分散管理型Federated CMDBの提案と評価

北島　信哉† 　松原　正純† 　大塚　浩† 　関口　敦二† 　和田　裕二†

† 株式会社富士通研究所 クラウドコンピューティング研究センター　〒211–8588 川崎市中原区上小田中 4-1-1
E-mail: †{shinya,matz,hotsuka,sekia,wada}@labs.fujitsu.com

あらまし　近年，コンピュータの高性能化および仮想化技術の発展に伴い，クラウドコンピューティングの普及が進んでいる．パブリッククラウドで必要とされる大規模データセンタの運用管理にあたっては，爆発的に増加するICTリソースを構成管理データベースを用いて管理することになる．そこで本稿では，既存の構成管理データベースのスケーラビリティを向上するため，分散管理型統合構成管理データベースを提案する．提案方式を用いることで，分散管理に伴うオーバーヘッドを最小限に留めつつ，管理可能なICTリソース数を向上できる．また，性能評価により，提案方式の有効性を検証する．

キーワード　データベース，CMDB，分散管理，コンシステント・ハッシュ

Proposal and Evaluation of Distributed Federated CMDB

Shinya KITAJIMA†, Masazumi MATSUBARA†, Hiroshi OTSUKA†, Atsuji SEKIGUCHI†, and Yuji WADA†

† Research Center for Cloud Computing, Fujitsu Laboratories Limited
4-1-1 Kamikodanaka, Nakahara, Kawasaki, Kanagawa 211–8588, Japan
E-mail: †{shinya,matz,hotsuka,sekia,wada}@labs.fujitsu.com

Abstract　Recent development of computers and virtualization techniques has lead to spread the cloud computing. To operate huge data center for the cloud computing, operators must manage millions of ICT resources. However, the existing management systems are not scalable enough to manage increasing resources. In this paper, we propose a federated management database which manages them distributedly. This database can improve the scalability of the management system, while the overhead for distributed management is kept small. Moreover, we confirm the effectiveness of our proposal method by performance evaluation.

Key words　database, CMDB, distributed management system, consistent hashing

1. はじめに

近年，コンピュータの高性能化および仮想化技術の発展に伴い，クラウドコンピューティングの普及が進んでいる．中でもパブリッククラウドでは，大規模データセンタにハードウェアを集約し，その上に構築した仮想化環境をコンピューティング資源として利用することにより，利用者はハードウェアを個別に管理する場合と比較して運用コストを大幅に削減できる [1]．

一方，このような大規模データセンタを提供する運用管理者側では，データセンタの構成管理，運用開始後の仮想リソース管理，サーバ監視，障害対処といった作業が必要となるが，大規模なデータセンタでは管理すべきサーバだけでも数千台規模となる．それらのサーバ構成やネットワーク構成，ソフトウェア情報，サービス情報，資産情報等の各要素レベルでは数十万以上の規模となることが考えられる．

従来，これらの構成情報はそれぞれの管理業務に最適化されたミドルウェアで管理され，各ミドルウェアが固有の構成管理データベース（CMDB: Configuration Management Database）を用いて管理していた [7], [8]．各CMDBは保持データのフォーマットやアクセス方法が異なっているため，CMDB間の連携は人手に頼らざるを得ず，構成情報の管理そのものに膨大な時間を要するといった問題があった．そこで，複数のCMDBを仮想的に統合し，運用管理の効率を向上する統合CMDB（FCMDB: Federated CMDB）が提案されている [5], [11]．しかし，既存のFCMDBでは，登録できる構成情報数がスケーラブルでないという課題があった．

そこで本稿では，FCMDBを拡張し，複数のFCMDBを用いて構成情報を分散管理することで管理可能な構成情報数を向上するF2CMDB（Federated FCMDB）を提案する．

以下，2. でCMDBについて述べ，3. で提案手法について説

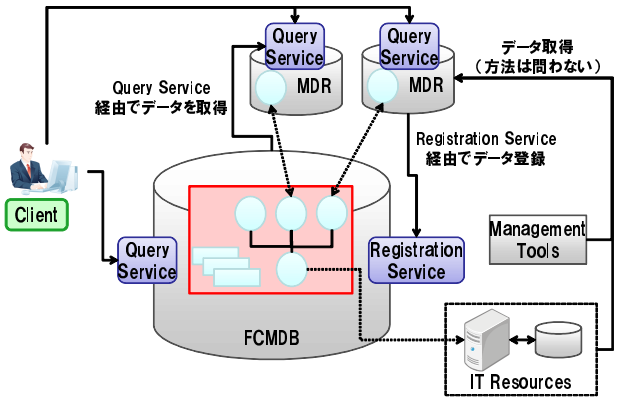

図 1　FCMDB
Fig. 1　FCMDB.

