MySQL Cluster 7.2 の新機能

Web スケールのパフォーマンスと
キャリアグレードの可用性を実現

MySQL® ホワイトペーパー

2012 年 2 月

目次

はじめに

MySQL Cluster は、スケーラブルで、リアルタムな、 ACID 準拠のトランザクション・データベースです。99.999% の高可用性とオープンソースの低総所有コスト（TCO）を兼ね備えています。分散型マルチマスタ・アーキテクチャにより、単一障害点を持ちません。MySQL Cluster はコモディティ・ハードウェアで水平スケールが可能で、読み取りおよび書き込み集中型の負荷に自動シャーディングで対応し、SQL および NoSQL インタフェースによるアクセスが可能です。

MySQL Cluster は、もともとキャリアグレードの可用性およびリアルタイムのパフォーマンスを必要とするネットワーク・アプリケーション向け組み込みデータベースとして設計されましたが、Web およびエンタープライズ・アプリケーション事例に対応するために、急速に機能が拡張されてきました。対応アプリケーションには以下があります。

- 大容量 OLTP
- リアルタイム分析
- e コマース、在庫管理、ショピング・カート、支払い処理、達成管理等
- 不正検出機能を備えた金融取引
- モバイルおよびマイクロペイメント
- セッション管理およびキャッシング
- フィード・ストリーミング、解析およびレコメンデーション
- コンテンツ管理および配信
- 大規模マルチレイヤ・オンラインゲーム
- コミュニケーションおよびプレゼンス・サービス
- サブスクライバ / ユーザー・プロファイル管理およびエンタイトルメント（権限付与）

このホワイトペーパーでは、MySQL Cluster 7.2 のリリースで追加された最新の拡張機能について、詳しく説明します。MySQL Cluster 7.2 では、以下の点が強化されています。

- 高度にダイナミックなサービスの開発をサポート:70 倍の複雑なクエリー・パフォーマンス、ネイティブ Memcached API、4 倍のデータ・ノードのスケーラビリティ、MySQL5.5 サーバーとの統合、および仮想マシン(Virtula Machine:VM)環境のサポート
- データ・スケーラビリティの向上:新しいマルチサイト・クラスタリングおよびアクティブ/アクティブ・レプリケーション
- プロビジョニングおよび管理の簡素化:ユーザー権限の統合

新機能概要

以下のリストは、最新の MySQL Cluster の全ての新機能および拡張機能のリストではありません。全ての機能については、こちらのドキュメントを参照してください: http://dev.mysql.com/doc/#cluster

新機能/向上機能	カテゴリ	リリース
アダプティブ・クエリー・ローカライゼーション（AQL）	次世代 Web サービス	MySQL Cluster 7.2
ネイティブ Memcached API NoSQL	開発の柔軟性	MySQL Cluster 7.2
データ・ノード・スケーラビリティの向上	パフォーマンス	MySQL Cluster 7.2
マルチサイト・クラスタリング	データセンター間スケーラビリティ	MySQL Cluster 7.2
アクティブ/アクティブ・レプリケーションの簡素化	データセンター間スケーラビリティ	MySQL Cluster 7.2
ユーザー権限の統合	プロビジョニングおよび管理の簡素化	MySQL Cluster 7.2
MySQL 5.5 サーバーとの統合	次世代 Web サービス	MySQL Cluster 7.2
仮想マシン（Virtual Machine）のサポート	プロビジョニングおよび管理の簡素化	MySQL Cluster 7.2
MySQLEnterprise Monitor – MySQL Cluster 向け機能向上	プロビジョニングおよび管理の簡素化	MySQL Enterprise Monitor 2.3.x
オンライン・ノード追加の自動化	プロビジョニングおよび管理の簡素化	MySQL Cluster Manager 1.1
1 ステップ Cluster 作成 (bootstrap)	プロビジョニングおよび管理の簡素化	MySQL Cluster Manager 1.1

MySQL Cluster アーキテクチャ

昨今の最もダイナミックなサービスへ対応するには、開発者およびアーキテクトは、データベースのパフォーマンス、スケーラビリティおよび可用性を多角的に検討する必要があります。

- 一カ所に統合されているか地理的に分散されているかにかかわらず、読み取りおよび書き込みに対して自動的にスケールする（それらのデータに対し複雑なクエリーの実行能力を犠牲にしない）。
- 要件に合わせて迅速にデータベース・サービスを適応させられる。たとえばデータベースへの容量およびパフォーマンスの増強、スキーマの展開を、全てダウンタイムなしで実行する。
- SQL および NoSQL インタフェースを備え、データベースへのアクセスに使用する API を柔軟に選択できることにより、開発者の生産性およびアプリケーションの柔軟性を向上する。
- 計画的なメンテナンス作業および予定外の障害発生時においても、継続的な可用性を維持する。

