計算機アーキテクチャ 141－13
（2001．1．26）

階層的キーワードベースの名前管理における キーワード管理手法

福井 一史 † 轟木 伸俊 † 多田 知正 †
樋口 昌宏 ‡ 谷口 健一 †

† 大阪大学
大学院基礎工学研究科 情報数理系専攻

‡ 近畿大学
理工学部 電気工学科

従来のファイルシステムにおいては，階層的名前管理が一般に用いられている．これに対し，著者らは、階層的キーワードベースの名前管理を提案している。これは，階層的に管理されたキーワードの集合をファイルに付与することによりファイルの識別を行うというものである．キーワードベースの名前管理では，階層的名前管理と比べて，名前解決の際の探索空間が非常に大きくなるため，名前解決を効率良く行うためには，キーワードの管理手法を工夫する必要がある．本稿では、この階層的キーワードベースの名前管理システムを実装するにあたり，名前解決を効率的に行うためのデータ構造および検索アルゴリズムについて検討する。

A Keyword Management Method on Hierarchical-Keyword-based Naming Scheme

Kazufumi Fukui † Nobutoshi Todoroki † Harumasa Tada †
Masahiro Higuchi ‡ Kenichi Taniguchi †

†Graduate School of Engineering Science
Osaka University

‡School of Science and Engineering
Kinki University

In file systems, hierarchical naming is generally used. We are studying another naming scheme, called hierarchical-keyword-based naming. In hierarchical-keyword-based naming, a file is named by a set of keywords and keywords are organized hierarchically. In keyword-based naming, the size of the search space will become much larger than in hierarchical naming. Therefore, it is an important issue how to manage hierarchical keywords. In this paper, we discuss data structures and algorithms to perform name resolution efficiently in hierarchical-keyword-based naming.

1. まえがき

近年では，二次記憶装置の容量の増大に伴い，計算機の保持するファイルの数も膨大なものとなっており，目的のファイルに到達することがより難しくなりつつある．また現在，テキストファイル以外の画像ファイル，音声ファイルを扱うことが一般的になっているが，これらのファイルには全文検索が使用できず，実際にオープンしなければファイルの内容はわからないため検索がより困難である．したがって，適切なファイルの名前によって，ファイルの内容を表し，ファイルの管理を行うことがより重要になってきている．

従来のファイルシステムでは，ファイルは木構造のディレクトリによって管理されており，ユーザはパス名によってファイルを指定する．このような名前管理を階層的名前管理という．

階層的名前管理は，実装が容易であるなどの利点がある．しかしファイル数の増加に伴って，ディレクトリ構造が深く複雑になると，目的のファイルを探し出すこと

が困難になる．このために，大量のデータを扱うことの多いデータベースシステムでは一般に階層的名前管理は用いられない．多くのデータベースシステムでは，属性を付与することでデータを管理する．ユーザはクエリを用いてデータを取得する．このようなタイプの名前管理は，属性ベースの名前管理と呼ばれる．

より適切なファイル管理のために，属性ベースの名前管理を階層的の名前管理に導入することが試みられている [1, 2, 3].

筆者らはこのような名前管理の 1 つとして階層的キーワードベースのファイル名管理 (以下，階層的キーワード名前管理) を提案している．これは階層的名前管理と属性ベースの名前管理の利点を合わせ持つ名前管理である．階層的キーワード名前管理では，ファイルにキーワードを付与することで分類を行う．ファイル指定の際には，キーワードを任意の順序で並べることができ，一部のキーワードによる指定も可能である．キーワードは階層的に管理されているため，キーワード集合はディレクトリ木のような階層構造をとっている．ユーザは目的のファイルに付与されたキーワードがわからないとき，キーワード階層を探索することでキーワードを見つけ出すことができる．

従来の階層的名前管理では，ユーザが目的のファイルに到達するためには，ユーザが適切だと考えるディレクトリを選択し，辿ることによってファイル検索を行う．このとき，サブディレクトリはディレクトリにより隠蔽されているため，ファイル探索空間はそれほど大きくなることはない．一方，階層的キーワードベースの名前管理では，このような隠蔽がないため，名前解決を行う際の探索空間が非常に大きくなり，名前解決を実用的時間で行うためには，キーワードの管理手法を工夫する必要がある．

本研究では，この階層的キーワード名前管理システムを実装するにあたり，名前解決を効率的に行うためのデータ構造および検索アルゴリズムについて検討を行った。

以下 2節では，従来の名前管理と提案する階層的キーワード名前管理の概要について述べる．3節では，従来の名前管理と階層的キーワード名前管理の相違点を述べ，階層的キーワード名前管理を行う際の留意点を提示する．4節では，階層的キーワードベースの名前管理における名前解決を行うための，いくつかのキーワード管理手法を提示し，比較を行う．最後に 5節で結論を述べる．

2. 従来の名前管理と階層的キーワード名前管理

2.1 名前管理手法と名前空間

ファイルの名前とは，ファイルを指定するために用いられる文字列のことをいう．従来の UNIX ファイルシステムにおける，絶対パス名，相対パス名は共にファイルの名前である．ファイル名とは，ファイルを識別するためにファイルに付与される文字列である．UNIX では，それを格納するディレクトリ内で一意の文字列 (*main.tex* や *fig.jpg* など) をいう．

名前管理手法はファイルの名前の表記法とその解釈のしかたを規定する．ファイルの名前をファイル名と区別するために，以降ではファイルの名前をファイル指定と呼ぶ．

2.2 従来の名前管理

従来用いられている名前管理には，階層的名前管理と属性ベースの名前管理がある．

階層的名前管理とは，UNIX に代表されるファイルシステムで用いられており，木構造をなすディレクトリに登録することでファイルを管理する名前管理である．それぞれのファイル (ディレクトリを含む) は，それを格納しているディレクトリ内で一意のファイル名をもつ．ファイルは一連のディレクトリ名とファイル名の並びであるパス名によって指定される．パス名はそのファイルが格納されている位置を示している．

階層的名前管理の利点は，以下のようなものである．

- ファイル探索の際に構造を把握しやすい
- システムが名前からファイルを得る際に探索範囲が小さくてすむ
- カレントディレクトリの概念により相対的なファイル指定が可能
- ファイル探索の際ディレクトリ構造が一種の道案内の役割を果たす

一方，属性ベースの名前管理とは，ファイルに 1 つ以上の属性を付与し，ユーザはクエリを用いて目的のファイルを取得するような名前管理のことをいう．

属性ベースの名前管理の利点は以下のようなものである．

- ファイル指定の際に属性の順序を気にしなくてよい
- 一部の属性でファイル指定ができる
  この特徴は次のような利点を与える．
  - 多くの属性を付与することで詳細な分類ができる
  - 付与された属性の一部でも記憶していれば素早いファイル検索ができる
  - 属性を追加しても以前と同じファイル指定ができる

2.3 階層的キーワード名前管理

階層的キーワード名前管理は，階層的名前管理と属性ベースの名前管理の双方の特徴を持つ名前管理である．階層的キーワード名前管理では，それぞれのファイルにはファイル名の他に，1つ以上のキーワードが付与される．それぞれのキーワードは“/”で始まり，“/”によって区切られた任意の文字列である．キーワードには親子関係が存在する．例えば，*/animal/dog* は */animal* の子である．これによりキーワード集合は階層構造を構成する．この階層構造をキーワード階層と呼び，それぞれのキーワードを階層的キーワードと呼ぶ．階層的キーワードに含まれる“/”の数をそのキーワードの深さという．例えば */animal* の深さは1，*/animal/dog/pug* の深さは3である．ファイル指定は，*/photo*, */animal/dog*, */people/child//fig1.jpg* のように，コンマで区切られた階層的キーワードの非順序リストとファイル名からなる．ファイル名はキーワードの非順序リストの最後に“//”で区切って書かれる．ファイル名を含まないキーワードのリストはファイルの集合を指定する．

ファイル指定により指定されたファイル集合は，キーワードリストがマッチするファイルを全て含む．ファイル指定は複数のファイルを指定することがある．「キーワードリストがファイルにマッチする」とは，キーワードリスト内のそれぞれのキーワードが，ファイルに付与された階層的キーワードの少なくとも1つにマッチすることをいう．キーワードは，そのキーワードおよびその子孫となる階層的キーワードにマッチする．例えば，キーワード */sports* は */sports/baseball* や */sports/swimming/butterfly* 等にマッチする．

3. 名前管理手法

本研究では，UNIX 上に階層的キーワード名前管理システムを実装することを考える．

3.1 従来の UNIX における名前管理手法

UNIX を例に取って，従来の階層的名前管理がどのように行われているかを示す．

UNIX ではファイルに関する情報は，i ノードが持っている．i ノードは，ファイルの内容のディスク上の位置，ファイルの所有者，アクセス許可，アクセス時刻などの情報を持つ．すべてのファイルにそれぞれ1つの i ノードが対応している．

ディレクトリはファイルシステムに階層構造を与えるファイルである．ディレクトリファイルの各項目は i ノード番号とそのディレクトリに含まれるファイルの名前から構成される．

open などのシステムコールでは，パス名を引数としてファイルにアクセスを行う．カーネル内部ではパス名でなく，i ノードを使用するので，ファイルにアクセスするために，システムコールの引数として与えられたパス名を i ノードに変換する．パス名の検索は，“//”でパス名が始まっている場合はルートディレクトリから，そうでない場合はプロセスの現ディレクトリから始まる．

階層的キーワード名前管理は，階層的名前管理と比べて，次のような違いがある．

- 探索空間が大きい
  階層的名前管理では，ファイル探索空間は1つのサブディレクトリ中だけであるため，高速に探索が可能であるが，階層的キーワード名前管理では，ファイル探索空間は制限されないため，単純な探索手法では非常に多くの時間を要し，この名前解決の部分がシステムのボトルネックになるおそれがある．

- キーワード間の関係の管理とキーワードとファイルの関係の管理に区別される
  通常の階層的名前管理では，ディレクトリはそれ自身がファイルであり，ディレクトリの親子関係と，ディレクトリに含まれるファイルの情報は統一的に扱われる．階層的キーワード名前管理においては，キーワードとファイルは異なるものとして扱われるために，キーワードの親子関係と，ファイルに付与されたキーワードの情報は異なる方法で扱うことができる．このため，それぞれに適した管理方法をとることが出来る．

- キーワードに関するより柔軟な操作を提供する
  階層的名前管理ではファイルを検索する際に行うことの出来る操作はディレクトリ中のファイルを一覧することのみである．しかし，階層的キーワード名前管理では，複数のキーワード全てを含むファイルを求めたり，ファイルに付与されている他のキーワードを一覧したりといったより柔軟な操作を提供するため，より複雑となる．

3.2 従来のキーワード管理手法

階層的キーワード名前管理はファイルの名前をキーワードの集合として扱っていることから，テキストデータベースや WWW 検索エンジン等で用いられるキーワード管理手法を応用するということが考えられる．大規模テキスト中からあらかじめキーワードとして切り出された単語の検索を高速に行う方法として，転置ファイルやシグネチャファイル等のインデックス構造を用いた方法がよく知られている [4, 5]．この2つの中でも，高速性を重視する場合は転置ファイルを用いることが一般的である [6]．

転置ファイルとは，文書中に現れる各単語に対してその出現位置のリストを格納したものである．出現位置は単語ごとに連続した領域に置かれ，その領域の先頭への

ポインタを単語ごとに保存しておく．検索の際には，各単語に関連のある情報に関してのみアクセスすればよいため，検索を高速に行うことが可能である．データベースの更新には大きな手間がかかるため，ある一定の周期あるいはある程度まとまった単位で更新を行うことが一般的である．

階層的キーワード名前管理は以下の点で既存のキーワード管理とは異なっているため，既存のキーワード管理をそのまま利用することは難しいと考えられる．

- キーワードが階層的に管理されている
  通常キーワードはそれぞれ独立しており，キーワード間の関係といったものは扱わない．階層的キーワード名前管理においては，キーワード間の関係を管理する必要がある．またファイル検索においては，キーワードはその子孫全てにマッチする．従来のキーワード管理手法はこのような場合については考慮されていない．

- キーワードの更新が即座にシステムに反映される必要がある
  WWW 検索エンジン等では，内容の更新の即時性は必ずしも要求されないため，ある一定の周期あるいはある程度まとまった単位での更新を行ってもシステムに支障をきたすことはない．しかし，階層的キーワード名前管理においてはファイルへの新たなキーワードの付与が即座にシステムに反映される必要がある．

4. 階層的キーワードベースの名前管理手法

階層的キーワード名前管理においては，ファイルテーブルとキーワードテーブルという 2 つのテーブルを用いる．キーワードテーブルは，階層的キーワードの親子関係を管理しており，ファイルテーブルは，個々の階層的キーワードがどのファイルに付与されているかという情報を管理している．

ファイルテーブルの 1 つのエントリは，1 つのファイルに対応し，ファイル名と i ノード番号を含んでいる．キーワードテーブルの 1 つのエントリは，1 つの階層的キーワードに対応し，階層的キーワードとその階層的キーワードに対応するファイルテーブルの i ノード番号を含んでいる．

4.1 キーワードテーブルの構成

キーワードテーブルの構成として，通常のディレクトリ階層と同様に，キーワードテーブルを階層的に構成する方法が考えられる．階層的キーワードごとにキーワードテーブルを 1 つのファイルとして設け，その中にその子にあたるキーワードに対応するファイルの i ノード番号を格納するというものである．この場合，通常のパス名の解決の場合と同様に，root キーワードテーブルから順番にキーワードテーブルを参照していくことによって対応するファイルのファイルテーブルの i ノード番号を得る．表 1にこの場合の root キーワードテーブルの例を示す．表中で，file はそのキーワードに対応するファイルテーブルの i ノード番号，kw は自分の子であるキーワードに対応するキーワードテーブルの i ノード番号を表す．

表 1　root キーワードテーブル

keyword	file	kw
animal	177702	124001
book	177930	124003
image	177800	124050

この手法において，深さ D のキーワードに対応するファイルテーブルの i ノード番号を得るためには，D 回のファイルへのアクセスが必要となる．キーワード階層が深い場合には，多くのファイルアクセスが必要となる．ここでキーワードテーブルは通常の UNIX のディレクトリのように，ファイルの i ノード番号を管理しているわけではなく，キーワードテーブルのエントリは，その直接の子にあたるキーワードのみであるため，個々のキーワードテーブルの大きさは，それほど大きくならないことが予想される．このため，キーワード階層の一部に含まれる階層的キーワードを 1 つのキーワードテーブルで管理する方法が考えられる．表 2に，*/animal* と */book* 以下の階層的キーワードをまとめて管理する root キーワードテーブルの例を示す．この例では，*/image* の子孫の階層的キーワードは表 3に示す別のファイルで管理されている．このようにすることで，キーワードテーブルにアクセスする回数を少なくできる．ただしキーワードテーブルが大きすぎると，不必要なエントリを大量にメモリ上にロードすることになり，キャッシュの効率も落ちることになる．ファイルはブロック単位で読みこまれるために，ファイルの読み込まれるブロックサイズに収まるように，1 つのキーワードテーブルを構成することが考えられる．キーワードテーブルのサイズをある一定の大きさに保つために，必要に応じて，キーワードテーブルを分割したり，統合したりする処理が必要となり，そのためのオーバーヘッドがかかる．ただしこれは，階層的キーワードを新規作成，削除するときにのみかかるものでそれほど大きな問題にならないと考えられる．

4.2 ファイルテーブルの構成

ファイルテーブルの構成には，いくつかの方法が考えられる．

表 2　大きな root キーワードテーブル

keyword	file	kw
animal	177702	
animal/cat	177708	
animal/dog	177715	
animal/dog/pug	177718	
book	177930	
book/nobel	177934	
book/nonfiction	177937	
image	177800	142800

表 3　/image のキーワードテーブル

keyword	file	kw
photo	177708	
photo/color	177708	
picture	177715	
painting	177718	

4.2.1　手法 1

手法 1 はある 1 つの階層的キーワードについての情報だけをファイルテーブルに格納しておく方法である．ファイルテーブルには，ある 1 つの階層的キーワードをキーワードに持つファイルの i ノード番号を格納しておく．表 4に */animal/dog* に対応するファイルテーブルの例を示す．このファイルテーブルの i ノード番号は 177715 である．

表 4　手法 1 における */animal/dog* のファイルテーブル

inode number	filename
195503	pochi.jpg
195780	eat.jpg
146887	jump.gif
158943	diary.txt
136789	cry.wav

手法 1 ではファイルは次のように検索される．各検索キーワードに対し，キーワード自身とその子孫全てに対応するファイルテーブルを参照し，その中に含まれる全ての i ノード番号の集合を作成する．それらの積集合として，検索キーワードに対応するファイルの i ノード番号の集合を求めることができる．しかし，この手法では検索対象のキーワードの子孫の数だけのファイルへのアクセスが生じることになる．一方，ファイルに対するキーワードの付与・除去によるテーブルの更新については，更新が必要な階層的キーワードのファイルテーブルだけを更新すれば良く，非常に効率が良い．

4.2.2　手法 2

手法 2 は階層的キーワードとその全ての子孫の情報を 1 つのファイルテーブルに格納しておく方法である．ファイルテーブルには，ある 1 つの階層的キーワードおよびそのキーワードの子孫をキーワードに持つファイル集合の i ノード番号のリストを格納しておく．これによりファイルアクセスの回数は階層的キーワードにつき一回ですむため，階層的キーワードの子孫が多い場合に検索の手間は大幅に軽減される．しかし，親子関係にある階層的キーワードに対応するファイルテーブルにおいてはエントリが重複し，より多くの記憶領域が必要となる．

一方ファイルに対するキーワードの付与・除去によるファイルテーブルの更新については，更新された階層的キーワードとその先祖全てのファイルテーブルを更新する必要があるため，手法 1 と比べて効率が悪い．

4.2.3　手法 3

手法 3 では，手法 2 におけるエントリの重複による記憶領域の増大を避けるため，子孫に対応するファイルテーブルを排除したものである．この手法では，ファイルテーブルにおいて，同じ階層的キーワードをもつファイルの i ノード番号は連続して格納しておく．また，キーワードテーブルを拡張し，深さ 2 以上の階層的キーワードについては，ファイルテーブル中の対応するエントリの位置を識別するため，キーワードテーブルには，ファイルテーブルの i ノード番号に加え，offset と number という 2 つの整数値を格納しておく．offset はファイルテーブル中のどの位置から対応するエントリが格納されているかを表し，number は対応するエントリの数を表す．表 5に表 2 のキーワードテーブルの一部を拡張した場合の例を示す．

表 5　拡張された root キーワードテーブル

keyword	file	kw	offset	number
animal	177702		1	60
animal/cat	177708		35	5
animal/dog	177715		40	7
animal/dog/pug	177718		45	2

検索の際には，それぞれの検索キーワードについて，

キーワードテーブルからファイルテーブルのiノード番号とoffsetとnumberを読み取り，ファイルテーブルから対応する部分のiノード番号を取り出す．しかし，この手法では，キーワードテーブルにファイルテーブルの内容に関する情報を記録しているために，キーワードの付与・除去によるファイルテーブルの更新によって，キーワードテーブルも更新する必要がある．例えば，表5の状態で，あるファイルに*/animal/cat*というキーワードが新たに付与されたとする．このとき，iノード番号が177702であるファイルテーブルの*/animal/cat*に対応する箇所に，新たにキーワードが付与されるファイルのiノード番号が追加される．それに伴い，キーワードテーブルの*/animal/cat*のnumberが7から8に変化する．さらにその変化に伴い，*/animal*の子である*/animal/dog*以下の全てのエントリのoffsetの値を1ずつ増やすという作業が必要になる．

