



A 3-Way Merging Algorithm and Optimization for Mobile Hosts

¹Mukesh Patel, ²Dr. K P Yadav, ³Dr. Abhijit Kulshreshtha

¹Department of Computer Engineering, Jodhpur National University, Jodhpur, Rajasthan, INDIA.

²Department of Computer Engineering, KCC Institute of Technology and Management (KCCITM),
Greater Noida, Uttar Pradesh, INDIA.

³Department of Applied Science, Sobhasaria Group, Sikar, Rajasthan, INDIA.

Abstract: Mobile Computing represents a new paradigm that aims to provide continuous network connectivity to users regardless of their location. A wide spectrum of portable, personal computing devices has recently been introduced in the market that range from laptop computers to handheld personal digital assistants. Coupled with the advent of wireless networking, this has given rise to a new style of computing wherein the computer can move with the user and yet maintain its network connections. The emergence of computer networking allowed isolated computers to share resources giving rise to distributed computing.

Keywords: Computer networking, Mobile Computing, algorithm, optimization

I. Introduction

Computer networking eliminated the physical isolation between computers, and enabled users connected to the system from one computer to use the resources available at another. This paved the way for distributed computing where a collection of networked computers cooperate to achieve a common goal. However, the access points to such a distributed system were still tethered machines and a user did not have the flexibility to move while accessing such a system. Mobile computing extends distributed computing in a direction where the services of such a system are available to a user regardless of location and more importantly, changes in location. Wireless networking eliminated the need to remain tethered to a wired, static infrastructure and yet avail of the services of a distributed system. A distributed system with mobile hosts thus consists of a wired infrastructure of static hosts (representing a conventional distributed system) that connects areas of wireless coverage ('cells') to mobile computers.

II. Review of literatures

The emergence of computer networking allowed isolated computers to share resources giving rise to distributed computing. Computer networking eliminated the physical isolation between computers, and enabled users connected to the system from one computer to use the resources available at another. This paved the way for distributed computing where a collection of networked computers cooperate to achieve a common goal. However, the access points to such a distributed system were still tethered machines and a user did not have the flexibility to move while accessing such a system. Mobile computing extends distributed computing in a direction where the services of such a system are available to a user regardless of location and more importantly, changes in location. Wireless networking eliminated the need to remain tethered to a wired, static infrastructure and yet avail of the services of a distributed system. A distributed system with mobile hosts thus consists of a wired infrastructure of static hosts (representing a conventional distributed system) that connects areas of wireless coverage ('cells') to mobile computers [5-6].

III. Location management for mobile users

To manage the changing location of mobile users, [10] proposes a hierarchy of regional directories, where each directory is based on a decomposition of the network into regions. A change in location of a mobile user updates "nearby" directories, which point to the user's new location. A forwarding pointer is left at the old location of the user. Searches that originate at "remote" directories are guided by this pointer to the user's current location. [1] examines strategies to reduce search costs and control the volume of location updates by employing user profiles. User profiles are used to define partitions and a location update is generated only when a user crosses partitions. The problem of location management is split into three components in [3]: search for the current location of an user, location updates after a move and location updates after a successful search. Multiple strategies are presented for each sub-problem which is then combined to formulate different location-management strategies. A different twist to the problem is explored in [4], viz. providing location information users may constitute an invasion of privacy of users, and therefore, users should be able to control how precisely their individual locations are revealed to location-based applications.

Fault tolerance: There has been very limited research work in this area [2] briefly discusses the issues involved in recovery from failures in a mobile environment, and suggests that the static infrastructure be used to back up the state of mobile hosts. The reverse problem, viz. failure of a static host that serves as an access point to the rest of the static infrastructure, can result in loss of communication with mobile hosts. As a solution, [4] suggests replicating the services of such a static host amongst several secondary static hosts so that a mobile host can switch its access point to one of the secondary's in case the primary fails.

IV. Algorithms relevant to 3DM

A. Check Pointing Algorithms

The overall check pointing algorithm consists of two parts: one executed by every MH to independently checkpoint its local state based on the two phase rule (with appropriate modifications) and the other initiated from a MSS (possibly, on behalf of a local MH) to collect one local checkpoint per MH at a common location to construct a global checkpoint.

