社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－MBL－36（51）
2006－UBI－10（51）
2006／2／17

マスターレスなDB同期システムとシームレスな動画視聴システムへの応用

荒木拓也[†]　小倉章嗣[†]　登内敏夫[†]

近年，一人のユーザが複数の機器を利用する状況が見られるようになるにつれ，複数機器をまたいでプログラム・サービスを継続して利用したいという要件が顕在化してきている．一方，機器単体の性能向上に伴い，機器内でデータベースを利用することが現実的となりつつある．これにより，アプリケーションにおけるデータ管理が容易となる．このような状況を踏まえ，本研究ではデータベースの内容を機器間で同期することで，サービスの継続利用を実現するアーキテクチャを提案する．本論文では，これを実現するためのシステムとして，DB 同期システムを試作した．本 DB 同期システムは，マスターレス（全機器が対等）であるという特徴を持つ．これにより，家庭内などマスターとなる機器を仮定しにくい状況での応用が可能となる．また，本 DB 同期システムを動画視聴システムに応用した．本動画視聴システムでは，据置機器とポータブル機器が存在する状況において，ポータブル機器をネットワークから切断・再接続することにより，動画の再生位置を据置機器とポータブル機器間でシームレスに移動する．これにより，本手法の有効性を確認した．

A masterless DB synchronization system and its application to a seamless video watching system

Takuya Araki,[†] Shoji Ogura[†] and Toshio Tonouchi[†]

Today, it is becoming common for one person to use multiple computing devices, which is revealing the requirement of using one program or service across to the devices continuously. On the other hand, rise of device performance makes it feasible to use a database within the devices, which makes it easy for application programs to manage their data. Based on these situations, we propose an architecture that enables continuous service usage by synchronizing the contents of database. In this paper, we made prototype of database synchronization system to enable this; this system has characteristics of masterless (all devices are equivalent) synchronization. This enables application of home environment where it is difficult to assume the existence of a master devices. In addition, we applied this DB synchronization system to a video watching system. There are stationary devices and a portable device within this system; by disconnection and reconnecting the portable device from/to the network, the device where the video is played is seamlessly changed. This system proved the effectiveness of the proposing architecture.

1. はじめに

近年，一人のユーザが複数の機器を利用する状況が見られるようになるにつれ，複数機器をまたいでプログラム・サービスを継続して利用したいという要件が顕在化してきている．これを実現するために，Web のようなサーバクライアントアーキテクチャを用いて継続するための状態をサーバ側に持たせる手法も考えられる．しかし，機器を持ち歩く場合には，ネットワーク接続が切断，あるいは帯域幅が極端に制限される可能性があるため，対応できない．

一方，機器単体の性能向上に伴い，機器内でデータベースを利用することが現実的となりつつある．これにより，アプリケーションにおけるデータ管理が容易となる．

このような状況をふまえた上で，我々は，

- プログラムはアプリケーション情報をデータベースに記録する．この中には，他デバイスでの実行継続に必要な情報が含まれる
- システムが，データベースの情報を各機器で同期することにより，全機器での実行環境を同一にする．これにより，ネットワークから切断される場合でも，複数機器をまたいだ実行の継続を実現する．

というアーキテクチャを提案する．

本論文では，これを実現するためのシステムとして，DB 同期システムを試作した．ここで，マスターとなる機器を仮定した上で DB 同期を行うシステムは，従来から存在するが，ユビキタスな利用環境を想定すると受け入れにくい．これは，マスターとなる機器の電

† NEC インターネットシステム研究所
NEC Internet Systems Research Laboratories

源断，ネットワーク断，故障などにより，他の機器に影響が及ぶ他，外出先などでの端末同士の同期もできないためである．

このため，本論文では，マスターレス（全機器が対等）であるという特徴を持つ DB 同期システムを提案する．これにより，家庭内などマスターとなる機器を仮定しにくい状況での応用が可能となる．

