A Mathematical Model and Scheduling Heuristics for Satisfying Prioritized Data Requests in an Oversubscribed Communication Network

Mitchell D. Theys, *Member*, *IEEE*, Min Tan, Noah B. Beck, H.J. Siegel, *Fellow*, *IEEE*, and Michael Jurczyk, *Member*, *IEEE*

Abstract—Providing up-to-date input to users' applications is an important data management problem for a distributed computing environment, where each data storage location and intermediate node may have specific data available, storage limitations, and communication links available. Sites in the network request data items and each request has an associated deadline and priority. In a military situation, the data staging problem involves positioning data for facilitating a faster access time when it is needed by programs that will aid in decision making. 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. The network is assumed to be oversubscribed and not all requests for data items can be satisfied. A mathematical model for the basic data staging problem is introduced. Then, three multiple-source shortest-path algorithm-based heuristics for finding a near-optimal schedule of the communication steps for staging the data are presented. Each heuristic can be used with each of four cost criteria developed. Thus, 12 implementations are examined. In addition, two different weightings for the relative importance of different priority levels are considered. The performance of the proposed heuristics are evaluated and compared by simulations. The proposed heuristics are shown to perform well with respect to upper and lower bounds. Furthermore, the heuristics and a complex cost criterion allow more highest priority messages to be received than a simple-cost-based heuristic that schedules all highest priority messages first.

Index Terms—Communication network, data management, Dijkstra's multiple-source shortest-path algorithm, distributed processing, heterogeneous computing, scheduling heuristics.

1 Introduction

The DARPA Battlefield Awareness and Data Dissemination (**BADD**) program [18] includes designing an information system for forwarding (staging) data to proxy servers prior to their usage as inputs to a local application in a distributed computing environment, 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 distributed 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 an example 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 as input for a program that plans 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 request is associated with a specific deadline and priority. It is assumed that not all requests can be satisfied by their deadline. In a military situation, the **data staging** problem involves positioning data for facilitating a faster access time when it is needed by programs that will aid in decision making.

Positioning the data before it is needed can be compli–cated 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

- *M.D. Theys is with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 851 S. Morgan St., RM 1120, Chicago, IL 60607-7053. E-mail: mtheys@uic.edu.*
- *M. Tan is with Dealist.com, 2059 Camden Ave., San Jose, CA 95124. E-mail: mintan@ureach.com.*
- *N.B. Beck is with Sun UltraSPARC Verification, Boston Design Center, 5 Omni Way, MS UCHL05-104, Chelmsford, MA 01824. E-mail: noah@noahsark.dyndns.com.*
- *H.J. Siegel is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285. E-mail: hj@purdue.edu.*
- *M. Jurczyk is with the Department of Computer Engineering and Computer Science, University of Missouri-Columbia, 121 Engineering Building West, Columbia, MO 65211. E-mail: mjurczyk@missouri.edu.*

Manuscript received 18 Dec. 1998; revised 12 Jan. 2000; accepted 3 Apr. 2000. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 108474.

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 [20]. 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 [18], [20]. 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 applications can obtain certain data (e.g., [22], [23]). Multiple copies of data items also provide an increased level of fault tolerance, in cases of links or storage locations going offline.

The simplified data staging problem addressed here 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. Each **node** in the system can be: 1) a source machine of initial data items, 2) an intermediate node for storing data temporarily, and/or 3) a final destination machine that requests a specific data item.

It is also assumed in this simplified version of the data staging problem that all parameter values for the communication system and the data request information (e.g., network configuration and requesting machines) 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. 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 a user's application can easily retrieve the data from this location.

Three multiple-source shortest-path algorithm-based heuristics for finding a near-optimal schedule of the communication steps for staging the data are presented. Each heuristic can be used with each of four cost criteria developed. Thus, 12 implementations are examined. The performance of the proposed heuristics are evaluated and compared by simulations. This research serves as a necessary step toward solving the more realistic and complicated version of the data staging problem involving fault tolerance, dynamic changes to the network configuration, adhoc data requests, sensor-triggered data transfers, etc.

Section 2 provides an overview 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 the multiple-source shortest-path algorithm-based heuristics for finding a near-optimal schedule of the communication steps for data staging. These heuristics adopt 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 heuristics outlined in Section 4. A BADD-like network environment has been used in developing the parameters for conducting this simulation study.

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 [13] in management science and operations research. Under the context of the construction of several new production facilities for components, a manufacturing firm needs to arrange the locations of the production facilities and assembly plants effectively, such that the total cost of transporting individual components from the production facilities to the assembly plants 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 the production facilities should be located. If an analogy is made between 1) the assembly 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 production 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., the use of a graph-based method to reduce the facility location problem to a shortest-path or minimum spanning tree problem).

However, when examining 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 an individual priority. Also, each component requested from a plant commonly does not have a corresponding individual deadline related factor, while in the data staging problem each data request has a deadline. For the data staging problem, the individual priority and individual 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 individual deadlines) is used as the optimization criterion in the mathematical model of the 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 individual deadline related factors (e.g., [8], [10], [14], [17], [19]). Furthermore, in the facility location problem, all constraints must be satisfied for the production to occur (e.g., all parts of a car must arrive). In this research, it is known that not all requests can be satisfied (e.g., some low priority data requests may be dropped). Thus, although lessons can be drawn from the design of algorithms for different versions of the facility location problem, there are significant differences between the facility location and the data staging problems in terms of their formulations and potential solutions.

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 US National Science Foundation (NSF) projects that new techniques for organizing cache memories and other buffering schemes are necessary to alleviate memory and network latency and to increase bandwidth [4]. 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]. It has been shown that several file caches could reduce file transfer traffic, and hence the volume of traffic on the internet backbone [11]. In addition, in distributed environments, ways to increase system performance with intelligent data placement are needed [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.

Work has been done to provide extensions to wormhole routing protocols that handle real-time messages. An off-line approach that schedules usage of the virtual channels by allowing higher priority messages to preempt lower priority messages is presented in [3]. Their research shows that they improve wormhole routing by employing such a protocol. The goal of the work in [3] is similar to the goal of the work presented here in that both give preference to messages that have higher priority. However, in [3] the focus is on wormhole routing protocols, while the work presented here 1) is for a general communication system, 2) attempts to find minimum paths over multiple links, and 3) uses a cost criterion that also considers how close a message is to its deadline.

There has been research done in the area of mapping tasks onto a suite of distributed heterogeneous machines (e.g., [6], [7], [12], [16], [25]). This task mapping research focuses on deciding what machine should execute each task, rather than assuming the task execution locations are known (as in the data staging situation). Thus, the basic problem being addressed by these task mapping studies is different than that of data staging.

Research has been performed concerning dynamic deadline scheduling in real-time systems. Many algorithms for this area have been based on earliest deadline first (EDF) scheduling [21], which is also referred to as earliest due date (EDD) scheduling [5]. Both of these approaches (EDD and EDF) are dynamic approaches to scheduling and often schedule periodic tasks, while the the research performed here is static in nature and deals with one-time requests for information. In addition, most EDF or EDD research has not included the notion of priorities as defined here, which is an integral part of this research.

Other research exploring heuristics for use in the BADD environment has been performed [15]. This work examines methods for scheduling efficiently the ATM-like channels of a possible BADD-like environment. It shows that "greedy" heuristics are effective tools for use in that BADD-like environment and uses a network simulator to corroborate this statement; however, those heuristics do not consider several parameters considered here, such as deadlines and data availability times. The work here differs from [15] in that: 1) here a detailed mathematical model is developed, and 2) the collection of heuristics and cost criteria studied here are based on a different set of assumptions about system structure and data request characterizations.

3 Mathematical Model

A quantitative mathematical model for a basic data staging problem is presented in this section. This model allows the heuristics introduced in Section 4 to be presented formally. As stated and discussed in Section 1, this document 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).

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 item size, list of sources, and a destination. A source of a data item consists of a node number and a time after which the data is available on that node. A destination for a data item contains a node number, priority, and deadline for the data request. This description of the network and associated data requests is used to formulate the mathematical model for solving the basic data staging problem. A glossary of notation is included in the Appendix for the reader's convenience. The reader is encouraged to reference this section as needed.

A communication system $\boldsymbol{M}$ consists of $\boldsymbol{m}$ machines $\{M[0], M[1], ..., M[m-1]\}$. Each machine can be a server that stores data elements and/or 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. $\boldsymbol{Cap}[i](t)$ represents the available memory storage capacity of machine $\boldsymbol{M}[i]$ $(0 \leq i < m)$ from time t_j to t_{j+1} (the interval from t_j to t_{j+1} is not necessarily equal to one time unit).

A network topology graph $\boldsymbol{G}_{nt}$ specifies the connectivity of the communication system for the machines in M. 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, where node $\boldsymbol{V}[i]$ corresponds to machine $M[i]$. Two machines may be connected directly by zero or more communication links. In this model, if two machines are connected by the same transmission link during nl non-overlapping and discontinuous time intervals, then nl 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 unidirectional. A bidirectional link between two machines is represented as two different virtual unidirectional links that correspond to the transmission link in each direction (for each time interval).

Let $\boldsymbol{Nl}[i,j]$ be the total number of unidirectional virtual communication links from $M[i]$ to $M[j]$. $\boldsymbol{L}[i,j][k]$ denotes the kth unidirectional virtual communication link from $M[i]$ to $M[j]$, where $0 \leq i,j < m$, $i \neq j$, and $0 \leq k < Nl[i,j]$. For each $L[i,j][k]$, a directed edge $\boldsymbol{E}[i,j][k]$from $V[i]$ to $V[j]$ is included in G_{nt}. All the included edges constitute the set of edges $\boldsymbol{E}$ of G_{nt}. Each $L[i,j][k]$ is associated with one unique time interval during which the corresponding physical link is available for communication. Let $\boldsymbol{Lst}[i,j][k]$ denote the time when link $L[i,j][k]$ becomes available (link start time) and $\boldsymbol{Let}[i,j][k]$ denote the time when link $L[i,j][k]$'s availability terminates (link end time). 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). Each virtual link also has an associated bandwidth.

Let a **data item** be a block of information that can be transferred between machines. For any data item $\boldsymbol{d}$, $|d|$ represents the size of the associated data item. Let $\boldsymbol{D}[i,j][k](|d|)$ denote the communication time for transferring data item d (from machine $M[i]$ to machine $M[j]$ through their kth dedicated virtual link during time interval $[Lst[i,j][k], Let[i,j][k]]$). The time $D[i,j][k](|d|)$ includes all the various hardware and software related components of the intermachine 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.

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 $\boldsymbol{n}$ is the number of data items with distinctive names (identifiers) available in the corresponding communication system M. Let $\boldsymbol{\Delta} = \{\delta[0], \delta[1], ..., \delta[n-1]\}$ be the set of these data items, where each $\boldsymbol{\delta}[i]$ 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 $\boldsymbol{N\delta}[i]$ be the number of different machines that the data item $\delta[i]$ is located at initially. $\boldsymbol{Source}[i,j]$ denotes the jth initial source location of the data item $\delta[i]$ (with no implied significance for the ordering of the sources), where $0 \leq i < n$, $0 \leq j < N\delta[i]$, and $0 \leq Source[i,j] < m$. Also, $\boldsymbol{\delta st}[i,j]$ denotes the time at which $\delta[i]$ is available at its jth initial source location (start time).

