delta[0]$ individually are shown in Figure 2(a), and the

shortest paths generated corresponding to δ[1] individually are shown in Figure 2(b). Note that Figure 2(a) and 2(b) correspond to the same machine suite, and that the links shown in Figure 2(a) and 2(b) collectively are a subset of all the inter-machine links. This example is used throughout the rest of this section to illustrate the proposed data staging heuristic.

A vertex $V[r]$ (corresponding to machine $M[r]$ in G_{nt}) is defined as a contingent vertex with respect to $V_S[i]$, if $V[r] \in V - V_S[i]$ and there is an edge entering into $V[r]$ that starts from a vertex in $V_S[i]$. For the example shown in Figure 2(a), $V[2]$ and $V[3]$ are contingent vertices with respect to $V_S[0] = \{V[0], V[1]\}$. For the example shown in Figure 2(b), $V[3]$, $V[4]$, $V[5]$, and $V[7]$ are contingent vertices with respect to $V_S[1] = \{V[0], V[2]\}$. $M[r]$ may be an intermediate machine along some paths from any source vertex in $V_S[i]$ to a set of destination vertices in $V_D[i]$. Suppose this set of destination (requesting) vertices associated with a data item $\delta[i]$ and a contingent vertex $V[r]$ is defined as $Drq[i,r]$. For any element $drq \in Drq[i,r]$, drq is an integer ($0 \le drq < Nrq[i]$), such that $V[Request[i,drq]] \in V_D[i]$. For the example shown in Figure 2(a), for $V[3]$ (corresponding to $M[3]$), $Drq[0,3] = \{0, 1, 2\}$ corresponding to requests for $\delta[0]$ from $M[7]$, $M[8]$, and $M[9]$, respectively. For $V[2]$ (corresponding to $M[2]$), $Drq[0,2] = \{3\}$ corresponding to the request for $\delta[0]$ from $M[6]$. Similarly, for the example shown in Figure 2(b), $Drq[1,3] = \{0\}$ corresponding to the request of $\delta[1]$ from $M[8]$, $Drq[1,5] = \{1\}$ corresponding to the request of $\delta[1]$ from $M[9]$, and $Drq[1,4] = Drq[1, 7] = \emptyset$.

After applying the multiple-source shortest-path algorithm for each requested data item $Rq[i]$ ($0 \le i < \rho$), individually, with the associated $V_S[i]$, $V_D[i]$, and G_{nt}, ρ sets of shortest paths are generated, one for each of the ρ different requested data items. Suppose the predecessor field of $V[r]$ corresponding to the shortest paths generated for $Rq[i]$ is (s,k). Each contingent vertex $V[r]$ that has an incident edge $E[s,r][k]$ from $V[s]$ in $V_S[i]$ and its associated $Drq[i,r] \neq \emptyset$ corresponds to a valid next communication step to be scheduled. This communication step is the one that specifies transferring $Rq[i]$ from $M[s]$ to $M[r]$ via link $L[s,r][k]$. For the example shown in Figure 2, there are four valid communication steps that can be scheduled (specified by asterisks). But different valid communication steps may have conflicting resource requirements (e.g., $M[0]$ cannot send $\delta[0]$ and $\delta[1]$ to $M[3]$ simultaneously due to network conflict for the example shown in Figure 2). Thus, a local optimization criterion is used to select one of the valid communication steps to be scheduled. The next paragraphs derive a cost function that is used as the basis for making this selection. The cost function will involve considerations of satisfiability, effective priority, and urgency as defined later.

Suppose $\tilde{A_T}[i,j]$ ($j \in Drq[i,r]$) denotes the time when $Rq[i]$ is received and available at its corresponding j-th requesting location. A satisfiability function $Sat[i,r](j)$, is defined as: $Sat[i,r](j) = 1$, if $\tilde{A_T}[i,j] \le Rft[i,j]$; and 0, if $\tilde{A_T}[i,j] > Rft[i,j]$. Note that this shortest path involves passing through vertex $V[r]$, so if a request in $Drq[i,r]$ is not satisfied, there is not other path that will cause it to be satisfied. That is, if $Sat[i,r](j) = 1$, then the data request of $Rq[i]$ from machine $M[Request[i,j]]$ is satisfied. Otherwise, the corresponding data request is not satisfied. As an example of the definition of $Sat[i,r](j)$, consider the shortest paths generated by selecting first the valid communication step for transferring $\delta[0]$ from $M[0]$ to $M[3]$ in the example shown by Figure 2(a). Suppose $T_0 = \tilde{A_T}[0,0] = A_T[0,7]$, $T_1 = \tilde{A_T}[0,1] = A_T[0,8]$, and $T_2 = \tilde{A_T}[0,2] = A_T[0,9]$. Also, assume that $T_0 \le Rft[0,0]$, $T_1 > Rft[0,1]$, and $T_2 \le T_f[0,2]$. Then, $Sat[0,3](0) = 1$, $Sat[0,3](1) = 0$, and $Sat[0,3](2) = 1$. T_0, T_1, and T_2 are calculated during the last execution of the Dijkstra's algorithm with respect to $\delta[0]$.

