行政院國家科學委員會專題研究計畫　成果報告

應用於嵌入式系統中以區塊為基礎之原位檔案重建模式

計畫類別：個別型計畫
計畫編號：NSC91-2213-E-002-062-
執行期間：91 年 08 月 01 日至 92 年 07 月 31 日
執行單位：國立臺灣大學資訊工程學系暨研究所

計畫主持人：劉邦鋒

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 92 年 10 月 31 日

行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　　※
※應用於嵌入式系統中以區塊為基礎之原位檔案重建模式
※　　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：■個別型計畫　　□整合型計畫

計畫編號：NSC 91-2213-E-002-062-

執行期間：2002 年 8 月 1 日至 2003 年 7 月 31 日

計畫主持人：劉邦鋒

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□赴大陸地區出差或研習心得報告一份

□出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

執行單位：國立台灣大學資訊工程學系

中　華　民　國　　九十二 年 十 月 三十一 日

行政院國家科學委員會專題研究計畫成果報告
應用於嵌入式系統中以區塊為基礎之原位檔案重建模式
In-place Block-based File Reconstruction for Embedded Systems

計畫編號：NSC 91-2213-E-002-062-
執行期限：91 年 8 月 1 日至 92 年 7 月 31 日
主持人：劉邦鋒　國立台灣大學資訊工程學系

一、中文摘要

隨著微處理器科技的蓬勃發展，嵌入式系統已經成為我們日常生活中不可或缺的一部份。嵌入式系統提供智慧型的控制機制，提供對各種工業及家用設施的掌控與監督。一個嵌入式系統的最核心部分即為其控制程式。大多數的嵌入式系統受限於成本並不具備大量的記憶體資源，所以如何在有限的記憶體中達成更新一嵌入式系統核心程式的目的是一個非常重要的研究課題。

本計畫的重點在於發展一以區塊為單元（block-based）且不允許重複複製（no multiple copy）之原位（in place）檔案更新架構。這個架構的著眼點在於在實際的檔案系統中，檔案皆以固定區塊為處理單元，同時業界通用的檔案規格大部分都有考慮到檔案的緊密性（compactness）。如此一來在傳統檔案更新問題中會出現的問題都可藉由我們所提出的較實際的架構得到解決。

針對此一較實際的架構，我們設計新的檔案更新演算法。初步規劃出的演算法可快速偵測檔案更新會出現之資料區塊複製依賴迴圈（dependency among copying of data blocks）。此種依賴迴圈是決定一個檔案能否在原位被更新的重要因素。一個檔案可被原位更新亦即系統不需額外記憶體資源即可在檔案原先的位置做更新。我們設計的演算法可偵測出假性依賴迴圈（false dependency cycle），並藉由重新安排資料複製的順序來打破依賴迴圈，以達成檔案原位更新的目的。即使此依賴迴圈為真（true dependency cycle），我們設計的演算法也可以偵測出此依賴迴圈的最小依賴量，使其打破依賴迴圈的成本降至最低。初步實驗成果證實此一新架構及對應演算法不但可偵測出傳統演算法無法解決的依賴迴圈，並可有效降低檔案原位更新所增加的資料複製成本。
關鍵詞：集檔案比對,原位檔案重建,嵌入式系統.

Abstract

The recent rapid developments of embedded systems have changed many aspects of our daily life. With this advancing embedded system technology more and more “smart” devices are able to provide inexpensive and reliable controlling capability. The central part of an embedded system is its control program, which may be subject to constant update. Therefore, an important issue in embedded system deployment is how easy it is to replace its controlling program, and whether this can be done without extra storage requirement.

This proposal addresses the in-place file reconstruction problem by providing a block-based, no-multiple-copy model. We suggest this practical model from the observation that file systems are organized in blocks and compact encoding of files illuminates the necessity of multiply copy of the same block. This model can avoid the theoretical problems that will not occur in practice but could restrict our algorithm designs.

We illustrate the advantages of our model by providing a much simplified, but yet practical mechanism to describe the dependency among copying of data blocks, and give bounds on the number of possible dependency edges in the model. We further provide evidences that some dependency graph may have “false” cycle, i.e., they can be patched in place in the new model by dividing the data into more manageable data blocks. Even if our new model cannot break up the cycle, we provide an efficient algorithm that identifies the blocks that actually causes the cycle in the new model, and add only the minimum amount of data to the patch file. Experiments are conducted to check for reduced patch file size and preliminary results indicate that our algorithm significantly decreases the patch file size while maintaining the in-place file reconstruction property.