MySQL Cluster のアーキテクチャは、これらの要件を以下の機能で対応するように設計されています。

- 書き込みのスケーラビリティを実現する自動シャーディング
- リアルタイム・パフォーマンスの最適化
- アクティブ/アクティブ遠隔地レプリケーション
- オンラインによるスケーリングおよびスキーマのアップデート
- SQL および NoSQL インタフェース
- 99.999%の可用性を実現する、クラスタを統合し、同期レプリケーションを備えた非共有分散型アーキテクチャ

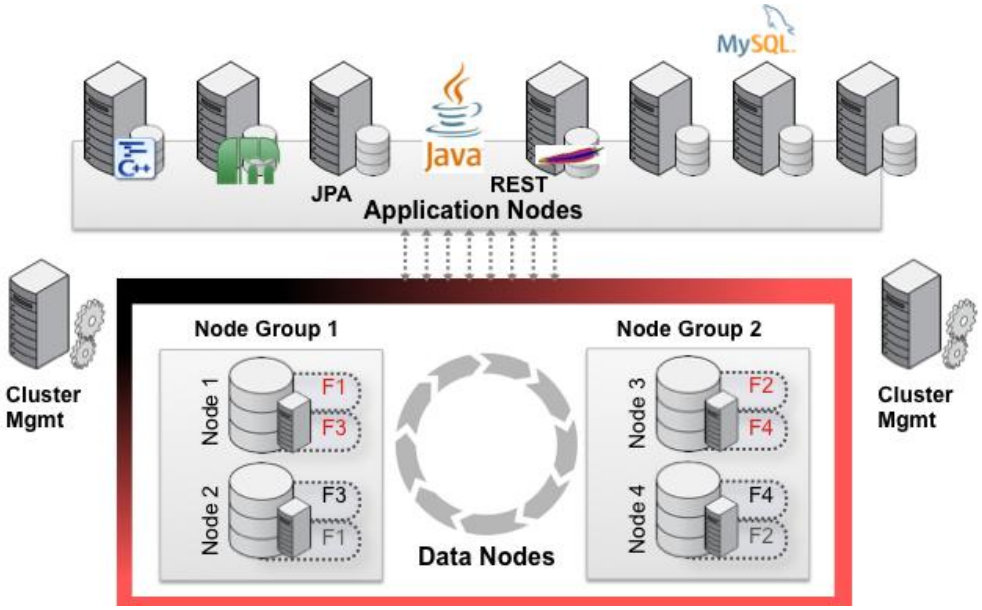

図 1 MySQL Cluster アーキテクチャ

MySQL Cluster には、3 つのノードのタイプがあり、これらが統合的にアプリケーションにサービスを提供します。アプリケーション側からは、これらノードの違いを認識する必要はありません。単にクラスタに接続すれば、シームレスにサービスが提供されます。以下は、3 つのノードのタイプの概要です。

データ・ノード - データの格納とデータへのアクセスを管理します。テーブルは自動的にノード間でシャードされ、透過的に負荷分散、レプリケーション、フェイルオーバーおよび自己修正も処理します。

アプリケーション・ノード - アプリケーション・ロジックからデータ・ノードへの接続性を提供します。複数の API がアプリケーションに提供されます。MySQL は、主要な全ての Web 開発言語およびフレームワークへの接続性をサポートする、標準 SQL インタフェースを提供します。Memchached、Java、REST/HTTP、C++ (NDB-API)、を含む、NoSQL インタフェースも備えています。

管理ノード - クラスタの構成に使用されます。ノードに障害が発生した場合には、アービトレータとしても動作します

MySQL Cluster のアーキテクチャの詳細は、こちらのホワイトペーパーを参照してください。

MySQL Cluster による Web データベース・スケーリング・ガイド:
http://www-jp.mysql.com/why-mysql/white-papers/mysql_wp_scaling_web_databases.php

MySQL Cluster 7.2 の新機能

アダプティブ・クエリー・ローカライゼーション（AQL） - 複雑なJOINの速度が70倍に

MySQL は長年にわたって、NoSQL データ・ストアで謳われているメリット（シンプルな API、リアルタイム応答、リニアな読み取り/書き込みのスケーラビリティを確保する自動シャーディング、オンライン作業、高可用性）を、確実に ACID に準拠したトランザクションや SQL インタフェースの柔軟性とともに実現してきました。 MySQL Cluster 7.2 では、この SQL インタフェースが最適化され、複数のパーティション（シャード）にまたがる大規模で複雑な JOIN（結合）の速度が飛躍的に向上しています。

