DEIM Forum 2012 F8-6

WebIndexアタッチエンジンの大規模分散構成

小須田達哉† 松崎 智子† 遠山 元道††

† †† 慶應義塾大学理工学部情報工学科 〒 223–8522 神奈川県横浜市港北区日吉 3–14–1
E-mail: †{kosuda,matsuzaki}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし 著者らはユーザ主体の Web 情報資源結合を実現する Web Index(WIX) システムを開発している. WIX システムとは, Web ドキュメントに対する Web 資源結合サービスを実現するシステムである. 本研究では, Web 資源結合動作であるアタッチ処理を行う WIX アタッチエンジンの分散構成の設計, 実装を行った.

キーワード WIX, Web 情報システム, 分散処理

Distiributed configuration of WebIndex Attach Engine

Tatsuya KOSUDA†, Tomoko MATSUZAKI†, and Motomichi TOYAMA††

† †† Department of Information and Computer Science, Keio University
Hiyoshi 3–14–1, Kouhoku-ku, Yokohama-shi, Kanagawa, 223–8852 Japan
E-mail: †{kosuda,matsuzaki}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

1. はじめに

著者らは Web における利用者主導による情報資源結合を実現するために, Web Index（以下 WIX とする）と呼ぶ情報資源表現形式の提案, 開発を行っている. WIX ファイルとは, キーワードとそれに対応する URL をペアとするエントリの集合であり, XML 形式で記述される. それを閲覧中の Web 文書に結合（アタッチ）することで, Web 文書中のキーワードが対応する URL へのハイパーリンクに変換される.

WIX アタッチエンジンは Find 処理 →Select 処理 →Decide 処理 →Rewrite 処理という処理の流れを持つ. WIX アタッチエンジンでは, 全ての WIX ファイルからエントリ情報を抽出し, Aho-Chorasick 法によるオートマトン（Find インデックス）を構築し辞書式マッチングを行なっている. しかし, この手法では WIX アタッチエンジン 1 台の性能には上限があり, 処理量の増加に対してスケールできない.

そういった背景を受けて, 全 WIX ファイルのエントリ集合の部分集合から Find インデックスを構築することでサーバ 1 台当たりの物理メモリ容量の上限に対応する設計方式を実現する. さらに, この Find インデックスを複数のサーバに複製し, WIX アタッチエンジンのスループット向上を目的とし, 負荷分散処理を実現する.

本論文の構成は以下の通りである. 2 章で WIX の概要について述べる. 3 章で WIX アタッチエンジンにおけるそれぞれの処理の概要について述べる. 4 章で WIX アーキテクチャについて説明する. 5 章で WIX アタッチエンジンの分散構成について説明する. 6 章, 7 章で評価 · まとめを行う.

2. WIX

WIX の概要は以下の通りである.

2.1 背 景

近年, Web の普及と共に人々は検索エンジンを利用して情報検索を行うようになった. ユーザは情報を得たい単語を検索エンジンに入力し, その結果を出力した Web 文書中から必要な情報得るのが一般的である. したがって, ユーザが Web 文書中の単語に対して新たな情報を得たいという要求が生じた場合, ユーザが更にその単語を検索エンジンなどにかけなければならない. このような単語が複数存在する場合, ユーザは何回も検索エンジンを使わねければならず, かなりの負担になってしまうと考えられる.

2.2 WIX ファイル

WIX ファイルは文章中のキーワードとそれに対応する URL からなるペアを一つのエントリとしたものの集合である. 実際には, 図 1 のような XML 形式で記述されている. キーワードを keyword 値に, URL を target 値に格納し, それらを一つとして entry タグで囲う.

2.3 アタッチ

WIX システムでは関係データベースの結合演算の考え方を Web の世界に持ち込むことによって, ユーザ主体の Web 情報源の結合を図っている. 結合演算の関係モデルでは, それ以前のデータモデルがアドレス / ポインタによって行っていた関連付けを, 値に基づいて主キー, 外部キーで実現した. 現在の Web

```
<WIX>
  <entry>
    <keyword>XXX</keyword>
    <target>http://www.XXX.com</target>
  </entry>
  <entry>
    <keyword>YYY</keyword>
    <target>http://www.YYY.co.jp/YYY.html</target>
  </entry>
  ...
</WIX>
```

図 1　WIX ファイル記述方式

におけるアンカー /URL は関係モデル以前の DB モデルと酷似している．そこで WIX ではアンカーテキストとリンクを Web ドキュメントから独立した WIX ファイルという情報源として保存し, それを適宜ドキュメントに対して結合を行うことによってハイパーリンクの生成を行う．この結合操作をアタッチと呼ぶことにする．

例えば, 閲覧中の Web 文書中にある英単語について英和辞書の情報を得たい場合, 従来なら英単語一つずつ検索エンジンなどで検索しなければならない．これに対し WIX では閲覧中の Web 文書に英和辞書の WIX ファイルを結合することで, 英単語からその英単語のページへのハイパーリンクが生成される．

