PROVIDING THE WEB OPERATING SYSTEM WITH AN EFFECTIVE SEARCH METHOD FOR SHARING AVAILABLE RESOURCES

DONALD W. DEARHOLT, UDAI KUMAR KUDIKYALA,
IOANA BANICESCU, and MARK L. BILDERBACK
Mississippi State University
Department of Computer Science
E-mail: {dearholt, kumar, ioana, bilderba}@cs.msstate.edu

Abstract
The world wide interconnect network (Internet), with its huge amount of resources, represents a fertile environment for the execution of applications on remotely connected networks. Recently, a Web operating system (WOS) has been proposed and is currently under development (WOSTM). It would allow some or all of the resources at participating sites to be assigned to those applications submitted to the WOS for execution. Due to the dynamic nature of Internet resource availability, locating these resources for assignment to submitted applications is an essential requirement of a WOS. Locating available resources requires a highly effective search strategy that is able to compensate for networks with high probability of failure rates. This paper proposes the integration of Pathfinder into the WOS. Pathfinder is an algorithm whose search provides fault-tolerance by offering alternative paths at relatively low costs.

Keywords: Pathfinder, Web operating system

1. INTRODUCTION

Recently it has become clear that the operating systems environments must evolve such that all machines in a given network appear to be controlled by the same environment. This has been necessitated by the rapid development of new forms and concepts of networked and mobile computing.

With the recent expansion of the world-wide interconnected network, the execution of applications on remotely connected networks has become more feasible. To achieve this goal, a new Web operating system (WOSTM, a specialized operating system for global computing) is currently under development [6].

A few questions about the applicability of the WOS for global computing have been raised [2]: a) will the overhead cost of the management for global computing outweigh the savings obtained by load sharing; b) will the cost of networking dominate the saving by sharing additional resources; c) will the demand for dependability, reliability, and privacy dominate the demand for savings and/or additional power; and d) will the market and technological constraints allow the development of new computer system layers optimized for global computing at the operating system, machine-instruction, or hardware layers.

Some of the essential issues to be studied in the context of global computing are load sharing, function sharing, performance aggregation, and fault tolerance. Resource management is the
central problem, because it requires the necessity of dealing with large number of physically distributed and statistically unpredictable load and resource data.

One of the main issues is the locating of necessary resources for application executions, since no global catalogue of all available resources exists. Locating resources requires a search of the exiting decentralized warehouses which were presented in [9]. Because of the dynamic nature of the World Wide Interconnected Network (resources connecting and disconnecting dynamically), referred to as the Internet, this requires the execution of an effective search algorithm relatively often. Using a graph to represent the Internet, locating resources translates into a shortest path problem.

Many shortest path algorithms need complete information about the graph and any changes to the graph results in extensive computations, even when the change is not directly related to the path being considered. This paper proposes the integration of Pathfinder as the preferred search algorithm for the WOS. Pathfinder [4] is a search technique which does not require complete knowledge of the graph, and only changes which occur within a bounded region require new computations. In addition, one of the main benefits of Pathfinder is its robustness due to its fault tolerant characteristics.

The rest of this paper is organized as follows. Section §2 will present the characteristics of a Web operating system WOSTM. Section §3 describes the Pathfinder paradigm, recent modifications to the Pathfinder algorithm that make it suitable to being the preferred search algorithm for the WOS, discusses its advantages over other shortest path algorithms. Finally, section §4 evaluates the potential of using Pathfinder as a competitive search technique for the WOSTM.

2. CHARACTERISTICS OF THE WOSTM

The Web operating system (WOSTM) [6] concept proposes a set of virtual operating system services that support and manage distributed, heterogeneous, adaptive and dynamically reconfigurable applications. The WOSTM benefits that enable it to support all requirements of distributed programming and computations are listed below.

- The development of distributed applications becomes easier and more efficient.
- The globally available resources could be efficiently used with the help of a family of adaptive, fault-tolerant load distribution and balancing methods.
- The cost/performance ratio (decreased network latency, shorter response times, etc.) is improved.
- Availability of services and applications is increased without complex administration and reconfiguration problems.

The WOSTM architecture consists of a set of WOS nodes representing a collection of deductive engines [13]. The structure of a WOS node is given in Figure 1. An essential condition for using the WOS is that users have to be registered. The network of registered users forms the WOS-Net. The deductive engines integrate the functionality of the client, the server, as
well as the resource managers on each machine of the WOS-Net. System information is kept in “warehouses” that are constantly updated.