Keywords: File comparison, in-place file reconstruction, and embedded systems.

二、具體成果

We propose a new model for the in-place reconstruction problem to more closely capture the feasibility of in-place reconstruction. First our model is block-based, i.e., all the segment starting offset and length are the multiple of a predefined block size. Second in our model the copying from a single block to multiple destinations is not allowed.

In contrast to our block-based model, the traditional model described earlier is command-based. In the dependency graph each node represents a copy command. If there is a cycle in the dependency graph and we would like to break it, we must delete a node, i.e., an entire copy command, from the cycle [7]. However, this may incur unnecessary overhead since it may not be necessary to transform the entire copy command into add since only part of that copy is in the dependency cycle.

Another advantage of block-based dependency is in the treatment of self-loop. It is easy to see that for a copy command x even when R(x) intersects W(x), it is still possible to complete x in-place since we can "shift" the data block within the reference file, so self-loop is not an issue.

It has been established in the literature that in-place file reconstruction is difficult [12]. One of the major difficulties lies in the assumption that a single data may be copied to different locations within the reference file. Should this be the case, and then the file structure of the version file is redundant since this duplication of data not only wastes storage, but also the processing time. Therefore we assume that the copy commands in a patch file copy *different* segments of data. In other words, the data being copied can be viewed as a permutation from the reference file to the version file. This makes practical sense since only the “common” part of file should be copied from the reference to the version file, and all the others should be obtained from add commands.

After establishing our file reconstruction model, we show that a simple property ensures that the resulting dependency graph is acyclic. First we define terminology that will be used in the description of the results. Due to the ``no multiple-copy'' requirement in our model, the data segments and copy commands have a one-to-one mapping. For two non-overlapping intervals x and y in the reference file, we define x < y if x has a smaller starting offset. We then define that two copy commands a and b are in order if and only if R(a) < R(b) if and only if W(a) < R(b). A patch file is in order if and only if any two of its copy commands are in order.

Theorem 1 If a patch file is in order, then its dependency graph is acyclic.

Next we discuss the impacts of our new model on the complexity of the dependency graphs. It is shown in [7] that due to multiple copying a dependency graph of n nodes can have $\Omega(n^2)$ dependency edges, the highest possible. We show that in our model the number of edges can be bounded by $O(n)$. This not only ensures the processing time of any operations on the graph but also indicates a more realistic estimation in practice.

Theorem 2 Let $G' = (V', E')$ be the dependency graph from $G = (V, E)$ after self loops being removed from G. Then the number of edges in G' is bounded by $O(|V'|)$.

Lemma 3 Let G = (V, E) be a dependency graph, G' = (V', E') be G after removing self-loops, and G* = (V*, E*) be the dependency graph after every copy command is partitioned into block size, then G' is cyclic if G* is.

With Lemma 3 we conclude that if there is no cycle in G' then we have eliminated all cycles in G*. It is easy to see that the minimum number of blocks that have to be removed is the number of cycles in G* (denoted by C*), therefore if we guarantee that every block we will convert into an add command breaks a cycle in G*, and we will convert C* blocks, then we can guarantee that the resulting dependency graph can be patched in-place with a minimum amount of data increase in the patch file. The algorithm can be described as follows.

Repeat until there is no cycle in the dependency graph.

1. For a newly found cycle, determine if it is true.
2. If the cycle is true, then remove the blocks that form the true cycle and convert them into an add command.
3. Divide the copy commands.

五、參考文獻

[1] M. Ajtai, R. Burns, R. Fagin, D. Long, L. Stockmeyer. Compactly Encoding Unstructured Inputs with Differential Compression.

[2] A. Alderson. A space-efficient technique for recording versions of data. Software Engineering Journal, 3(6):240-246, 1988.

[3] P. Bernstein, V. Hadzilacos, N. Goodman. Concurrency Control and Recovery in Database. Addison-Welsley Publishing Co., 1987.