また，本 DB 同期システムを動画視聴システムに応用した．本動画視聴システムでは，据置機器とポータブル機器が存在する状況において，ポータブル機器をネットワークから切断・再接続することにより，動画の再生位置を据置機器とポータブル機器間でシームレスに移動する．

すなわち，ポータブル機器をネットワークから切断すると，据置機器で再生中の動画がポータブル機器で継続して再生され，ポータブル機器をネットワークに再接続すると，ポータブル機器で再生中の動画が据置機器で再生される．

これにより，DB 同期システムを用いてサービスを継続利用するという手法の有効性を確認した．

本論文の構成は以下の通りである．第 2 節で本論文で提案するマスターレス DB 同期システムについて説明する．次に第 3 節において，動画視聴システムへの応用について説明する．第 4 節において，試作したシステムの評価を行う．第 5 節で関連研究について述べた後，最後に第 6 節でまとめを行う．

2. マスターレス DB 同期システム

2.1 概　　要

前節で述べたとおり，アプリケーションは，他デバイスでの実行継続に必要な情報を含め，アプリケーションの実行に必要な情報をデータベースに記録するそして，システムは，データベースの情報を各機器で同期することにより，複数機器をまたいだ実行の継続を実現する．

ここに保持する情報は，アプリケーション情報（動画視聴なら，どのようなコンテンツがあるか，どこまで見たか，など）や，環境情報（端末の設定や，認証情報など）などが考えられる．

各機器はローカルにデータベースを持ち，その中に各機器で同じ情報を保持する．これにより，ネットワーク非接続時での実行が可能となる．

機器同士がネットワークで接続されている際に行われた更新は，直ちにすべての機器に伝播され，各機器が持つ情報が同期される．

機器がネットワークから外れている際でも，参照，更新は可能である．ネットワークから外れている際に行われた更新は，次にネットワークに接続された際に，各機器に伝播される．

以上の様子を図 1 に示す．

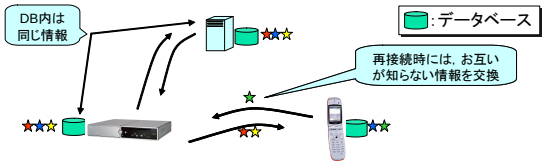

図 1　マスターレス DB 同期システムの概要

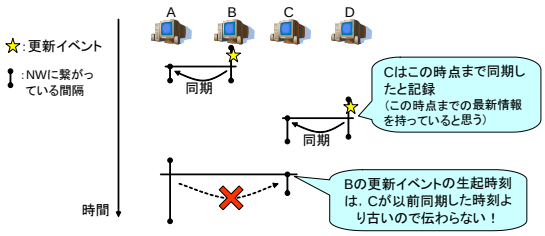

図 2　情報同期に失敗する例

本技術の特徴として，マスターとなるデータベースを持つ機器が不要であるということがあげられる．すなわち，すべての機器が対等の関係にある．これにより，特にホームでの応用を考えた場合，サーバとなるマシンを常時起動しておかなくてもすむ．また，外出先で端末同士で同期を行うということも可能になる．

次に，以上の機能をどのようにして実現するかについて説明する．

更新を接続中の機器に伝播することは，問題にはならない．更新が行われた行の情報を，各機器に個別に伝播するか，あるいはマルチキャストすればよい．問題は，ネットワーク非接続中に行われた更新について，差分情報のみをやり取りする方法である．

マスターとなるデータベースがあれば，例えば最後にマスターと同期した時刻を記録しておき，その時刻以降に行われた更新情報を交換すればよい．しかし，マスターの存在を仮定しない場合，情報がうまく伝播しない可能性がある．この様子を図 2 に示す．

マスターがある場合と同様に考えると，機器はいずれかの機器と同期を行った時刻を記録しておき，その時刻以降に起こった更新についてのみ，交換すればよいということになる．しかし，この例の場合，機器 C は D と同期を行っているが，C, D はそれ以前に B で起こった更新について情報を持っていない．その後 B で起こった更新について情報を持っている機器と接続したとしても，C は以前 D と同期を行った時点までの情報をすべて持っていると扱われるため，B で起こった更新についての情報が伝播されない．