また, 同じ英単語をキーワードとする WIX ファイルでも, 例えば英和辞書の他にも英英辞書, 英仏辞書, または同じ英和辞書でも会社が異なる辞書まで WIX ファイルを提供すれば, 利用者は必要に応じて自由に WIX ファイルを選択し, 自分の閲覧している文書にアタッチすることで状況に応じた情報を簡単に切り替えることができる．これにより Web ページ作成者の意思とは独立した, ユーザ主体でハイパーリンクの生成を行うことができる．

2.4　WIX システム (クライアント)

WIX システムのクライアントサイドは, FireFox add-on や Chrome Extension によって実装されている．以下の図は FireFox add-on の例である．

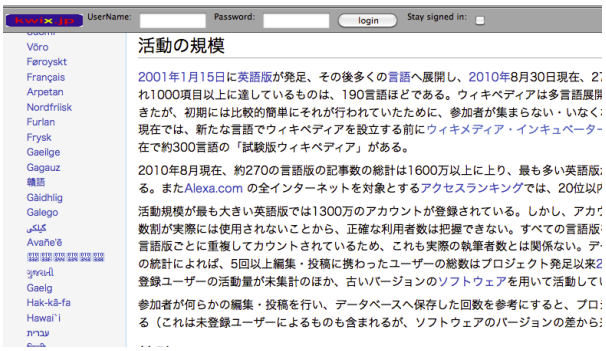

図 2　Firefox add-on(1)

WIX システムでは, ユーザは予め目的に適った WIX ファイルをブックマークしておく必要がある．図 2, 図 3 のようにログインするとツールバーにアタッチボタンが生成される．このアタッチボタンはブックマークに対応し, ボタン一つに対して複数の WIX ファイルを登録できる．アタッチボタンをクリックすることで, 図 3 の記事にはなかったハイパーリンクが図 4 の記事には生成される．これによって, WIX ファイル内の target タグに記述されている URL と結合されたことになる．

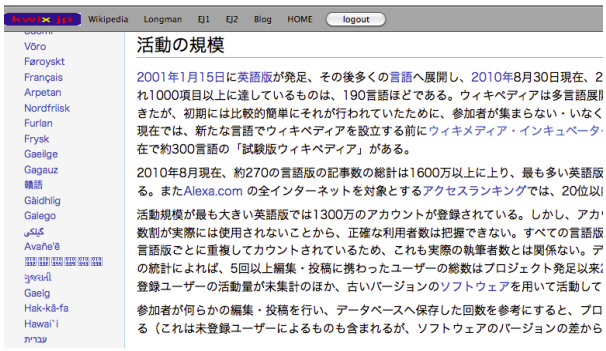

図 3　Firefox add-on(2)

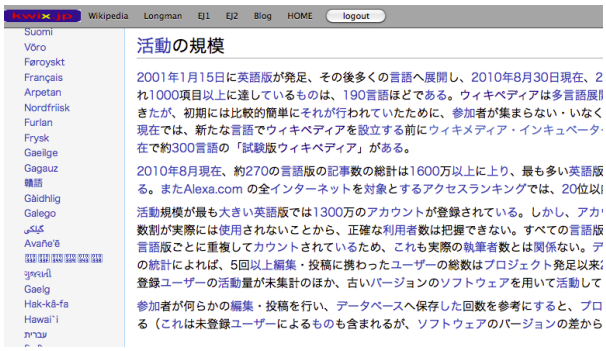

図 4　Firefox add-on(3)

3.　WIX アタッチエンジン

3.1　FSDR 処理

システムの Web 資源結合動作であるアタッチ処理について述べる．

アタッチ処理は以下の 4 フェーズに分けられる．この一連の流れを FSDR 処理とする．

- Find 処理
- Select 処理
- Decide 処理
- Rewrite 処理

3.1.1　Find 処 理

Find 処理では Web 文書と WIX ファイル集合を入力として, 全 WIX ファイル中の全てのエントリのうち文書中に存在するキーワードを持つエントリを Web 文書中の出現位置とセットにして返す．このセットの集合のことを Find 結果と呼ぶ．Find 結果はその出現位置によってソートされ, 次処理に引き継がれる．

3.1.2　Select 処理

Select 処理では Find 結果とユーザのブックマーク情報を入力として, Find 結果からユーザのブックマークしている WIX ファイルのエントリのみに絞り込む．さらに, 同一出現位置の

Find 結果の要素に対しては最長一致をとる.

3.1.3 Decide 処理

Decide 処理では Select 結果から更にキーワードの周辺文字列を利用して, より Web 文書のトピックとマッチするエントリを抽出するなどの研究がされている.

3.1.4 Rewrite 処理

Rewrite 処理では Decide 結果を用いて入力 Web 文書を新たなハイパーリンクのついた Web 文書に書き換える. これによって, アタッチが完了する.

4. WIX アーキテクチャ

現在の WIX システムのアーキテクチャを図 5 に示す.

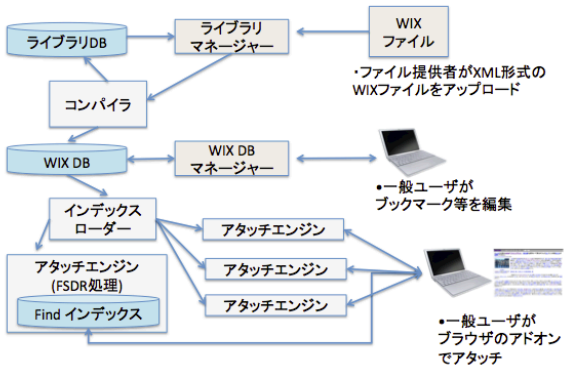

図 5 WIX アーキテクチャ

また, WIX システムでは WIX ファイルの情報をライブラリ, WIX DB, Find インデックスの 3 つの異なる形態で管理する. これら 3 形態の比較概念図を図 6 に示す.

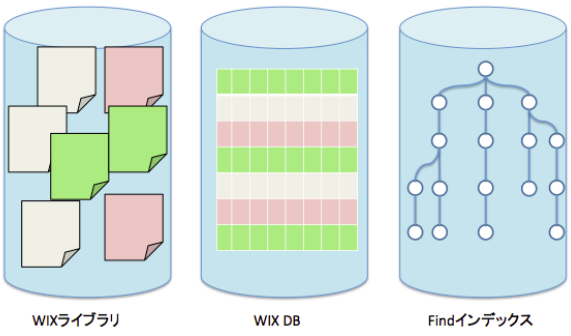

図 6 形態比較概念図

4.1 WIX ライブラリ

ライブラリでは, 全ての WIX ファイルを過去のヴァージョンも含めて XML テキストそのまま保存する (図 6 左). 即ち, ライブラリではファイル単位での情報管理となっている. しかしながらアタッチにおいては全ての WIX ファイルのエントリに対して, 辞書式マッチングを行わなければならない. 故に WIX ファイルをエントリ単位に分解し格納する WIX DB が必要となる.

4.2 WIX DB

WIX DB においては, ライブラリで管理している WIX ファイルをエントリ単位に分解し, RDB にタプルとして管理する (図 6 中央). 即ち, WIX DB ではエントリ単位での情報管理となっている. ここで, WIX ファイルの情報を扱う entry テーブルについて述べる.

entry テーブルでは WIX ファイルのもつエントリ情報の管理を行う (表 1). そのエントリが所属する WIX ファイルの wid, エントリの id である eid, 辞書語となるキーワードの keyword とそれに対応する URL である target を属性として持つ.

表 1 entry テーブル

wid	eid	keyword	target
1	1	細貝萌	http://ja.wikipedia···
1	2	川島永嗣	http://ja.wikipedia···
1	3	アウクスブルク	http://ja.wikipedia···
2	1	イチロー	http://ja.51channel.tv···
3	1	細貝萌	http://samuraiblue.jp/···
3	2	川島永嗣	http://samuraiblue.jp/···
···			

4.3 Find インデックス

Find インデックスでは, WIX DB の entry テーブルからエントリ情報をメモリ上に展開する (図 6 右). WIX システムでは Aho-Corasick 法に基づくオートマトンを構築し, 辞書式マッチングを行う.

5. WIX アタッチエンジンの分散構成

5.1 概 要

現在の WIX システムのアーキテクチャは第 4 章で述べた通りである. この時 Find インデックスでは, WIX DB 内の全てのエントリ情報から Aho-Corasick 法に基づくオートマトンを構築し, 辞書式マッチングを行う. これを統合インデックス方式と呼ぶことにする. WIX システムにおける先行研究 [2] では, 全ユーザが共通のインデックスを利用するため, 統合インデックス方式を採用している. これに対して, 本研究では全てのエントリ集合の部分集合からオートマトンを構築し, 辞書式マッチングを行う方式を実現する. これを部分インデックス方式と呼ぶことにする. 部分インデックス方式では, サーバの物理メモリ容量の上限に対応することを目的とする.

ここで, どのように全 WIX ファイルのエントリ情報から部分集合を作るかという問題がある. この問題に対して, WIX ファイルのエントリ情報におけるターゲットのコンテンツによって部分集合を作る手法を実現する. ターゲットのコンテンツとは, ターゲットの指す Web ページの言語などが考えられる. これを WIX ファイルのコンテンツ分散と呼ぶことにする.

WIX システムでは, ユーザはターゲットが同一コンテンツのものを 1 つのブックマーク内にブックマークすると考えられるため, WIX ファイルのコンテンツ分散の結果として, Find インデックスの局所化の向上に繋がると考える. また, 部分インデックス方式による Find インデックスを複製することで WIX

アタッチエンジンのスケールアウトを図る．この時，同一エントリ集合から Find インデックスを構築したアタッチエンジンサーバの集合をサーバグループと定義する．サーバグループを定義することによって，サーバグループ内の WIX アタッチエンジンにおいてアタッチ処理の負荷分散を実現する．

ここで，部分インデックス方式による Find インデックスの分散化及びアタッチ処理の負荷分散を実装した WIX アタッチエンジンの分散構成におけるサーバサイド処理として，アタッチ処理である FSDR 処理の分散化を実装する．

WIX アタッチエンジンの分散構成におけるサーバグループでは，WIX ファイルのコンテンツ分散による Find インデックスの局所化が実現されているため，1 つのサーバグループでユーザのアタッチ要求に応えられることが理想的となる．しかし，現実的に 1 つのサーバグループでユーザのアタッチ要求に応えられない場合が考えられる．ゆえに，複数のサーバグループにおける Decide 処理の結果を統合する手法を実装する．これにより，WIX ファイルのコンテンツ分散の方式に関わらずユーザのアタッチ要求に応えられる WIX アタッチエンジンの分散構成を実現する．

5.2 部分インデックス方式

Find インデックスはメモリ上に展開するため，WIX ファイルの増加に伴いメモリ不足になることが考えられる．表 2 に日本語 Wikipedia と英語 Wikipedia の記事タイトル集合から作成した WIX ファイルをもとに Find インデックスを構築した場合の消費メモリ量を示す．

表 2　エントリ件数と消費メモリ量

WIX ファイル	エントリ数 (件)	更新機能あり Find インデックス 消費メモリ量 (GB)	更新機能なし Find インデックス 消費メモリ量 (GB)
日本語 Wikipedia.wix	100 万	4.5	1.6
英語 Wikipedia.wix	100 万	8.9	1.7
英語 Wikipedia.wix	200 万	20.8	4.9
英語 Wikipedia.wix	400 万	38.8	8.7
英語 Wikipedia.wix	800 万	65.3	14.9

Find インデックスでは WIX ファイルの追加 · 更新に対して，高速かつ動的な差分更新を行なっている．この差分更新機能がある場合とない場合の消費メモリ量の違いを表 2 に示している．差分更新機能がある場合の英語 Wikipedia.wix の 800 万件のエントリをもとに Find インデックスを構築した場合，65.3GB ものメモリ量が必要となる．また，WIX ファイルはこの他にも多数存在するため，アタッチエンジンにはハイエンドサーバを用い，また，WIX ファイルの増加に応じて順次サーバをスケールアップする必要がある．これは WIX システムを運営するにあたってコスト面で望ましくない．

ここで，アタッチエンジンサーバをスケールアウトすることを考える．この時，各サーバでは部分インデックス方式による Find インデックスを構築することで，メモリ量に関して，1 台のサーバにかかるはずの負担を複数のサーバに分散するため，1 台のサーバの物理メモリ容量の上限に対応する設計方式が実現される．ここで，部分インデックス方式による WIX ファイルの情報管理における 3 形態の比較概念図を図 7 に示す．

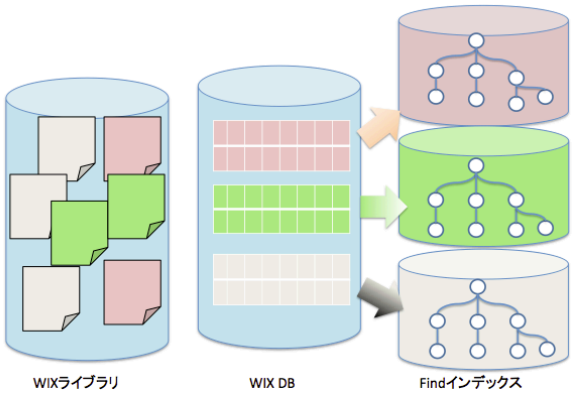

図 7　WIX アタッチエンジンの分散構成における形態比較概念図

5.2.1 WIX ファイルのコンテンツ分散

部分インデックス方式において全エントリ集合から部分集合を作成する際，エントリ情報におけるターゲットのコンテンツに着目し部分集合を作成する．これを WIX ファイルのコンテンツ分散と呼ぶ．ターゲットのコンテンツとは，そのターゲットのページ内言語などが考えられる．この時，ユーザは予め目的に適った WIX ファイルをブックマークする必要があるため，WIX システムでは同一ブックマーク内には同一コンテンツのものをブックマークすると考える．ゆえに，1 台のサーバの物理メモリ容量の上限に対応することを目的とした WIX ファイルのコンテンツ分散の結果として Find インデックスの局所化が実現される．

5.3 サーバ管理システム

部分インデックス方式の導入により，Find インデックスの局所化を達成できるような WIX ファイルのコンテンツ分散を実現しなくてはいけない．また，ライブラリマネージャから登録された WIX ファイルは適宜部分集合に含める必要がある．さらに，WIX ファイルの追加 · 更新に対して Find インデックスの動的更新を行うために，どのサーバグループがどの WIX ファイルのエントリ集合から Find インデックスを構築しているのかという情報管理が必要となる．

5.3.1 サーバグループ内の WIX ファイル管理

WIX ファイルのコンテンツ分散は，Find インデックスの局所化の達成，サーバメモリ資源の効率よい利用，ユーザからの処理要求の負荷分散等，WIX システム運用状況における様々な要素を考慮する必要がある．例えば，表 2 のように，Wikipedia 記事タイトル集合から作成した WIX ファイルを元に Find インデックスを構築した場合，多くのメモリ容量を必要とする事実を考慮し，他の WIX ファイルとは別のサーバグループに属させることが考えられる．(表 3)

表 3　WIX ファイルコンテンツ分散の一例

サーバグループ A	サーバグループ B	サーバグループ C	サーバグループ D
日本語 Wikipedia	日本語 Wikipedia 以外	英語 Wikipedia	英語 Wikipeida 以外

このように, WIX システムの運用状況に応じて WIX ファイルのコンテンツ分散の方式は様々なものが考えられることから, サーバグループ内の WIX ファイル管理を自動化することは困難である. ゆえに, システム管理者の手動管理で行われる.

5.3.2 Find インデックスの更新

Find インデックスの更新には, リフレッシュとリロードの 2 種類がある. リフレッシュとは WIX ファイルの追加・更新に対して高速かつ動的な差分更新を行う手法である. リロードとは, 現在起動中のオートマトンを一時中断し, 一から Find インデックスを構築し直す手法である. 表 4, 5 にそれぞれにかかる時間を示す.

表 4　リフレッシュ処理にかかる時間

更新エントリ件数（件）	更新時間 (ms)
1000	742
5000	1649
10000	1786
50000	6030

表 5　リロード処理にかかる時間

WIX ファイル	エントリ件数（件）	更新時間 (ms)
日本語 Wikipedia.wix	100 万	23265
英語 Wikipedia.wix	100 万	24110
英語 Wikipedia.wix	200 万	43949
英語 Wikipedia.wix	400 万	108959
英語 Wikipedia.wix	800 万	225640

リフレッシュには, WIX システムを停止することなく Find インデックスの更新ができるという利点があるが, 表 2 のように, この差分更新のためには多くの物理メモリ容量を必要とする. そこで, サーバグループ内では差分更新機能がないタイプの Find インデックスを併用することで, サーバメモリ資源を効率よく利用する. この時, 差分更新機能がないタイプの Find インデックスを更新する場合, リロードを行う必要がある. 表 5 で分かるように, エントリ数に応じてリロードには最大数分間を要する. 単一のサーバでは問題となるが, サーバグループ内で適切な負荷分散処理を行うことで, 一方の稼働中アタッチエンジンサーバにアタッチ処理を要求することで WIX システムを停止させることなくユーザのアタッチ要求に対応できる.

5.4 WIX アタッチエンジンの分散構成におけるアーキテクチャ

WIX アタッチエンジンの分散構成におけるアーキテクチャを図 8 に示す.

5.5 FSDR 処理の分散化

ここで, WIX アタッチエンジンの分散構成におけるサーバサイド処理として FSDR 処理の分散化を実装する. まず, 図 8 のマスターノードには Rewrite モジュールを配置する. また, スレーブノードには Find/Select/Decide モジュールを配置する. ここで, スレーブノードの持つモジュールを FSD モジュールと呼ぶことにする. ジョブクライアントは, まずマスターノードにアタッチの実行を要求する. 次にマスターノード

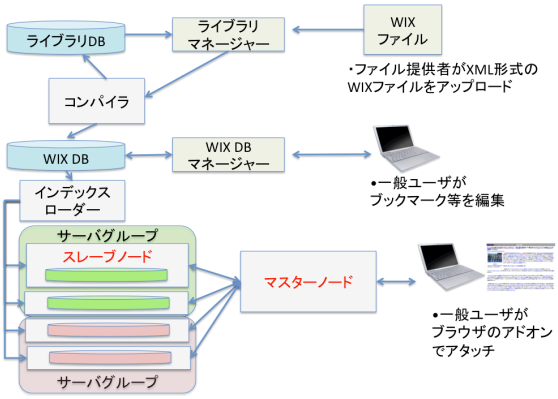

図 8　WIX アタッチエンジンの分散構成におけるアーキテクチャ

では, Find/Select/Decide タスクを生成し, それを起動状態にする. タスクの生成時に, ジョブクライアントからマスターノードへの要求はそのままスレーブノードに引き継がれることになる. スレーブノードでは, タスクの起動状態を確認し, あらかじめ配置された FSD モジュールを実行する. 最後に, その出力がマスターノードへと引き継がれ, 必要に応じてデータの統合を行い, マスターノードに配置された Rewrite モジュールを実行し, ジョブクライアントに新たなハイパーリンクつき Web 文書を提供することでアタッチの実行を終了とする.

5.5.1 サーバグループの選択

マスターノードは, どのサーバグループ内のスレーブノードの持つ FSD モジュールを実行するかを決定しなくてはいけない. そこで, サーバグループの選択方法について説明する. まず, 分散構成におけるスレーブノードの数を n とすると, Find インデックスの数は n となる. ここで, 分散構成におけるある 1 つのサーバの持つ Find インデックスのエントリ情報の集合を F_τ とすると, 全体としては以下のように集合族として表すこととする.

$$\{F_\tau \mid \tau \in T\} \quad (\ T = \{1, 2, \cdots, n\}\ ) \tag{1}$$

しかし, $\{F_\tau \mid \tau \in T\}$ の中には重複要素を含む. 例えば $F_1 = F_2$ ということがあるということである. これをサーバグループと定義した. ゆえに, Find インデックスを分散・複製するにあたってサーバグループ単位で考える. サーバグループ数を k とすると, 全体としては以下の集合族で表すこととする.

$$\{S_\lambda \mid \lambda \in \Lambda\} \quad (\ \Lambda = \{1, 2, \cdots, k\}\ ) \tag{2}$$

また, 部分インデックス方式において, 全てのユーザがアタッチできるようにしなくてはいけない. ゆえに, WIX DB の持つ全エントリ情報の集合を E とすると, 以下の関係がある.

$$E = \bigcup_{\lambda \in \Lambda} S_\lambda \tag{3}$$

サーバグループの選択にはそのサーバグループの持つ Find インデックスのエントリ情報の集合と, ユーザがアタッチ要求時に選択したブックマークが持つ WIX ファイル内のエントリ

情報の集合の包含関係を用いる．ゆえに, ユーザがアタッチ要求時に選択したブックマークが持つ WIX ファイル内のエントリ情報の集合を B とすると次の関係がある．

$$B \subseteq \bigcup_{\lambda \in P} S_\lambda \quad (P \subseteq \Lambda \text{ and } P \neq \emptyset) \tag{4}$$

5.5.2 スレーブノードの選択

マスターノードはサーバグループの選択後, サーバグループ内でスレーブノードを決定する必要がある．この時, サーバグループ内ではアタッチ処理の負荷分散を実現する．

WIX システムにおいて, 1 リクエスト当たりの負荷は, ユーザがアタッチ要求時に閲覧中の Web 文書によって異なる．そこで, スレーブノードにおいてアタッチ要求時に入力とした Web 文書のバイトサイズを測定し, アタッチ要求毎に Web 文書のバイトサイズを加算していき, これをスレーブノードにおけるアタッチ処理の負荷とする．この時, 負荷分散処理としてこの加算されたデータサイズが最小のスレーブノードにマスターノードはアタッチを要求する．これにより, サーバグループ内でアタッチ処理の負荷分散が実現されたことになる．

5.6 Decide 結果の統合

式（4）において, $|P| = 1$ である時,

$$B \subseteq S_\lambda \tag{5}$$

となる．この時, 1 つのサーバグループを選択することでユーザのアタッチ要求に応えることができる．よって, この時にマスターノードで Rewrite モジュールを実行する時の入力は, サーバグループ内のスレーブノードから引き継がれた Decide 結果をそのまま使えばいいことになる．1 台のサーバの物理メモリ容量の上限に対応することを目的とした WIX ファイルのコンテンツ分散の結果として, Find インデックスの局所化が実現されているため, $|P| = 1$ となることが理想的である．

しかし, WIX ファイルのコンテンツ分散の方式によっては 1 つのサーバグループではユーザのアタッチ要求に対応できない場合が考えられる．例えば, ユーザの 1 つのブックマーク内にはエントリ情報におけるターゲットのページ内言語が同一であると考え, 全エントリ集合からターゲットのページ内言語毎に部分集合を作成し, 部分インデックス方式による Find インデックスを構築したとする．しかし, 全ユーザが 1 つのブックマーク内にターゲットのページ内言語が同一のものをブックマークするとは限らない．ゆえに, $|P| \neq 1$ の時も考慮する必要がある．この時, 複数のサーバグループを選択することになるため, 複数の Decide 結果がマスターノードに引き継がれることになる．ここで, WIX ファイルのコンテンツ分散の方式に関わらずユーザのアタッチ要求に対応できる WIX アタッチエンジンの分散構成を実現するために, Decide 結果の統合を実装する．

Decide 結果を統合する際に, 以下の 2 つの場合について考える．

$$\{S_\lambda \mid \lambda \in P\} \quad (S_\lambda \text{ がそれぞれ互いにに素である}) \tag{6}$$

$$\{S_\lambda \mid \lambda \in P\} \quad (S_\lambda \text{ がそれぞれ互いにに素でない}) \tag{7}$$

まず, (6) の場合, $\bigcup_{\lambda \in P} S_\lambda$ は直和であり, $\{S_\lambda \mid \lambda \in P\}$ は E の直和分解である．ゆえにそれぞれの Decide 結果の要素に重複要素が存在することはないため, Decide 結果の統合後, 出現位置でソートのみすればいいことになる．

次に, (7) の場合, $\{S_\lambda \mid \lambda \in P\}$ は E の直和分解でないため, それぞれの Decide 結果の要素に重複要素が存在する場合がある．よって, Decide 結果の統合後, 出現位置でソート及び重複削除の処理を行う必要がある．

また, Find 結果の要素を $x_{s,e}$ とする．添え字 s はその要素が持つキーワードの Web 文書中における出現位置の開始位置を表し, e は終了位置を表す．ここで, 次の場合も考慮する必要がある．

$$x_{s_1,e_1} \in S_{\lambda_1} \text{ and } x_{s_2,e_2} \in S_{\lambda_2} \text{ and } x_{s_1,e_1} \notin S_{\lambda_2} \tag{8}$$

$s_1 \leq s_2 < e_1$ かつ $e_2 - s_2 < e_1 - s_1$ である場合, Find インデックスとして S_{λ_1} を持つ FSD モジュールの実行結果は要素 x_{s_1,e_1} を出力し, Find インデックスとして S_{λ_2} を持つ FSD モジュールの実行結果は要素 x_{s_2,e_2} を出力する．これは, 第 3 章の第 1 節で述べた FSDR 処理の Select 処理における最長一致による．ここで Decide 結果を統合すると, 統合後の Decide 結果の要素に x_{s_1,e_1} と x_{s_2,e_2} の両方を持つことになってしまう．ゆえに, 再び最長一致をとり x_{s_2,e_2} を削除する処理を行う必要がある．この処理は (6) と (7) のどちらの場合においても実行する必要がある．

6. 評価実験

6.1 実験目的

本研究の評価として, WIX アタッチエンジンの分散構成における 1 リクエスト当たりのアタッチ実行速度と, 分散構成前の WIX アタッチエンジンの 1 リクエスト当たりのアタッチ実行速度を比較し, WIX アタッチエンジンの分散構成におけるアタッチ処理に対するオーバーヘッドが小さく, 以前のアタッチエンジンと同等のアタッチ実行速度であることを示す．この時, WIX アタッチエンジンにおけるアタッチ処理に対するオーバーヘッドとして以下のものが考えられる．

- サーバグループの選択によるオーバーヘッド
- サーバグループ内のスレーブノードの選択によるオーバーヘッド
- マスターノードにおけるスレーブノードからの Decide 結果の取得によるオーバーヘッド
- Decide 結果の統合によるオーバーヘッド

6.2 実験条件

比較対象として分散構成前の WIX アタッチエンジン, 即ち統合インデックス方式による Find インデックスを持つ WIX アタッチエンジンサーバ (以下, 比較対象とする) を 1 台用意した．また, 分散構成として 5 台のサーバを用意した．そのうち, 1 台は物理サーバ (以下, マスタとする) であり, 残りの 4 台は仮想サーバ (以下, スレーブ A〜D とする) である．ここで, マスターノードとして物理サーバ, スレーブノードとして仮想サーバを選択した．

WIX ファイルのコンテンツ分散の方式としてエントリ情報に

おけるターゲットのページ内言語に着目し, 表 6 のようにサーバグループを作成した. この時, 日本語 Wikipedia の記事タイトル集合から作成した WIX ファイルを独立したサーバグループとしているのは, 他の WIX ファイルに比べエントリ件数が多いことを考慮したためである.

表 6　WIX ファイルのコンテンツ分散方式 (1)

サーバグループ A	サーバグループ B	サーバグループ C	サーバグループ D
日本語	英語	その他言語	日本語 Wikipedia

ここで, Decide 結果の統合によるオーバーヘッドを考慮する設計にするために, あえてサーバグループ D を 2 つに分割する. よって, 最終的に表 7 のような 5 つのサーバグループを作成した.

表 7　WIX ファイルのコンテンツ分散方式（2）

サーバ グループ A	サーバ グループ B	サーバ グループ C	サーバ グループ D	サーバ グループ E
日本語	英語	その他言語	日本語 Wikipedia (1)	日本語 Wikipedia (2)

それぞれのサーバグループに対して表 8 のようにスレーブ A〜D を配置する. この時, 比較対象及びマスタに対してアタッチ要求時に選択するアタッチボタンとして, 日本語 Wikipedia の記事タイトル集合から作成した WIX ファイルをブックマークすることで, それぞれのアタッチ処理速度を比較した場合に, 分散構成におけるアタッチ処理に対するオーバーヘッドとして第 1 節で示した 4 つのオーバーヘッドを考慮しなくてはいけない設計にすることができる.

表 8　サーバグループとサーバの対応表

サーバ グループ A	サーバ グループ B	サーバ グループ C	サーバ グループ D	サーバ グループ E
			スレーブ A スレーブ B	スレーブ C スレーブ D

6.3 実験結果

まず, 比較対象とマスタにおいて異なる Web 文書サイズに対しアタッチを実行した時の, アタッチ実行速度の比較表を表 9 に示す.

表 9　アタッチ実行速度の比較

バイト数 (KB)	比較対象 (ms/req)	マスタ (ms/req)
80	368.2	370
150	419.8	411.7
220	895.6	898.5

表 9 より, 分散構成のマスタにおける 1 リクエスト当たりのアタッチ実行速度が比較対象と同等のものになっていることが分かる.

ここで, 第 1 節で示した分散構成において考えられる, アタッチ処理に対するオーバーヘッドの測定結果を表 10 に示す.

まず, サーバグループの選択によるオーバーヘッドがどの場合においても 0ms となっていることが分かる. これは, 集合演算プログラムをアタッチ処理とは独立した処理として実行し, アタッチ処理内では, 予め集合演算プログラムによって計算された値を参照することで, そのブックマークに対応するサーバグループの情報が即座に得られるためだと考える.

表 10　アタッチ処理に対するオーバーヘッド

バイト数 (KB)	サーバグループの選択 (ms)	スレーブノードの選択 (ms)	Decide 結果の取得 (ms)	Decide 結果の統合 (ms)
80	0	6.3	-35.3	32.6
150	0	6.2	-32.5	28.8
220	0	6.4	-55.5	64.4

また, スレーブノードの選択は平均 6.3ms となっている. これは, それぞれのスレーブノードにおけるデータサイズを参照する処理が, マルチスレッドで動作するためだと考える. 即ち, スレーブノード数に依存することなくサーバグループ内でスレーブノードの選択をすることが可能である.

よって, サーバグループの選択及びサーバグループ内でのスレーブノードの選択による, アタッチ処理に対するオーバーヘッドが小さいものであることが分かる.

次に, Decide 結果の取得によるオーバーヘッドが負の値を取っていることが分かる. ここで, Decide 結果の取得によるオーバーヘッドについて説明する. 比較対象ではローカルで FSD モジュールを実行し, Decide 結果を得られるのに対し, 分散構成ではマスターノードがスレーブノードに対し FSD モジュールの実行要求を出し, スレーブノードにおいて FSD モジュールを実行し, その計算結果をマスターノードでは参照することで Decide 結果を得る. よって Decide 結果の取得には, それぞれのスレーブノードにおける FSD モジュールの実行速度及びマスターノード · スレーブノード間の通信速度が影響すると考えられる.

そこで, 比較対象とサーバグループ D, E におけるスレーブノードの Find 結果の要素数の比較表を表 11 に示す.

表 11　Find 結果の要素数の比較

バイト数 (KB)	比較対象	サーバグループ D	サーバグループ E
80	6614	3413	3282
150	5571	3281	2285
220	14463	9279	5940

表 11 より, それぞれのスレーブにおける Find 結果の要素数が比較対象のものよりも小さくなっていることが分かる. よって, 分散構成のそれぞれのスレーブにおける FSD モジュールの実行速度が, 比較対象のものよりも速くなっていることが分かる. これにより, マスターノード · スレーブノード間の通信速度が影響したとしても, マスターノードにおいてスレーブノードから Decide 結果の取得にかかる実行速度が, 比較対象において Decide 結果を取得する実行速度よりも速くなり, 表 10 における Decide 結果の取得によるオーバーヘッドが負の値を取っていると考える. よって, Decide 結果の統合によるオーバーヘッドがあったとしても, 表 9 のように, 分散構成におけるそれぞれの 1 リクエスト当たりのアタッチ実行速度が, 比較対象のものと同等となったと考える.

7. おわりに

本研究では, 既存の統合インデックス方式を部分インデックス方式に変更することで, 1 台のサーバの物理メモリ容量の上限に対応する設計方式を実現した. また, 部分インデックス方式におけるエントリ集合の部分集合の作成方式として, WIX ファイルのコンテンツ分散を提案した. その結果として Find インデックスの局所化を実現した. さらに部分インデックス方式による Find インデックスを構築した WIX アタッチエンジンサーバを複製することで, サーバグループを定義した. サーバグループ内で更にサーバをスケールアウトすることにより, アタッチエンジンのスループット向上を目的として, 負荷分散処理を実現した.

また, 部分インデックス方式及び負荷分散処理を実装した WIX アタッチエンジンの分散構成におけるサーバサイド処理として, アタッチ処理である FSDR 処理の分散化を実現した.

しかし, WIX ファイルのコンテンツ分散の方式によっては 1 つのサーバグループではユーザのアタッチ要求に応えられない場合が考えられる. そこで, Decide 結果の統合を実装することで, WIX ファイルのコンテンツ分散の方式に関わらずユーザのアタッチ要求に応えられる分散構成を実現した.

評価実験では, 分散構成におけるアタッチ処理に対するオーバーヘッドが小さいことを示した. これにより, 統合インデックス方式による WIX アタッチエンジンの 1 リクエスト当たりのアタッチ実行速度と同等のものとなる WIX アタッチエンジンの分散構成が実現されたと言える.

文　献

[1] 林　昌弘, 青山　峻, 朱　成敏, 遠山　元道. Keio WIX システム (1) ユーザインターフェース. データ工学ワークショップ, DEIM2010. 2010.

[2] 森　良介, 藪　達也, 朱　成敏, 遠山　元道. Keio WIX システム (2) サーバーサイド実装. データ工学ワークショップ, DEIM2010. 2010.

[3] 市東　隼人, 分部　亮太, 朱　成敏, 遠山　元道. Keio WIX システム (3) コンテンツ作成. データ工学ワークショップ, DEIM2010. 2010.