Suppose $\boldsymbol{\rho}$ is the number of *requested* data items with distinctive names (identifiers) in the corresponding communication system M, where $(0 \leq \rho \leq n)$. Let $\boldsymbol{Rq} = \{Rq[0], Rq[1], \ldots, Rq[\rho-1]\}$ be the set of the requested data items. Each $\boldsymbol{Rq}[j]$ $(0 \leq j < \rho)$ is the name of a data item and there must exist an i $(0 \leq i < n)$, 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 $\boldsymbol{Nrq}[j]$ denote the number of different requests for $Rq[j]$.

$\boldsymbol{Request}[j,k]$ denotes the number of the machine from which the kth request for data item $Rq[j]$ originates (with no implied order among the requests), where $0 \leq j < \rho$, $0 \leq k < Nrq[j]$, and $0 \leq Request[j,k] < m$. (It is assumed that a given machine generates at most one request for a given data item.) Also, $\boldsymbol{Rft}[j,k]$ denotes the finishing time (or deadline) after which the data item $\boldsymbol{Rq}[j]$ on its kth requesting location is no longer useful (e.g., data items may be needed before a specific time when certain decisions must be made). The deadlines are set by the users' applications and also the users' position in the command hierarchy. Two requests for the same data item, terminating on two different destinations, could result in different deadlines for each of the destinations.

A machine functioning as an intermediate node for a data item $Rq[j]$ does not need to keep the data item local indefinitely. Instead, at γ time units after the latest deadline for the requested data item $Rq[j]$, the data item is removed from the storage of any intermediate nodes (i.e., garbage collection is performed). In this way, storage capacity is reclaimed by removing data items after they are no longer needed, and a level of redundancy is provided in the system in cases where a link, an intermediate node, or a destination might lose their copy of $Rq[j]$. The scheduling heuristics do not remove a data item from any of its sources or destinations because it is considered outside the scope of responsibility of the scheduler.

Suppose the priority of each data request is between 0 and P, where $\boldsymbol{P}$ is the highest priority possible (i.e., P corresponds to the class of most important requests). $\boldsymbol{Priority}[j,k]$ denotes the priority for the data request of the data item $Rq[j]$ on its kth requesting location. The priority of each request would be set by the command structure present in the problem, and two different requests for the same data item can have different priorities (e.g., if a general and a private both request the air traffic orders, obviously the general would have a higher priority than the private).

Suppose $\boldsymbol{W}[i]$ $(0 \leq i \leq P)$ denotes the relative weight of the ith priority. These weightings allow system administrators to specify the relative importance of a priority α data request versus priority β data request, where $0 \leq \alpha, \beta \leq P$.

Assume that the scheduling procedure of the communication steps starts at time 0. Let $\boldsymbol{S} = \{S_0, S_1, ..., S_{\sigma-1}\}$ denote a set of $\boldsymbol{\sigma}$ distinct schedules for the communication steps of transmitting requested data items. Consider a specific schedule $\boldsymbol{S_h}$, where $0 \leq h < \sigma$. The kth 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 $\boldsymbol{Srq}[S_h]$ denote the set of two-tuples $\{(j,k) \mid k\text{th request of the data item } Rq[j] \text{ is satisfiable using schedule } S_h\}$.

The effect, $\boldsymbol{E}[S_h]$, of the scheduling scheme S_h is defined as

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

Given this mathematical model, the **global optimization criterion** used for data staging in this document, 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 Heuristics

4.1 Introduction

The heuristics for solving the data staging problem are based on Dijkstra's algorithm for solving the multiple-source shortest-path problem on a weighted and directed graph [9]. Dijkstra's algorithm takes as input a directed graph with weighted edges, and produces as output the shortest path from a set of source nodes to every other node of the graph. More detailed information about Dijkstra's algorithm can be found in [9].

All necessary communication steps are scheduled by the data staging heuristics. These heuristics consider all data requests together and utilize the following strategies collectively:

1. find the shortest path for each data item as if it is the only requested data in the system,
2. resolve conflicting requests,
3. maximize the weighted sum of the priorities of the potentially satisfiable data requests, and
4. consider the urgency of a request as its deadline approaches.

The model used for the heuristics and implementation details about the heuristics are presented in Sections 4.2 through 4.4. Sections 4.5 through 4.7 discuss the three heuristics that have been developed. These heuristics are built upon Dijkstra's multiple-source shortest-path algorithm. The heuristics iteratively pick which data item to transfer next based on a cost function. Section 4.8 presents background information about the cost criteria components, and details the four cost criteria used in this research.

4.2 Adaptation of Dijkstra's Algorithm

For each requested data item $Rq[i]$, an instantiation, $\boldsymbol{G}_{nt}[i]$, of the graph G_{nt} (defined in Section 3) is created. Let $\boldsymbol{V}_S[i]$ be the set of source nodes corresponding to the machines that are the initial locations of the data item $Rq[i]$. Let $\boldsymbol{V}_D[i]$ be the set of destination nodes corresponding to the machines that are making data requests for $Rq[i]$ (i.e., machines $Request[i,k], 0 \leq k < Nrq[i]$). (It is assumed that $V_S[i] \cap V_D[i] = \emptyset$.) The weight on an edge $E[b,j][k]$ of $G_{nt}[i]$ is the communication time required to transfer $Rq[i]$ from machine b to machine j over virtual link $L[b,j][k]$. Let the length of a path from a source node $\boldsymbol{v}_s \in V_S[i]$ to a destination node $\boldsymbol{v}_d \in V_D[i]$ be defined as the difference between the time when data item $Rq[i]$ is available on v_s and the time data item $Rq[i]$ arrives at v_d (via the machines and the communication links along the path). This time can be calculated using the various parameters defined in Section 3. With the defined $G_{nt}[i]$, $V_S[i]$, and $V_D[i]$, a separate multiple-source shortest-path problem is well-defined for each requested data item in the context of the data staging problem.

Dijkstra's algorithm, in general, is applied to a directed graph with weighted edges and a set of source nodes [9]. For each node in the graph, the algorithm generates a shortest path from any of the sources of the data item to that node (using the weighted edges). The heuristics examined here begin by applying Dijkstra's algorithm to $G_{nt}[i]$ for each requested data item $Rq[i]$. More detailed information about Dijkstra's algorithm can be found in [9]. Then the heuristics consider all data items collectively.

An example of how Dijkstra's algorithm functions is illustrated in Fig. 1. In the figure, nodes s_0 and s_1 are both sources for the requested data item $Rq[i]$ at time zero, i.e., $V_S[i] = \{s_0, s_1\}$. In general, different sources can have differing available times for the same data item. For the purposes of this study, all available times, for a particular data item, are created to be the same. Dijkstra's algorithm will find the shortest path from a source to each of the other three nodes in the graph, u, v, and x. In Fig. 1, the network graph shows the links that are available, with their associated cost noted. The four parts of Fig. 1 show four states of $G_{nt}[i]$ during the execution of the algorithm.

The following notation is used to show the state of the algorithm at each time. The dotted lines are those next links that can be used to schedule the movement of the data item from a source to the other nodes. The dashed line shows the next link to be scheduled. The set $\boldsymbol{V}_F$, which is initially $V_S[i]$, contains all nodes currently scheduled to receive the data items. When all the nodes in the graph reside in the set V_F, the algorithm is finished. For each node j adjacent to a node in V_F, the current estimate of the earliest arrival time for the data item at node j is $\boldsymbol{e}[j]$.

Fig. 1a shows the initial state for the example. In Fig. 1b, the earliest arrival time for the data item at the three nodes is noted. The algorithm selects the node with the smallest $e[]$ value to next receive the data item. Thus, the dashed line shows that the next link to be scheduled by the algorithm will be moving the data to node x from s_0, the node with the earliest arrival time. After the data item is scheduled to node x, x is added to the set V_F.

In Fig. 1c, the algorithm then attempts to reduce the earliest arrival time, $e[]$, for all nodes that are adjacent to node x. It is then determined that the path from s_0 through node x to node u is shorter than the direct path from s_0 to node u and $e[u]$ is updated. Because $e[v] < e[u]$, the link from s_1 to node v will be the next scheduled.

Lastly, in Fig. 1d, the process is repeated by examining those nodes adjacent to node v. The link from node x to node u is scheduled to transfer the data item. When the algorithm completes, the shortest path to each node (machine) from one of the elements of $V_S[i]$ (a source) is shown by the dashed lines in Fig. 1d.

The implementation of Dijkstra's algorithm in the heuristics checks: 1) that all machines have enough memory capacity, Cap, to hold the data item being transferred until the garbage collection scheme schedules its removal; 2) that

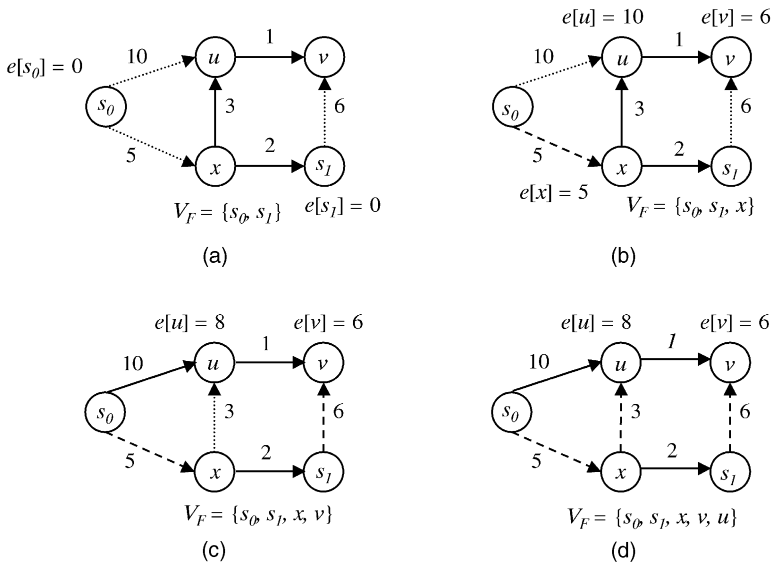


Fig. 1. An example execution of Dijkstra's algorithm for a given requested data item $Rq[i]$. The earliest arrival time for the data item at a node is represented as e[]. Dotted lines are those next links that can be scheduled to move data; dashed lines show the links that have been scheduled. The solid lines are other links in the system. The set V_F contains nodes scheduled to receive the data item. The number next to each link is the corresponding communication time for a data item to traverse the link. The figure components (a) through (d) show four states of the algorithm.

the communication links are available; and 3) the initial time that the data item is available on a source. This information is for the shortest path from some source node to a destination. At the completion of Dijkstra's algorithm, the shortest path from any source to all nodes in the network is known (clearly, $V_D[i] \subseteq$ all machines). The shortest path time $\boldsymbol{e}[i, s]$ for $Rq[i]$ to arrive at node $V[s]$ is initially obtained from executing Dijkstra's algorithm, and is a lower bound for the arrival time when requests for all data items are considered collectively.

Suppose $e[i, s]$ and $e[i, r]$ are known and there are virtual links $L[s, r][k] (0 \leq k < Nl[s, r])$ from $V[s]$ to adjacent node $V[r]$. Let $\boldsymbol{A_L[s, r][i][k]}$ denote the time when the requested data item $Rq[i]$ can be available on machine $V[r]$ via fetching the copy from $V[s]$ through the virtual link $L[s, r][k]$. Attempting to reduce the shortest path time $e[i, r]$ with respect to the edges from $V[s]$ to $V[r]$ based on the known $e[i, s]$ and $e[i, r]$ is implemented by the C-style pseudocode in Fig. 2.