この問題を解決するため，本研究では以下のような方式で情報の同期を行う．

まず，データベースの各行に，同期情報として，

- 更新を行った機器
- その機器で行った更新のシリアル番号

を付加する．この様子を図 3 に示す．

図 3 同期情報

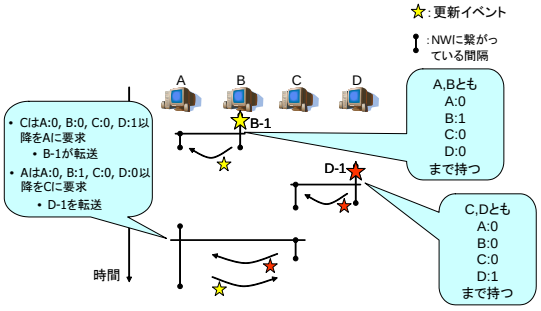

図 4 同期の様子

シリアル番号は各機器で独立であり，更新が起こるたびに 1 ずつインクリメントしたものが付与される．ネットワーク接続に伴い，データベースの同期を行う際には，

- 同期情報を検索し，更新した機器ごとに「シリアル番号何番のものまで持っている」，という情報を相手側機器に渡す
- 同期要求を受けた側は，同期情報を検索し，更新機器ごとに指定以降に起こった更新を渡す

という手順で情報の授受を行う．この様子を図 4 に示す．

この例では，まず機器 B においてシリアル番号 1 番の更新が行われた後，A と B で同期が行われる．この時点で，機器 A, B 共に,「A, C, D は 0 番，B は 1 番まで更新情報を持つ」という状態になる．その後同様に，D でシリアル番号 1 番の更新が行われた後 C と D で同期が行われ，共に「A, B, C は 0 番，D は 1 番まで更新情報を持つ」という状態になる．

その後，A と C が同期を行うと，C は「A, B, C は 0 番，D は 1 番以降の更新」を A に要求する．A は B の 1 番という更新を持っているので，これを C に転送する．同様に，A は「A, C, D は 0 番，B は 1 番以降の更新」を C に要求する．C は D の 1 番という更新を持っているので，これを A に転送する．

以上により，差分情報のみ転送することで，更新情報の同期を行うことができる．

2.2 実　　装

次に，本研究で行った実装について説明する．

本実装では，以下のソフトウェアを利用した．

- OS： Linux OS
- UPnP スタック：Linux UPnP SDK[1)]
  Intel から無償で提供されている UPnP スタック
- データベース：MySQL
  MySQL を用いた理由は，後述する MythTV が内部で MySQL を利用しているためである．より小型の関係データベースが必要であれば，例えば SQLite[2)] などの利用も考えられる

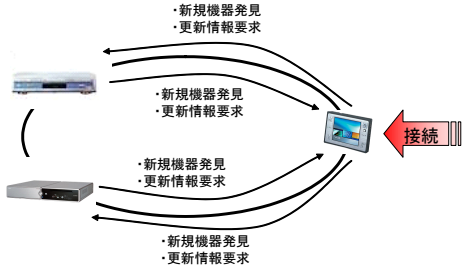

図 5 新規機器発見と情報更新

上記で述べたとおり，本ライブラリは UPnP を利用している．UPnP では，マルチキャストを用いて機器の接続や切断を認識したり，SOAP を用いて機器間で通信を行う機能がある．本ライブラリではこれらの機能を用い，以下のように動作する：

図 5 にネットワークに複数の機器がある状態で，ポータブル機器を接続した様子を示す．

今回の実装では，すでにネットワークに接続されているか，新たに接続された機器かにかかわらず,「新しい機器を発見したら，その機器に対して自分が知らない更新情報を要求する」というアルゴリズムで動作する．

