

Scheduling Heuristics for Data Requests in an Oversubscribed Network with Priorities and Deadlines

Mitchell D. Theys
University of Illinois at Chicago
Elec. Engr. and Comp. Sci. Dept. (MC 154)
851 S. Morgan St. RM 1120
Chicago, IL 60607-7053, USA
mtheys@eecs.uic.edu

Michael Jurczyk
Dept. of Comp. Engr. and Comp. Sci.
University of Missouri - Columbia
Columbia, MO 65211, USA
mjurczyk@cecs.missouri.edu

Noah B. Beck and Howard Jay Siegel
School of Elec. and Comp. Engr.
Purdue University
West Lafayette, IN 47907-1285, USA
{noah, hj}@purdue.edu

Min Tan
Segue Software
142 South Santa Cruz Ave.
Los Gatos, CA 95030, USA
mtan@segue.com

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. 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 given. 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, twelve implementations are examined. In addition, two different weightings for the relative importance of different priority levels are considered. The performance of the proposed heuristics is 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.

1. Introduction

The DARPA Battlefield Awareness and Data Dissemination (**BADD**) program [11] 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

This research was supported by the DARPA/ISO BADD Program and the Office of Naval Research under grant number N00014-97-1-0804, by NRaD under contract number N66001-96-M-2277, and by the DARPA/ITO AICE Program under contract numbers DABT63-99-C-0010 and 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.

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 complicated by the dynamic nature of data requests and network congestion; the limited storage space at certain sites; the limited bandwidth of links; the changing availability of links and data; the time constraints of the needed data; the priority of the needed data; and the determination of where to stage the data [13]. Also, the associated garbage collection problem (i.e., determining which data will be deleted or reverse deployed to rear-sites from the forward-deployed units) arises when existing storage limitations become critical [11, 13]. The storage situation becomes even more difficult when copies of data items are allowed to reside on different machines in the network so that there are more available sources from which the requesting applications can obtain certain data (e.g., [15]). Multiple copies of data items also provide an increased level of fault tolerance, in cases of links or storage locations going off-line.

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 machine 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. The full description of the model is presented in [16]; portions are included here to aid in the presentation of the heuristics detailed in this paper.

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, twelve 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.

One of the heuristic/cost-criterion pairs was examined in [16]. That investigation used a simpler model (e.g., there was no garbage collection), and a different set of network parameters (e.g., the number of communication links that can connect to a machine). The research presented here expands the work initiated in [16] by developing two additional heuristics, three cost criteria, and improving the parameter values to better model a BADD-like environment.

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 provided. Section 4 presents the multiple-source shortest-path algorithm based heuristics for finding a near-optimal schedule of the communication steps for data staging. A simulation study is discussed in Section 5, which evaluates the performance of the proposed heuristics outlined in Section 4.

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. Due to space constraints, the reader is referred to [17] for a more thorough discussion of the related work. A problem that is, at a high level, remotely similar to data staging is the facility location problem in management science and operations research (e.g., [9]). Data management problems similar to data staging for the BADD program are studied for other communication systems [1, 2, 4, 8]. Other areas that are somewhat related include modifying routing schemes [3], mapping tasks onto a suite of distributed heterogeneous machines (e.g., [5, 6, 18]), and earliest deadline first [12, 14] scheduling for real-time systems. Lastly, other research exploring heuristics for use in the BADD environment have been performed [10]. All of this research is related, but does not develop a mathematical model like the one researched here nor do they examine a network similar to BADD-like network being used in this research.

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 communication system \underline{M} consists of m machines $\{M[0], M[1], \dots, M[m-1]\}$. 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. $\underline{Cap}[i](t)$ represents the available memory storage capacity of machine $M[i]$ ($0 \leq i < m$) 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 \underline{G}_{nt} specifies the connectivity of the communication system for the machines in M . A set of m vertices $\underline{V} = \{V[0], V[1], \dots, V[m-1]\}$ is generated that corresponds to the m machines in the communication system, where node $V[i]$ 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 uni-directional. A bi-directional link between two machines is represented as two different virtual uni-directional links that correspond to the transmission link in each direction (for each time interval).