B. Tree Differencing Algorithms

Suppose we have two trees T_1 and T_2 , and that T_2 has been obtained by editing or transforming T_1 in some unknown way. The tree differencing problem is to recover the set of changes between T_1 and T_2 so that this set of changes can be used to obtain T_2 from T_1 . The tree differencing problem is typically approached by defining a set of operations to express the changes (e.g. node insert, delete, and update) and optimality criteria on the set of changes. Intuitively, the optimality criteria allow us to express that we want to have a meaningful set of changes. For instance, assuming that $T_1 = (R; a\ b)$ and $T_2 = (R; a\ (b; c))$ a set of changes deleting all nodes in T_1 and then inserting all nodes in T_2 is not desirable, whereas a set expressing that c is inserted as a child of b is. The tree differencing problem is a generalization of the string (or ordered sequence) comparison problem, in the sense that strings can be thought of as ordered trees of height 1. Although techniques from the domain of string comparison can be applied to the tree differencing problem string comparison algorithms are not directly suitable for tree differencing. The possibility that the tree differencing problem can be reformulated in the string domain is, however, still not formally ruled out. Tree differencing can be divided into two subcategories dealing with ordered or unordered trees. In unordered tree differencing, the order of child nodes is not important. For instance, if the trees $(R; a\ (b; c\ d))$ and $(R; (b; d\ c)\ a)$ are considered unordered, there is no difference between them. Even simple variants of the unordered tree differencing problem have been shown to be NP-hard. In 3DM, we share the assumption of [CDH00, IAW01, and CRGW96] that the trees are ordered.

The assumption is based on computational feasibility, the likely ordered nature of the input (e.g. text documents) and the fact that XML has no inherent support for expressing when element children should be considered ordered or unordered. The first non-exponential algorithm for solving the ordered tree differencing problem is due to, who also introduced the concept of edit distance to measure the difference between two trees. As the majority of subsequent work (e.g. on tree differencing is based on this distance, we shall examine it in some detail. At the same time we will define the related concepts of edit script and matchings' between trees.

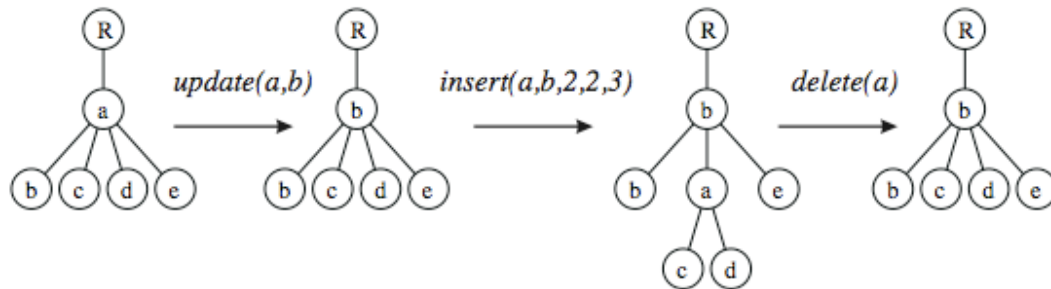
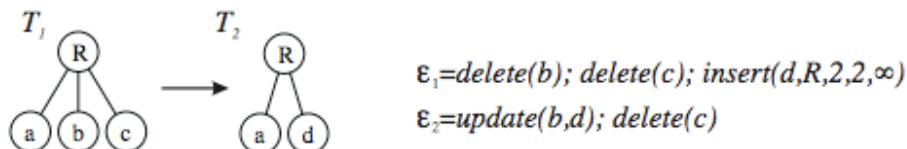


Figure 4.1: Illustration of the tree edit operations *update*, *insert* and *delete*.



C. Tree Merging Algorithms

Suppose we have three trees T_B , T_1 and T_2 . T_1 and T_2 have both been obtained from T_B through editing or some other transformation: T_B is thus the base version of which T_1 and T_2 are branches. No record of the edits or transformations exists, only the final trees, and we want to construct the 3-way merge of T_B , T_1 and T_2 . This is the tree merge problem, as defined in section 3.3. Recall that an edit script transforming a tree T into another, T' , may be obtained by running a structural differencing algorithm (see section 4.4.1) on the trees T and T' . We can