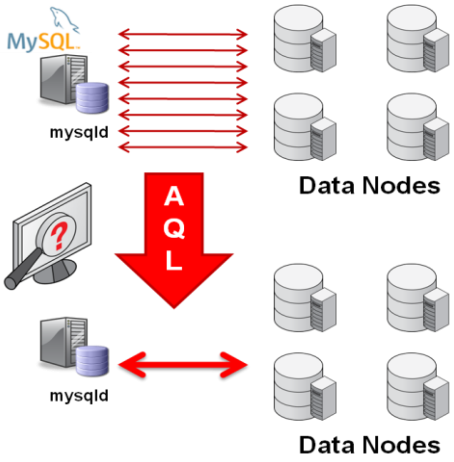

図2 AQLではJOINをデータ・ノードで処理

デフォルトでは、JOIN 処理は MySQL サーバーで実行されます。そのため、データがローカルに保存されている場合には高いパフォーマンスを発揮します。 一方、MySQL Cluster では、データは複数の冗長データ・ノードにわたって分散されます。その結果、MySQL サーバーでネステッド・ループ結合を実行する場合、各ステップで繰り返しデータ・ノードにアクセスする必要があります。JOIN の深さが増すにつれ、あるいは中間結果セットのサイズが大きくなるにつれ、データ・ノードに対するメッセージ数も急速に増加するため、クエリー実行速度が大幅に低下する可能性があります。

アダプティブ・クエリー・ローカライゼーション（AQL）機能では、MySQL サーバーからのクエリーがデータ・ノードに送信されます。このデータ・ノード上でデータのローカル・コピーに対してクエリーが並列に実行され、その後マージされた結果セットが MySQL サーバーに返されます。その結果、ネットワーク・トリップが減少し、パフォーマンスが大幅に向上します。

複雑なクエリーを高スループットの OLTP 処理と同時に、1 つのアプリケーションまたはデータベース・ソリューションで実行する必要がある場合や、複数のアプリケーションが同じデータにアクセスする一方、アクセス/クエリー要件はそれぞれのアプリケーションで異なるような場合に、AOL を利用すれば MySQL Cluster の効果が高まります。

この機能を支えるパラメータ化クエリーを、アプリケーションで NDB API を直接操作して使用することもできます。こうすることで、必要となるデータベース・アクセス数を削減し、さらにパフォーマンスを改善できます。

AQL のしくみ

すべての API ノードからデータ・ノードへのアクセスに、ネイティブ C++ NDB API が使用されます。MySQL サーバー（さらには新機能の Memcached Cluster API）はそのような API ノードの一種です。この API は、許可されたパラメータ化クエリーやリンク・クエリー（クエリーの入力がそれ以前のクエリーに依存するもの）にまで拡張されています。

以前のブログ投稿[1]に基づいて例を示します。MySQL サーバーによって処理される、JOIN を実装する従来の方法は次のとおりです。

```
SQL > select t1.b, t2.c from t1,t2 where t1.pk=22 and t1.b=t2.pk;
ndbapi > read column b from t1 where pk = 22;
[round trip]
(b = 15)
ndbapi > read column c from t2 where pk = 15;
[round trip]
(c = 30)
[ return b = 15, c = 30 ]
```

新機能を使用すると、単一のネットワーク・ラウンドトリップで同じ処理を実行できるようになります。このラウンドトリップでは、2 つ目の読み取り操作が最初の読み取り操作の結果に依存します。

```
ndbapi > read column @b:=b from t1 where pk = 22;
         read column c from t2 where pk=@b;
         [round trip]
         (b = 15, c = 30)
         [ return b = 15, c = 30 ]
```

このような複雑な処理は SQL を使用する場合はまったく表面化しませんが、アプリケーションで NDB API を使用する場合は、このようなパラメータ化クエリーの使用方法を理解しておくと役に立ちます。

さらに、MySQL Cluster から MySQL サーバーに、利用可能なインデックスに関する適切な情報が提供されるようになりました。この情報を MySQL オプティマイザで利用することで、より適切なクエリー実行計画を自動的に生成できます。 以前のバージョンでは、オプティマイザに対するヒントの提供はユーザーに委ねられていました。

実環境のテストで、さまざまなクエリーのパフォーマンスが通常、70 倍程度向上することが示されています。11 個のテーブルを結合する複雑なクエリーを実行するオンライン・コンテンツ保存/管理システムの例において、MySQL Cluster 7.1 ではこのクエリーの処理に 87 秒以上かかっていましたが、MySQL Cluster 7.2 でテストするとわずか 1.2 秒まで短縮されました。実際のクエリーの内容、テスト環境、全結果については、次の記事を参照してください。

http://www.clusterdb.com/mysql-cluster/70x-faster-joins-with-aql-in-mysql-cluster-7-2-dmr/