明する．4. で提案手法の性能評価を行い，最後に 6. で本稿のまとめと今後の課題について述べる．

2. CMDB

CMDB（Configuration Management Database）は，ICT システムの構成要素に関する様々な情報を保持する概念的なデータベースであり，ITIL フレームワークにおける構成管理プロセスの中枢部とされている [3]．ITIL（IT Infrastructure Library）とは，システム運用管理，IT サービス管理に関するベストプラクティス（成功事例）を集めた手引集である [7], [8]．

2.1 CI と Relationship

CMDB は，構成アイテム（CI: Configuration Item）と CI 間の関係情報を表す Relationship の 2 種類から構成されるデータベースである．CI は ICT システムの部品を表し，例えばサーバやストレージ，ソフトウェア等の詳細を表す．一方，Relationship は CI 間の関係を定義するものであり，例えば A というサーバを B という管理者が管理している，といった関係情報を表す．Relationship において，関係情報の関係元 CI と関係先 CI を，それぞれ source，target と呼ぶ．以降，CI と Relationship をあわせてエンティティと呼ぶ．各エンティティは一意に識別するための ID（エンティティID）をもつ．各エンティティは，サーバやスイッチ，ユーザといった種類ごとにタイプを定義でき，各タイプのエンティティがもつ属性を設定できる．ここで属性とは，例えばサーバであれば名前や IP アドレス，OS といった，そのエンティティに関する個々の情報を表している．属性の名前を属性名，属性の値を属性値と呼ぶ．つまり，Server タイプの CI における「IP アドレス」であれば，属性名が「IP アドレス」，属性値が「192.168.0.1」となる．

2.2 FCMDB

これまでのデータセンタの運用管理では，インフラやアプリケーション，業務など，レイヤーごとに異なる管理者が存在していたため，サーバ構成やネットワーク構成，ソフトウェア情報，サービス情報，資産情報等を固有の CMDB で管理していた．各 CMDB は保持データのフォーマットやアクセス方法が異なっているため，CMDB 間の連携は人手に頼らざるを得ず，人的ミスによる情報の記述漏れや不整合を招いたり，エンティティ管理そのものに膨大な時間を要するといった問題があった．そこで，DMTF 配下の「CMDB Federation Working Group」[11] において，図 1 に示すような統合 CMDB（FCMDB: Federated CMDB）が標準化され，現在は同組織によって策定された標準仕様 CMDBf バージョン 1.0.1 が公開されている．FCMDB を用いることで，複数の CMDB を仮想的に統合し，運用管理の効率を向上できる．

FCMDB では，クライアントからの検索要求に対し，複数の MDR（Management Data Repository）が保持するエンティティをマージし，結果を返す．MDR とは，サーバ構成やネットワーク構成，ソフトウェア情報，サービス情報，資産情報等のデータを保持するデータベースである．MDR 内のデータは各ミドルウェアによって管理されているため，物理的に同じものを表すエンティティが複数の MDR に登録されている場合がある．例えば，あるサーバの設計情報は設計情報 MDR に，実運用時点での情報は実態情報 MDR に，というように管理されている場合が考えられる．この場合，あるサーバが設計通りに構築されているかを調べたい場合，設計情報 MDR と実態情報 MDR にアクセスし，各 MDR の情報を付き合わせて比較する必要があるため，非常に煩雑である．

そこで，物理的に同一のものを表すエンティティを 1 つにマージすることで，この問題を解決できる．つまり，同じサーバに関する設計情報と実態情報をマージし，1 つのデータとしてデータベースに格納することで，容易に比較が行えるようになる．このマージ処理をリコンサイルと呼び，リコンサイルの際に用いる「同じ MAC アドレスをもつサーバは同一サーバと見なす」といったルールに含まれる属性を **Identifying Property（IPROP）**と呼ぶ．この例では「MAC アドレス」という属性が IPROP である．リコンサイル処理を行う際には複数のエンティティが 1 つにマージされるため，エンティティ ID は元となったいずれかのエンティティのものを用いる．

エンティティの登録は，MDR 経由で行う．つまり，各 MDR が新規に登録するエンティティのデータを各種ミドルウェア等を通じて取得し，取得したデータを FCMDB に登録する．このとき，MDR ごとに取得したデータ形式が異なる場合は，FCMDB に登録する際に統一形式にデータ変換を行い，FCMDB を利用する側からはデータ形式の違いを認識する必要がないようにする．

2.3 FCMDB の課題

FCMDB に格納可能なエンティティ数は，エンティティ・データの保持に用いるデータベースや記憶装置の容量に依存する．しかし，クラウドコンピューティング向けデータセンタの運用管理に FCMDB を用いることを考えた場合，FCMDB は膨大な数のエンティティを保持する必要がある．この問題を解決するためには，あらかじめ巨大なデータベースを用意しておく方法や，データを分散管理する方法が考えられる．

前者の場合，巨大なデータベース構築のための費用が非常に高価であったり，最終的に登録されるエンティティ数が予測しづらいため，データベース構築にどの程度のハードウェアリソースを割くべきか判断ができないといった問題がある．後者の場