Let $Nl[i, j]$ be the total number of unidirectional virtual communication links from $M[i]$ to $M[j]$. $L[i, j][k]$ denotes the k -th 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 $E[i, j][k]$ from $V[i]$ to $V[j]$ is included in \underline{G}_{nt} . All the included edges constitute the set of edges \underline{E} of \underline{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 $\underline{Lst}[i, j][k]$ denote the time when link $L[i, j][k]$ becomes available (link start time) and $\underline{Let}[i, j][k]$ 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 $\underline{Lst}[i, j][k]$ (starting time) and $\underline{Let}[i, j][k]$ (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 \underline{d} , $|d|$ represents the size of the associated data item. Let $\underline{D}[i, j][k](|d|)$ denote the communication time for transferring data item d (from machine $M[i]$ to machine $M[j]$) through their k -th dedicated virtual link within the time interval $[\underline{Lst}[i, j][k], \underline{Let}[i, j][k]]$. Time $\underline{D}[i, j][k](|d|)$ includes all the various hardware and software related components of the inter-machine communication overhead (e.g., network latency and the time for data format conversion between $M[i]$ and $M[j]$ when necessary). Machines $M[i]$ and/or $M[j]$ may be intermediate nodes for transferring d rather than the original source or the final destination node of d .

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 may relax this assumption.

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

Suppose ρ is the number of *requested* data items with distinctive names (identifiers) in the corresponding communication system M , where ($0 \leq \rho \leq n$). Let $\underline{Rq} = \{Rq[0], Rq[1], \dots, Rq[\rho-1]\}$ be the set of the requested da-

ta items. Each $Rq[j]$ ($0 \leq j < \rho$) 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 $Nrq[j]$ denote the number of different requests for $Rq[j]$. $Request[j, k]$ denotes the number of the machine from which the k -th 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, $Rft[j, k]$ denotes the finishing time (or deadline) after which the data item $Rq[j]$ on its k -th requesting location is no longer useful (e.g., data items may be needed before a specific time when certain decisions must be made). The deadlines are set by the users' applications. 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 its 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 P is the highest priority possible (i.e., P corresponds to the class of most important requests). $Priority[j, k]$ denotes the priority for the data request of the data item $Rq[j]$ on its k -th requesting location. 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 $W[i]$ ($0 \leq i \leq P$) denotes the relative weight of the i -th 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 $\underline{S} = \{S_0, S_1, \dots, S_{\sigma-1}\}$ denote a set of σ distinct schedules for the communication steps of transmitting requested data items. Consider a specific schedule S_h , where $0 \leq h < \sigma$. The k -th request for data item $Rq[j]$ is satisfiable with respect to S_h if $Rq[j]$ can be obtained by the requesting machine, $M[Request[j, k]]$,

before the deadline, $Rft[j, k]$. Let $Srq[S_h]$ denote the set of two-tuples $\{(j, k) \mid k\text{-th request of the data item } Rq[j] \text{ is satisfiable using schedule } S_h\}$.

The effect, $E[S_h]$, 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 [7]. 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 [7].

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 Subsections 4.2 through 4.4. Subsections 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. Subsection 4.8 presents background information about the cost criteria components, and details the four cost criteria used in this research. The mathematical model developed for the simplified data staging problem appears in [24], where one of the twelve heuristic/cost-criterion pairs was examined using a simpler system configuration. Some information from that paper appears in this section as necessary background for the discussion of the data staging heuristics.

4.2. Adaptation of Dijkstra's Algorithm

For each requested data item $Rq[i]$, an instantiation, $G_{nt}[i]$, of the graph G_{nt} (defined in Section 3) is created.

Let $V_S[i]$ be the set of source nodes corresponding to the machines that are the initial locations of the data item $Rq[i]$. Let $V_D[i]$ 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 $v_s \in V_S[i]$ to a destination node $v_d \in V_D[i]$ 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 [7]. 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 [7]. Then the heuristics consider all data items collectively. A more detailed description of how Dijkstra's algorithm functions in the heuristics can be found in [25].

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 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).