Suppose $Efp[i,j]$ denotes the effective priority for the data request of $Rq[i]$ from its j-th requesting location, where $Efp[i,j] = Sat[i,r](j) \times W[Priority[i,j]]$. Suppose $Urgency[i, j]$ denotes the urgency for the data request of $Rq[i]$ from its j-th requesting location, where $Urgency[i,j] = - Sat[i,r](j) \times (Rft[i,j] - \tilde{A_T}[i,j])$, where smaller $Urgency[i]$ implies that it is less urgent to transfer $Rq[i]$ to the j-th requesting location. Suppose $W_E \ge 0$ is the relative weight for the effective priority factor and $W_U \ge 0$ is the relative weight for the urgency factor in the scheduling. Readers should notice that (a) applying Dijkstra's algorithm to obtain $A_T[i,r]$ through shortest paths, (b) maximizing $Efp[i,j]$, and (c) maximizing $Urgency[i,j]$ follow the three strategies for designing data relocation heuristics recommended in (i), (ii), and (iii), respectively, at the beginning of this subsection. The cost, $Cost[s,r][i,j][k]$, for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$ is defined as:

$$Cost[s,r][i,j][k] = - W_E Efp[i,j] - W_U Urgency[i,j].$$

The next chosen communication step should be the one that has the smallest associated cost among all valid next communication steps for transferring all $Rq[i]$ where $0 \le i < \rho$, and $Sat[i,r]$ is not 0 for all r. If $Sat[i,r]$ is 0 for all r, that request receives no resources and the data does

not move from its current locations. The request is not eliminated from the network. Currently the heuristic is applied to a static system, as this constraint is loosened and a dynamic system is explored, links might become available that would facilitate the delivery of an otherwise unsatisfiable request. So requests that are at one point in time unsatisfiable, might become satisfiable at a later point in time. For the valid communication step for transferring $\delta[0]$ from $M[0]$ to $M[3]$ shown in Figure 2(a), $Efp[0,0] = W[Priority[0,0]]$ and $Urgency[0,0] = -(Rft[0,0] - T_0)$.

The rationale for choosing the above cost for local optimization is as follows. First, only a valid next communication step whose associated $Sat[i,r]$ is not 0 for all r will facilitate satisfying data request(s). $Cost[s,r][i,j][k]$ attempts to maximize the total priority of the satisfiable data requests. Furthermore, in order to satisfy as many data requests as possible, intuitively it is necessary to transfer a specific data item to the requesting locations whose deadlines are close to expire. This intuition is captured by the inclusion of the urgency factor. Thus, collectively with the consideration of the total priority of the satisfiable data requests and the urgency of those data requests in this local optimization step, this data staging heuristic should generate a suboptimal S_h that reasonably achieves the global optimization criterion presented in Section 3.

Suppose that the current chosen communication step for S_h according to $Cost[s,r][i,j][k]$ is to transfer the requested data item $Rq[i]$ from $M[s]$ to $M[r]$ via link $L[s,r][k]$ during the time interval between t_0 and t_1. Before repeating the above multiple-source shortest-path based heuristic for determining the next communication step of S_h, the following four categories of information need to be updated.

(1) Update the network topology graph G_{nt} — Delete the edge $E[s,r][k]$ corresponding to link $L[s,r][k]$ in G_{nt}. Also, add up to two more edges from $V[s]$ to $V[r]$ that correspond to two more virtual links. One with the link starting time as $Lst[s,r][k]$ and link ending time as t_0, and another with link starting time as t_1 and link ending time as $Let[s,r][k]$. If $t_0 = Lst[s,r][k]$ and/or $t_1 = Let[s,r][k]$, then the above first and/or second additional virtual links are not needed.

(2) Update $Cap[r](t)$ — $Cap[r](t)$ is decremented by $|Rq[i]|$, where $|Rq[i]|$ is the size of $Rq[i]$, because machine $M[r]$ keeps a copy of the data item $Rq[i]$ (e.g., $Cap[3](t)$ is decremented by $|\delta[0]|$ for the communication step shown in Figure 2(a)).