4.3 考察

手法1の長所は，テーブルの更新が容易に行うことが出来るということである．短所は，検索の際に，検索キーワード自身だけでなく，その子孫全てに対応するファイルテーブルにアクセスしなければならないということである．キーワード階層中の各キーワードが子孫を多数持つ場合は，手法1は実用的ではないが，各キーワードの子孫の数がそれほど多くない場合は利用することも可能であると考えられる．

手法2の長所は，検索の際のファイルアクセス回数が少なくてすむため，手法1と比較して高速な検索が可能であるということである．短所は，エントリの重複によって，より多くの記憶領域が必要となるとなることである．より高速な検索を提供するために記憶領域を犠牲にしているが，近年の2次記憶装置の大容量化を考慮すれば，手法2は現実的に使用可能な手法であると考えられる．

手法3の長所は，手法2と大差ない検索速度を提供できるということと記憶領域が少なくて済むということである．短所は，テーブルの更新に非常に手間がかかるということである．更新がボトルネックにならない程度の速度で可能ならば，実用的であるといえる．

このようにそれぞれの手法に長所と短所があげられるが，実際にどの手法が有効であるかは，

- 実際に作られるキーワード階層の形状
- 1つのファイルに付与されるキーワードの個数
- 検索に用いられるキーワード数とその深さ
- キーワードの更新頻度

等に依存すると考えられる．しかしこれらは実際に運用してみなければ把握できないものである．現在UNIX上に階層キーワードベースの名前管理システムのプロトタイプシステムを実装しており，実際の運用を通じてこれらのデータを収集し，これをもとに，どの手法が有効であるかを評価する予定である．

5. あとがき

本研究では，階層的キーワード名前管理におけるキーワード管理手法について検討を行った．階層的キーワード名前管理は，キーワード間の関係の管理とキーワードとファイルの関係の管理に区別されるという点とキーワードに関するより柔軟な操作を提供するという点で従来の階層的名前管理とは異なっており，キーワードが階層的に管理されているという点とキーワードの更新が即座にシステムに反映される必要があるという点で従来のキーワード名前管理手法とは異なっている．そこでこれらの階層的キーワード名前管理の特徴をふまえた名前管理手法の考案を行い，それらの定性的評価を行った．

今後は，現在作成中のプロトタイプを用いて，ユーザが実際に階層的キーワード名前管理システムを利用した場合のデータを収集し，それをふまえて最適なキーワード管理手法の検討を行う予定である．

References

[1] S. Sechrest and M. McClennen, "Blending hierarchical and attribute-based file naming", IEEE Proc. of the 12th IEEE Intl. Conf. on Distributed Computing Systems, pp.572-580, Yokohama, Japan, Jun 1992.

[2] D.K. Gifford, P. Jouvelot, M. A. Shedon and J. W. O'Toole, "Semantic File Systems", ACM Proc. of the 13th ACM Symp. on Operating Systems Principles, pp.16-25, Pacific Grove, CA, Oct 1991.

[3] B. Clifford. Neumann and Steven. Seger. Augart, "Prospero: A Base for Building Information Infrastructure", Proc. INET, 1993.

[4] R. Baeza-Yates, B. Ribiero-Neto, "Modern Information Retrieval", Addison-Wesley, ISBN 0-201-39829-X, 1999.

[5] C. Faloutsos. "Access methods for text", ACM Comput.Surv.,17(1):49-74,March 1985.

[6] J. Zobel,A. Moffat, K. Ramamohanarao, "Inverted Files Versus Singnature Files for Text Indexing" ACM Trans. Database Syst. Vol.23, No.4, pp.453-490, 1998.