As illustrated by Step 10 in Fig. 2, the exact virtual link $L[s, r][\boldsymbol{k_l}]$ used for updating the shortest path estimate to $M[r]$ needs to be recorded, due to the existence of multiple virtual links between $V[s]$ and $V[r]$. Thus, the **predecessor-node** $\pi[r]$in the usual description of the Dijkstra's algorithm (used to record the shortest path) is extended as a **predecessor field** and is defined as a two-tuple (s, k_l) in this data staging heuristic, where s gives the source machine and k_l stores the virtual link used.

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 [9] is used to implement the priority queue, the worst case asymptotic complexity of Dijkstra's algorithm is $O(|E| + |V| \log |V|)$. For the network topology graph G_{nt} terminology described in Section 3, $|E| = \sum_{0 \leq i \neq j < m} Nl[i, j]$ and $|V| = m$. Because it is necessary to apply the multiple-source shortest-path algorithm to all the requested data items $Rq[i], (0 \leq i < m)$, the worst case asymptotic complexity of this heuristic for determining one communication step in S_h is $O[\rho(v \log v + \sum_{0 \leq i \neq j < m} Nl[i, j])]$.

4.3 Combining Paths for Multiple Data Items

After applying the multiple-source shortest-path algorithm for each of the ρ requested data items $Rq[i]$ individually, ρ sets of shortest paths are generated. For the example shown in Fig. 3, 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., machine $M[0]$ cannot send data items $Rq[0]$ and $Rq[1]$ to machine $M[3]$ over the same virtual link simultaneously due to a link conflict). Thus, a local optimization criterion is used to select one of the valid communication steps to be scheduled (refer to Section 4.8 for more information).

Readers should notice that it may be impossible to use the individually shortest paths to all destinations for each data item due to possible communication link and memory space contention in the network when transferring other data items during the same time interval. Also, a multiple-source shortest-path algorithm for $G_{nt}[i]$ attempts to minimize the time when only a given requested data item $Rq[i]$ is obtained by its corresponding requesting locations. But as stated in Section 1, request deadlines and the priorities of *all* potentially satisfiable data requests must be taken into account (as well as the sharing of the memory capacity of machines and the communication links by multiple data items).

```
1.  for all $k$ $(0 \leq k < Nl[s,r])$, do steps 2 to 7 (for a given $Rq[i]$)
2.    $A_L[s,r][i][k] = \infty$   /* initialize value */
      if $(e[s] > Lst[s,r][k])$ {
      /* if $Rq[i]$ is obtained by $M[s]$ after $L[s,r][k]$ is available */
3.      if $([e[s] + D[s,r][k](|Rq[i]|)] \leq Let[s,r][k])$
        /* if the available time interval is long */
        /* enough to transfer $Rq[i]$ via $L[s,r][k]$ */
4.        if $(Cap[r](e[s])$ through $(Cap[r](\max_{b \in V_D[i]} Rft[i,b] + \gamma) \geq |Rq[i]|)$
          /* if $M[r]$ has enough storage capacity for $Rq[i]$ */
            $A_L[s,r][i][k] = e[s] + D[s,r][k](|Rq[i]|)$
            /* 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 $([Lst[s,r][k] + D[s,r][k](|Rq[i]|)] \leq Let[s,r][k])$
        /* if the available time interval is long */
        /* enough to transfer $Rq[i]$ via $L[s,r][k]$ */
7.        if $(Cap[r](Lst[s,r][k])$ through $(Cap[r](\max_{b \in V_D[i]} Rft[i,b] + \gamma) \geq |Rq[i]|)$
          /* 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 $(e[r] > \min_{0 \leq k < Nl[s,r]} A_L[s,r][i][k])$ {
        /* if smaller shortest-path estimate is found */
9.      $e[r] = \min_{0 \leq k < Nl[s,r]} A_L[s,r][i][k]$
        /* 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 $A_L[s,r][i][k]$ */
```

Fig. 2. Pseudocode for attempting to reduce the shortest path time with respect to the edges from $V[s]$ to $V[r]$ for $Rq[i]$.

4.4 Garbage Collection

Leaving a copy of $Rq[i]$ on machine $M[r]$ after it is sent to the next machine on the shortest path allows this copy to be used as an intermediate copy for forwarding $Rq[i]$ to some other machines. For the example communication step of transferring $Rq[0]$ from $M[0]$ to $M[3]$ shown in Fig. 3a, by tracing the shortest paths generated for $M[7]$, $M[8]$, and $M[9]$, the set of intermediate machines can be determined as $\{3, 5\}$. At some time after the latest deadline for $Rq[i]$, as discussed in Section 3, the available memory capacity of $M[r]$ is incremented by $|Rq[i]|$ to simulate the removal of $Rq[i]$ from its memory. So for the example, $M[3]$ and $M[5]$ would keep $Rq[i]$ in their local memory for γ time units after the latest deadline among machines $M[6]$, $M[7]$, $M[8]$, and $M[9]$. The data item is kept on the intermediate machines for this time duration to provide a level of fault tolerance in cases when communication links or storage locations become unavailable, or in the case where a link, an intermediate node, or a destination loses its copy of the data.

4.5 Partial Path Heuristic

Each iteration of this heuristic involves: 1) performing Dijkstra's algorithm for each data request individually, 2) for the valid next communication steps, determining the "cost" to transfer a data item to its successor in the shortest

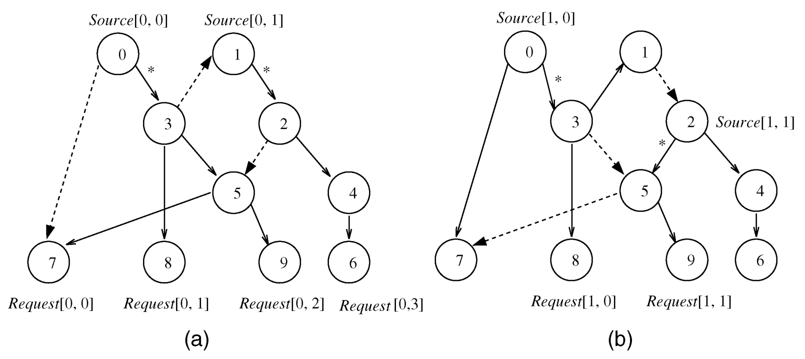


Fig. 3. An example communication system that requests (a) $Rq[0]$ and (b) $Rq[1]$. $Source[i, j]$ denotes the jth initial source location of the data item $Rq[i]$. $Request[i, j]$ denotes the machine from which the jth request for data item $Rq[i]$ originates. Solid lines show shortest paths for a given data item to all nodes (even nonrequesters), and dashed lines show unused links for a given data item.

path, 3) picking the lowest cost data request and transferring that data item to the successor machine (making this machine an additional source of that data item), 4) updating system parameters to reflect resources used in 3), and 5) repeating 1) through 4) until there are no more satisfiable requests in the system. In some cases, Dijkstra's algorithm would not need to be executed each iteration for a particular data transfer, i.e., if the data transfer did not use resources needed for any future transfers. This optimization is not yet considered, because this research is not focusing on minimizing the execution time of the heuristics themselves.

This heuristic will schedule the transfer for the single "most important" request that must be transferred next, based on a cost criterion. The heuristic is called the **partial path heuristic** (referred to as **partial** in Figs. 6, 7, and 10), because only one successor machine in the path is scheduled at each iteration. If a data item is partially scheduled through the system and because of other scheduled transfers the requesting destination's deadline is no longer satisfied, the scheduled transfers remain in the system (the initial transfers were scheduled because the deadline could have been satisfied). Reasons the schedule for this now unsatisfiable request is not removed include: 1) in a dynamic situation, a change in the network could allow the request to be satisfied, and 2) removing the already scheduled transfers would require restarting the scheduling for all data requests because of conflicts that might have occurred. Sections 4.6 and 4.7 present the other two heuristic methods explored. The various cost criteria used with the heuristics are described in Section 4.8, and the heuristics are evaluated in Section 5.

4.6 Full Path/One Destination Heuristic

The **full path/one destination heuristic** produces a communication schedule that avoids partial paths that are later blocked. The behavior of the partial path heuristic showed that if a data item $Rq[i]$ was selected for scheduling a transfer to its next intermediate location (a "**hop**"), in the following iteration, the same requested data item, $Rq[i]$, would typically be selected again to schedule its next hop. The full path/one destination heuristic (referred to as **full_one** in Figs. 6, 8, and 10) attempts to exploit this trend by selecting a requested data item with one of the cost criterion discussed in Section 4.8 and scheduling all hops required for the data item to reach its lowest cost destination before executing Dijkstra's algorithm again.

Considering the example communication system in Fig. 3a, data item $Rq[0]$ would be scheduled only from $M[0]$ to $M[3]$ before executing Dijkstra's algorithm again in the partial path heuristic. In the full path/one destination heuristic, data item $Rq[0]$ would be scheduled from $M[0]$ through $M[3]$ and $M[5]$ to $M[9]$ (assuming $M[9]$'s cost is lower than $M[7]$ or $M[8]$) before executing Dijkstra's algorithm again. This results in reducing the number of executions of Dijkstra's algorithm by three for this example. A savings proportional to the average length of a data item's path from a source to a destination is expected from this heuristic.

Considering again the communication system in Fig. 3a, if this heuristic initially schedules the transfer of data item $Rq[0]$ from $M[0]$ to $M[9]$, $M[3]$ and $M[5]$ would become sources for $Rq[0]$. In the next iteration, $M[7]$ could receive $Rq[0]$ from $M[5]$, and $M[8]$ could receive $\delta[0]$ from $M[3]$, without having to schedule a transfer from the original source, $M[0]$. The schedules generated by the partial path and the full path/one destination heuristics are compared in Section 5.

The partial path heuristic may construct a partial path (of many links) that it later cannot complete (due to network or memory resources being consumed by other requested data items). However, until this is determined, the part of the path constructed may block the paths of the other requested data items, causing them to take less optimal paths or be deemed unsatisfiable. The full path/one destination heuristic avoids this problem. An advantage the partial path approach does have over the full path/one destination approach is that it allows the link-by-link assignment of each virtual link and each machine's memory capacity to be made based on the relative values of the cost criteria for the data items that may want the resource.

4.7 Full Path/All Destinations Heuristic

The **full path/all destinations heuristic** builds on the full path/one destination heuristic and requires fewer executions of Dijkstra's algorithm then the other two heuristics. In the full path/one destination heuristic, a data item is transferred from a single source to a single destination, even if there are multiple destinations requesting the same data item. For the example communication system in Fig. 3a, $Rq[0]$ is requested by machines $M[7]$, $M[8]$, and $M[9]$, and the shortest path for these three destinations all originate at machine $M[0]$ and pass through machine $M[3]$. The full path/all destinations heuristic (referred to as **full_all** in Figs. 6, 9, and 10), will schedule all paths for a single data item that share the next machine in the path as an intermediate machine. In Fig. 3, the data item $Rq[0]$ would be scheduled for all three destinations (machines $M[7]$, $M[8]$, and $M[9]$) at the same time. By scheduling the path to multiple destinations, two fewer executions of Dijkstra's algorithm are required as compared to the full path/one destination heuristic. A savings is expected that is proportional to the average number of destinations for a data item whose shortest path intermediate machine sets share a common machine.

This approach was considered because it was expected to generate results comparable to the full path/one destination heuristic, but with a smaller heuristic execution time. The schedules generated by the three heuristics are compared in Section 5.

4.8 Cost Criteria

Four cost criteria that use "urgency" and "effective priority" have been devised for the three heuristics presented. Each of these cost criteria was chosen so as to vary the effect these two parameters will have in determining the next communication step. This section begins by defining "urgency" and "effective priority," and then the four cost criteria used are presented.

Recall that $A_T[i,j]$ denotes a lower bound on the time when $Rq[i]$ is received and available at its corresponding jth requesting location (as mentioned in Section 3) and $Rft[i,j]$ denotes the finishing time (or deadline) after which the data item $Rq[i]$ on its jth requesting location is no longer useful (as mentioned in Section 3). Assume $M[r]$ is the next machine in the shortest path from a given source to the jth destination to receive data item $Rq[i]$. Let the set of all such destinations j be called $Drq[i,r]$. A **satisfiability function** $Sat[i,r](j)$ is 1 if a request for data item $Rq[i]$ is scheduled to be received at the jth requesting destination (through machine $M[r]$) before its deadline; and 0 if the request for data item $Rq[i]$ is scheduled to be received after its deadline. Note that if the request cannot be satisfied using the shortest path, there is no other path that will cause it to be 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 $Rq[0]$ from $M[0]$ to $M[3]$ in Fig. 3a. For this example, $Drq[0,3] = \{M[7], M[8], M[9]\}$. Suppose that the request deadlines for $Rq[0]$ are as follows (in some abstract time units): 10 for $M[7]$, 15 for $M[8]$, and 5 for $M[9]$. Suppose further that the shortest path estimate has shown that the network can deliver $Rq[0]$ at time: 12 for $M[7]$, 11 for $M[8]$, and 8 for $M[9]$. Then, $Sat[0,3](0) = 0$, $Sat[0,3](1) = 1$, and $Sat[0,3](2) = 0$.

Recall from Section 3 that $Priority[i,j]$ denotes the priority for the data request for the data item $Rq[i]$ on its jth requesting location and that $W[k]$ $(0 \leq k \leq P)$ denotes the relative weight of the kth priority. Let $\boldsymbol{Efp}[i,r](j)$ denote the **effective priority** for the data request of $Rq[i]$ from its jth requesting location, where $Efp[i,r](j) = Sat[i,r](j) * W[Priority[i,j]]$.

Suppose $\boldsymbol{Urgency}[i,r](j)$ denotes the urgency for the data request of $Rq[i]$ from its jth requesting location, where $Urgency[i,r](j) = -Sat[i,r](j) * (Rft[i,j] - A_T[i,j])$, where smaller $Urgency[i,r](j)$ implies that it is less urgent to transfer $Rq[i]$ to the jth requesting location. Note that the urgency would take into consideration the impact of factors such as the number of intermediate nodes and the bandwidths of the links between nodes. The unit of measure for the $Urgency$ term is seconds.

Four cost functions for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via an available direct virtual link are each defined using urgency and effective priority, as above defined. Readers should notice that 1) applying Dijkstra's algorithm to obtain $A_T[i,r]$ through shortest paths, 2) maximizing $Efp[i,r](j)$, and 3) maximizing $Urgency[i,r](j)$ follow the three strategies for designing data relocation heuristics recommended in 1), 2), and 4), respectively, in Section 4.1.

If $Sat[i,r](j)$ is 0 for all r that correspond to valid next machines that receive $Rq[i]$ and the associated values of j, 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 heuristics are 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. Thus, requests that are at one point in time unsatisfiable, might become satisfiable at a later point in time.

The following information about the links in the network is available: bandwidth available on a link, duration a virtual link is available, and size of each data item. In the model used here, the time interval a virtual link is needed to transfer a specific data item $Rq[i]$ is determined by dividing the size of the data item, $|Rq[i]\ |$, by the bandwidth available on the link.

Suppose that the current chosen communication step is to transfer the requested data item $Rq[i]$ from $M[s]$ to $M[r]$. Before repeating the above heuristics for determining the next communication step(s), the following information must be updated: the list of virtual links and their start and stop times, the available memory capacity on any machines that $Rq[i]$ has been placed, the sources of $Rq[i]$ must now include all machines that $Rq[i]$ has been moved to/through, and the time at which $Rq[i]$ can be removed from any intermediate machines.

The cost criteria are designed so that 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 $0 \leq i < \rho$. Suppose $\boldsymbol{W}_E \geq 0$ is the relative weight for the effective priority factor and $\boldsymbol{W}_U \geq 0$ is the relative weight for the urgency factor in the scheduling. For the first cost criterion (**C1**), the $\boldsymbol{Cost}_1[s,r][i,j][k]$, for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$, via link $L[s,r][k]$, for the jth destination, is defined as:

$$\begin{aligned} Cost_1[s,r][i,j][k] = &- W_E * Efp[i,r](j) \\ &- W_U * Urgency[i,r](j). \end{aligned}$$

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 will facilitate satisfying data request(s). The first term of $Cost_1[s,r][i,j][k]$ attempts to give preference to a satisfiable data request with a priority higher than the other requests. Furthermore, 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 sooner. This intuition is captured by the inclusion of the urgency term. Thus, collectively with the consideration of the priority of satisfiable data requests and the urgency of those data requests in this local optimization step, using this cost criterion in the data staging heuristic should generate a near-optimal communication schedule that reasonably achieves the global optimization criterion.

The second criterion (**C2**) examines the $\boldsymbol{Cost}_2[s,r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$:

$$\begin{aligned} Cost_2[s,r][i][k] = &- W_E * (\sum_{j \in Drq[i,r]} Efp[i,r](j)) \\ &- W_U * (\max_{j \in Drq[i,r]} Urgency[i,r](j)). \end{aligned}$$

This cost function considers all requests for $Rq[i]$ whose shortest path passes through machine $M[r]$ and sums their weighted priorities. Rather than summing all of the urgency terms for these destinations, the most urgent satisfiable request is added in $Cost_2$. This method of capturing the urgency is used as a heuristic to maximize the sum of the weighted priorities of satisfied requests because if the most urgent request for an item passing through $M[r]$ is satisfied, it is more likely that all requests for this data item passing through $M[r]$ will be satisfied.

The $\boldsymbol{Cost}_3[s,r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$ (**C3**) is:

$$Cost_3[s,r][i][k] = \sum_{j \in Drq[i,r]} Efp[i,r](j)/Urgency[i,r](j).$$

The third criterion takes the weighted priority for a destination and divides it by the urgency for this destination, and then sums over all the destinations with satisfiable requests for data item $Rq[i]$ on a path through machine $M[r]$. This cost is a sum of the weighted priorities of satisfiable requests normalized by the urgency of each request. Note that this heuristic does not use W_E or W_U.

	0	1	2	3	4	5	6	7	8	9
0	○	●	○	●	○	●	○	○	●	●
1	○	○	●	○	●	●	○	●	●	○
2	●	○	○	○	●	●	○	○	○	●
3	○	●	●	○	●	○	○	●	●	●
4	●	○	●	○	○	○	●	○	●	●
5	○	●	●	●	○	○	○	●	●	○
6	●	●	○	○	●	●	○	○	●	●
7	●	●	●	●	○	○	○	○	●	●
8	○	○	●	●	○	○	●	○	○	○
9	○	○	●	●	●	●	○	○	○	○

Fig. 4. An example adjacency matrix for one of the networks used in the simulation. A • at entry i, j implies there is a connection from machine i to machine j.

This is because the effective priority is divided by the urgency and so W_E divided by W_U acts as a scaling factor that would not affect the relative cost of the requests. For example, if two data items i_1 and i_2 are competing for the use of $L[s,r][k]$, the relative value of $Cost_3[s,r][i_1][k]/Cost_3[s,r][i_2][k]$ will be unchanged by including any given W_E to weight the $Efp[i,r](j)$ factors and any given W_U to weight the $Urgency[i,r](j)$ factors.

The $\boldsymbol{Cost}_4[s,r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$ (**C4**) is:

$$\begin{aligned} Cost_4[s,r][i][k] = &- W_E * \sum_{j \in Drq[i,r]} Efp[i,r](j) \\ &- W_U * \sum_{j \in Drq[i,r]} Urgency[i,r](j). \end{aligned}$$

This last criterion sums the weighted priorities of all satisfiable requests for data item $Rq[i]$ on a path through machine $M[r]$ and combines that with the sum of the urgency for those same satisfiable requests. Comparing $Cost_2$ and $Cost_4$, it should be noted that the urgency term for each destination whose shortest path shares an intermediate node $M[r]$ is summed in $Cost_4$, whereas $Cost_2$ simply takes the maximum of the urgency terms over this same set of destinations. The benefit of $Cost_4$ is demonstrated by the following example. The first data item, $Rq[i]$, is requested by four machines that all have identical priorities, and have an A_T that is very close to their deadlines. The second data item, $Rq[j]$, is also requested by four destinations that have the same identical priorities, but only one destination has an A_T that is close to its deadline. $Cost_2$ will be unable to differentiate between these two data requests, but $Cost_4$ will chose to schedule $Rq[i]$ before $Rq[j]$.

All four of these cost criteria are used in conjunction with the partial path heuristic and the full path/one destination heuristic. For the full path/all destinations heuristic, $Cost_1$ is not used because it does not capture the fact that a data item can be sent to multiple destinations.

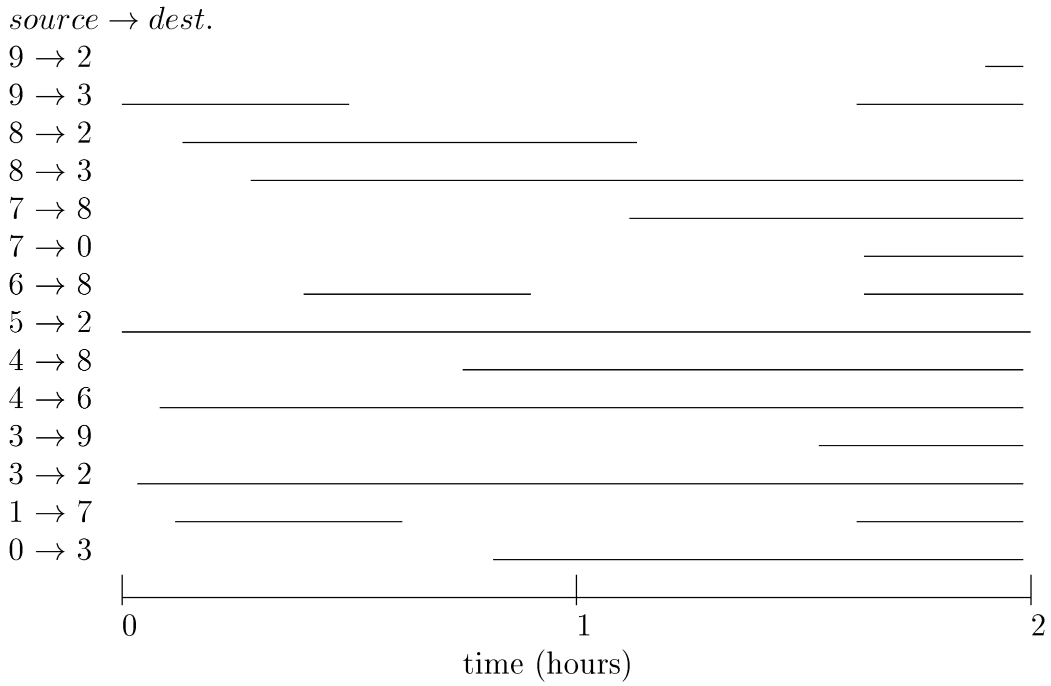


Fig. 5. An example of the initial link availability for one of the networks used in the simulation study.