then use the edit script as the edit history between T and T' , and may thus also consider merge algorithms based on edit histories, although they are not explicitly available. Algorithms based on merging T_1 and T_2 without at least implicit knowledge of TB or the edit histories are not considered here, because such algorithms cannot perform automatic merging. In a method for merging unordered trees and identifying conflicts is outlined. In addition to the base and branch trees the method uses the edit scripts that transform the base version into the branches. The edit operations allowed in the script are node insert, delete and update. As the method is one of the few references to a general structured merge algorithm that the author is aware of, we will now examine it in some detail. Let the edit script $TB \ominus T_1$ be EB_1 and the script $TB \ominus T_2$ be EB_2 . In the first step, we add information to each node in T_1 and T_2 , indicating if the node has been inserted, deleted or changed compared to the base version. The paper does not give any details on how this is implemented, but it could be accomplished by for instance running the edit scripts EB_1 and EB_2 on TB , and keeping track of the modified nodes. If a node has modified children, it is itself also considered to be modified. These way modifications are reflected all the way to the root of T_1 and T_2 . We then continue by comparing the roots of T_1 and T_2 . If the content of either root has changed, the changed content is used. If both roots have changed, a conflict has occurred. Next, we compare the children of the roots. Any new (i.e. inserted) children of the root in either tree are also inserted in the merged tree, and conversely any child of either root that is deleted is removed from the merged tree. For instance, if the root of TB has the children a b c and the roots of T_1 and T_2 have the children a b c d and b c respectively, the merged child list is b c d . The method can now be applied recursively to the children of the root in the merged tree. The paper does not state specific details of the recursive step, but the general idea should be clear. The method described in the paper handles moves and copying of nodes as node insert and delete operations, and thus lacks the ability to successfully handle the move/update case. The paper furthermore presents a method for detecting conflicts in the context of the merge method described above, and also proposes how merge conflicts should be handled. For instance, if a node is deleted in one tree, and changed in the other, how should this be handled? The author suggests that one of the branches could be used as a main alternative, whose edits are always carried out in the case of a conflict.

A similar method of merging structural data is sketched in, having the same shortcomings regarding node move and copy operations. In a system for merging and conflict resolution, called MARC, is presented. MARC operates in the context of a CSCW system named CoNus, whose data model consists of objects with attributes and links between these objects. The links can be of two types: hierarchical or semantic. The hierarchy links connect the objects into a tree structure. MARC is based on edit histories.

When two object trees need to be synchronized, the edit histories gathered since the branch point is used. The edit histories are then analyzed in order to detect and resolve conflicts. Based on the analysis and conflict resolution, two new edit histories are subsequently produced, which when applied to the trees bring them into sync (i.e. in effect a merge). Compared to the merge methods presented in and, the one used in MARC treats the tree vertices (i.e. the hierarchical links between the objects) as entities that may be created and deleted. This allows for a more powerful merge that can handle the move/update case. On the downside, copies are not handled and the merged structures may be forests rather than trees. A system for comparing and querying structural documents is briefly presented in. The system is based on insert-delete-update tree differencing algorithm, but also supports moves through post processing of the algorithm output. The paper mentions that the system has two merge-related operations: mergeable (TB , T_1 , T_2) for determining if a 3-way merge is possible, and merge(TB , T_1 , T_2) to perform a 3-way merge. Unfortunately no further details on this system seem to be available [7-10].

V. Conclusion

Mobile Computing represents a new paradigm that aims to provide continuous network connectivity to users regardless of their location. A wide spectrum of portable, personal computing devices has recently been introduced in the market that range from laptop computers to handheld personal digital assistants. Coupled with the advent of wireless networking, this has given rise to a new style of computing wherein the computer can move with the user and yet maintain its network connections.

References

1. Badrinath, B. R., Imielinski, T., and Virmani, A. Locating strategies for personal communication networks. *IEEE Globecom 92 Workshop on networking of personal communications applications* (1992).
2. Krishna, P., Vaidya, N. H., and Pradhan, D. K. Recovery in distributed mobile environments. In *IEEE Workshop on Advances in Parallel and Distributed Systems* (October 1993).
3. Krishna, P., Vaidya, N. H., and Pradhan, D. K. Location management in distributed mobile environments. In *Proc. of the 3rd Intl. Conf. on Parallel and Distributed Information Systems* (October 1994).
4. Spreitzer, M., and Theimer, M. Architectural considerations for scalable, secure, mobile computing with location information. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems* (June 1994).
5. Acharya, A., and Badrinath, B. R. An efficient protocol for ordering broadcast messages in distributed systems. In *The 3rd IEEE Symposium on Parallel and Distributed Processing* (1991).
6. Acharya, A., and Badrinath, B. R. Delivering multicast messages in networks with mobile hosts. In *Proc. of the 13th Intl. Conf. on Distributed Computing Systems* (May 1993).
7. Agrawal, D., and Abbadi, A. E. An efficient and fault tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems* 9, 1 (Feb 1992).

8. Alagar, S., Rajagoplan, R., and Venkatesan, S. Tolerating mobile support station failures. Tech. rep., University of Texas at Dallas, November 1993.
9. Alonso, R., and Ganguly, S. Query optimization for energy efficiency in mobile environments. In Proc. of the 1993 Intl. Workshop on Foundations of Models and Languages for Data and Objects.
10. Alonso, R., and Korth, H. Database system issues in nomadic computing. Tech. Rep. TR-36-92, MITL, December 1992.