(3) Update the set of source vertices $V_S[i]$ for $Rq[i]$ — $V_S[i] = \{\text{latest } V_S[i]\} \cup \{V[r]\}$. Also, set the starting time when $Rq[i]$ is available on $M[r]$ as $A_T[i,r]$. For the example communication step for transferring $\delta[0]$ from $M[0]$ to $M[3]$ shown in Figure 2(a), $M[3]$ becomes a source of $\delta[0]$ and its starting time is $A_T[0,3]$.

(4) Update the garbage collection related information — The copy of $Rq[i]$ on machine $M[r]$ is used as an intermediate copy for forwarding $Rq[i]$ to some other machines. Suppose this set of machines is defined as $Im[i,r]$. $Im[i,r]$ can be determined by tracing the shortest paths generated above for each $v_d \in V_D[i]$. For the example communication step of transferring $\delta[0]$ from $M[0]$ to $M[3]$ shown in Figure 2(a), by tracing the shortest-paths generated for $V[7]$, $V[8]$, and $V[9]$, $Im[0,3]$ can be determined as $\{5, 8\}$. After all those machines in $Im[i,r]$ have received the copy of $Rq[i]$ from $M[r]$, the copy of $Rq[i]$ on machine $M[r]$ can be deleted. Suppose the time for the last machine in $Im[i,r]$ to receive its copy of $Rq[i]$ from $M[r]$ is $R_T[i,r]$, then at this time, $Cap[r](t)$ is incremented by $|Rq[i]|$. This above procedure implements the garbage collection scheme in data staging.

A single iteration of this multiple-source shortest-path based heuristic starts at the step for generating shortest paths for all remaining data requests (based on the Dijkstra's algorithm and the current system status (e.g., data source locations and link availability)). The iteration ends at the step for updating the system information described above. Then with the new $Cap[r](t)$, G_{nt}, and $V_S[i]$, execute the above multiple-source shortest-path based heuristic repeatedly to determine the rest of the communication steps in S_h. The heuristic terminates when all remaining data requests are not satisfiable.

For the complexity analysis of this multiple-source shortest-path based heuristic for determining one communication step in S_h, suppose that $|E|$ is the number of edges and $|V|$ is the number of vertices in the network topology graph G_{nt}. If a Fibonacci heap [5] is used to implement the priority queue, the worst case asymptotic complexity of Dijkstra's algorithm is $O(|E| + |V|\lg|V|)$. For the network topology graph G_{nt} terminology described in (2) of Section 3, $|E| = \sum_{0 \le i \ne j < m} Nl[i, j]$ and $|V| = m$. Because in the worst case it is necessary to apply the multiple-source shortest-path algorithm to all the requested data items $Rq[i]$ $(0 \le i < \rho)$, the worst case asymptotic complexity of this heuristic for determining one communication step in S_h is

$$O[\rho(m \lg m + \sum_{0 \le i \ne j < m} Nl[i,j])],$$

where ρ is the total number of requested data items defined in Section 3.

Given the heuristic approach presented in this section uses Dijkstra's algorithm in conjunction with a minimization criteria, it is called the Dijkstra/minimization heuristic. It is evaluated in the next section.

5. Simulation study

To perform the simulation study, network topologies and data requests must be generated, values for W_E and W_U must be determined, and other scheduling schemes need to be created to compare to the Dijkstra/minimization heuristic. Rather than just choosing one network topology and set of data requests, because one can not accurately reflect the changing data requests and network availability with one case, 40 test cases were generated and the Dijkstra/minimization heuristic was executed using each of these cases and the results were averaged. The data requests and the underlying communication systems were randomly generated over a set of parameters (corresponding to the notation introduced in Section 3) as specified below. All parameters are randomly generated with uniform distributions in predefined ranges representing systems in a BADD-like environment. The sources and requesting machines for all data items are also generated randomly. The test generation program guarantees that the generated communication system is strongly connected [5], such that there is a path consisting of physical transmission links between any pair of nodes in both directions.

These randomly generated patterns of data requests and the underlying communication systems are used for three reasons: (1) it is beneficial to obtain cases that can demonstrate the performance of the Dijkstra/minimization heuristic presented over a broad range of conditions; (2) a generally accepted set of data staging benchmark tasks does not exist; and (3) it is not clear what characteristics a ''typical'' data staging task would exhibit. Determining a representative set of data staging benchmark tasks remains an unresolved challenge in the research field of data staging and is outside the scope of this paper.

Finding optimal solutions to data staging tasks with realistic parameter values are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the above Dijkstra/minimization heuristic with those found by exhaustive searches in which optimal answers can be obtained by enumerating all the possible schedules of communication steps. Also, to the best of the authors' knowledge, there is no other work presented in the open literature that addresses the data staging problem and presents a heuristic for solving it (based on a similar underlying model). Thus, there is no other heuristic for solving the same problem with which to make a direct comparison of the Dijkstra/minimization heuristic presented in this paper.