The data item that the full path/one destination heuristic chooses to send is sent from its current location (machine ($M[s]$) in each of the cost criteria) over as many virtual links as required to reach its destination machine (machine $M[j]$ for one value of j). For $Cost_1$, the choice of j (i.e., which requesting destination should be satisfied) is trivial; $Cost_1$ only takes into account a single requesting destination. All other cost criteria identify a set $Drq[i,r]$ of destinations, and one destination $M[j]$ must be selected from that set to satisfy. For cost $Cost_2$, the value of j chosen is the one satisfying the condition

$$\max_{j \in Drq[i,r]} Urgency[i,r](j)$$

from the equation describing $Cost_2$. For cost $Cost_3$, the value of j chosen is the one satisfying the condition

$$\min_{j \in Drq[i,r]} \frac{Efp[i,r](j)}{Urgency[i,r](j)}$$

from the equation describing $Cost_3$. For cost $Cost_4$, the data item is sent first to machine $M[r]$, and if no request was satisfied, the cost is applied a second time for the same data item $Rq[i]$, but setting the new $M[s]$ (data source machine) to the old $M[r]$ (the machine to which the data was just scheduled). The minimum cost is then taken over all values of (r) (possible next staging locations). The value of (r) with minimum cost determines the machine $M[r]$ that the data is sent to next. This process continues until the data item has reached one requesting destination $M[j]$.

5 Simulation Study

5.1 Introduction

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 heuristics discussed in Sections 4.5 through 4.7. Rather than just choosing one network topology and set of data requests, 40 test cases are generated because one test case cannot reflect the range of possible data requests and network configuration scenarios. The three heuristics are executed using each of these cases and the results are averaged. The properties for data requests and the underlying communication systems are randomly generated with uniform distributions in predefined ranges representing a subset of the systems in a BADD-like environment (see Section 5.3). 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 [9], such that there is a path consisting of unidirectional physical transmission links between any pair of machines 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 heuristics over a broad range of conditions, 2) a generally accepted set of data staging benchmark tasks does not exist, and 3) the system details of actual environments where these data heuristics could be employed are constantly changing as new technologies are introduced. Determining a representative

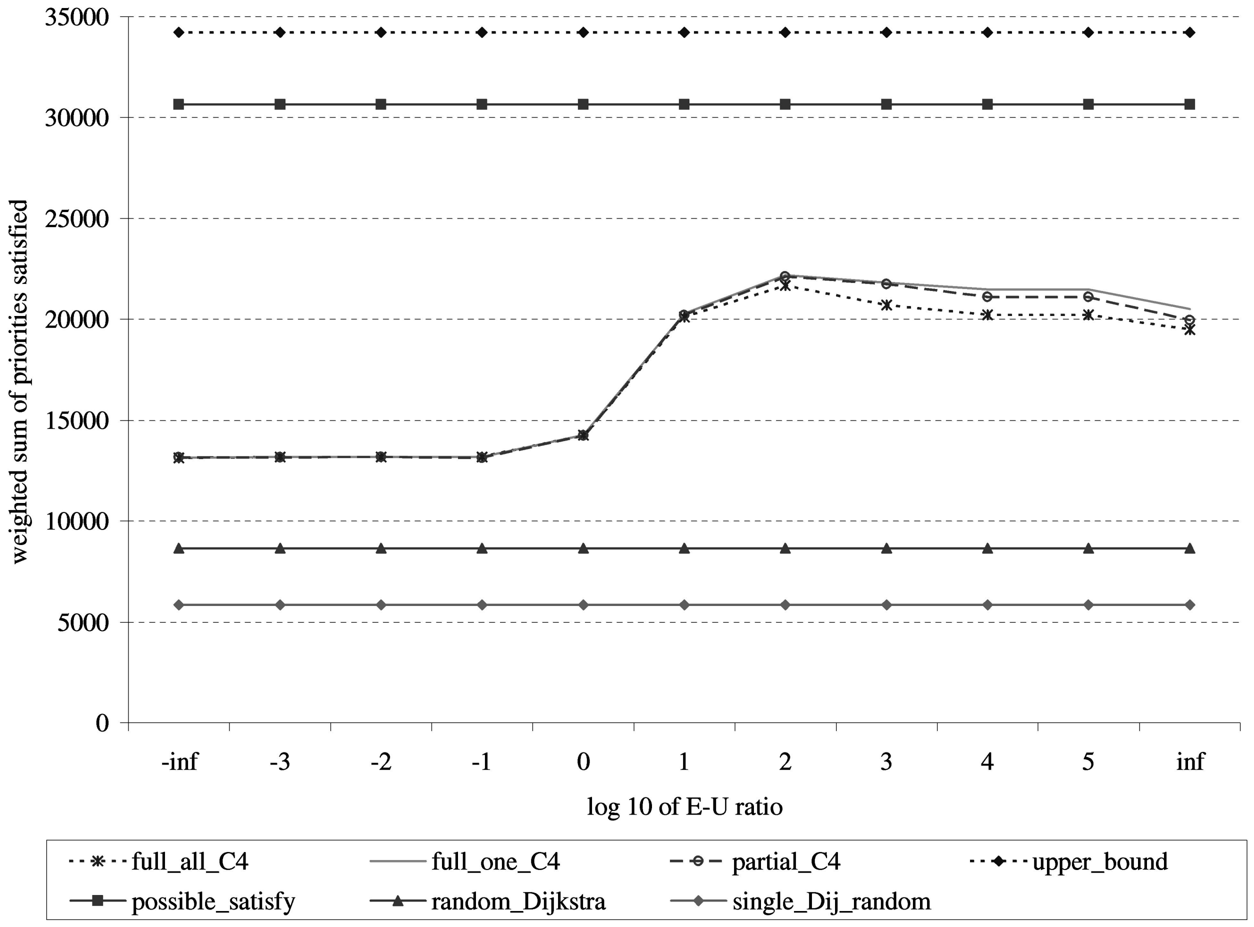


Fig. 6. Comparison of the heuristics' best cost criterion performance for the 1, 10, 100 weighting scheme.

set of data staging benchmark tasks remains an unresolved challenge in the research field of data staging and is outside the scope of this document.

As an example of the communication network structure generated, one sample adjacency matrix is shown in Fig. 4. A • at position (i, j) in the matrix represents the fact that a physical link between machine i and machine j exists; a ◦ means there is no such link. The links utilized here are unidirectional. This corresponds to some links being satellite-based links. Bidirectional links can be represented as two unidirectional links. Thus, a link from machine i to machine j does not imply that there is a link from machine j to machine i, nor does it disallow such a link.

Fig. 5 shows, for a subset of all the links in the system, the time intervals that the physical links are available before any scheduling of data items occurs. The varying availability of some links correlates to certain links being satellite based and not always available. Other links can be reserved for specific functions at certain times (such as teleconferencing) and unavailable for transferring data items. Not every link shown in Fig. 4 is available during the simulation period shown in Fig. 5.

Finding optimal solutions to data staging tasks with realistic parameter values are intractable problems. Therefore, it is currently impractical to directly compare the quality of the solutions found by the three heuristics 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 heuristics presented in this document. To aid in the evaluation of these heuristics, two lower bounds and two upper bounds on the performance of the heuristics are provided.

5.2 Lower and Upper Bounds

To provide lower bounds for the performance of the three heuristics presented here, two random search based scheduling procedures were devised. The first (looser) lower bound is a random-search based scheduling procedure that 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 $Rq[i+1]$ (where the

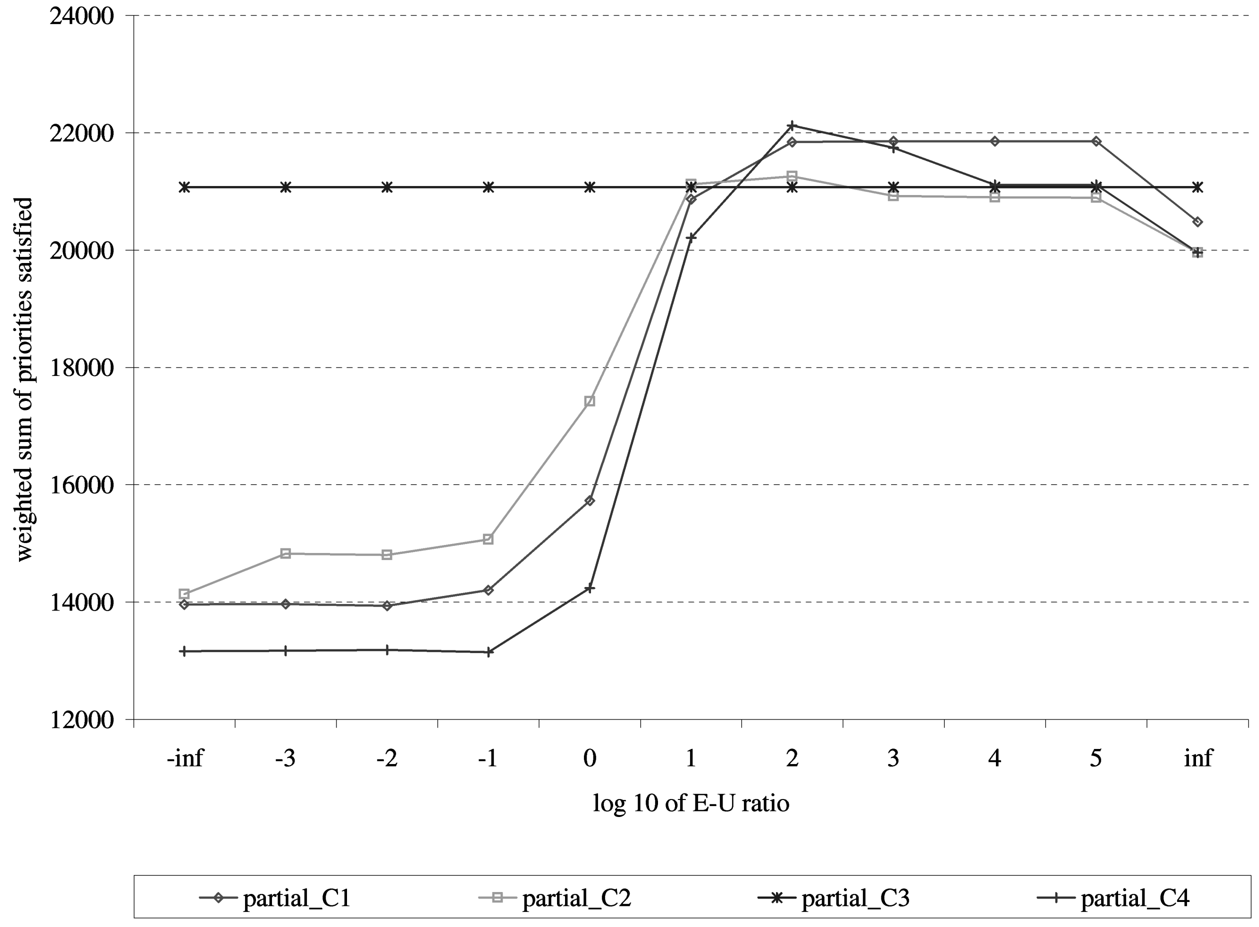


Fig. 7. The partial path heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

ordering of the data items is arbitrary). If a conflict arises, e.g., the link a transfer is attempting to schedule is no longer available, the request is dropped and not satisfied. This approach is referred to as **single Dijkstra random** (shown as **single_Dij_random** in Fig. 6) because Dijkstra's algorithm is only executed once for each data item. This random based method is used to illustrate that executing Dijkstra's algorithm more than once, with updated communication system information, is advantageous.