この例では，すでにネットワークにあった機器は，UPnP によって新しい機器が接続されたことを認識する．よって，これらの機器は，新たに接続されたポータブル機器に対し，自分が知らない更新情報を要求する．

また，新しく接続されたポータブル機器は，ネットワーク上に 2 つの機器を新たに見つける．したがって，ポータブル機器は，これら 2 つの機器に対して更新情報を要求する．

この方法は，どの機器も同じアルゴリズムで動作するという点でシンプルであるが，通信効率の改善の余地は存在する．例えば，すでにネットワークに存在していた 2 つの機器は，同じデータを持っていると仮定できるので，ポータブル機器からの更新情報受信をマルチキャストにし，一度の通信を複数機器で受信することで，通信回数を削減することができる．また，ポータブル機器がすでにネットワークに存在している機器から更新情報を受け取る際も，複数の機器から別々に更新情報を受け取る必要は無く，一度受け取るだけで良い．しかし，今回の実装では単純さを優先し，このような実装とした．

次に更新情報送出要求の実現について説明する．まず，同期用データの構成について図 6 に示す．

この図では，左側がアプリケーションが通常使うテーブル，右側が同期用のデータとなっている．これらのテーブルの間の関連付けは，主キーによって行う（主

主キー	データ
1	9
2	7
3	11
4	3

主キー	更新者	イベント番号	更新時刻	削除
1	マシンA	12	…6:00	N
2	マシンA	11	…4:00	N
3	マシンC	5	…1:00	N
4	マシンB	7	…3:00	N

図 6 同期用データ

キーでなくても，一意のキーがあればよい）．このようにアプリケーションが使うデータと同期用のデータを別テーブルに分けることで，既存のアプリケーションでも容易に本システムに対応できるようにしている．

同期用データの中で，更新者およびイベント番号は，前節で述べたように用いられる．

更新時刻は，ネットワーク非接続時に複数の場所で更新された際，どちらのデータを優先するか判断するために用いられる（更新時刻が新しい方が優先される）.「削除」と書かれた列は削除されたテーブルかどうかを表すフラグである．行を削除する際，実際に削除してしまうと，まだ削除されていない機器と同期する際に，その行が削除されたのか挿入されたものなのか区別がつかない．そのため，行を削除する際は実際には削除せず，削除フラグを Y にすることで対応している（アプリケーションデータ側のテーブルでは実際に削除してもよい）．

この同期用データは，アプリケーションがアプリケーション用データを更新する際に同時に更新されなければならない．そのため，アプリケーションが SQL 文でデータベースをアップデートした直後に，ライブラリが提供する関数（propagate 関数）を呼ぶよう，アプリケーションを変更する必要がある．

propagate 関数は，引数に主キーを取り，対応する同期用データを適切に更新する．同時に，ネットワークに接続されている機器がある場合は，それらの機器に更新されたデータを伝播する．

このような同期用データの存在のもと，UPnP で定義されている「アクション」を利用して，更新情報要求を実現した．アクションの実体は SOAP 呼び出しであり，その引数に「更新した機器ごとに「シリアル番号何番のものまで持っている」という情報」を与える．動作の例を図 7 に示す．

この例では，マシン A とマシン B がお互いが持つ更新情報を交換する様子を示している．まずマシン B がマシン A に対し，更新情報を送信するように要求する．この際,「更新マシン毎に最も大きなイベント番号」を検索し，その情報をマシン A に伝える．この場合「マシン A については 11 番，マシン B については 8 番，マシン C については 5 番」という情報をマシン A に伝える．同期情報はデータベース内にあるため，この情報の検索は，データベースの機能を用いることで高速に行うことができる．

マシン A はこの情報を受け取り，それ以降に起こった更新情報を検索する．この場合，主キーが 3 の行がマシン C によって更新されており，そのイベント番

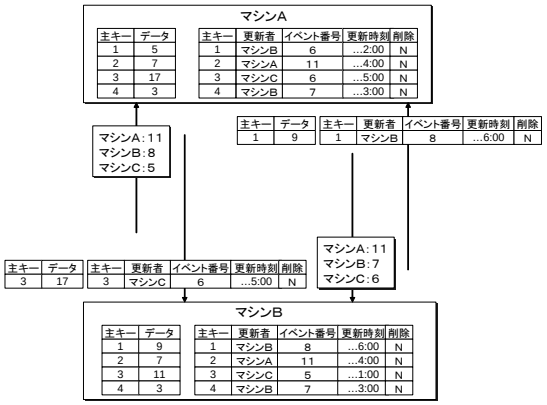

図 7 同期の例

号が 6 であるので，該当する．したがって，マシン A はこの行をマシン B に渡す．この情報の検索も関係データベースの機能を用いることで高速に行うことができる．

マシン A がマシン B に対して更新情報送信を要求する場合も同様である．

以上により，マシン間で更新情報のやり取りを行うことができる．

3. シームレスな動画視聴システムへの応用

次に，以上述べたマスターレス DB 同期ライブラリを MythTV に適用した，シームレスな動画視聴システムについて述べる．

まず，MythTV について説明する．MythTV は Linux 上で動作するフリーの HDD レコーダソフトである[3)]．MythTV ではアプリケーションデータの記録に MySQL を利用しているため，このデータをマスターレス DB 同期ライブラリを用い，機器間で同期することになる．

次に，第 1 節で述べた「シームレスなコンテンツ視聴」を実現するために，どのような情報を同期するかについて述べる☆

まず，ポータブル機器の取り外し時を考える．取り外した後，ポータブル機器では，据置機器（リビングにある機器など）で視聴中のコンテンツを続きから再生する必要がある．取り外しは予告無く行われるため，同期のタイミングはそれ以前に行わざるを得ない．したがって，

- 視聴中のプログラム
- 視聴位置
- 視聴状態（再生，早送り，巻き戻しなど）
- 視聴状態変更時刻

を同期情報とし，再生開始，停止，早送り等の操作が

☆ 今回の実装では，単純化のため，再生に必要となる動画ファイルは各マシンローカルに持つものとした．

行われた際に DB の内容を更新，各機器と同期する．

ポータブル機器側では，視聴状態変更時刻と現時刻との差から，経過時間を知ることができる．これを用いて，現時点での視聴位置を計算し，再生を開始することができる．(早送り時や巻き戻し中に取り外した場合，ポータブル機器側は，早送りや巻き戻しを開始する．)

据置機器側では，ポータブル機器の切断を認識すると，再生を停止する．

再接続時も同様の方法で同期，再生の開始を行う．

今回の実装では MythTV が管理するデータベースを拡張し，上記の情報を記録するようにした．

また，本ユースケースにおいては，機器の接続，切断をシステムが認識する速度が重要となる．UPnP レベルでも機器の接続，切断は認識されるがこれは定期的に送出される “advertise” パケットを利用するもので，特に切断の認識にはタイムアウトを待つ必要があるため，やや遅い．そのため，本実装では高速に機器の接続，切断を認識するため，“ifplugd” というソフトウェアを同時に利用した[4]．このソフトウェアにより，ポータブル機器側では高速にネットワーク接続，切断を認識することが可能となる．

4. 評　　価

作成したシステムの評価を行った．

まず，マスターレス DB 同期ライブラリ単体での評価として，情報同期に要した時間を接続台数及び同期対象の行数を変更しながら評価した．この結果を，表 1 および表 2 に示す（単位：秒）．

これらの評価は，いずれも「初めて参加する機器を 1 台接続した際に要した時間」である．すなわち，3 台での評価では，ネットワーク上に同期済みの機器が 2 台ある状態で，未初期化の機器を 1 台，ネットワークに接続した際に同期に要した時間を表す．再接続時には表の「初期化」部分の時間はかからない．

また，ここで言う「接続」には OS によるネットワーク認識時間は含まれない．すなわち OS レベルでの接続が確立していることを前提とし，その後評価用プロセスを起動してからの経過時間となる（従って，前述の ifplugd は利用していない）．

評価に用いたマシンは，いずれも Red Hat Linux 9.0 を利用し，Athlon 2 GHz, 512 MB の RAM を持つマシンである．

この表において,「デバイス起動」は，UPnP SDK における UPnP デバイスの初期化のほか，advertise パケットの送出やデバイスのサーチ等が含まれる.「UPnP Subscription」は接続中の機器からデータ更新が起こったという情報を得るための，UPnP レベルでイベントの Subscription を行うのにかかった時間である.「DB 同期」における「UPnP 関連」は UPnP SDK 内で費

表 1　情報同期時間（2 台）

同期行数	1	5	10
DB 初期化	0.02	0.02	0.01
UPnP 初期化	0.03	0.06	0.06
デバイス起動	1.10	1.09	1.09
UPnP Subscription	0.39	0.25	0.44
DB 同期（UPnP 関連）	0.74	0.75	0.77
DB 同期（UPnP 以外）	0.01	0.01	0.01
合計	2.28	2.17	2.39

表 2　情報同期時間（3 台）

同期行数	1	5	10
DB 初期化	0.02	0.01	0.01
UPnP 初期化	0.06	0.06	0.06
デバイス起動	1.10	1.09	1.10
UPnP Subscription	0.09	0.07	0.06
DB 同期（UPnP 関連）	0.74	0.83	0.84
DB 同期（UPnP 以外）	0.02	0.03	0.04
合計	2.01	2.09	2.10

表 3　ポータブル機器切断・接続時の動画再生・停止までの時間

		切断時	接続時
ポータブル	ifplugd	1 秒	1.5〜4 秒
	MythTV	3 秒	0.5 秒
	合計	4 秒	2〜4 秒
据置	UPnP 機器検出	—	2〜3 秒
	DB 同期	—	0.4 秒
	MythTV	—	3 秒
	合計	4 秒	5.4〜6.4 秒

やされた時間を表し,「UPnP 関連以外」は，DB からの同期データの抽出,（UPnP で通信するための）XML へのエンコードとデコード，および DB への同期データの書き込みを表す．

接続機器が 2 台および 3 台の場合は性能にほとんど変化が見られない．また，同期対象行数が増えても性能に大きな影響は与えていないことが分かる．

また，同期に要した時間の主な部分は，UPnP SDK に由来するものであることが分かる（「デバイス起動」および「DB 同期」の「UPnP 関連」）．したがって，UPnP SDK として他のものを使う，あるいは利用中の UPnP SDK をチューニングすることなどで，性能向上を行う余地があるものと考えている．

つぎに，MythTV を用いたユースケースの評価を示す．評価に用いたシステムはライブラリ評価の際のものと同様である．ただし，今回は実際にシステムをネットワークから切り離すため，ポータブル機器側で “ifplugd” を用いている．また，UPnP レベルでも高速に機器発見を行うため，advertise パケット送出間隔を短く調整している．

表 3 にポータブル機器をネットワークから切断，ネットワークに再接続した際の，ポータブル機器および固定機器側の動画再生，停止までにかかった時間を示す．

切断時は，ポータブル機器側，固定機器側共に 4 秒

程度で動画の再生，停止が行われている．ポータブル機器側ではその時間の多くが MythTV の起動に費やされている（3 秒）．

接続時は，測定によりばらつきがあるが，ポータブル機器側では 2〜4 秒停止までにかかり，固定機器側では 5.4〜6.4 秒再生までにかかっている．再生までにより多くの時間がかかっている理由は，ポータブル機器が接続されたと認識するまでに 2〜3 秒かかっていること（UPnP レベルで認識している），また再生状況（動画ファイル名や視聴位置など）の同期に 0.4 秒かかっていることによる．やはり，MythTV の起動には 3 秒程度費やされている．