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 Figure 1, 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 Subsection

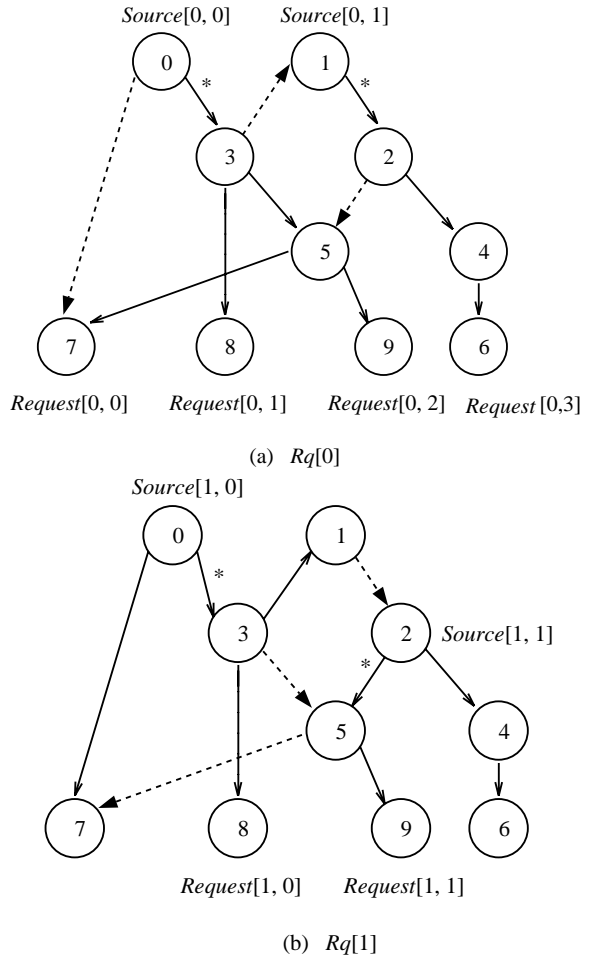


Figure 1. An example communication system that requests (a) $Rq[0]$ and (b) $Rq[1]$.

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).

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 Figure 1(a), 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 duration 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 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 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 Figures 2 and 3) 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. Subsections 4.6 and 4.7 present the other two heuristic methods explored. The various cost criteria used with the heuristics are described in Subsection 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 Figures 2 and 4) attempts to exploit this trend by selecting a requested data item with one of the cost criterion discussed in Subsection 4.8 and scheduling all hops required for the data item to reach its lowest cost destination before executing Dijkstra’s algorithm again.

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, the part of the path constructed may block the paths of the other requested data items, causing them to take less optimal paths or causing them to 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 than 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. The full path/all destinations heuristic (referred to as full_all in Figures 2 and 5) will schedule all paths for a single data item that share the next machine in the path as an intermediate machine. By scheduling the path to multiple destinations, fewer executions of Dijkstra’s algorithm are required as compared to the full path/one destination heuristic.

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 subsection begins by defining “urgency” and “effective priority” and then the four cost criteria used are discussed.

To begin, define $A_T[i, j]$ as the shortest path time obtained from the last execution of Dijkstra’s algorithm, which is a lower bound for the arrival time when requests for all data items are considered collectively. If $Rq[i]$ were the only request in the system, $A_T[i, j]$ is the time for the request to reach machine $Request[i, j]$. Let $Rft[i, j]$ denote the finishing time (or deadline) after which the data item $Rq[i]$ on its j -th 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 j -th 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 j -th 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 Figure 1(a). 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 j -th requesting location and that $W[k]$ ($0 \leq k \leq P$) denotes the relative weight of the k -th priority. Let $Efp[i, r](j)$ denote the effective priority for the data request of $Rq[i]$ from its j -th requesting location, where $Efp[i, r](j) = Sat[i, r](j) * W[Priority[i, j]]$.

Suppose $Urgency[i, r](j)$ denotes the urgency for the data request of $Rq[i]$ from its j -th 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 j -th 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 defined above. 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), (3), and (4), respectively, in Subsection 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 $W_E \geq 0$ is the relative weight for the effective priority factor and $W_U \geq 0$ is the relative weight for the urgency factor in the scheduling.