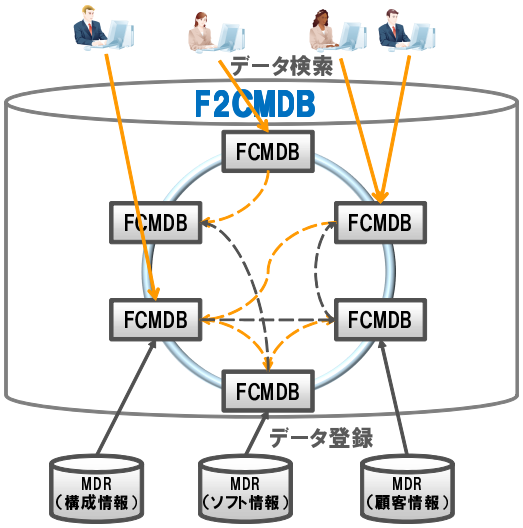

図 2　F2CMDB
Fig. 2　F2CMDB.

合，安価なデータベースサーバを必要なだけ用意すればよく，保持するエンティティ数の増加に応じて容易にハードウェアリソースを追加できるため，コスト面や柔軟性にメリットがある．また，高負荷時の負荷分散が期待できる，検索内容によっては並列に処理することで検索速度が向上できる，耐故障性を向上できる，といった利点も考えられる．しかし，ネットワーク経由でデータのやり取りを頻繁に行う必要があるため，登録や検索の際にオーバーヘッドが生じるという問題が懸念される．

本稿では，登録よりも検索のほうが圧倒的に多い環境で FCMDB を利用することを想定し，検索時間の短縮効果を見込んで後者の分散管理によるスケールアウトの方法を検討する．例えばこのような環境としては，サーバの状態やエラー情報，各サーバで稼動しているサービスの情報など，頻繁に閲覧する必要のある情報が FCMDB に格納されている場合が挙げられる．

3.　F2CMDB

本章では，提案する F2CMDB について説明する．F2CMDB は FCMDB を拡張し，エンティティの分散管理を可能としたものである．提案する F2CMDB の基本アーキテクチャを図 2 に示す．F2CMDB は，クライアントや MDR に対しては従来の FCMDB とまったく同じ振る舞いをするように見える．しかし，実際は複数の FCMDB が相互にネットワークを介して接続している．これら FCMDB 群は，いずれかがマスターとなるわけではなく，各々が協調して動作する．クライアントや MDR は，任意の FCMDB に接続できる．

3.1　データ登録

データ登録の際には，MDR から送られてきたデータに対して分散リコンサイルを行った後，コンシステント・ハッシュ法に従ってエンティティを保持する FCMDB を決定する．以下に，詳しく説明する．

3.1.1　コンシステント・ハッシュ法

F2CMDB では，データの登録先 FCMDB を決定する方法として，コンシステント・ハッシュ法 [6] を用いる．コンシステント・ハッシュ法を用いることで，効率的にデータ分散を実現できる．図 3 では，6 台の FCMDB から構成される F2CMDB をリング状のハッシュ値空間にマップした例を示している．各 FCMDB は，一意な ID（例えばホスト名）をもっており，各々の ID に基づいてハッシュ値空間上に配置される．同様に，エンティティもエンティティ ID に基づいて配置される．各 FCMDB は，1 つ前の FCMDB との間にマップされたエンティティを管理する．すなわち，これらのエンティティを自 FCMDB 配下の MDR で保持する．

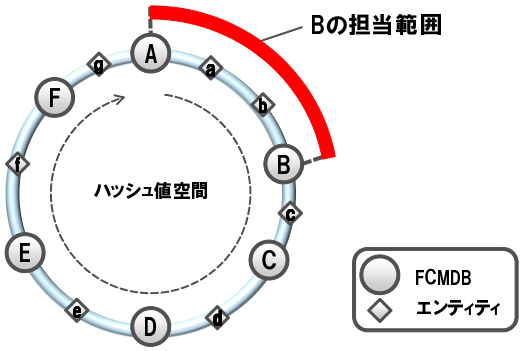

図 3　コンシステント・ハッシュ法
Fig. 3　Consistent hashing.

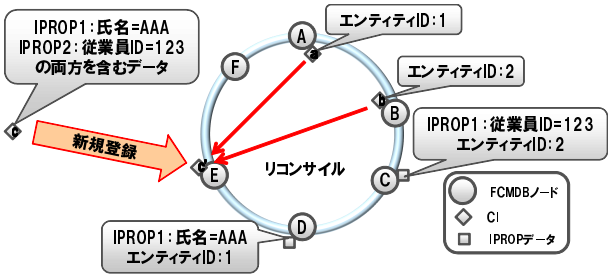

図 4　分散リコンサイル
Fig. 4　Distributed reconciliation.

3.1.2　データ登録処理

データ登録を行う際，MDR は F2CMDB に参加している任意の FCMDB に対してデータを送る．データを受け取った FCMDB（受付 FCMDB）は，3.1.3 で説明する分散リコンサイル処理を行う．このとき，当該データのエンティティ ID が同時に決定されるため，そのハッシュ値を計算する．各 FCMDB が管理するハッシュ値の区間はコンシステント・ハッシュ法により決定されているため，そのエンティティを管理する FCMDB（担当 FCMDB）が求まる．そこで，受付 FCMDB は担当 FCMDB にエンティティ・データを転送する．担当 FCMDB 側では，エンティティ ID のハッシュ値をキーに受け取ったエンティティ・データを格納する．既に同一エンティティ・データが登録されていた場合は，リコンサイルした後に格納する．

3.1.3　分散リコンサイル

登録するエンティティのうち，いずれかの IPROP の属性値が合致したエンティティ同士は同一のエンティティと見なし，リコンサイルを行う．リコンサイル処理を行う際，他の FCMDB

が管理する全エンティティを毎回参照して IPROP の属性値を確認すると，リコンサイル処理にかかる負荷が高くなってしまう．そこで，IPROP の情報もエンティティ・データと同様にコンシステント・ハッシュ法を用いて分散管理することで，参照すべき IPROP の情報を管理する FCMDB を一意に決定できるため，リコンサイル処理の負荷を軽くできる．この方法を分散リコンサイルと呼ぶことにする．

IPROP の情報をハッシュ空間上で管理するために，IPROP の属性名および属性値から生成したハッシュ値をキーとし，対応するエンティティID を値として，各 FCMDB が保持する．図 4 に，分散リコンサイルの概要を示す．データ登録の際には，登録するエンティティの IPROP の情報を参照し，IPROP の属性名と属性値からハッシュ値を生成する．そのハッシュ値を担当する FCMDB に問い合わせることで，リコンサイルすべきエンティティが存在するか，また，そのエンティティ・データをどの FCMDB が保持しているかが容易に判別できる．このとき，リコンサイルすべきエンティティが存在する場合には，対象となるエンティティID を取得し，そのエンティティ・データを管理する FCMDB 上でリコンサイルを行う．一方，リコンサイルすべきエンティティが存在しない場合には，一意なエンティティID を新たに発行し，エンティティを新規に登録する．

3.2 データ更新・削除

データ更新の場合，データ登録の場合と動作は同じである．更新された属性値が，すでに登録されている属性値に上書きされて登録される．一方，データ削除の場合，CI を削除する際に，その CI を source または target とする Relationship が存在する場合には，該当する Relationship も同時に削除する．

3.3 データ検索

データ検索は，エンティティID を指定してエンティティ・データを取得する ID 指定検索と，それ以外の通常検索で手順が異なる．

3.3.1 ID 指定検索

ID 指定検索の場合，次のような検索式がクライアントから送られてくる．

entity-id(‘Server001’)

上記の検索式は，エンティティID が ‘Server001’ であるエンティティを検索するものである．受付 FCMDB はエンティティID のハッシュ値をもとにクライアントが要求するエンティティ・データを保持する担当 FCMDB を求め，担当 FCMDB にクエリ式を転送してデータを取得し，クライアントに返す．

3.3.2 通常検索（XPath 検索）

通常検索におけるクエリ式は，次のような形式になる．

%Server[··· /@status=‘error’]
/&ManagedBy/%Person/··· /@tel

通常検索は，どの FCMDB に該当データが格納されているのか検索式を見ただけではわからない，という点が ID 指定検索とは異なる．よって，すべての FCMDB に対して問い合わせる必要があるが，各 FCMDB はエンティティ単位でデータを管理しているため，クライアントから送られてきた検索式のまま各 FCMDB で検索しても正しい結果を得ることはできない．そこで，受付 FCMDB において，検索式をエンティティ検索（CI または Relationship 取得）単位のサブクエリに分解する．そして，サブクエリを順に全 FCMDB で処理し，最後のサブクエリを実行した結果がクライアントに返すデータとなる．

表 1　評価に用いたサーバの性能
Table 1　Parameters.

パラメータ名	値
CPU	Intel® Xeon® 5130 2.0GHz
メモリ	3GB
OS	Microsoft® Windows Server™2003 R2

表 2　評価に用いたデータ
Table 2　Data.

エンティティ	タイプ	個数
CI	Server	500
	Storage	350
	Switch	100
	User	50
Relationship	ConnectedTo	500
	ManagedBy	500

4. 評　　価

F2CMDB の有効性を示すため，FCMDB，F2CMDB をそれぞれ実装し，比較評価を行った．評価指標は以下の 4 項目である．

- **登録上限数**

用意したエンティティを順に登録していき，それ以上エンティティを登録できなくなった際の登録エンティティ数．

- **登録時間**

登録リクエストを出してから，実際に登録処理が行われ，そのレスポンスがクライアントに返ってくるまでの時間の総和．

- **検索時間**

検索リクエストを出してから，実際に検索処理が行われ，その結果がクライアントに返ってくるまでの時間の総和．

- **検索スループット**

1 秒間あたりにクライアントに結果が返ってきた検索リクエスト数．

検索スループットに関しては，F2CMDB においてエンティティを分散管理することにより，検索スループットも向上するのではないかという副次的効果が期待されるため，測定を行った．

4.1 評 価 環 境

表 1 に，評価で用いたサーバの性能を示す．性能測定では同一のサーバを 9 台準備し，それらを同一ルータにギガビットイーサネットで接続した．データ登録の際は，データファイルからエンティティを読み込み，順次登録を行った．1 ファイルあたりには 2000 件のデータを含み，内訳は CI，Relationship それぞれ 1000 件ずつであった．用意したエンティティの 1 ファイルあたりの内訳を表 2 に示す．1 ファイルのサイズは約 1MB であり，1 エンティティのサイズは約 0.5KB だった．

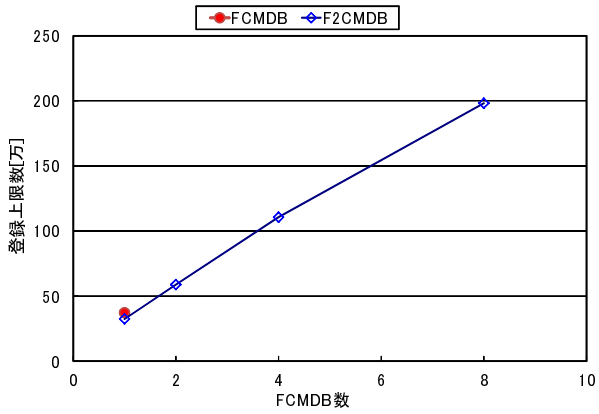

図 5　登 録 性 能

Fig. 5　Limit number of adding entities.

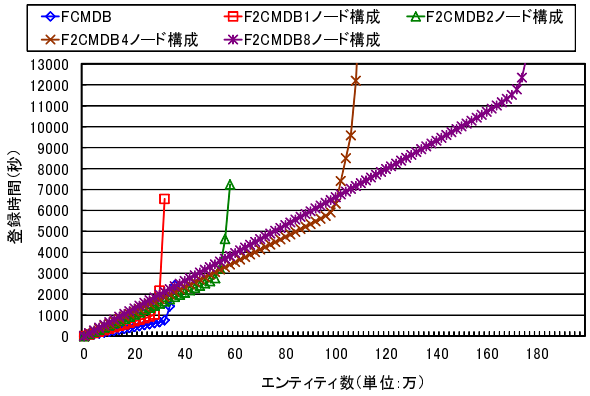

図 6　登 録 時 間

Fig. 6　Performance of adding entities.

評価にあたって，FCMDB，F2CMDB をそれぞれ実装した．実装に用いた言語は Java 1.5，Scala 2.8 [10] である．エンティティ等の保持には特定のデータベースは用いず，すべてメモリ上に保持するものとした．また，評価の際には F2CMDB を構成する各 FCMDB の保持データ数がほぼ均等となるようにした．これは，各 FCMDB がもつエンティティ数に偏りがある場合，性能に与える影響が顕著となるためである．

以降，n 個の FCMDB からなる F2CMDB を，F2CMDBn ノード構成と呼ぶ．F2CMDB を構成する際，FCMDB を複数台使用する場合には，各 FCMDB をそれぞれ異なるサーバに割り当てた．例えば，FCMDB4 ノード構成の場合，サーバを 4 台用い，各サーバを 1 つの FCMDB として利用した．また，各 FCMDB が管理する MDR は，FCMDB と同一サーバ上に構築した．

4.2　登録上限数

FCMDB と F2CMDB について，登録上限数の評価を行った結果を図 5 に示す．F2CMDB では，FCMDB 数を 1，2，4，8 と変化させた．

図 5 から，登録上限数は FCMDB と比べ，F2CMDB1 ノード構成の場合は 10%程度少ないものの，FCMDB 数が 2 以上の場合には増えていることがわかる．F2CMDB では，FCMDB には存在しなかった IPROP の情報を保持する必要があることから，1FCMDB あたりの登録上限数はやや少なくなる．しかし，F2CMDB 全体の登録上限数は，FCMDB 数にほぼ比例して増やすことができる．

4.3　登 録 時 間

FCMDB と F2CMDB について，登録時間の評価を行った結果を図 6 に示す．F2CMDB では，FCMDB 数を 1，2，4，8 と変化させた．グラフは，登録し終えたエンティティの総数を横軸に，登録開始からの経過時間を登録時間として縦軸にプロットしている．例えば，FCMDB8 ノード構成の場合，100 万エンティティ登録するまでに約 6000 秒かかったことを表している．

図 6 から，どの場合においても，あるところまでは登録時間は登録エンティティ数に比例して増加し，それ以降は急激に登録時間が増えている．これは，登録上限数に近づくにつれ，ガーベッジコレクションの発生頻度が増加し，それにともなって登録時間が延びていることを表している．

登録時間は FCMDB が最も短く，F2CMDB では FCMDB 数が増えるにしたがって登録時間が増加している．F2CMDB1 ノード構成では FCMDB 間通信によるオーバーヘッドは発生しないが，内部でのメッセージ通信によるオーバーヘッドのため，登録時間が増加している．また，F2CMDB2 ノード構成では，約半数のデータが受付 FCMDB にそのまま登録されるが，それ以外の半数のデータはネットワークを介して他の FCMDB に登録されるため，FCMDB 間通信によるオーバーヘッドが発生し，登録時間はさらに遅くなる．FCMDB ノード数が増えるほど，受付 FCMDB 以外にデータが登録される割合が増えるため，登録時間は遅くなっていく．登録時間については，複数の FCMDB で並列に登録処理を行うことで，単位時間あたりのデータ登録数は改善できるため，実運用上は問題ないと考えられる．

4.4　検 索 時 間

FCMDB と F2CMDB について，検索時間の評価を行った．検索式によって検索時間が大きく変化するため，想定環境において使用頻度が高いと考えられる以下の 3 つの検索式を用いた．

検索式 1：%User[#*/rc:User/@id =‘user*****’]
検索式 2：&GRelation[@source = ‘gid*****’]
検索式 3：entity-id(‘gid*****’)[#default]/
&GRelation/%User

検索式 1 は，全 User タイプの CI から，特定の ID をもつものを検索する式，検索式 2 は，全 Relationship から，source が特定の ID をもつエンティティであるものを検索する式，検索式 3 は，全エンティティから特定の ID をもつエンティティを検索した後，そのエンティティを source または target にもつ Relationship を検索し，さらにその Relationship の source もしくは target が User タイプの CI であるものを検索する式である．これらの検索式は XPath を独自に拡張したもので，本想定環境において必要となる複数の検索パターンにおいて，検索式を容易に記述できるように最適化している．

評価の際には，あらかじめ一定数のエンティティを登録した

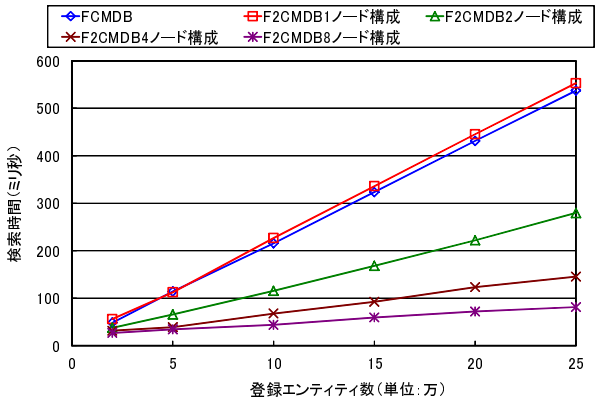

図 7　検索時間：検索式 1

Fig. 7　Performance of searching entities for query expression 1.

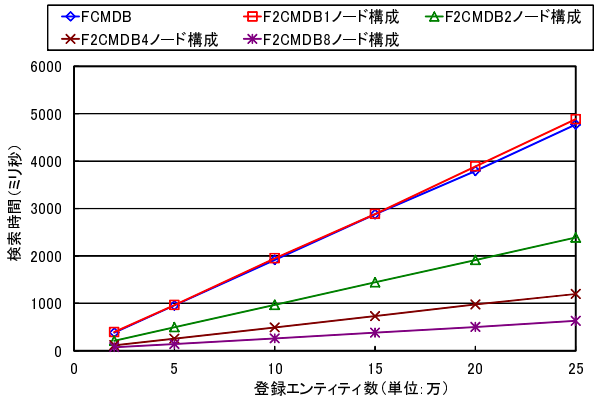

図 8　検索時間：検索式 2

Fig. 8　Performance of searching entities for query expression 2.

後，登録したエンティティに対して検索を行った．エンティティ数は，2 万，5 万，10 万，15 万，20 万，25 万と変化させた．また，F2CMDB では，FCMDB ノード数を 1，2，4，8 と変化させた．

4.4.1　検 索 式 1

検索式 1 を用いて検索時間について評価を行った結果を，図 7 に示す．図 7 から，すべての場合において，登録したエンティティ数が増えるほど，検索時間が長くなることがわかる．これは，検索式 1 の場合，User タイプの CI すべての中から検索を行うため，登録エンティティ数が増えるほど対象となる User タイプの CI が多くなるため，検索時間が長くなっている．

F2CMDB1 ノード構成の場合，検索時間は FCMDB より若干長くなっている．これは，登録性能の評価時と同様に，内部でのメッセージ通信によるオーバーヘッドのためである．F2CMDB2 ノード構成の場合，検索時間は F2CMDB1 ノード構成の約半分となっている．これは，F2CMDB2 ノード構成ではエンティティ・データが各 FCMDB ノードに分散して登録されるため，全探索を要する検索式の場合，1FCMDB あたりが探索するエンティティ数が約半数になるためである．つまり，検索時間は FCMDB 数に反比例し，短くなるといえる．

4.4.2　検 索 式 2

検索式 2 を用いて検索時間について評価を行った結果を，図 8 に示す．図 7 と図 8 から，検索式 2 を用いて検索した場合の検索時間の傾向は，検索式 1 を用いる場合と同様であり，登録エンティティ数に比例して検索時間が長くなり，FCMDB 数に反比例して検索時間が短くなるといえる．検索式 1，検索式 2 は，ともに一定数のエンティティの中から，特定の ID をもつエンティティを検索する検索式である．そのため，検索時間は同様の傾向を示したと考えられる．また，母数が多いほど検索にかかる時間は延びるため，1000 エンティティあたり 50 の User タイプの CI に対して検索を行う検索式 1 の場合と比べ，1000 エンティティあたり 500 の Relationship すべてに対して検索を行う検索式 2 の場合の方が，検索時間は長くなっている．

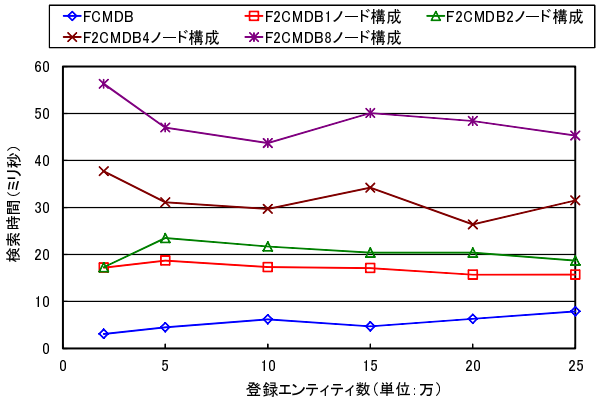

図 9　検索時間：検索式 3

Fig. 9　Performance of searching entities for query expression 3.

4.4.3　検 索 式 3

検索式 3 を用いて検索時間について評価を行った結果を，図 9 に示す．図 9 から，登録エンティティ数が増加しても，検索時間はほぼ一定であることがわかる．これは，検索式 3 では初めに特定の ID をもつエンティティを検索するが，FCMDB，F2CMDB では ID とそれに対応するエンティティをハッシュテーブルで管理していることから，このエンティティは常に一定時間で得られる．また，得られたエンティティには自身を source もしくは target としてもつ Relationship の ID が記述されているため，これらの Relationship も同様に一定時間内に取得できる．さらに，これらの Relationship には自身の source もしくは target となる CI の ID が記述されているため，同様に一定時間で取得できる．結果として，検索式 3 は登録エンティティ数にかかわらず，検索時間は一定になる．

また，FCMDB の場合が最も検索時間が短く，F2CMDB1 ノード構成では FCMDB と比べ，検索時間は 10 ミリ秒程度長くなっていることがわかる．F2CMDB では，内部でのメッセージ転送や IPROP 情報の管理など，FCMDB では必要ない処理が発生するため，この若干のオーバーヘッドにより検索時間が長くなったと考えられる．さらに，FCMDB 数が多いほど，検索時間は長くなっている．F2CMDB では FCMDB 数が多くなるほど，検索を受け付けた FCMDB が検索結果となるエンティティを保持している確率が低くなり，ネットワークを介して他の FCMDB からエンティティを取得する場合が多くなるためである．

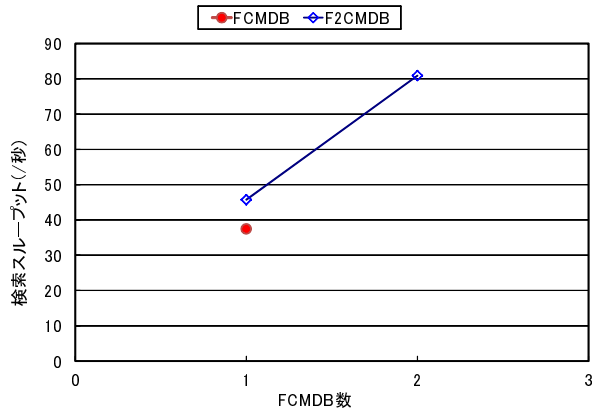

図 10　スループット測定：検索式 1

Fig. 10　Throughput of searching entities for query expression 1.

4.5　検索スループット

FCMDB と F2CMDB について，検索スループットの評価を行った．複数の検索リクエストを同時に受け付けた場合の影響を見るために，検索式は単純な検索式 1 を用いた．検索スループットを測定する際には，検索リクエストを送信するクライアントを 4 台用意した．各クライアントは 1 度検索リクエストを送信すると，検索結果を受け取るまで待機する．その後，検索リクエストを受け取ると，すぐに次の検索リクエストを送信する，という動作を繰り返した．また，F2CMDB では，FCMDB 数が複数の場合には，検索リクエストの送信先となる FCMDB はランダムに選択するものとした．

検索式 1 を用いて検索スループットについて評価を行った結果を，図 10 に示す．F2CMDB では，FCMDB 数を 1，2 と変化させた．

図 10 より，F2CMDB1 ノード構成の検索スループットは，FCMDB と比べ，10%程度向上している．これは，F2CMDB 実装時にロック方式の変更等最適化を行ったことによるものと考えられる．また，F2CMDB2 ノード構成では，F2CMDB1 ノード構成と比べてスループットが 2 倍程度となっている．これは，FCMDB 数が増えたことにより，処理できる検索リクエストの数が増加したためである．F2CMDB では，登録上限数を向上することを目的としてエンティティを複数の FCMDB を用いて分散管理するが，複数の FCMDB を検索に用いることができるため，副次的な効果として検索スループットの向上という効果も得られることがわかった．

5.　関 連 研 究

本研究に関連するものとして，分散型データベースや大規模データベース，連合データベースが挙げられる．分散型データベースに関する研究は従来から盛んに行われており，文献 [2] などが著名である．また，本研究でも取り入れたキー・バリュー型データストアやコンシステント・ハッシュといった技術は近年注目を集めており，盛んに研究が行われている．これらの技術は汎用的であるが，本想定環境では複数のエンティティをマージする必要があるため，これらの技術をそのまま適用することは難しい．本稿では，分散リコンサイルという新たな手法を提案し，分散環境におけるエンティティのマージを実現している．

数百 GB 以上のデータを管理する大規模データベース技術についての研究開発も盛んに行われている．Oracle 社では，パーティション化と呼ばれる手法によりデータベースを小規模な単位に分割することで，大規模化に伴う問題点を解決している [9]．2.3 節でも述べたように，本研究の想定環境では大規模データベースの利用は適さないため，F2CMDB との性能比較は行っていない．また，本稿では特定のデータベースを用いずに評価を行ったが，F2CMDB においてバックエンドに大規模データベースを用いることで，さらなるスケールアウトが可能になると考えられる．

Fedarated database（連合データベース）[4] とは，複数のデータベースを仮想的に統合したデータベースである．概念は本稿で述べた FCMDB と差異はなく，FCMDB は連合データベースの一種と見ることができる．両者の差異としては，FCMDB はエンティティ・データそのものを格納するものの他に，配下の MDR がもつエンティティ・データへのポインタのみをもつものも考えられており，より柔軟にデータベースを仮想統合できると言える．また，FCMDB では更新データは各データベースから統合データベースに入力し，検索時は統合データベースもしくは各データベースに対して行うことができる．

一方，CMDB に関する技術は，富士通や IBM，HP といったベンダによって研究開発が行われ，すでに製品にも組み込まれている．しかし，本稿で提案した F2CMDB のように，複数の FCMDB を用いてエンティティを分散管理することにより膨大な数のエンティティを管理する手法は，F2CMDB の他にはないと思われる．

6.　ま　と　め

本稿では，FCMDB を拡張し，複数の FCMDB を用いてエンティティを分散管理する F2CMDB の提案と評価を行った．評価の結果から，登録時間の増加をできるだけ抑えながら，登録上限数を改善したことを示した．また，検索時間については，検索式によっては大幅に短縮できることを示した．さらに，検索スループットについては，検索式によっては大幅に向上できることを示した．

以上の結果から，構成情報管理データベースの大規模化対応は実現できたといえる．今後は登録・検索スループットの詳細な測定や，F2CMDB に最適なデータベースについて検討する予定である．また，F2CMDB に対し，動的に FCMDB を追加・削除する機能を実装する予定である．

文　　献

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz,A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing”, Communications of the ACM, Vol. 53, No. 4, pp. 50–58, 2010.

[2] P. A. Bernstein and N. Goodman, “Concurrency Control in Distributed Database Systems”, ACM Computing Surveys, Vol. 13, No. 2, pp. 185–221 , 1981.

[3] M. Brenner, M. Garschhammer, M. Sailer, and T. Schaaf,

"CMDB - Yet Another MIB? On Reusing Management Model Concepts in ITIL Configuration Management", In Proceedings of the 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006), pp. 269–280, 2006.

[4] D. Heimbigner and D. McLeod, "A federated architecture for information management", ACM Transactions on Information Systems (TOIS), Vol. 3, No. 3, pp. 253–278, 1985.

[5] ITIL ベースの構成管理を支援する統合 CMDB 技術 : 富士通, http://jp.fujitsu.com/about/journal/middleware/technologies/200810.shtml.

[6] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", In Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 654–663, 1997.

[7] Office of Government Commerce (OGC), ed, "Service Support, IT Infrastructure Library (ITIL)," The Stationary Office, Norwich, UK, 2000.

[8] Office of Government Commerce (OGC), ed, "Introduction to ITIL, IT Infrastructure Library (ITIL)," The Stationary Office, Norwich, UK, 2005.

[9] http://www.oracle.com/technetwork/jp/index.html

[10] The Scala Programming Language, http://www.scala-lang.org/.

[11] Welcome to the CMDB Federation Workgroup, http://www.cmdbf.org/.