A successful implementation of a WOS requires flexibility and efficiency [13]. To satisfy this requirement, there is a need for cooperation between the resource and load management of interactive and distributed programs. The efficiency is dictated by the implementation of communication, synchronization, and monitoring within the operating system and the other computer system layers. In addition, the organization of the resource management plays an important role in obtaining an efficient implementation. However, existing systems based on client-server architecture need a centralized service. This translates into large overheads and therefore long waiting response times when wide area networks such as the Internet, the World Wide Web, or subsets of these are considered.

Recently, an implementation of WOS under the Message Passing Interface (MPI) has been proposed (WOSMPI) and discussed in detail in [1]. The WOS™ concept gives great possibilities to use a large number of resources of the Internet for distributed computing without need of a-priori knowledge and administration work, while MPI is building a comfortable and already accepted Application Programming Interface. Creating a MPI under the WOS environment, would offer the advantages of both systems, and would overcome many difficulties.

3. THE PATHFINDER-KLIG PARADIGM

The Pathfinder paradigm began as a means of modeling aspects of human semantic memory [14]. An extension proposed [4] makes use of much of the same computations used to compute a Pathfinder network to provide routing information for the most economical route(s) through a network [7] has compared this method with alternative routing algorithms.
A K-Local Image Graph (KLIG) [4] in an arbitrary network is a subgraph in which each node has sufficient memory to store all nodes and interconnecting edges within K edges of itself. It is the responsibility of each node to determine all costs and cost changes of communication with its adjacent neighbor and also transmitting these changes to all nearest neighbors within K edges of itself whenever changes occur. All costs associated with communication through a node are attributed to the edges incident with the node, and each node is capable of executing algorithms using graph structure as data. Each node in the network is assumed to have coordinates and the knowledge of coordinates of all the other nodes within K edges. An effective routing algorithm should not only find the shortest path, reducing the search space while searching for the shortest network routes, but it should also store information about alternate paths which will be useful in case of node or link failures. The resource-sharing problem with the WOS is very similar to the routing (shortest path) problem in an arbitrary network, so the algorithms appropriate for the latter will be effective for the former.

The most frequently used routing algorithms are those of Dijkstra and Bellman-Ford. The Dijkstra and Bellman-Ford algorithms are for single-source shortest-path problems [3]. The Distributed Bellman-Ford algorithm is a modified version of the original algorithm to compute the shortest paths in communication networks in a distributed fashion where links and nodes can fail and recover at arbitrary times. This means that there is no node that is a central controller. In this algorithm the necessary information about the topology of the network is stored in route tables. The information stored in the route table is not complete information about the topology of the network but only the information about the next neighboring node through which the message can reach the destination in the shortest distance. Apart from the route table each node maintains a distance table with an entry for each destination from that node through the links to the neighboring nodes. It is to be noted that the route table is derived from the distance table [12]. The Floyd-Warshall algorithm [3], solves the all-pairs shortest path problem through a dynamic-programming approach. This algorithm considers the intermediate vertices of the shortest path.

The Pathfinder-KLIG algorithm [4] computes the shortest path from a source node to a destination node. In that sense it is a single source shortest path algorithm, but this algorithm also uses an all-pairs shortest path algorithm called Pathfinder to achieve the goal. The goal here is to find the shortest or the minimum cost path from a given source to a given destination. It has to be noted that the cost of a path is the sum of the edge weights along that path. The Pathfinder algorithm [14] eliminates the edges that violate the generalized triangle inequality, thus reducing the search space for finding the shortest paths between any pair of nodes within KLIGi. The Pathfinder algorithm also stores some of the alternate path information between any two nodes within the KLIGi. This information will be useful to consider alternate paths in case of node or link failures. The Pathfinder-KLIG algorithm first selects a subgraph (KLIGi) within K edges of the source node, and utilizes search within that KLIGi and a heuristic estimate of the shortest path outside its local view (KLIGi) in an A* search for the shortest path. If there are no failures, the paradigm guarantees finding the shortest path. In case of failures, the computations within the KLIGi provide information helpful in routing around failures, and can provide close to optimal paths.

In more detail, suppose a resource is to be made available from a target node to a source node with minimum cost. The steps to identify each KLIGi and search for the minimum cost path to the needed resource are as follows:

1. The KLIGi is determined by applying the Node Inclusion Test [4] for each node Ni. Node Nq is in KLIGi iff
(a) $N_q$ is within $K$ edges of message node $N_m$, and
(b) $d(N_q, N_t) \leq d(N_m, N_t)$ where $N_t$ is the target node.