For the first cost criterion (C1), the $Cost_1[s, r][i, j][k]$, for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$, via link $L[s, r][k]$, for the j -th destination, is defined as: $Cost_1[s, r][i, j][k] = -W_E * Efp[i, r](j)$

$-W_U * Urgency[i, r](j)$. 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 $Cost_2[s, r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s, r][k]$: $Cost_2[s, r][i][k] = -W_E * (\sum_{j \in D_{rq}[i, r]} Efp[i, r](j)) - W_U * (\max_{j \in D_{rq}[i, r]} Urgency[i, r](j))$. 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 $Cost_3[s, r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s, r][k]$ (C3) is: $Cost_3[s, r][i][k] = \sum_{j \in D_{rq}[i, r]} Efp[i, r](j) / Urgency[i, r](j)$. 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 . 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 $Cost_4[s, r][i][k]$ for transferring the requested data item $Rq[i]$ from machine $M[s]$ to $M[r]$ via link $L[s, r][k]$ (C4) is: $Cost_4[s, r][i][k] = -W_E * \sum_{j \in D_{rq}[i, r]} Efp[i, r](j) - W_U * \sum_{j \in D_{rq}[i, r]} Urgency[i, r](j)$. 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.

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 Subsection 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 [7], 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 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.

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 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 Figure 2) 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 Figure 2) 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 Figure 2) 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 Figure 2). 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 10MB to 20GB 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.

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 10KB to 100MB. 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 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 1.5Mbit/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 ten percent. The number of virtual communication links is then determined by taking the time the link is available during the day

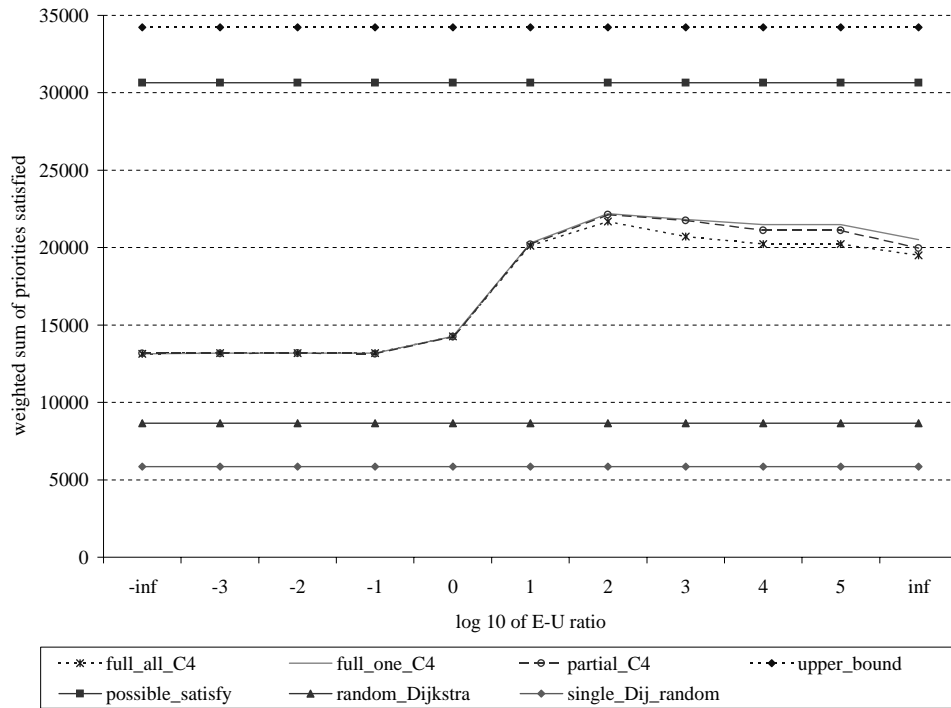


Figure 2. Comparison of the heuristics' best cost criterion performance for the 1, 10, 100 weighting scheme.