The performance of the Dijkstra/minimization heuristic is compared with two random-search based scheduling procedures. The only difference between the first random procedure and the above Dijkstra/minimization heuristic is that, instead of choosing a valid communication step using $Cost[s,r][i,j][k]$ as discussed for the Dijkstra/minimization heuristic, the Dijkstra random heuristic randomly chooses an arbitrary valid communication step to schedule.

The second random-search based scheduling procedure performs Dijkstra once for each requested data item, assuming it is the only requested item in the network. Then the paths through the network are scheduled for each data item, finishing $Rq[i]$ before $Rq[i+1]$. If a conflict arises, i.e., the time frame in which a particular link was originally scheduled by the independent Dijkstra's for a given request is unavailable, the request is dropped and not satisfied. This approach is referred to as single Dijkstra random because Dijkstra's algorithm is only executed once for each data item.

Each test set for the results shown in Figures 3 and 4 was generated with the following parameters. The number of machines in the communication system is between ten and twenty. Each machine has between 10MB to 20GB memory storage capacity. The maximum outbound degree of a machine $M[i]$ (i.e., the number of machines that $M[i]$ can transfer data items to directly through physical transmission links) is five. There are at most two physical transmission links between any two machines (there can be none). The total number of data requests is one to ten times the number of machines in the system. There can be up to three sources and three destinations for each of the data requests. Each data item size ranges from 10KB to 1MB. The priority of each data item is 1, 5, or 10, and the relative weight of a priority is equal to the priority itself (i.e., $W[i] = i$, for all $0 \le i \le P$). Each request has its own priority. The bandwidth of each physical transmission link is between 10KB/sec and 10MB/sec. The link starting and ending times and the data item starting (available) and finishing times (or deadlines) are modeled building on information about the underlying communication infrastructures and data

request patterns in [12, 13]. These parameters were used as they capture the information about the network that is necessary to show the functionality of the heuristic over a variety of network configurations.

Let the E−U ratio be W_E/W_U. As shown by the cost function $Cost[s,r][i,j][k]$ introduced in Subsection 4.3, the E-U ratio may affect the performance of the Dijkstra/minimization heuristic. Figures 3 and 4 show the performance of the Dijkstra/minimization heuristic when the E-U ratio ranges from 0.001 to 1000 (shown by dashed-dotted lines). Figure 3 uses the average sum of the weighted priorities of the satisfiable data requests and Figure 4 uses the average number of the satisfiable data requests. Both are averaged over 40 randomly generated test cases.

In this study, the Dijkstra/random heuristic is executed ten times for each of 40 randomly generated cases (the same 40 cases as used for Dijkstra/minimization). Then, its average sum of the weighted priorities of the satisfiable data requests and its average number of the satisfiable data requests over all ten runs for all 40 cases are calculated. As shown in Figures 3 and 4, the Dijkstra/minimization heuristic consistently outperforms the Dijkstra/random heuristic (shown by the dotted lines). The difference shows the advantage of using a minimization criteria to resolve conflicting demands for a link.

Also shown in Figures 3 and 4, single Dijkstra random performs poorly (shown by the pluses) compared to the Dijkstra/minimization heuristic and the Dijkstra/random heuristic. The difference between the Dijkstra/minimization and the single Dijkstra random shows the advantage of the process of interleaving link demands from multiple data items.

Solid lines in Figures 3 and 4 show the average sum of the weighted priorities of all data requests and the average number of all data requests, respectively. Readers should notice that not all data requests for a randomly generated test case can be satisfied even with the optimal scheduling scheme for data staging. The asterisks show the average of all the requests that could be satisfied if each had exclusive use of the network, i.e., it was the only request in the network. Thus, the asterisks in Figures 3 and 4 represent a loose upper bound for the performance of any data staging heuristic. The difference between the solid line and the asterisks represent those requests that could never be satisfied due to insufficient resources of the network, i.e., links or storage. This information is useful as a prediction tool to determine changes to the network that would increase the number of satisfied requests.