The only difference between the second random procedure and the partial path heuristic is that, instead of choosing a valid communication step using a cost function as discussed for the partial path heuristic, the **Dijkstra random heuristic** (shown as **random_Dijkstra** in Fig. 6) randomly chooses an arbitrary valid communication step to schedule. This heuristic is used to show the importance of using a cost criterion for decision making.

The first (looser) upper bound used for comparison (shown as **upper_bound** in Fig. 6) is the total weighted sum of the priorities of all requests in the system (it assumes all requests can be satisfied). The second upper bound represents those requests that could be satisfied if each were the only request in the system (shown as **possible_satisfy** in Fig. 6). The loose upper bound is not equal to the upper bound because some requests cannot be satisfied due to lack of link bandwidth and/or machine storage (even when it is the only request in the system).

5.3 Parameters Used in Experiments

Creating the properties of a network structure that are expected to occur in the field is a difficult endeavor. The parameters used were chosen to reflect a representative subset of a BADD-like environment. The number of machines in the communication system is between ten and twelve. Each machine has between 10 MB to 20 GB memory storage capacity. The 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 between four and seven. There are at most two physical unidirectional transmission links between any two machines (there can be none). The adjacency matrix is created by selecting each machine in the network and randomly determining the outbound degree of the machine to be between the bounds mentioned above. Once the outbound degree is chosen, the end machines for the links are randomly generated. The network creation software makes sure that a link does not originate and end at the same machine.

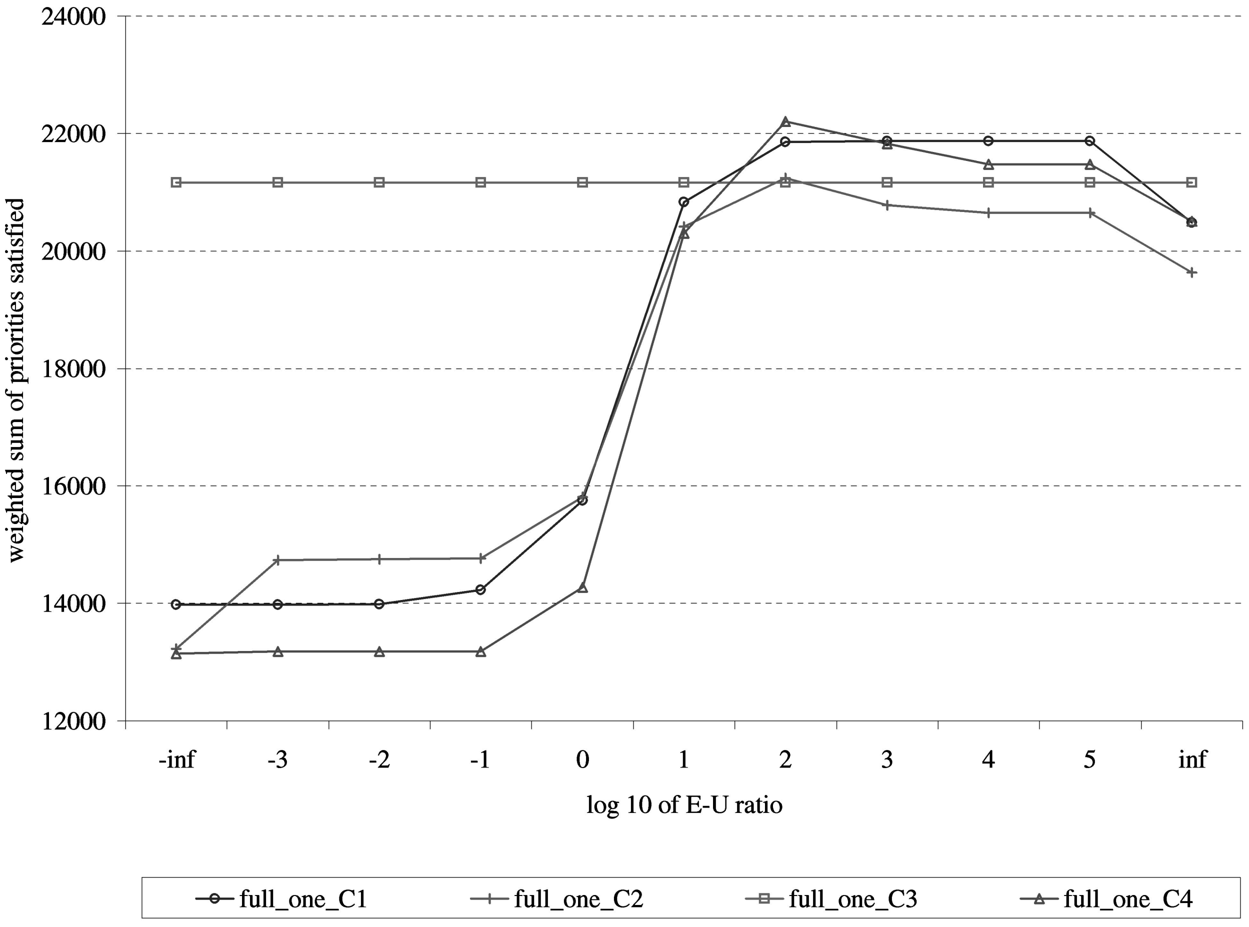


Fig. 8. The full path/one destination heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

The total number of data requests is 20 to 40 times the number of machines in the system. The sources and destinations for a data item are randomly selected from the set of machines in the system such that: 1) there are at most five sources, 2) there are at most five destinations, and 3) a destination for a data item is not also a source of the same data item. Each data item size ranges from 10 KB to 100 MB. The simulations that were performed utilized two different priority weightings. The first used a weighting of 1, 5, and 10 for low, medium, and high priority requests, and the second used a weighting of 1, 10, and 100 for low, medium, and high priority requests. Each data item that a destination requests has an associated priority; therefore, two destinations that request the same data item may have differing priorities for that data item.

The bandwidth of each physical transmission link is between 10Kbit/sec and 1.5 Mbit/sec. The link availability times were generated as follows. First, a duration for a particular virtual communication link between two machines was chosen from the set {30 minutes, one hour, two hours, four hours}. The percentage of the day (24 hours) that a given physical link is available is then chosen between 50 and 100 percent of the day, in increments of 10 percent. The number of virtual communication links is then determined by taking the time the link is available during the day (percentage available $*$ 24 hours) and dividing by the virtual link duration chosen above (for a given communication link, all virtual links will have the same duration). The starting time of the first virtual link is randomly chosen between 0 and (1/3) $*$ (total unavailable time of the communication link). The unavailable time between virtual links is randomly chosen such that: 1) no two virtual links for the same physical link overlap in time, 2) the percentage of the day the physical link is available is as chosen above, and 3) the number of virtual links is as above calculated.

The starting time for a data item is sometime between 0 and 60 minutes (0 signifying some start time of the scheduling period, such as midnight or 6 a.m.). The deadline of a request for a data item is 15 to 60 minutes after the data item's available time. Thus, the effective duration of the simulation is two hours. The time duration parameter for garbage collection, γ, was set to six minutes. Therefore, a particular intermediate machine $M[r]$ will keep a data item in its local memory for six minutes after the latest deadline for $Rq[i]$. Sources and final destinations hold data for the remainder of the simulation.

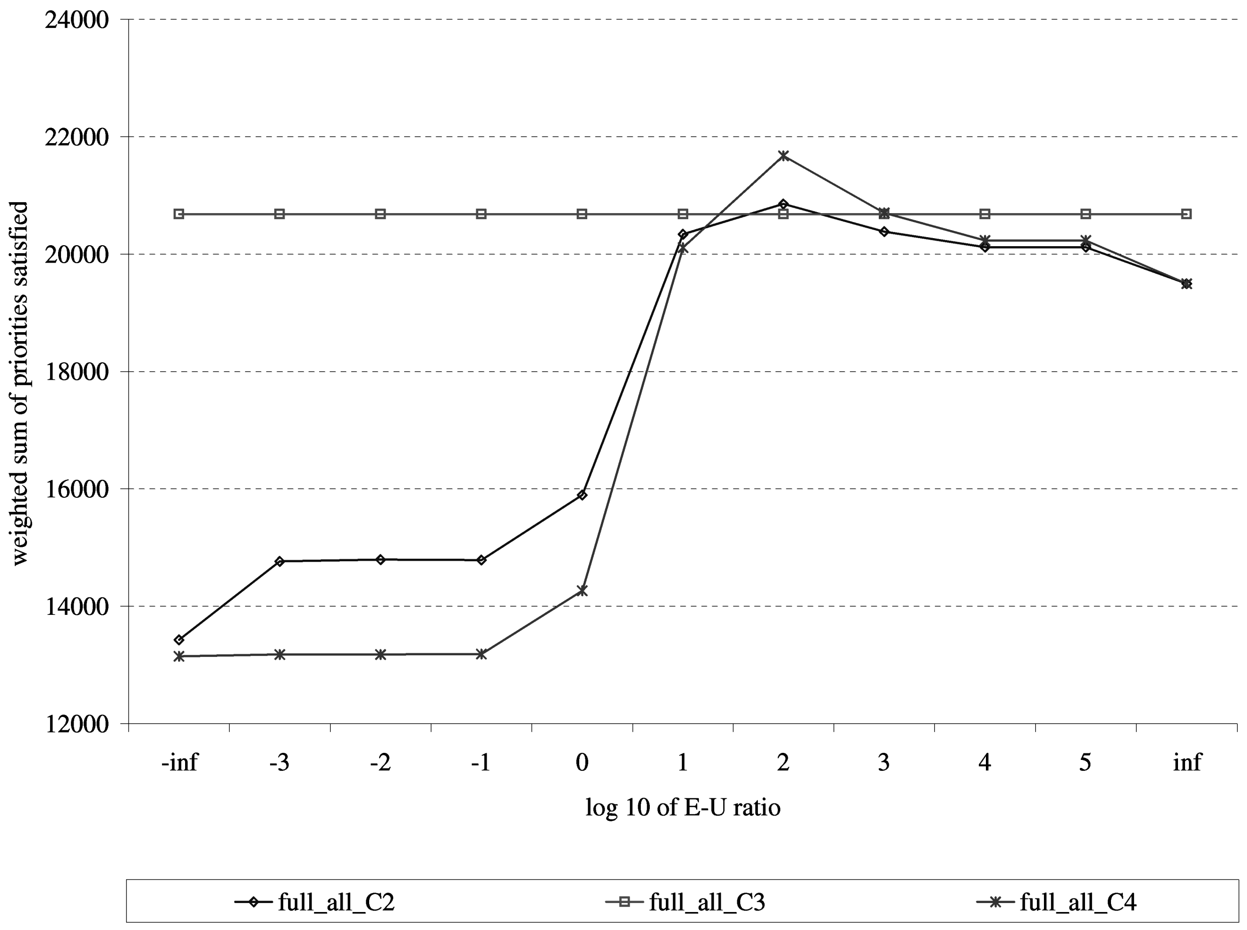


Fig. 9. The full path/all destinations heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

Let the **E-U ratio** be W_E/W_U. As shown by the cost functions introduced in Section 4.8, the E-U ratio may affect the performance of the cost criterion ($Cost_1$, $Cost_2$, and $Cost_4$). Figs. 6, 7, 8, 9, and 10 show the performance of the heuristic/cost-criterion pairings examined. It can be seen from the figures how the E-U ratio affects the performance of the heuristic/cost-criterion pairings. All data points in Figs. 6, 7, 8, 9, and 10 are the average of the same 40 randomly generated test cases.