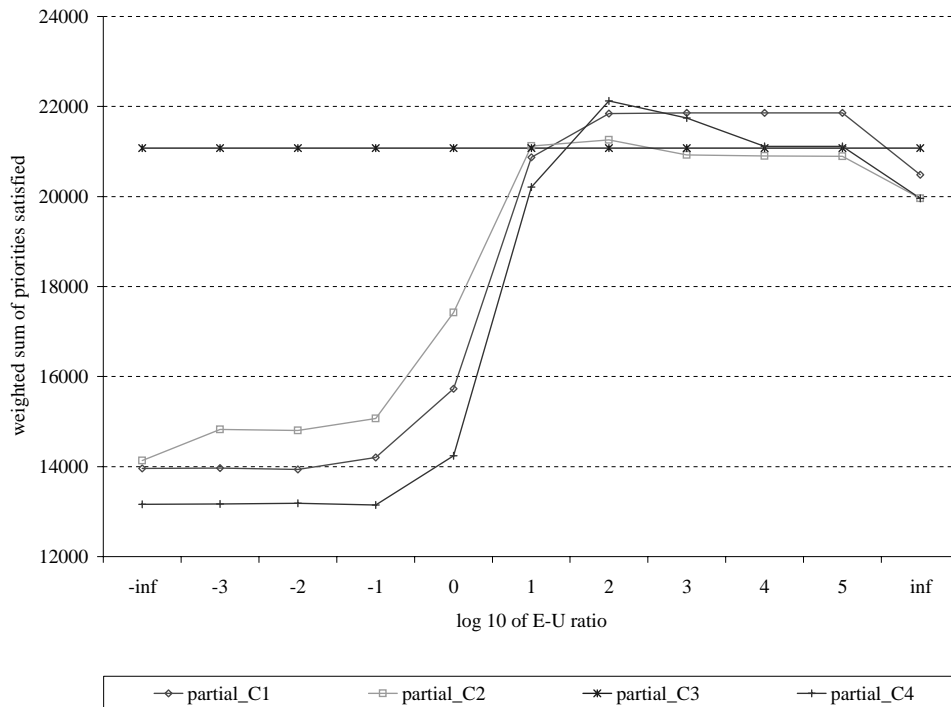


Figure 3. The partial path heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

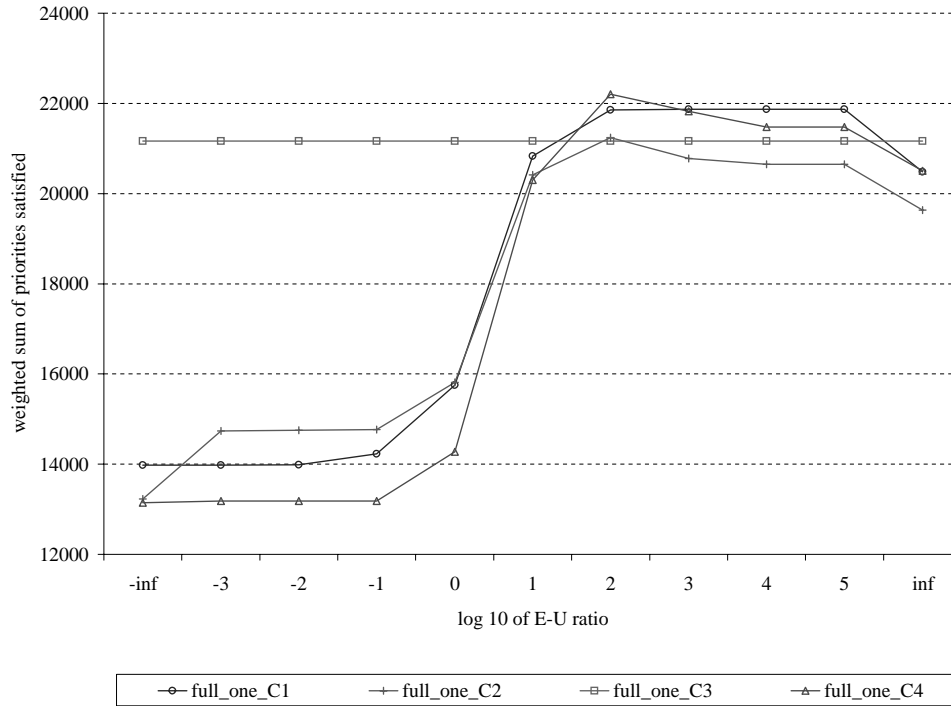


Figure 4. The full path/one destination heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

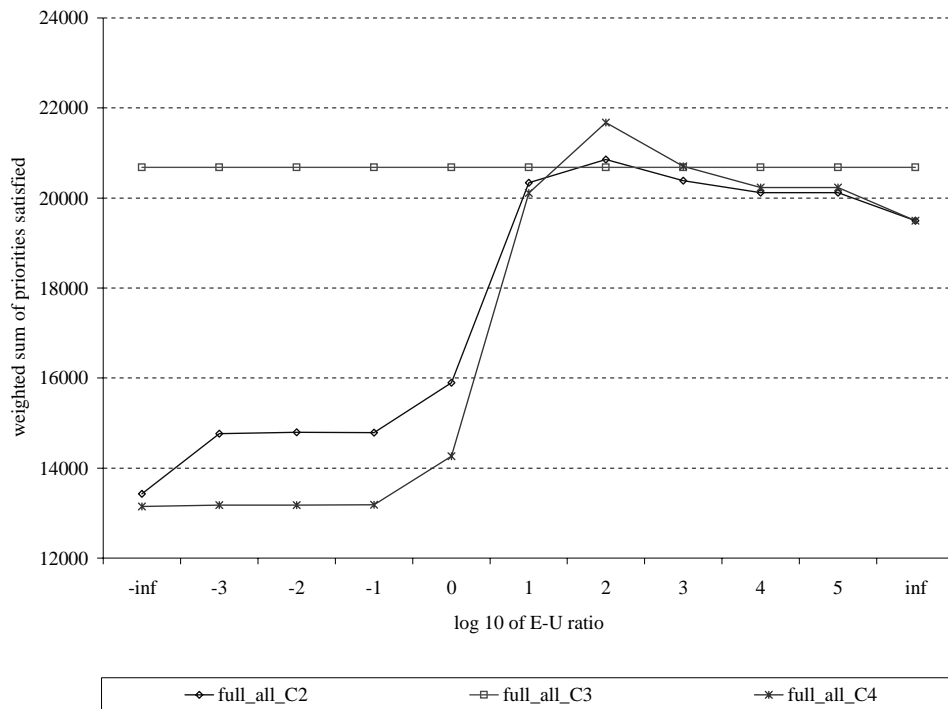


Figure 5. The full path/all destinations heuristic results for the 1, 10, 100 weighting scheme and various cost criteria.

(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 $(1/3) * (\text{total unavailable time of the communication link})$. 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, (3) the number of virtual links is as calculated above, and (4) the unavailable time is > 0 .

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.

Let the E-U ratio be W_E/W_U . As shown by the cost functions introduced in Subsection 4.8, the E-U ratio may affect the performance of the cost criterion ($Cost_1$, $Cost_2$, and $Cost_4$). Figures 2 through 5 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 Figures 2 to 5 are the average of the same 40 randomly generated test cases.

5.4. Evaluation of Simulations

The results shown in Figures 2 through 5 are for the 1, 10, 100 weighting scheme. The results of the 1, 5, 10 weighting are similar and are not shown here (see [17]). 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.

Figure 2 shows average lower and upper bounds (defined in Subsection 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 [17].) The performance of the partial path heuristic is shown in Figure 3, the full path/one destination heuristic in Figure 4, and the full path/all destinations heuristic in Figure 5 (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 Figures 3, 4, and 5, show that this was not the case, most likely due to scaling (i.e., one very small $Urgency[i, j]$ may have 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 gives 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.

Additional results pertaining to execution time and average number of links traversed was collected. This information can be found in [17] due to space constraints.