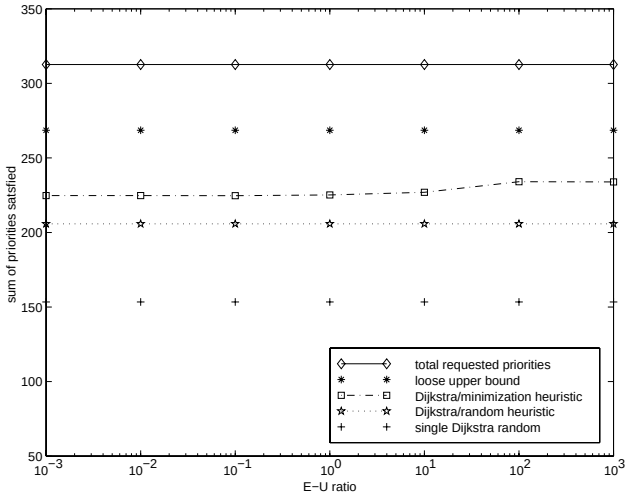

Figure 3: Comparison in terms of the average sum of the priorities of the satisfiable data requests for a lightly loaded network.

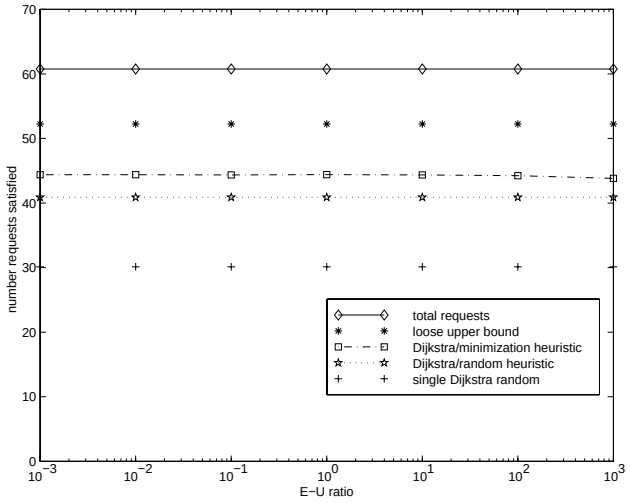

Figure 4: Comparison in terms of the average number of the satisfiable data requests for a lightly loaded network.

Figure 5 shows the average sum of the priorities satisfied for more heavily congested network topologies and Figure 6 shows the average of the number of requests satisfied. The network topologies that were used for these cases had fewer nodes in the network (ten to twelve), but had more requests (20 to 40 times the number of nodes), as well as one to five different sources and one to five different requesting machines. This

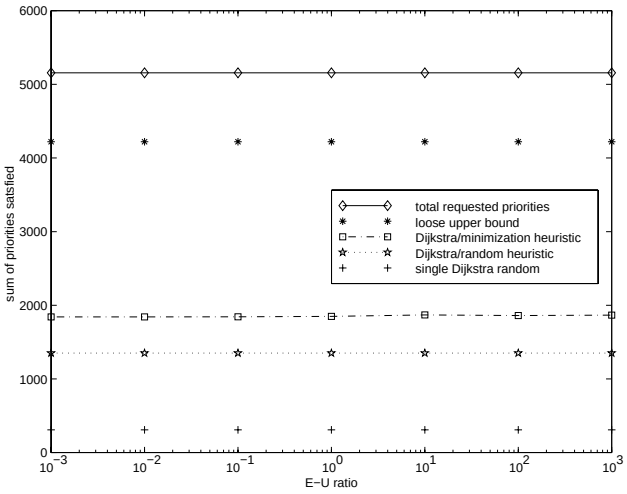

Figure 5: Comparison in terms of the average sum of the priorities of the satisfiable data requests for a heavily congested network.

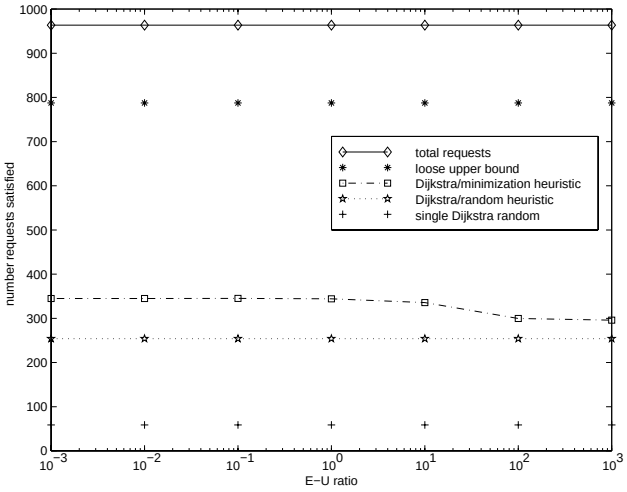

Figure 6: Comparison in terms of the average number of the satisfiable data requests for a heavily congested network.

caused the number of total requested data items, and the total priorities requested, to be at least an order of magnitude higher than in our previous experiments. The E-U ratio was again varied between 0.001 and 1000 to see what effect this would have on the results. It can be observed from Figures 3 and 5 that varying the E-U ratio has only a small impact, and that either $Efp[i,j]$ or $Urgency[i,j]$ by itself would be a sufficient criterion. As with the lightly loaded case, the scheduling the Dijkstra/minimization heuristic created for the heavily congested network is better than that of single Dijkstra random, and that of Dijkstra/random heuristic.