5.4 Evaluation of Simulations

The results shown in Figs. 6, 7, 8, and 9, are for the 1, 10, 100 weighting scheme. The results of the 1, 5, 10 weighting are similar and are not shown here (see [24]). In the interval from -3 to 5, the horizontal axis contains the $\log_{10}$ of the E-U ratio. The points $\inf$ and $-\inf$ are the two extremes, where inf only considers the effective priority term, while $-inf$ only considers the urgency term.

Fig. 6 shows average lower and upper bounds (defined in Section 5.2) and the average performance of the best cost criterion for each of the heuristics, which happens to be $Cost_4$. (The minimum and maximum values for the performance of these heuristics over the 40 individual test cases with $Cost_4$ are presented in [24].) The performance of the partial path heuristic is shown in Fig. 7, the full path/one destination heuristic in Fig. 8, and the full path/all destinations heuristic in Fig. 9 (recall that $Cost_1$ does not capture that a data item can be sent to multiple destinations and is therefore not considered in the full path/all destination heuristic). These graphs highlight the performance of the four cost criteria for each of the heuristics implemented.

Each of the cost criteria was developed for specific features. The best cost criterion of the four investigated is $Cost_4$. This cost criterion combines the sum of the priorities and the sum of the urgencies of multiple destinations whose shortest path passes through a particular node. $Cost_1$ performs worse than $Cost_4$, because it does not consider moving data to satisfy multiple requests, which $Cost_4$ does. The drawback of $Cost_2$ is that the minimum urgency term allows nonurgent data items within Drq to become scheduled and block more urgent requests for other data items. Using the ratio of priority and urgency in $Cost_3$ was meant to directly associate the priority of a request with its particular urgency. This allows a nonurgent and urgent request for a particular data item to be represented fairly (i.e., take care of the problem with $Cost_2$). The results in Figs. 7, 8, and 9, show that this was not the case, most likely due to scaling (i.e., one very small $Urgency[i, j]$ may have

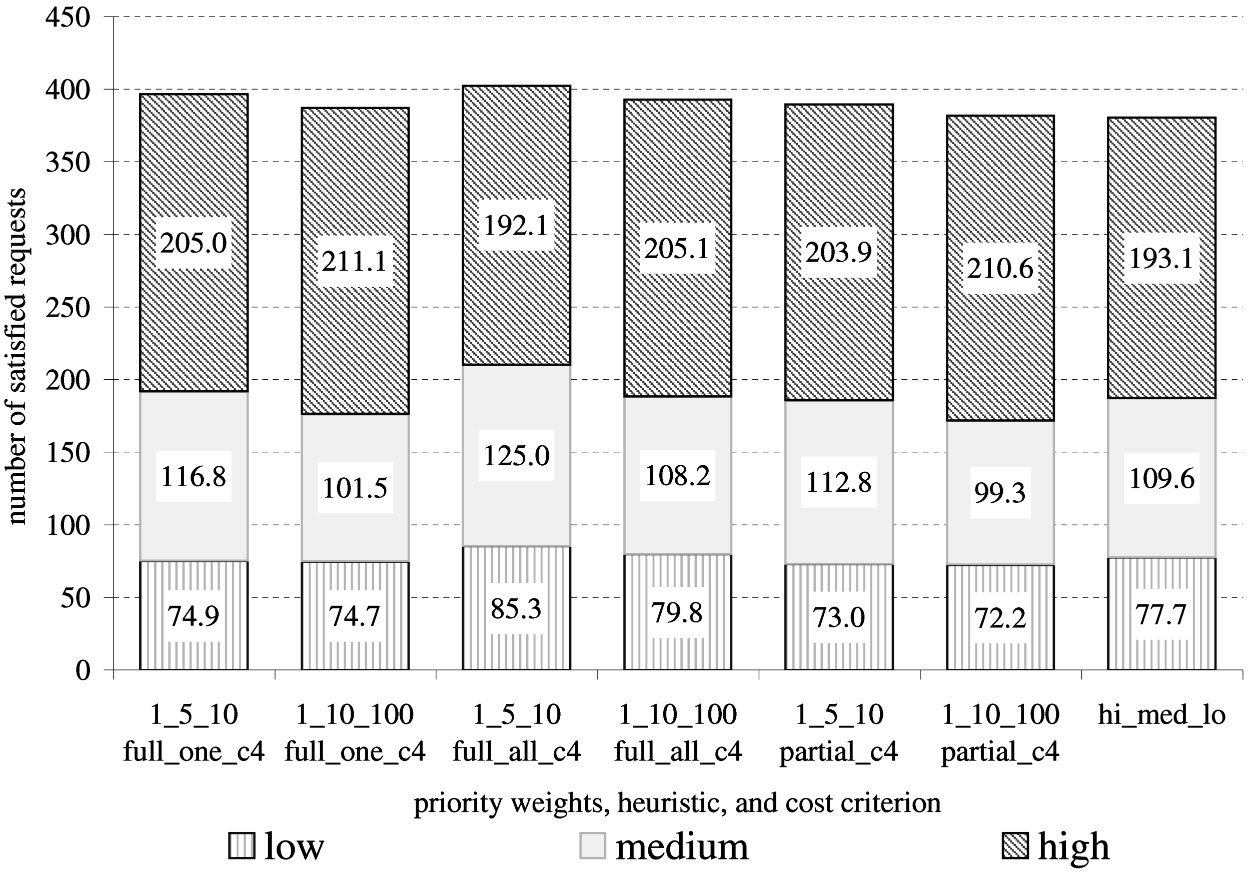


Fig. 10. A comparison of three heuristic's best E-U ratio/cost criterion combinations using different priority weighting terms.

too much impact on the total cost). Future cost criteria might be designed to capture the original intent.

An advantage of $Cost_3$ is that it gave results that were close to those of $Cost_4$ with its best E-U ratio, while being independent of the E-U ratio. Thus, in environments where it is difficult to predict which E-U ratio to use, $Cost_3$ may be preferred.

When examining the schedule that the partial path heuristic created, it was noted that often when a data item started being scheduled, this particular data item was repeatedly transferred until it reached a destination. Because of this trend, the full path/one destination heuristic was tested. A problem with the partial path heuristic is that a data item can be partially scheduled through the network, and then a second data item can be partially scheduled through the network. At this point, the first data item may attempt to finish being scheduled, and may be blocked by the second data item from reaching a destination. The full path/one destination heuristic corrects this problem and outperforms the partial path heuristic, as can be seen in Fig. 6. In addition, the full path heuristics were intended to reduce the number of runs of Dijkstra's algorithm, hence reducing execution time, as was found to be true in the experiments.

Because the full path/one destination heuristic performed so well, it was thought that scheduling a data item to all of the destinations whose path passes through a particular node would perform even better, the definition of the full path/all destinations heuristic. This heuristic did not perform as well as the other two. This may be because this heuristic would schedule less important requests (by sending to all requesting destinations using the intermediate node under evaluation) and block other more important requests from being satisfied. Fig. 6 showcases these trends. The full path/all destination heuristic did have the smallest execution time, as was expected.

For the best performing E-U ratio, for each of the heuristic/cost-criterion pairs, the average number of links a satisfied data item traverses from a source to a destination was measured in the range 1.5 to 1.6. All potentially satisfiable data items have an average shortest path length that is also in the above range. The maximum path length taken by a satisfied data item was measured as nine links.

The various heuristics have differing execution times. The ranges on execution time because each of the 40 test cases correspond to a different scheduling problem with a different optimal solution. The average execution times for the three heuristics using the 1, 10, 100 weighting scheme (for each heuristic's best cost criterion and best average E-U ratio) are: partial path—74 seconds, full path/one destination—53 seconds, and full path/all destinations—50 seconds. The random Dijkstra heuristic execution time (157 seconds) is larger than the cost-based heuristics because the random Dijkstra heuristic will spend time processing data items that cannot possibly satisfy their associated deadlines, whereas the cost-based heuristics will eliminate such data items from consideration.

The last figure, Fig. 10, shows the comparison between the two priority weighting schemes for the best combination of heuristic, cost criterion, and E-U ratio. It can be seen that the 1, 10, 100 weighting satisfies more higher priority requests and fewer medium and low priority requests than the 1, 5, 10 weighting scheme, as was expected.

The last column in Fig. 10 **(hi_med_lo)** shows the number of satisfied requests in the system by processing all the high priority requests before any of the medium priority requests, and all of the medium priority requests before any of the low priority requests. Each request follows its Dijkstra's shortest path (and Dijkstra's algorithm is rerun after each request is scheduled). To implement this process, the full path/one destination heuristic was used in conjunction with $Cost_1$, a $\log_{10}(W_E/W_U)$ of infinity, and a priority weighting scheme of 1, 10, 100. Choosing this combination causes only the priority term of the cost function to be used in determining which single request will get network resources. This method provides a cost-guided (versus arbitrary) approach to basing scheduling decisions only on the priority of individual requests. Recall that the priority of the requests was determined such that each priority level was assigned to approximately one-third of all the requests.

The other heuristics shown in Fig. 10 (with $Cost_4$) satisfy more high priority requests than the hi_med_lo method. This is due to $Cost_4$'s collective view of destinations and the incorporation of the urgency term. Furthermore, the total weighted sum of priorities satisfied for hi_med_lo is the lowest of the techniques shown for each of the priority weighting schemes.

6 Conclusions

Data staging is an important data management issue for distributed computer systems. It addresses the issues of distributing and storing over numerous geographically dispersed locations, both repository data and continually generated data through an oversubscribed network, where not all data requests can be satisfied. 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.

The performance of eleven heuristic/cost-criterion pairs were shown, and compared to upper and lower bounds. Two different weightings for the relative importance of different priority levels were considered. Each heuristic and cost criterion had advantages. The results presented show that for the system parameters considered (e.g., priority weighting, network loads), the combination of the $Cost_4$ and the full path/one destination heuristic performed the best, when using the measure of weighted sum of priorities satisfied.

Because each heuristic/cost-criterion pair has advantages, the pair that performs best may differ depending on the system parameters (i.e., the actual environment where the scheduler heuristic/cost-criterion pair will be deployed). If one was also concerned with execution time, the full path/all destinations heuristic results in a comparable weighted sum of priorities satisfied with a slightly faster execution time. Both of these heuristics allowed more highest priority messages to be received than a simple-cost-based heuristic that schedules all highest priority messages first. Future work will explore how the heuristics perform when varying the congestion of the network and when additional priority weighting schemes are considered.