今後は，切断・接続の認識の高速化，および MythTV 起動の高速化を行うことにより，再生・停止までの時間を短縮することができると考えられる．

5. 関 連 研 究

まず，データベースの同期に関して議論する．マスターを利用するものとしてはさまざまな先行例があり，商品としても Oracle Lite や DB2 everywhere などが存在する．しかしマスターを利用しないものはあまり知られていない．

マスターを利用しない同期機構として，Microsoft による WINGMAN がある[5]．WINGMAN では同期情報として各行に「世代」番号を付加する．世代番号は，（同期時を含め）データが更新されると増やされる．ここで，世代番号はマシンにローカルである．すなわち，同じデータがマシン毎に異なる世代番号を持つことがありうる．（我々の手法では，同じデータは同期情報も含め，マシン間でも同じデータになる．）

各機器は，前に他の機器と同期した際の世代番号を，機器ごとに覚えておく．次に同期した際は，その世代番号以上の情報を授受する．

しかし，この手法では，無駄な転送が発生する可能性がある．すなわち，別のマシンを経由して間接的にすでに得られた情報であっても，直接授受したことがなければ，すべての情報が送られる．

その他の同期技術として，Bayou がある[6]．Bayou が用いているプロトコルは本研究とほぼ同様である．すなわち，各マシンごとにどこまでの更新情報を持っているということを伝え，その後に生起した更新情報を取得する．

しかし，Bayou では更新をログ形式で管理する．すなわち，自マシンが同期対象マシンで生起した更新情報をどこまで持っているかを知るなどのために，ログをサーチする必要があり，非効率である．本研究では，これら同期情報の管理もデータベースで行うため，効率的に行うことができる．

次に複数機器をまたいだプログラムの継続利用について議論する．この要件を対象とした研究は数多くなされている（例えば 7) で言及されているデスクトップテレポーティング）．実現手法にも多種の方法があり，Java などを仮定しプログラム自体をコピーするもの（モバイルエージェントなど）や，サーバとの接続が継続していることを仮定する方法などがある．

本論文で提案した手法は，アプリケーションにより同期すべきデータをデータベースに保存しなければならないが，同期すべきデータ量が少ないことや，言語を仮定しないこと，アプリケーション移動後のネットワーク接続が不要であることなどの利点がある．

6. お わ り に

本論文ではデータベースの内容を機器間で同期することで，サービスの継続利用を実現するアーキテクチャを提案し，そのための技術としてマスターレスな DB 同期技術について提案した．さらに本技術を試作した上，シームレスな動画視聴システムに適用し，その有効性を確認した．

今回提案手法を適用したアプリケーションは比較的単純なものである．今後の課題として，より複雑なアプリケーションに対し，本手法を適用することでその有効性を確認することがあげられる．

また，マスターレスな DB 同期システムは，本論文で述べた以外の分野にも適用可能であると考えられる．このような応用への適用も今後の課題としてあげられる．

参 考 文 献

1) SourceForge.net: UPnP SDK for Linux. http://sourceforge.net/projects/upnp/.
2) sqlite: SQLite home page. http://www.sqlite.org/.
3) Richards, I.: MythTV. http://www.mythtv.org/.
4) Poettering, L.: ifplugd 0.28. http://www.stud.uni-hamburg.de/users/lennart/projects/ifplugd/.
5) Hammond, B.: WINGMAN A Replication Service for Microsoft Access and Visual Basic. research.microsoft.com/∼gray/WingMan_Replicas.doc.
6) Petersen, K., Spreitzer, M. J., Terry, D. B., Theimer, M.M. and Demers, A.J.: Flexible Update Propagation for Weakly Consistent Replication, *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)* (1997).
7) 佐藤一郎：スマートスペースのプログラミング，情報処理学会論文誌， Vol.45, No.12, pp.2655–2665 (2004).