6. Summary and future work

Data staging is an important data management problem for information systems. It addresses the issues of distributing and storing over numerous geographically dispersed locations both repository data and continually generated data. When certain data with their corresponding priorities need to be collected together at a site with limited storage capacities in a timely fashion, a heuristic must be devised to schedule the necessary communication steps efficiently.

A rigorous mathematical model was created to describe a simplified static version of the data staging problem. This model is a first attempt at addressing this problem. The Dijkstra/minimization heuristic was introduced in Section 4 to solve this version of the data staging problem. Section 5 presented the results of simulation testing that shows the performance of the proposed Dijkstra/minimization heuristic over the Dijkstra/random heuristic and single Dijkstra random for a class of data staging tasks.

There are many issues that must be resolved before a complete heuristic for solving the entire data staging problem can be presented. The mathematical model described in this paper serves as a starting point for a rigorous model for the general data staging problem and will evolve over time. Also, when dynamic scheduling is necessary, methods need to be devised to include run-time information into the selection criterion. Fault tolerance issues must be considered as well in order to build a robust heuristic. The results of this research and its extensions may impact web management procedures, as well as the DARPA BADD program.

7. Glossary of notation

$A_L[s,r][i][k]$: time when $Rq[i]$ can be available on machine $M[r]$ via fetching the copy from $M[s]$ through the virtual link $L[s,r][k]$

$A_T[i,j]$: the earliest possible time found so far when $Rq[i]$ is available on $M[j]$

$\tilde{A_T}[i,j]$: time when $Rq[i]$ is received at its corresponding j-th requesting location with respect to the generated shortest path

$Cap[i](t)$: available memory storage capacity of machine $M[i]$ at time t

$Cost[s,r][i,j][k]$: cost for transferring the requested data item $Rq[i]$ associated with the j-th destination, from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$

$|d|$: size of the associated data item d

$d[v]$: shortest-path estimate for vertex v in Dijkstra's algorithm

$D[i,j][k](|d|)$: communication time for transferring data item d with size $|d|$ from $M[i]$ to $M[j]$ through their k-th dedicated virtual communication link

Δ: set of data items available in the communication system

$\delta[i]$: i-th data item available in the communication system

$\delta st[i,j]$: starting time at which $\delta[i]$ is available at its j-th initial source location

$Drq[i,r]$: set of destination vertices associated with a data item $\delta[i]$ and a contingent vertex $V[r]$, $Drq[i,r] \subseteq \{0, 1, ..., Nrq[i] - 1\}$

E: set of edges in G_{nt} that corresponds to all virtual communication links among machines

$\tilde{E}$: set of edges in $\tilde{G}$

$|E|$: number of edges in the network topology graph G_{nt}

$Efp[i,j]$: effective priority for the data request of $Rq[i]$ from its j-th requesting location

$E[i,j][k]$: k-th direct edge from $V[i]$ to $V[j]$ in G_{nt}

$E[S_h]$: effect of the scheduling scheme S_h

$\tilde{G}$: a weighted and directed graph

G_{nt}: network topology graph of the communication system that illustrates the connectivity of the machines

$Im[i,r]$: set of machines that $M[r]$ will forward its copy of $Rq[i]$ to according to the generated shortest paths

k_l: the argument k that minimizes $A_L[s,r][i][k]$

$L[i,j][k]$: k-th direct virtual communication link from $M[i]$ to $M[j]$

$Let[i,j][k]$: link ending time when $L[i,j][k]$'s availability terminates

$Lst[i,j][k]$: link starting time when $L[i,j][k]$ becomes available

m: number of machines in the communication system

$M[i]$: i-th machine in the communication system, $0 \le i < m$

n: number of the data items with distinctive values available in the communication system

$N\delta[i]$: number of different machines that the data item $\delta[i]$ is located at initially

$Nl[i,j]$: total number of direct virtual communication links from $M[i]$ to $M[j]$

$Nrq[j]$: number of different machines where a request for $Rq[j]$ is initiated

P: highest priority possible and implies to be most important for any data request

$\pi[v]$: predecessor field (or predecessor vertex) of vertex v

$Priority[j,k]$: priority for the data request of the data item $Rq[j]$ on its k-th requesting location