Here the distances are Euclidean. A node in the backsphere is either on the surface equidistant from the target node with the message node, or has the links cut by the locus of the sphere defined by (b) above.

2. The Pathfinder algorithm [14] is performed on the $KLIG_i$ to determine PFN($\tau = 1$, $q = n-1$), where the $\tau$-metric is the Minkowski metric, and the $q$-parameter takes on the value of the number of nodes in the KLIG less one. This step eliminates the edges that violate generalized triangle inequalities and also generates and stores alternative path information that will be useful in case of node or edge failures.

3. The actual network distances are computed from the message node $N_m$ to every frontier node for the $KLIG_i$. The frontier of the $KLIG_i$ is the set of nodes which is furthest from the source node at the limit of the $K$-edge constraint. The Euclidean (heuristic) distance is calculated from each frontier node $N_f_i$ to the target node $N_t$. The cost of each route from the message node to the target node is estimated as: $d = d_n(N_m, N_f_i) + d_h(N_f_i, N_t)$ [4] where $d_n$ is the actual network distance within and $d_h$ is the heuristic distance. The Euclidean cost can be converted into an approximate network cost by multiplying the Euclidean distance with lowest cost per unit length of the network distance.

4. Move the message one edge along the minimum-cost path computed in step 3.

5. Steps one through four are repeated until the message reaches the target node. The search procedure is a modified version of the $A^*$ algorithm.

Several criteria can be applied to compare the performance of the candidate algorithms:

1. Running time: This criterion will measure the order of growth of the running time of the candidate algorithms as a function of the size of the input, i.e. the number of nodes in the network. The comparison will be in terms of worst-case running time.

2. Communication complexity: This criterion represents the number of messages that are generated to get a consistent view of the network and compute shortest paths after detecting a link/node failure [10].

3. Memory requirement: This criterion is concerned with storage requirement of each node to compute an optimal path from that node to the target node. The smaller the amount of memory required to compute the optimal and near-optimal paths, the better will be the performance of that particular algorithm with respect to this criterion.

4. Multiple paths: This criterion requires that the algorithm supports the computation of alternative paths for a destination. This is particularly helpful in case access to a resource must be routed through multiple paths to prevent congestion and oscillations and also in case of link and node failures where alternate routes may be necessary. In addition, the information about multiple paths to a destination will aid in splitting traffic over multiple paths which is proven mathematically to reduce average delay of traffic in a network [5].

Let's consider the candidate algorithms performance with respect to each of the criteria mentioned above separately.
Running time

The Dijkstra's algorithm has a complexity of $O(V^2)$ to find a single-source shortest path and has a complexity of $O(V^3)$ to find all-pairs shortest paths without link/node failures [3].

The Bellman-Ford algorithm has a complexity of $O(VE)$, where $V$ is the set of vertices in the graph and $E$ is the set of all the edges in that graph [3] for single-source shortest path and has $O(V^2E)$ for all-pairs shortest path computation. In general, the distributed Bellman-Ford algorithm's complexity is upper bounded as an exponential function of $V (V^K$, for some constant $K)$, a polynomial function of maximum degree of any node, say $d_{max}(a_{max})$, for $a \geq 1$, and linear function of the number of topological changes [12].

The Floyd-Warshall algorithm has a complexity of $\Theta(V^3)$, which means that it is asymptotically tightbound [3].

The complexity of the Pathfinder algorithm that the Pathfinder-KLIG algorithm uses at each step before moving the message towards the destination is $O(V^4)$ [4]. The complexity of this algorithm in case of topological changes will depend on the value of $K$, which determines the selection of a subgraph of $K$ edges within which the changes have to be propagated to guarantee a near-optimal path.

Storage complexity

In the distributed Dijkstra algorithm [8] each node must maintain complete information of the topology of the network in order to compute its distance table and the route table. This requires that each node not only has the distances from itself to other nodes, but also the information about other nodes distances to every other node. Thus, the storage requirement is of the order of $O(V^2)$ [11].

The distributed Bellman-Ford algorithm requires only partial knowledge of the network in order to compute distance and route tables. It keeps information regarding the paths to the destination node only through the edges incident on it. The storage requirement at each node is of the order of $O(V^e)$, where $e$ is the average degree of the node in a network [11].

The Floyd-Warshall algorithm should have available complete information of the network to compute the all-pair shortest paths and hence has storage requirements of the order of $O(V^2)$ [3].