Lastly, when comparing the performance of the heuristic/cost criterion combinations for differing priority weighting schemes, the 1, 10, 100 weighting satisfies more higher priority requests and fewer medium and low priority requests than the 1, 5, 10 weighting, as was expected. This performance was then compared to a simplified scheme that scheduled all high priority requests before any medium priority requests, and all medium priority requests before any low priority requests. This method provides a cost-guided (versus arbitrary) approach to basing scheduling decisions only on the priority of individual requests. It was found that the heuristic/cost criterion combinations performed better than this simplified scheduling scheme in all cases.

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. Ten of the twelve possible combinations have been first explored in this paper. The eleventh was explored in [16] utilizing a simpler network model. The network model used here has been modified from [16] to allow a more realistic version of garbage collection to be implemented and utilizes network parameters that more closely match a BADD-like environment. The twelfth heuristic/cost-criterion (the full path/all destinations and $Cost_1$) did not make sense and was not examined in this research. 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. This heuristic allowed more highest priority messages to be received than a simple-cost-based heuristic that schedules all highest priority messages first.

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). Future work will explore how the heuristics perform when varying the congestion of the network and when additional priority weighting schemes are considered.

Acknowledgments The authors thank B. Beaton, B. Douglass, S. Chatterjea, R. F. Freund, D. Hensgen, and J. Rockmore for their comments. We also thank the anonymous reviewers for the helpful suggestions they provided.

7. References

- [1] S. Acharya and S. B. Zdonik, "An efficient scheme for dynamic data replication," CS-93-43, CS Dept., Brown Univ., Sept. 1993, 25 pp.
- [2] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm, "Enhancing the web's infrastructure: From caching to replication," *IEEE Internet Computing*, Vol. 1, No. 2, Mar.-Apr. 1997, pp. 18-27.
- [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. 2, July 1998, pp. 664-678.
- [4] A. Bestavros, "WWW traffic reduction and load balancing through server-based caching," *IEEE Concurrency*, Vol. 5, No. 1, Jan.-Mar. 1997, pp. 56-67.
- [5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, 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," *IEEE Workshop on Advances in Parallel and Distributed Systems*, Oct. 1998, pp. 330-335 (included in the proceedings of the 17th IEEE Symp. Reliable Distributed Systems, Oct. 1998).
- [6] 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," *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, Apr. 1999, pp. 15-29.
- [7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.
- [8] P. Danzig, R. Hall, and M. Schwartz, "A case for caching file objects inside internetworks," Technical Report CU-CS-642-93, CS Dept., Univ. of Colorado, Mar. 1993, 15 pp.
- [9] P. C. Jones, T. J. Lowe, G. Muller, N. Xu, Y. Ye, and J. L. Zydiak, "Specially structured uncapacitated facility location problem," *Operations Research*, Vol. 43, No. 4, July-Aug. 1995, pp. 661-669.
- [10] M. J. Lemanski and J. C. Benton, *Simulation for SmartNet Scheduling of Asynchronous Transfer Mode Virtual Channels*, Master's Thesis, CS Dept., Naval Postgraduate School, June 1997.
- [11] A. J. Rockmore, "BADD functional description," Internal DARPA Memo, Feb. 1996, 9 pp.
- [12] M. A. Bonuccelli and M. C. Clo, "EDD algorithm performance guarantee for periodic hard-real-time scheduling in distributed systems," *13th Int'l Parallel Processing Symp. and 10th Symp. Parallel and Distributed Programming (IPPS/SPDP'99)*, Apr. 1999, pp. 668-677.
- [13] SmartNet/Heterogeneous Computing Team, "BC2A/TACITUS/BADD integration plan," Internal NRaD Naval Lab Report, Aug. 1996, 16 pp.
- [14] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers, Boston, MA, 1998.
- [15] 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, Aug. 1997, pp. 857-871.
- [16] M. Tan, M. D. Theys, H. J. Siegel, N. B. Beck, and M. Jurczyk, "A mathematical model, heuristic, and simulation study for a basic data staging problem in a heterogeneous networking environment," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 115-129.
- [17] 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," TR-ECE 99-2, ECE School, Purdue, Jan. 1999, 58 pp.
- [18] 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, Nov. 1997, pp. 1-15.