$Request[j,k]$: k-th location of the request for data item $Rq[j]$, $0 \le Request[j,k] < m$

$Rft[j,k]$: finishing time (or deadline) after which the data item $Rq[j]$ on its k-th requesting location is no longer useful

ρ: number of the requested data items with distinctive values in the corresponding communication system

$Rq[j]$: j-th requested data item in the communication system

$R_T[i,r]$: time for the last machine in $Im[i,r]$ to receive its copy of $Rq[i]$ from $M[r]$

$Sat[i,r](j)$: satisfiability function associated with a the j-th requesting location for data item $\delta[i]$ and a contingent vertex $V[r]$

S_h: a specific schedule for the communication steps of transmitting requested data items

σ: number of distinct schedules for the communication steps of transmitting requested data items

$Source[i,j]$: j-th initial source location of the data item $\delta[i]$

$Srq[S_h]$: set of two-tuples $\{(j,k) \mid k$-th request of the data item $Rq[j]$ is satisfiable$\}$

$Urgency[i,j]$: urgency for the data request of $Rq[i]$ from its j-th requesting location

V: set of m vertices for G_{nt} that corresponds to m machines

$|V|$: number of vertices in the network topology graph G_{nt}

$\tilde{V}$: set of vertices in $\tilde{G}$

v_d: a specific destination vertex

V_D: set of destination vertices

$V_D[i]$: set of destination vertices corresponding to $Rq[i]$

V_F: set of vertices whose final shortest paths from any $v_s \in V_S$ have been determined during the execution of Dijkstra's algorithm

$V[i]$: i-th vertex of G_{nt} that corresponds to machine $M[i]$

v_s: a specific source vertex

V_S: set of source vertices

$V_S[i]$: set of source vertices corresponding to $Rq[i]$

$W[i]$: relative weight of the i-th priority

W_E: relative weight for the effective priority factor in the scheduling

W_U: relative weight for the urgency factor in the scheduling

References

[1] S. Acharya and S. B. Zdonik, ''An efficient scheme for dynamic data replication,'' Tech. Report CS-93-43, Dept. of Computer Science, Brown Univ., 1993, 25 pp.

[2] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, ‘‘Enhancing the web’s infrastructure: From caching to replication,’’ *IEEE Internet Computing*, Vol. 1, No. 2, March-April 1997, pp. 18-27.

[3] A. Bestavros, ‘‘WWW traffic reduction and load balancing through server-based caching,’’ *IEEE Concurrency*, Vol. 5, No. 1, January-March 1997, pp. 56-67.

[4] R. Chandrasekaran and A. Dauchety, ‘‘Location on tree networks: P-centre and n-dispersion problems,’’ *Mathematics of Operations Research*, Vol. 6, No. 1, February 1981, pp. 50-57.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, MA, 1990.

[6] G. Cornuejols, G. L. Nemhauser, and L. A. Wolsey, ‘‘Worst-case and probabilistic analysis of algorithms for a location problem,’’ *Operations Research*, Vol. 28, No. 4, July-August 1980, pp. 847-858.

[7] P. Danzig, R. Hall, and M. Schwartz, ‘‘A case for caching file objects inside internetworks,’’ Tech. Report CU-CS-642-93, Computer Science Dept., Univ. of Colorado, 1993, 15 pp.

[8] A. P. Hurter and J. S. Martinich, *Facility Location and The Theory of Production*, Kluwer Academic Publishers, Norwell, MA, 1989.

[9] P. C. Jones, T. J. Lowe, G. Muller, N. Xu, Y. Ye, and J. L. Zydiak, ‘‘Specially structured uncapacitated facility location problem,’’ *Operations Research*, Vol. 43, No. 4, July-August 1995, pp. 661-669.

[10] M. J. Lemanski and J. C. Benton, *Simulation for SmartNet Scheduling of Asynchronous Transfer Mode Virtual Channels*, Master’s Thesis, Dept. of Computer Science, Naval Postgraduate School, June 1997 (Advisor: D. Hensgen).

[11] I. D. Moon and S. S. Chaudhry, ‘‘An analysis of network location problems with distance constraints,’’ *Management Science*, Vol. 30, No. 3, March 1984, pp. 290-307.

[12] A. J. Rockmore, ‘‘BADD functional description,’’ Internal DARPA Memo, February 1996.

[13] SmartNet/Heterogeneous Computing Team, ‘‘BC2A/TACITUS/BADD integration plan,’’ Internal Report, August 1996.

[14] D. R. Shier, ‘‘A min-max theorem for p-center problems on a tree,’’ *Transportation Science*, Vol. 11, No. 3, August 1977, pp. 243-252.

[15] H. J. Siegel, H. G. Dietz, and J. K. Antonio, ‘‘Software support for heterogeneous computing,’’ in *The Computer Science and Engineering Handbook*, edited by Allen B. Tucker, Jr., CRC Press, Boca Raton, FL, 1997, pp. 1886-1909.

[16] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, ‘‘Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system,’’ *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8, No. 8, August 1997, pp. 857-871.

Biographies

Min Tan received his PhD degree from the School of Electrical and Computer Engineering at Purdue University, West Lafayette, Indiana, USA in 1997. He currently is with the technical staff of Cisco Systems, Inc. He attended Shanghai Jiao Tong University, Shanghai, People’s Republic of China, in 1988. In 1991, he went to Western Maryland College, Maryland, USA, and received a BA degree in Mathematics and Physics in 1993. In 1994, he received an MSEE degree from the School of Electrical Engineering at Purdue University. While at Purdue, he received the ‘‘Estus H. and Vashti L. Magoon Outstanding Teaching Assistant Award’’ in 1996. He also served as a program committee member for the 1998 Heterogeneous Computing Workshop. His research interests include data source management in heterogeneous computing, data staging issues for network communication, video compression and financial applications on parallel and distributed systems, and dynamic partitionability for reconfigurable parallel processing machines. He has authored or coauthored 15 technical papers in these and related areas. He is a member of IEEE, the IEEE Computer Society, and the Eta Kappa Nu honorary society.

Mitchell D. Theys is a PhD student and Research Assistant in the School of Electrical and Computer Engineering at Purdue University. He received a Bachelor of Science in Computer and Electrical Engneering from the school of Electrical Engineering in 1993 with Highest Distinction from Purdue University, and an MSEE from the school of Electrical Engineering at Purdue University in 1996. He has received a Benjamin Meisner Fellowship from Purdue University for the 1996-1997 academic year, and an Intel Graduate Fellowship for the 1997-1998 academic year. He is a member of the Eta Kappa Nu honorary society, IEEE, and IEEE Computer Society. He was elected President of the Beta Chapter of Eta Kappa Nu at Purdue University and has held several various offices during his stay at Purdue. He has held positions with Compaq Computer Corporation, S&C Electic Company, and Lawrence Livermore National Laboratory. His research interests include design of single chip parallel machines, heterogeneous computing, parallel processing, and software/hardware design.

H. J. Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE (1990) and a Fellow of the ACM (1998). He received BS degrees in both electrical engineering and management (1972) from MIT, and the MA (1974), MSE (1974), and PhD degrees (1977) from the Depart-

ment of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing* (second edition 1990). He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing* (1989-1991), and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* (1993-1996) and the *IEEE Transactions on Computers* (1993-1996). He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Prof. Siegel's heterogeneous computing research includes modeling, mapping heuristics, and minimization of inter-machine communication. He is the Principal Investigator of a joint ONR-DARPA/ISO grant to design efficient methodologies for communication in the heterogeneous environment of the Battlefield Awareness and Data Dissemination (BADD) program. He is an Investigator on the MSHN project, supported by the DARPA/ITO Quorum program to create a management system for a heterogeneous network of machines.

Prof. Siegel's other research interests include parallel algorithms, interconnection networks, and the PASM reconfigurable parallel machine. His algorithm work involves minimizing execution time by exploiting architectural features of parallel machines. Topological properties and fault tolerance are the focus of his research on interconnection networks for parallel machines. He is investigating the utility of the dynamic reconfigurability and mixed-mode parallelism supported by the PASM design ideas and the small-scale prototype.

Noah B. Beck is pursuing an MSEE degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is data staging in heterogeneous networks. He has held many positions supporting Siemens Stromberg-Carlson's technical support staff, and has also worked as a Design Engineer in one of Intel Corporation's microprocessor design groups. Noah received his BS degree from Purdue University in 1997 in Computer Engineering. His research interests include parallel computing, computer architecture and organization, and heterogeneous computing. He is an active member of the Eta Kappa Nu honorary society.

Michael Jurczyk studied Electrical Engineering at Purdue University and the University of Bochum, Germany, where he received his Diploma in 1990. He obtained his PhD in Electrical Engineering from the University of Stuttgart, Germany, in 1996, where he studied parallel simulation and performance issues of interconnection networks. In 1996, he was a visiting assistant professor at the School of Electrical and Computer Engineering at Purdue University. Currently, he is an assistant professor at the Computer Engineering and Computer Science Department at the University of Missouri - Columbia. His research interests include parallel and distributed systems, interconnection networks for parallel and communication systems, ATM-networking, and networked multimedia.