Appendix

Glossary of Notations

$A_L[s,r][i][k]$: 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 $M[j]$

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

$Cost_1[s,r][i,j][k]$: a cost for transferring the requested data item $Rq[i]$ associated with the j-th destination, from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$

$Cost_2[s,r][i][k]$: a cost for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$

$Cost_3[s,r][i][k]$: a cost for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$

$Cost_4[s,r][i][k]$: a cost for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s,r][k]$

$\mid d \mid$: size of the associated data item d

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

$Drq[i,r]$: set of destination machines requesting a data item $Rq[i]$ whose shortest paths go through machine $M[r]$

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

$e[j]$: current estimate of earliest arrival time for data item at node j in Dijkstra's algorithm

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

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

$Efp[i,r](j)$: effective priority for the data request of $Rq[i]$ from its j th requesting location, that passes through intermediate machine $M[r]$

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

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

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

$G_{nt}[i]$: an instantiation of G_{nt} for $Rq[i]$

γ: number of time units after latest deadline for a data item when it is removed from any intermediate machines

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

$L[i,j][k]$: kth 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

$M[i]$: ith machine in the communication system, $0 \leq i < m$

n: number of the data items with distinctive names (identifiers) 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

$Priority[j,k]$: priority for the data request of the data item

$\pi[v]$: predecessor field (or predecessor vertex) of vertex v $Rq[j]$ on its kth requesting location

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

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

ρ: number of the *requested* data items with distinctive names (identifiers) in the corresponding communication system

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

$Sat[i,r](j)$: satisfiability function associated with the jth requesting location for data item $Rq[i]$ and using intermediate machine $M[r]$

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

$Source[i,j]$: jth 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 using schedule S_h

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

$Urgency[i,r](j)$: urgency for the data request of $Rq[i]$ from its jth requesting location that passes through machine $M[r]$

v_d: a specific destination vertex

v_s: a specific source vertex

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

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

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 $v_s \in V_S[i]$ have been determined during the execution of Dijkstra's algorithm

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

V_S: set of source vertices

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

$W[i]$: relative weight of the ith priority

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

W_U: relative weight for the urgency factor in the scheduling

ACKNOWLEDGMENTS

This research was supported by the DARPA/ISO BADD Program and the US Office of Naval Research under grant N00014-97-1-0804, by NRaD under contract N66001-96-M-2277, and by the DARPA/ITO AICE Program under contracts DABT63-99-C-0010 and DABT63-99-C-0012. M.D. Theys was also supported by a Purdue Benjamin Meisner Fellowship, an Intel Fellowship, and an AFCEA Fellowship. Equipment used in this research was donated by Intel.

The authors would also like to thank Bob Beaton, Bob Douglass, Surja Chatterjea, Richard F. Freund, Debra Hensgen, and Joe Rockmore for their comments and the anonymous reviewers for the helpful suggestions they provided. Preliminary versions of some portions of this material were presented at the Seventh Heterogeneous Computing Workshop (HCW '98) and at the 20th Conference on Distributed Computing Systems.

REFERENCES

[1] S. Acharya and S.B. Zdonik, "An Efficient Scheme for Dynamic Data Replication," Technical Report CS-93-43, Dept. of Computer Science, Brown Univ., Sept. 1993.

[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, pp. 18-27, Mar.-Apr. 1997.

[3] S. Balakrishnan and F. Özgüner, "A Priority-Driven Flow Control Mechanism for Real-Time Traffic in Multiprocessor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 7, pp. 664-678, July 1998.

[4] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-Based Caching," *IEEE Concurrency,* vol. 5, no. 1, pp. 56-67, Jan.-Mar. 1997.

[5] M.A. Bonuccelli and M.C. Clo, "EDD Algorithm Performance Guarantee for Periodic Hard-Real-Time Scheduling in Distributed Systems," *Proc. 13th Int'l Parallel Processing Symp. and 10th Symp. on Parallel and Distributed Programming (IPPS/SPDP '99),* Apr. 1999.

[6] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bööni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," *Proc. IEEE Workshop Advances in Parallel and Distributed Systems,* pp. 330-335, Oct. 1998.

[7] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A Comparison Study of Static Mapping Heuristics for a Class of Meta-Tasks on Heterogeneous Computing Systems," *Proc. Eighth IEEE Workshop on Heterogeneous Computing Systems (HCW '99),* pp. 15-29, Apr. 1999.

[8] R. Chandrasekaran and A. Dauchety, "Location on Tree Networks: P-Centre and n-Dispersion Problems," *Math. Operations Research,* vol. 6, no. 1, pp. 50-57, Feb. 1981.

[9] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* Cambridge, Mass.: MIT Press, 1990.

[10] 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, pp. 847-858, July-Aug. 1980.

[11] P. Danzig, R. Hall, and M. Schwartz, "A Case for Caching File Objects Inside Internetworks," Technical Report CU-CS-642-93, Computer Science Dept., Univ. of Colorado, Mar. 1993.

[12] D. Hensgen, T. Kidd, D. St. John, M. Schnaidt, H.J. Siegel, T. Braun, M. Maheswaran, S. Ali, J. Kim, C. Irvine, T. Levin, R. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, "An Overview of MSHN: The Management System for Heterogeneous Networks," *Proc. Eighth IEEE Workshop on Heterogeneous Computing Systems (HCW '99),* pp. 184-198, Apr. 1999.

[13] A.P. Hurter and J.S. Martinich, *Facility Location and the Theory of Production.* Norwell, Mass.: Kluwer Academic Publishers, 1989.

[14] 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, pp. 661-669, July-Aug. 1995.

[15] M.J. Lemanski and J.C. Benton, "Simulation for SmartNet Scheduling of Asynchronous Transfer Mode Virtual Channels," masters of science thesis, Dept. Computer Science, Naval Postgraduate School, June 1997.

[16] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *J. Parallel and Distributed Computing: Special Issue on Software Support for Distributed Computing,* vol. 59, no. 2, pp. 107-121, Nov. 1999.

[17] I.D. Moon and S.S. Chaudhry, "An Analysis of Network Location Problems with Distance Constraints," *Management Science,* vol. 30, no. 3, pp. 290-307, Mar. 1984.

[18] A.J. Rockmore, "BADD Functional Description," internal DARPA memo, Feb. 1996.

[19] D.R. Shier, "A Min-Max Theorem for p-Center Problems on a Tree," *Transportation Science,* vol. 11, no. 3, pp. 243-252, Aug. 1977.

[20] SmartNet/Heterogeneous Computing Team, "BC2A/TACITUS/BADD integration plan," internal NRaD Naval laboratory report, Aug. 1996.

[21] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazzo, *Deadline Scheduling for Real-Time Systems.* Boston: Kluwer Academic Publishers 1998.

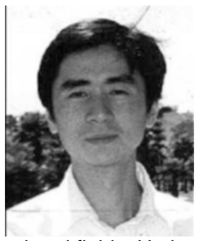
[22] M. Tan, H.J. Siegel, J.K. Antonio, and Y.A. Li, "Minimizing the Application Execution Time through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, pp. 857-871, Aug. 1997.

[23] M. Tan and H.J. Siegel, "A Stochastic Model for Heterogeneous Computing and Its Application in Data Relocation Scheme Development," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 11, pp. 1,088-1,101, Nov. 1988.

[24] M.D. Theys, M. Tan, N. Beck, H.J. Siegel, and M. Jurczyk, "Heuristics and a Mathematical Framework for Scheduling Data Requests in a Distributed Communication Network," Technical Report TR-ECE 99-2, School of Electrical and Computer Eng., Purdue Univ., Jan. 1999.

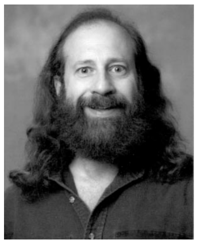
[25] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *J. Parallel and Distributed Computing,* vol. 47, no. 1, pp. 1-15, Nov. 1997.

Mitchell D. Theys received the BS degree in computer and electrical engineering in 1993, the MS degree in electrical engineering in 1996, and the PhD in electrical engineering in 1999, all from Purdue University. He is currently a professor in the Electrical Engineering and Computer Science Department at the University of Illinois at Chicago. His current research interests include: distributed computing, heterogeneous computing, network scheduling, parallel processing, VLSI design, and computer architecture. During his college career, he held various intern positions with Caterpillar Inc., Compaq Computer Corporation, and Lawrence Livermore National Laboratory. In addition, during his undergraduate work, he participated in the cooperative education program and worked in the sales, marketing, quality assurance, and research and design departments of S&C Electric Company. He has published several journal papers and also had several documents reviewed and accepted at conferences, such as the International Conference on Parallel Processing and the Heterogeneous Computing Workshop. He has received support from the US Defense Advanced Research Projects Agency (DARPA), Intel, Microsoft, and the US Armed Forces Communications and Electronics Association (AFCEA). Dr. Theys is a member of the IEEE, the IEEE Computer Society, Eta Kappa Nu, and Tau Beta Pi.



Min Tan received the BA in mathematics and physics from Western Maryland College, as well as the MS degree in electrical engineering and the PhD degree in computer and electrical engineering from Purdue University. During his academic career, his research interests have included parallel and distributed processing, heterogeneous computing, and network communication. He has authored or coauthored more than 20 technical papers in these and related fields. He has worked at Cisco Systems and Segue Software as a research and development engineer and a technical sales consultant. He is currently a cofounder of Dealist.com, an Internet service provider for Greater China and Asian business-to-business communities.

Noah B. Beck received the BS in computer engineering in 1997 and the MS in electrical engineering in 1999, both from Purdue University. While an undergraduate, he spent semesters working for Siemens Stromberg-Carlson on the EWSD telephone switch, and for Intel Corporation on the Willamette x86 microprocessor core. As a graduate student, he researched data staging in heterogeneous networks. After completing the MS degree, Mr. Beck went to work for Sun Microsystems at the Boston Design Center, where he is currently a verification engineer working on a future version of the UltraSPARC microprocessor. His research interests include microprocessor functional and performance verification, as well as computer architecture, parallel computing, and heterogeneous computing.



H.J. Siegel received the BS degrees in electrical engineering and management from the Massachusetts Institute of Technology (MIT), and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He is a professor in the School of Electrical and Computer Engineering at Purdue University. In August 2001, he will hold the endowed chair position of Abell Professor of Electrical and Computer Engineering at Colorado State University. He is a fellow of the IEEE and a fellow of the ACM. Professor Siegel has coauthored more than 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a coeditor-in-chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. Professor Siegel's research interests include heterogeneous parallel and distributed computing, communication networks, parallel algorithms, interconnection networks, and reconfigurable parallel computer systems. He was program chair/cochair of three international conferences, general chair/cochair of four international conferences, and chair/cochair of four workshops. He is a member of the Eta Kappa Nu electrical engineering honorary society and the Sigma Xi science honorary society. He is an international keynote speaker and tutorial lecturer, as well as a consultant for government and industry.

Michael Jurczyk studied electrical engineering at Purdue University and the University of Bochum, Germany, where he received his diploma in 1990. He obtained the 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. He is currently an assistant professor in the Computer Engineering and Computer Science Department as well as an adjunct professor at the Electrical Engineering 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. He is a member of the IEEE and the IEEE Computer Society.