In the Pathfinder-KLIG algorithm, the storage requirement at each node depends on the value of $K$ and the degree of the node in $KLIG_i$. Each node should have sufficient memory to store all the nodes and the interconnecting edges within $K$ edges of itself [4] and the coordinates of the destination node. The algorithm makes a heuristic estimate of the shortest path outside its local view ($KLIG_i$) [4] by combining the shortest path within its local view and Euclidean distance from the frontier to the destination node. The amount of memory required at each node is of the order of $O(V_i^2)$, where $V_i$ is the number of vertices in the $KLIG_i$ determined by the Node Inclusion Test (NIT).

Multiple paths

The Distributed Dijkstra algorithm stores information about alternate paths to a destination from the same source node if we consider a source/destination pair [5]. Alternate paths to a destination are available in the distributed Bellman-Ford algorithm [5]. In the Floyd-Warshall algorithm, modifications to the original algorithm must be made to extract the alternative path information. The Pathfinder-KLIG algorithm uses the Pathfinder algorithm in its computation for the shortest paths and hence this algorithm explicitly stores the alternative paths information.
To specify date, time, or period.

2005 Date/time/period qualifier, M, an..3

137 Document/message date/time
   Date/time when a document/message is issued.

2380 Date/time/period, C, an..35

2379 Date/.. format qualifier, C, an..3

102 CCYYMMDD
   Calendar date: C = Century;
   Y = Year; M = Month; D = Day.

<DTM DTM2005="137">
<DTM2380>19981218</DTM2380>
<DTM2379>102</DTM2379>
</DTM>

Figure 3: UN/EDIFACT segment names and a corresponding message

With the common term set defined, a special communication service is needed, which is offered by each business application system whose services are part of a certain product. The communication service translates the application systems output to the common term set and receives messages via the tuplespace that preserves messages regarding to the common term set. Thus, the communication service consists of two parts: an input service, and an output service. The output service translates the messages of a certain business application system to XML-based UN/EDIFACT, and has to be implemented for each application system individually. To implement the input service, that translates XML-based UN/EDIFACT messages to a format that can be understood by the respective application system, the XML parsers described above can be reused. Again, input and output service are "accessed" via WCML components that are capturing the code needed for initiating communication (figure 4), and may be implemented as, e.g. Java-based software agents as proposed in [3].

For implementation, we use a tuple space approach [9], which serves as a distributed and shared data space. The data items that the tuplespace is capable to handle are tuples. Marking a tuple semantically can be done by using XML, and thus enhancing the tuplespace to deal with XML tuples, which are simply XML documents, especially XML/EDI data.

The XML tuplespace provides the functionality to store structured data and to enable distributed processes, e.g. the mentioned software agents, to access and modify XML tuples.

Data conform with a DTD (Document Type Definition) is said to be valid XML. In this case, an XML parser could check incoming data against the rules defined in the DTD and check if the data was structured correctly. Data is known as well-formed XML if it is structured as defined by XML and sent without a DTD. The XML tuplespace supports the handling of both, valid and well-formed XML, giving the processes connected to the XML tuplespace the freedom to exchange and transfer any information.
The functions provided by the XML tuplespace are *in*, *out*, and *read*, and are provided by an XML tuplespace that is addressed by the given XML tuplespace URL, as described in the following:

- **out**(XML tuplespace URL, DTD, XML tuple)
  The function *out()* puts an XML document, specifically an XML/EDI document into the XML tuplespace without overwriting existing XML tuples. If the XML tuple is sent along with a DTD, the XML tuple is said to be a valid XML tuple, otherwise a well-formed XML tuple. Valid XML tuples ensure that the data is conform to a given grammar, provided by the DTD.

- **in**(XML tuplespace URL, DTD, XML tuple)
  The function *in()* returns an XML tuple from the XML tuplespace if a XML tuple matching to the DTD and the given XML tuple can be found. The returned XML tuple will still remain in the XML tuplespace.

- **read**(XML tuplespace URL, DTD, XML tuple)
  The function *read()* returns a XML tuple from the XML tuplespace if a XML tuple matching to the DTD and the given XML tuple can be found, and deletes the XML tuple in the XML tuplespace.

To access an existing XML tuple the XML tuplespace provides three kinds of searches:

- **Actual**
  Actual XML tuples are all XML tuples that do not contain a placeholder tag, e.g. `<*>`. An actual XML Tuple is: