RO-002

スレッドローカル変数を用いた Web アプリケーションのマルチテナント化方式の開発と仮想化方式との比較

Development of a Technique for Enabling Multi-Tenant Web Applications Using Thread Local Variables and Comparison with Virtualization

乾 敦行† 高野 英樹† 秋藤 俊介†
Atsuyuki Inui Hideki Takano Shunsuke Akifuji

1. はじめに

SaaS（Software as a Service）はソフトウェア機能をインターネット上で提供するサービスである [1]．SaaS の利用顧客は自社内でシステムを持つ必要がなく，低コストですぐに利用できる利点があり，近年需要が高まっている．一方 SaaS ベンダは，これらの利点を顧客に提供するための工夫としてマルチテナント化が必須である [2]．マルチテナント化とは，複数のユーザ企業（テナント）がシステムを共有し，ハードウェアリソースを有効活用することである．

従来マルチテナントアプリケーションは新規に開発する必要があり，開発工数がかかるという課題がある．我々は既存の Web アプリケーションをソースコードの修正なしにマルチテナント化する方式を開発した．これにより，既存のパッケージソフトウェアを持つ企業にとっては，SaaS 事業を始めるコストが下がることが期待できる．また，既存の Web アプリケーションをマルチテナント化するためには，仮想化によってインスタンスを複数用意する（マルチインスタンス）方法もある．本研究では仮想化によるマルチテナントシステムと提案方式との性能比較を行った．

本論文の構成は以下の通りである．第 2 章でマルチテナント化方式の概要について述べる．第 3 章で本方式の評価を行う．第 4 章で関連研究について述べたあと，第 5 章でまとめを行う．

2. マルチテナント化方式の概要

マルチテナント化対象の Web アプリケーションは，企業のシステムで良く利用されている，Servlet 形式で，JDBC でデータベースにアクセスするものとした．

2.1. データの管理方式

ここでは，アプリケーションが使う複数のテーブルをまとめたものをデータベースと呼ぶ．Web アプリケーションはアプリケーションごとにデータベースを 1 つ指定する（図 2.1）．マルチテナントにおけるデータの管理方式は大きく 3 つある．

1. データベースをテナントごとに分ける
2. テーブルをテナントごとに設ける
3. テナントを区別するカラムをテーブルに追加する

我々はカスタマイズや管理（バックアップなど）のし易さを考えて 2 の方式を選択した．具体的には，テナント名を付加したテーブルをテナントごとに用意する．例えばオリジナルのアプリケーションにテーブル A，B があり，テナントは X，Y があるとき，X_A，X_B，Y_A，Y_B というテーブルを用意する．

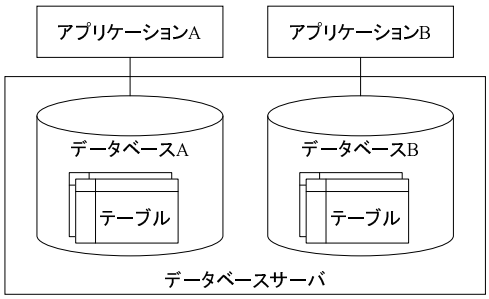

図 2.1 データベースとテーブルの概念

2.2. マルチテナント方式の構成

図 2.2 に本マルチテナント化方式の構成を示す．

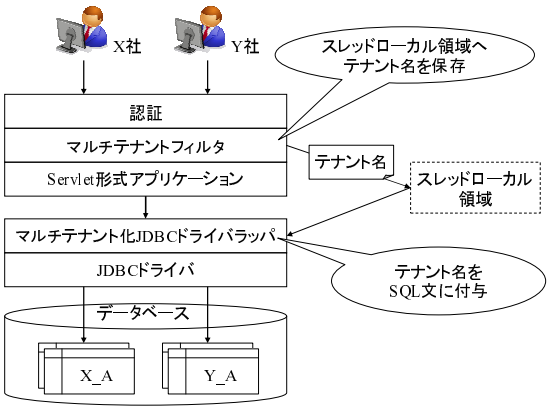

図 2.2 マルチテナント化方式の構成

本方式ではテナントごとのテーブル（図 2.2 の X_A，Y_A）を用意し，「マルチテナントフィルタ」と「マルチテナント JDBC ドライバラッパ」をアプリケーションに組み込む．マルチテナントフィルタがリクエスト元のテナント名を特定し，マルチテナント JDBC ドライバラッパがテナントに応じてアクセスするテーブルを切り替える．

マルチテナントフィルタが特定したテナント名を JDBC ドライバラッパが取得する必要があるが，既存のソースコード（Servlet アプリケーション）を修正しないことを目標としているため，このままではマルチテナントフィルタから Servlet アプリケーション，Servlet アプリケーションから JDBC ドライバラッパへテナント名を引き渡すことができない．

†株式会社 日立製作所 システム開発研究所

そこで，Servlet アプリケーションに対する 1 つのリクエストは 1 つのスレッドで処理されることに注目し，2 つの異なるプログラムであるマルチテナントフィルタと JDBC ドライバラッパ間でテナント名を共有するためにスレッドローカル変数を用いた．

スレッドローカル領域へのアクセスは Java 標準 API の ThreadLocal でできる．ThreadLocal は Thread オブジェクトから値へのマップを保持しており，値の設定と取得はそれぞれ set メソッドと get メソッドで行う．

以降，スレッドローカル変数を用いたマルチテナントフィルタ，マルチテナント化 JDBC ドライバラッパの詳細を説明する．

2.3. マルチテナントフィルタの設計と実装

マルチテナントフィルタはリクエスト元のテナント名を特定し，その情報を JDBC ドライバラッパに渡すためにリクエストスレッドにテナントを設定する．マルチテナントフィルタは次の情報を入力として受け取る．

1. リクエストパラメータ
2. セッション情報
3. リクエストヘッダ
4. Form 認証のフィールド名

これらのうち 1〜3 は Servlet アプリケーションがマルチテナントフィルタに渡す．4 の Form 認証のフィールド名はアプリケーションの設置者がアプリケーションの外から設定情報として与える．マルチテナントフィルタは Basic 認証と Form 認証に対応するため，図 2.3 のフローチャートにしたがって処理を行う．リクエストを受け付けたとき，セッションが確立されているかどうかを調べて認証済みのユーザからのリクエストかどうかを判断する．認証済みのユーザからのリクエストであった場合は，ユーザ名をセッション情報から取得し，リクエストスレッドへ設定する．セッションが確立されておらず，認証されていないユーザからのリクエストであった場合，Form 認証，Basic 認証の順でユーザ名を取得することを試みる．ユーザ名を取得できればセッションとリクエストスレッドにユーザ名をセットする．

ログインユーザ名にはテナント名を付加しておく．例えば X 社の aaa というユーザのログインユーザ名は aaa@X などとする．

セッションを用いる理由は，Form 認証の場合ログイン時にしかアプリケーションに認証情報が渡ってこないためである．

マルチテナントフィルタはアプリケーションサーバのフィルタ機能を用いて実装した．javax.servlet.Filter を implements して Filter クラスを作成する．実装しなければならないメソッドは表 2.1 の通りである．

表 2.1　Filter クラスで実装するメソッド

#	型	メソッド名
1	void	destroy()
2	void	doFilter(ServletRequest, ServletResponse, FilterChain)
3	void	init(FilterConfig config)

Form 認証のフィールド名を設定するために，アプリケーションサーバの設定ファイル web.xml ファイルに図 2.4 のように記述することにした．これは Form 認証のフィールド名が nickname であることを表している．

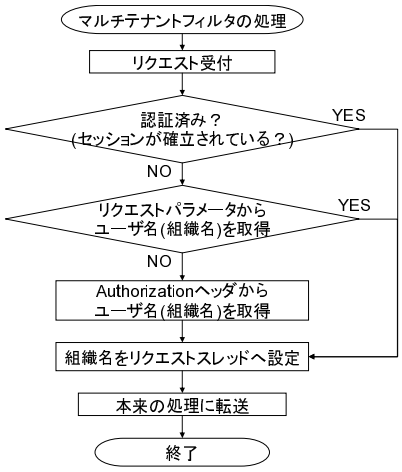

図 2.3　マルチテナントフィルタの処理フロー

```
<context-param>
  <param-name>Username</param-name>
  <param-value>nickname</param-value>
</context-param>
```

