組み込み Linux 上のソフトウェア更新方式(2)
－圧縮データへの差分適用－

河相英典† 深澤司†† 小谷亮†† 清原良三††

三菱電機株式会社 †モバイルターミナル製作所 ††情報技術総合研究所

1.はじめに

近年、組み込み Linux が様々な機器で用いられており、それらの機器に対してソフトウェアの不具合を修正する手段が必要とされている。また、不具合の修正に必要なデータは、通信回線を用いて送信される場合もあるため、小さい方が望ましい。従来のソフトウェアの更新方式として、ソフトウェアの修正点を示す差分データを送信し、端末側で差分適用する方式が提案されていた[1][2]。しかし、Linux とそのファイルシステムの導入により、従来手法では問題が生じる場合がある。

組み込み Linux で用いられるファイルシステムのひとつに、今回対象とする Cramfs[3]がある。Cramfs は、Linux の一般的な圧縮アルゴリズムである zlib[4]で、データを圧縮し ROM 上に保管する。従来方式で圧縮データ部分の差分を抽出すると差分データが増加する。そこで、本論文では、圧縮データに対する差分データの増加を抑制できる差分適用法を検討する。

2.圧縮データへの差分適用の問題点

圧縮後のデータイメージは、圧縮前のデータに含まれる文字の出現数や順序に大きく影響されるため、少しでも変更されると圧縮後のデータイメージは大きく異なる。

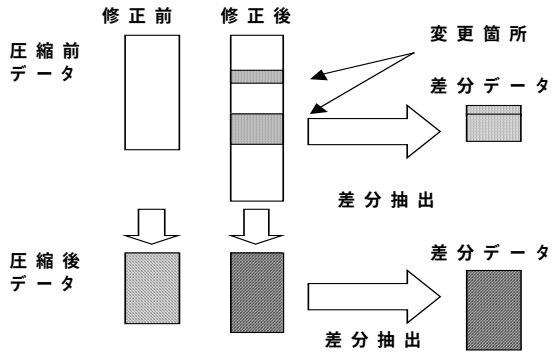

図 1 圧縮データと非圧縮データでの差分抽出

実際に組み込み Linux 上のバイナリファイルに対して、zlib で圧縮した状態と、非圧縮状態のそれぞれの差分データを求めた。例として、バイナリ形式 14KB、変更としてデータサイズの 10%にあたる 1.4KB のバイナリデータを中央に挿入した結果を示す。実験の結果、非圧縮状態では、挿入されたデータサイズに対して抽出される差分データが 1.2KB とほぼ等しいのに比べ、このデータを圧縮し 7.7KB になったデータに対して差分を抽出すると、8.0KB の差分データが発生した。このように、圧縮データが存在する場合、差分データが大きくなる場合があることを確認した。

A Software Update Method for Embedded Linux Systems.

Hidenori Kawai† Tsukasa Fukasawa†† Akira Kotani†† Ryozo Kiyohara††

Mitsubishi Electric Corp. †Mobile Terminal Center ††Information Technology R&D Center

3.圧縮データへの差分適用法

圧縮データに対する差分サイズの問題を解決する方法として以下の 4 方式を検討した。

(1)解凍適用方式

組み込み端末側の圧縮データを RAM 上に解凍し、非圧縮状態での差分を適用後、再圧縮して ROM に書き込む方式。圧縮データに対応するために、書換えデータに解凍状態と圧縮状態を示す情報を追加する必要がある。

(2)ビット比較方式

抽出ツールをビット単位での比較を可能にする方式。圧縮データでは、データに変更があった場合、ビットずれが起きただけで、元のビット列と変化していない部分が存在する場合があるため、この方式によりビットずれ部分を抽出可能にする。

(3)同一辞書ハフマン符号化方式

ハフマン符号化のみで圧縮する方式。ハフマン符号化では、符号辞書が作成され、各文字はそれに基づき対応した符号に置換される。符号辞書は、データごとに作成されるため、修正後に再圧縮を行うと修正前と異なる辞書が構成される場合がある。辞書が異なると、今まで置換されていた符号が変わるため、データ全体に影響が及ぶ。そこで影響を局所化するため、符号化に用いる辞書を以前と同一のものにする。この方式では、修正後のデータに新しい文字が現れた場合符号化できないため、あらかじめ全ての文字が含まれる符号辞書を作成しておく必要がある。

(4)ブロック圧縮方式

圧縮対象のデータをあるサイズごとのブロックに分割し、ブロック単位で圧縮する方式。この方式により図 2 のように、変更が起こる前のブロックについては、差分発生を防ぐことができる。

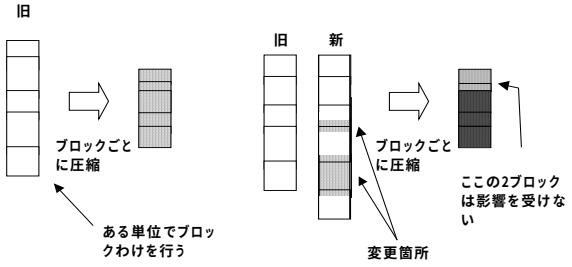

図 2 ブロック圧縮

4.差分適用法の検討

表 1 に各方式における、あるバイナリファイルの差分

データの見積もり及び実験結果を示す。

表 1　各方式での差分データ

データサイズ	通常差分	圧縮差分	(1)	(2)	(3)	(2)+(3)	(4)
196608	13429	70042	13800	68500	93391	23000	73769

4.1 解凍適用方式

(1)の方式の問題点は、解凍-圧縮を組み込み端末側で行う必要がある点が挙げられる。組み込み端末側は 元々メモリの量や CPU の性能に限界があるため、処理に時間がかかるためである。

zlib では、参考文献[4]より圧縮の処理が重いことがわかっている。そこで、PC 上で圧縮にかかる時間を求めた。Pentium3、1.26GHz メモリ 1GB のマシンで行った結果では動作周波数 1MHz 当り、毎秒 4.7KB の処理が行える結果となった。組み込み端末上では上記の結果よりもさらに遅いものになると予想される。しかし、ベンチマーク結果などから性能の劣化は最悪でも 2 倍程度と予想され、50MHz 程度の周波数であっても、通信速度と比較した場合高速であり、圧縮の処理のために一時的に処理を止めても問題の無い組み込み機器では有効である。

また、メモリも組み込み端末では制限が大きいが、zlib で圧縮時に使用するメモリサイズは、パラメータで設定できるため問題にはならない。

4.2 ビット比較方式

ビットの 0、1 を 1 バイトの 0、1 に置換した仮想データで差分を抽出し、得られた差分データに含まれているコマンドの分布から(2)の方式での差分データの解析を行った （図 3）。

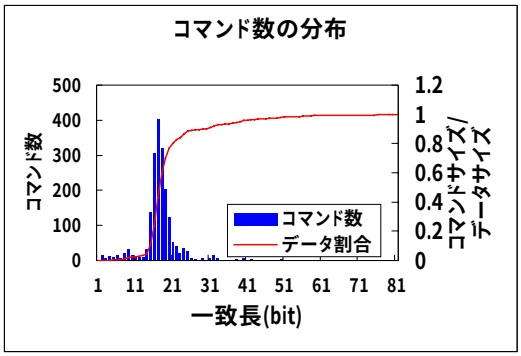

図 3 マンド分布例

修正の前後で変化していない部分を、組み込み端末側にあるデータで表現する copy コマンドを用いることで、差分データ量の削減が可能である。このコマンドは、アドレス情報や、データのサイズなどを含むため 40 ビット程度必要となる。従って、40 ビット以上の一致長があれば、このコマンドによる置換効果がある。

一致部分が、差分データの中にどれだけ含まれているか解析した結果、zlib に対してビット比較を行うと、40 ビット以下の一致長がほとんどを占めていることが明らかとなった。この結果から、zlib で圧縮したものに対して(2)の方式を行っても、差分データ削減の効果が無いことを確認した(表 1)。

また、この方式の問題点として、適用側、抽出側双方でビットに対する処理を行う必要があるため、計算量が増加する点がある。

4.3 同一辞書ハフマン符号化方式

(3)の方式での差分データを調べた結果（表 1）、この方法で圧縮したもので差分抽出しても差分データを削減できなかった。これは、圧縮の性質上ビットずれが発生しているためだと考えられる。そこで、(2)の方式と組み合わせて差分を抽出した。

(2)と(3)の方式を併用して抽出された差分データを解析した結果、長い一致が多く検出されたため、copy コマンドで差分データを削減できることがわかった（表 1)。

この方法では、4.2 で挙げた(2)の問題点に加えて、圧縮率が zlib に対して劣るため、メモリを(1)の時ほど効率的に利用できないことが問題である。

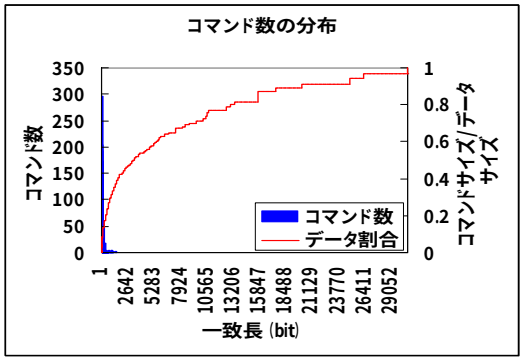

図 4 コマンド分布例

4.4 ブロック圧縮方式

表 1 によれば、方式(4)では差分データは増加している。この方法は、単純に前からある一定バイト毎に区切り、圧縮を行うため、データ挿入削除があるとブロックの単位がずれてしまうことが原因だと考えられる。

この方法で有効に差分を削減するためには各ブロックに余裕領域を持たせ、挿入削除の影響を吸収できるようにする必要がある。また、余裕領域をもつブロックでも動作するファイルシステムにしなければならない。

5.まとめ

組み込み端末側に負担はかかるものの、差分データに関しては(1)の方法が最も効果があることを示した。(2)、(3)は、単独では効果が無く、(2)と(3)を組み合わせる事で差分データを削減できるが、実現に必要なステップ、計算量などの点で問題がある。(4)はそのままでは差分を小さくできず、改良した場合でも全体に少しずつ変更があった場合に問題がある。

差分データから考えると、(1)の方式と、(2)と(3)を組み合わせた方式の二つが候補になるが、メモリ効率や、抽出した差分データの再圧縮の可能性などを考慮すると(1)の方法が最も有効であると考える。

今後、(1)について実装し効果を確認する予定である。

参考文献

[1]清原他、「携帯電話のＳＷ更新に関する検討」、情報処理学会MBL22-13,PP,93-100,2002

[2]栗原他、「携帯電話ＳＷのバージョン間差分データに関する検討」情報処理学会DICOMO2003, 4A-074

[3] http://sourceforge.net/projects/cramfs/

[4] http://www.gzip.org/zlib/