[4] R. Burns. Differential compression: A generalized solution for binary files. Technical Report, Master's thesis, University of California at Santa Cruz, Department of Computer Science, 1997.

[5] R. Burns, Darrell D. Long. Efficient Distributed Backup with Delta Compression. In I/O in Parallel and Distributed Systems , pages 27-36, 1997.

[6] R. Burns, Darrell D. Long. A linear time, constant space differencing algorithm. In Proceedings of the 1997 International Performance,Computing and Communications Conference, February, 1997.

[7] R. Burns, Darrell D. Long. In-Place Reconstruction of Delta Compressed Files. In Symposium on Principles of Distributed Computing, pages 267-275, 1987.

[8] L. Clausen, U. Schultz, C. Consel, G. Muller. Java Bytecode Compression for Embedded Systems. Technical Repor RR-3578, Institut de Recherche en Informatique et Systemes Aleatoires, 1988.

[9] T. Corman, C. Leiserson, R. Rivest. Introduction to Algorithms. The MIT Press, 1989.

[10] M. Delco and M. Ionescu, xProxy: A transparent caching and delta transfer system for web objects . Technical Report, UC Berkeley class project: CS262B/CS268., 2000.
[11] J. Hunt, K. Vo, W. Tichy", An empirical study of delta algorithms. In IEEE Software Configuration and Maintenance Wks., 1996.
[12] R. Karp, Reducibility among combinatorial problems. R. Miller and J. Thatcher, editors, Plenum Press, 1972.
[13] D. Knuth, Fundamental Algorithms, The Art of Computer Programming, Addison Wesley, 1968.
[14] D. Korn, K. Vo. The VCDIFF Generic Differencing and Compression Data Format. Internet Draft draft-korn-vcdiff02. txt, Nov. 2000. 2000.
[15] P. Liu, W. Hu. A randomizaed block-based file comparison algorithm for in-place file reconstruction. Manuscript.
[16] J. MacDonald , Versioned File Archiving, Compression, and Distribution. UC Berkeley. Available via http://www.cs.berkeley.edu/ jmacd, 1999.
[17] J. MacDonald, File system support for delta compression. UC Berkeley. Masters thesis. 2000.
[18] U. Manber. Finding Similar Files in a Large File System. Proceedings of the USENIX Winter 1994 Technical Conference, pages 1-10, San Fransisco, CA, USA, 17-21 1994.
[19] B. Marsh, F. Douglis, P. Krishnan. Flash Memory File Caching for Mobile Computers. In Proceedings of the 27th Hawaii Conference on Systems Science, 1994.
[20] W. Masek, M. Paterson. A faster algorithm computing string edit distance. Journal of Computer and System Science. 20(1), 1980.
[21] W. Miller, E. Myers. A file comparison program. Software--Practice and Experience, 15(11):1025-1040, 1985.
[22] J. Mogul, F. Douglis, A. Feldmann, B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In Proc. of SIGCOMM, pages181-194, 1997.
[23] C. Reichenberger. Delta Storage for Arbitrary Non-Text Files, In Peter H. Feiler, editor, Proceedings of the 3rd International Workshop on Software Configuration Management. , pages 144-152. IEEE Computer Society Press, 1991.
[24] M. Rochkind. The Source Code Control System. IEEE Transaction on Software Engineering.SE-1(4):364-370, December, 1975.
[25] P. Rodriguez, K. Ross, E. Biersack. Distributing Frequently-Changing Documents in the Web: Multicasting or Hierarchical Caching. In Computer Networks and ISDN Systems. Selected Papers of the 3rd International Caching Workshop, pages 2223-2245, 1998.
[26] D.Severance and G. Lohman. Differential files: Their application to the maintenance of
large databases. ACM Transaction on Database Systems,1(2), September 1976.
[27] W. Tichy. The string-to-string correction problem with block moves. ACM Transactions on Programming Languages and Systems,6(4):309-321, 1984.
[28] W. Tichy. RCS -- A system for version control. Software Practice and Experience, 15(7):637-- 654, July 1985.
[29] A. Tridgell, P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Canberra 0200 ACT, Australia, 1996.
[30] T. Yu. Data Compression for {PC} Software Distribution. Software Practice and Experience, 26(11):1181-11**95, 1996.**