図 2.4　web.xml の設定

ここで指定した値は ServletContext クラスの getInitParameter メソッドで取得できる．Filter 内で ServletContex のオブジェクトを取得するには，init メソッド内で FilterConfig クラスの getServletContext メソッドを呼ぶ．これをインスタンスフィールドに保持しておけば，doFilter 内から ServletContex のオブジェクトを参照できる．

Basic 認証の Authorization ヘッダを解析するには Base64 でデコードする必要があるが，我々は sun.misc パッケージの BASE64Decoder クラスを用いた．セッション情報にアクセスするには javax.servlet.http パッケージの HttpSession クラスを用いる．このオブジェクトは doFilter メソッド内で以下のようにして取得できる．

```
HttpSession session =
  ((HttpServletRequest)request).getSession();
```

このマルチテナントフィルタを有効にするには，アプリケーションサーバの設定ファイル web.xml に図 2.5 の記述を追加する．

```
<filter>
  <filter-name>Kate Filter</filter-name>
  <filter-class>kate.Filter</filter-
class>
</filter>
<filter-mapping>
  <filter-name>Kate Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

図 2.5　マルチテナントフィルタの設定

2.4 JDBC ドライバラッパの設計と実装

JDBC ドライバラッパは Servlet アプリケーションから受け取った SQL 文のテーブル名にテナント名を付加し，本来の JDBC ドライバに渡すことを行う．

Web アプリケーションの設置者はアプリケーションが使用するデータベースを設定することができる．具体的には JDBC の java.sql.Driver クラスを指定することになる

ため，本マルチテナント化方式を使用するにはマルチテナント化 JDBC ドライバラッパクラスを指定する．

マルチテナント化 JDBC ドライバラッパクラスが行うのは，SQL 文にテナント名を付加する機能を持つ java.sql.Statement オブジェクトを返すことである．java.sql.Driver インタフェースは表 2.2 のメソッドを定義している．

表 2.2　Driver インタフェースのメソッド

#	型	メソッド名
1	boolean	acceptsURL(String url)
2	Connection	connect(String url, Properties info)
3	int	getMajorVersion()
4	int	getMinorVersion()
5	DriverPropertyInfo[]	getPropertyInfo(String url, Properties info)
6	Boolean	jdbcCompliant()

我々が注目するのは 1 の acceptsURL と 2 の connect である．acceptsURL は指定された URL に接続できるかどうかを判断するメソッドであり，マルチテナント化 JDBC ドライバラッパを示す URL が渡されたときに true を返すことにする．また connect メソッドは指定された URL に対応する java.sql.Connection オブジェクトを返す．ここではラップした java.sql.Connection オブジェクトを返す．

SQL 文を実行するのは java.sql.Statement クラスであるため，java.sql.Statement インタフェースを実装したクラスを実装する．ほとんどのメソッドは本来の JDBC ドライバの Statement オブジェクトに委譲し，execute メソッドをオーバーライドする．

テナント名をテーブル名に付加するには SQL 文を構文解析し，テーブル名に相当する部分を見つける必要がある．本研究では実装を簡単にするため，図 2.6 のフローチャートに示す簡易アルゴリズムを用いた．この簡易アルゴリズムで，3 つのオープンソース Web アプリケーションをマルチテナント化することができた．

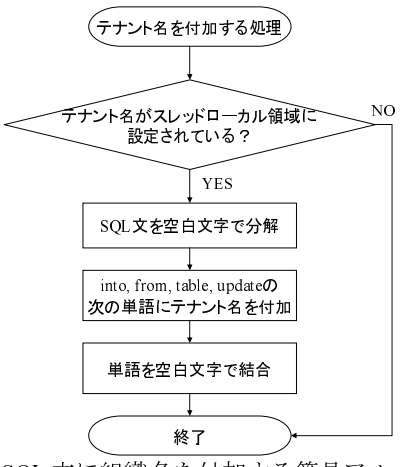

図 2.6　SQL 文に組織名を付加する簡易アルゴリズム

☆Apache Software Foundation の商標または登録商標
*米国 Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標

3．マルチテナント化方式の評価

オープンソースの Web アプリケーションをマルチテナント化することを試みた．実装に用いたソフトウェアは表 3.1 の通りである．

表 3.1　プロトタイプの実装に用いた環境

#	項目	ソフトウェア
1	アプリケーションサーバ	Tomcat☆ 5.5
2	データベース	MySQL* 5.0

マルチテナント化できたのは Eberom[3]，Hipergate[4]，Cream[5]である．各アプリケーションのテーブル数，プログラムの規模，認証方式は表 3.2 の通りである．改造コストはいずれもアプリケーションサーバの設定ファイルを 10～20 行変更するだけであった．

表 3.2　マルチテナント化できたアプリケーション

#	名前	テーブル数	規模	認証
1	Eberom	36	170K ステップ	Basic
2	Hipergate	200	10K ステップ	Form
3	Cream	58	200K ステップ	Form

次にマルチテナント化した Eberom を用いて，マルチテナント化後の性能測定をした．性能測定に用いたマシン環境は表 3.3 の通りである．測定ツールには JMeter[6]を用いた．

表 3.3　性能測定に用いたマシン環境

#	項目	サーバ	クライアント
1	CPU	Intel Xeon 2.4GHz	Intel Xeon 2.4GHz
3	メモリ	16GB	16GB
4	HDD	500GB	500GB
5	OS	CentOS5.4	Windows 2003 Server
6	Java	JDK 1.6.0_18	JDK 1.6.0_16

測定対象のマルチテナント化方式は，本方式と仮想化方式である．仮想化によるマルチテナント方式では，仮想化ソフトは Xen[7]を用い，1 テナントにつき仮想マシン 1 つを起動する．仮想マシン 1 台につき 1GB のメモリを割り当てた．

本方式によるマルチテナント化の測定方法は図 3.1 の通りである．

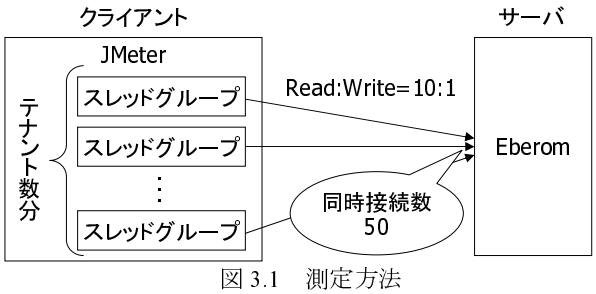

図 3.1　測定方法

1 スレッドグループあたり Read:Write = 10:1 のリクエスト送信を 5,000 回繰り返す．テナント数を 1，10，25，50 と変化させ，同時接続数が 50 になるように 1 スレッドグループが出すリクエスト数を設定し，スループットとレ

スポンスタイムを計測した．テナント数が変わっても，トータルの同時接続数は同じである．

仮想化方式によるマルチテナント化の測定方法は，クライアント側は図 3.1 と同様で，サーバ側はテナントの数だけアプリケーションサーバが存在する．各スレッドグループが各アプリケーションサーバに対して 1 対 1 でリクエストを送信する．

測定結果を図 3.2，図 3.3 に示す．図 3.2 はテナント数ごとのスループット比較を表し，図 3.3 はテナント数ごとのレスポンスタイムを比較したものである．

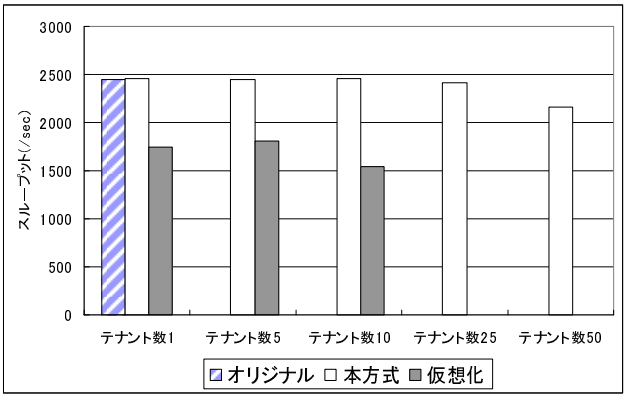

図 3.2　テナント数ごとのスループット比較

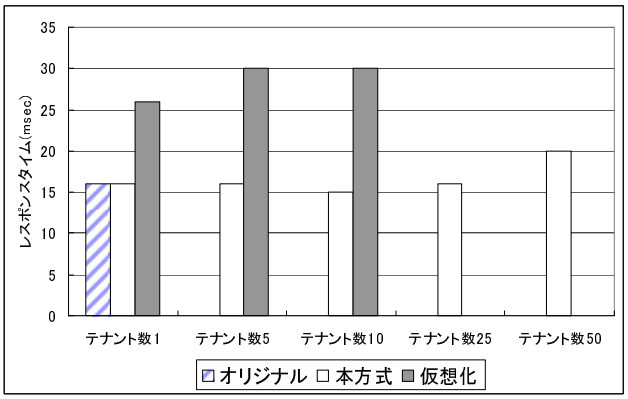

図 3.3　テナント数ごとのレスポンスタイム比較

図 3.2 より，仮想化方式ではオリジナルと比べてスループットが最大 37%低下したのに対し，本方式でのマルチテナント化後の性能劣化は，オリジナルと比べてテナント数が 25 のときは 1%，テナント数が 50 のときは 12%となった．

図 3.3 より，仮想化方式ではオリジナルと比べてレスポンスタイムが最大 87%遅くなったのに対し，本方式ではテナント数 25 までは遅れはなく，テナント数 50 のときで 25%の遅れとなった．

仮想化方式では PC の動作が重くなり仮想マシンを 25 個以上起動することができなかったため，テナント数 25 以上のケースは試せなかった．仮想化方式が遅くなる理由は，仮想マシンの切り替え，ディスク IO，仮想マシンが動作すること自体のオーバヘッドが挙げられる．

以上から，CPU 性能やメモリ容量が許せば，仮想化方式は改造コストがゼロである利点があるが，テナント数が増えれば，性能劣化しない本方式が有利である．

４．関連研究

Grails[8]は Groovy（Java 上で動くスクリプト言語）で実装された Web アプリケーションフレームワークである．Hibernate（OR マッパ）や Spring（Web アプリケーションフレームワーク）など既存の Java 技術を使用している．Grails 向けのマルチテナント化プラグインが Web 上で公開されている．本方式との違いはデータの管理方法である．テナントごとにテーブルは分けず，tenantID カラムを追加することで区別する．Hiberntae に割り込みを入れ，データベースにリクエストを出す前に検索条件に tenantID を追加する．Tenant Resolver クラスがどのテナントからのリクエストであるかを解決する．テナントごとにテーブルを用意する必要がないという利点があるが，テナントごとにスキーマをカスタマイズすることができない．

文献 2)はマルチテナント技術の特徴と考慮点を分類，整理し，新規にマルチテナントシステムを開発した事例を紹介している．Java の事例では 1 つのデータベースにテナントごとのテーブルを用意しており，本方式と同じ管理方法である．

Salesform.com[9]のマルチテナントデータベースはすべてのデータを 1 つのテーブルに格納する特殊な方式であり，アプリケーションも新規に作る必要がある [11]．一方，本方式はテナントごとのテーブルを用意すれば既存アプリケーションのソースコードを修正する必要はないという利点がある．

５．まとめ

スレッドローカル変数を利用して，ソースコードの修正が不要な Web アプリケーションのマルチテナント化方式を開発した．本方式はマルチテナント化フィルタと JDBC ドライバラッパをアプリケーションに組み込む．改造コストはアプリケーションサーバの設定ファイルを変更するだけである．

性能評価したところ，スループットの低下は 12%を達成した．仮想化方式との比較を行ったところ，テナント数が 10 以上に増えた場合，仮想化と比較して本方式が性能面で有効であることが分かった．

参考文献

[1] 河合輝欣，児西清義，米村征洋: ASP・SaaS の動向と普及促進の状況（前編），情報処理，Vol.49，No.11，pp.1325-1333(2008).

[2] 中村 誠吾，田中 要: マルチテナント型アプリケーション開発の実装考慮点，UNISYS TECHNOLOGY REVIEW，第 103 号，pp.91-101(2010)

[3] Eberom: http://sourceforge.net/projects/eberom/

[4] Hipergate: http://www.hipergate.org/

[5] Cream: http://www.campware.org/en/camp/cream_news/

[6] JMeter: http://jakarta.apache.org/jmeter/

[7] Xen: http://www.xen.org/

[8] Grails: http://www.grails.org/

[9] Weissman, C.D. and Bobrowski, S.:The Design of the Force.com Multitenant Internet Application Development Platform, ACM Proceedings of the 35th SIGMOD international conference on Management of data(SIGMOD), ACM, pp.889-896(2009).