AQL の利用方法

AQL を利用するかどうかは、グローバル変数 ndb_join_pushdown で制御します。デフォルトではこの変数はオンに設定されています。

JOIN 処理で AQL を利用できる（つまり、データ・ノードに「処理を移行」する）ようにするには、次の条件を満たす必要があります。

---

[1]http://messagepassing.blogspot.com/2011/01/low-latency-distributed-parallel-joins.html

- 結合する列はすべて、まったく同じデータ型である必要がある （たとえば、INT 列と BIGINT 列を結合する場合は、JOIN 処理を移行できない）
- BLOB 列または TEXT 列を参照するクエリーはサポート対象外である
- 明示的ロックはサポート対象外だが、NDB ストレージエンジン特有の暗黙的行ベース・ロックが強制的に実行される
- JOIN 処理を移行するには、`ref`、`eq_ref`、`const` のいずれかのアクセス・メソッド（またはこれらのメソッドの組み合わせ）を使用して結合の子テーブルにアクセスする必要がある
- [LINEAR] HASH、LIST または RANGE によって明示的にパーティショニングされたテーブルを参照する JOIN 処理は、現時点では移行できない

ある JOIN 処理が移行可能かどうかは、`EXPLAIN` を使用して確認できます。JOIN 処理が移行可能な場合は、出力結果の Extra 列に、移行される JOIN 処理への参照が表示されます。

AQL の利用可能頻度に関する集計を示すために、NDBINFO データベースのカウンタ・テーブルに多数の新規エントリが追加されています。

このカウンタから、AQL 機能の利用可能頻度や、次の SQL によるクエリー実行可能頻度に関する情報を得ることができます。

```
mysql> SELECT node_id, counter_name, val FROM ndbinfo.counters WHERE block_name= "DBSPJ";
+---------+-----------------------------------+------+
| node_id | counter_name                      | val  |
+---------+-----------------------------------+------+
|       2 | READS_RECEIVED                    |    0 |
|       2 | LOCAL_READS_SENT                  |    0 |
|       2 | REMOTE_READS_SENT                 |    0 |
|       2 | READS_NOT_FOUND                   |    0 |
|       2 | TABLE_SCANS_RECEIVED              |    0 |
|       2 | LOCAL_TABLE_SCANS_SENT            |    0 |
|       2 | RANGE_SCANS_RECEIVED              |    0 |
|       2 | LOCAL_RANGE_SCANS_SENT            |    0 |
|       2 | REMOTE_RANGE_SCANS_SENT           |    0 |
|       2 | SCAN_BATCHES_RETURNED             |    0 |
|       2 | SCAN_ROWS_RETURNED                |    0 |
|       2 | PRUNED_RANGE_SCANS_RECEIVED       |    0 |
|       2 | CONST_PRUNED_RANGE_SCANS_RECEIVED |    0 |
|       3 | READS_RECEIVED                    |    0 |
|       3 | LOCAL_READS_SENT                  |    0 |
|       3 | REMOTE_READS_SENT                 |    0 |
|       3 | READS_NOT_FOUND                   |    0 |
|       3 | TABLE_SCANS_RECEIVED              |    0 |
|       3 | LOCAL_TABLE_SCANS_SENT            |    0 |
|       3 | RANGE_SCANS_RECEIVED              |    0 |
|       3 | LOCAL_RANGE_SCANS_SENT            |    0 |
|       3 | REMOTE_RANGE_SCANS_SENT           |    0 |
|       3 | SCAN_BATCHES_RETURNED             |    0 |
|       3 | SCAN_ROWS_RETURNED                |    0 |
|       3 | PRUNED_RANGE_SCANS_RECEIVED       |    0 |
|       3 | CONST_PRUNED_RANGE_SCANS_RECEIVED |    0 |
+---------+-----------------------------------+------+
```

ネイティブMemcached API NoSQLインタフェース

MySQL Cluster では MySQL サーバーではなくデータ・ノードにテーブルが保存されるため、データベース・アクセスに複数のインタフェースを利用できます。

どの API に対しても、任意の SQL および NoSQL のアクセス・メソッドを混合して、同じデータ・セットに同時に使用できるという非常に柔軟な開発環境を提供します。そのため、MySQL Cluster は、以下のどのサービスの組み合わせもリアルタイムに利用可能です。

- SQL API を使用したリレーショナルなクエリー
- REST/HTTP および memcached API を使用したキー・バリュー型 Web サービス
- ClusterJ および JPA API によるエンタープライズ・アプリケーション;
- NDB API を使用したリアルタイム Web サービス (例：プレゼンス、ロケーション・ベース)

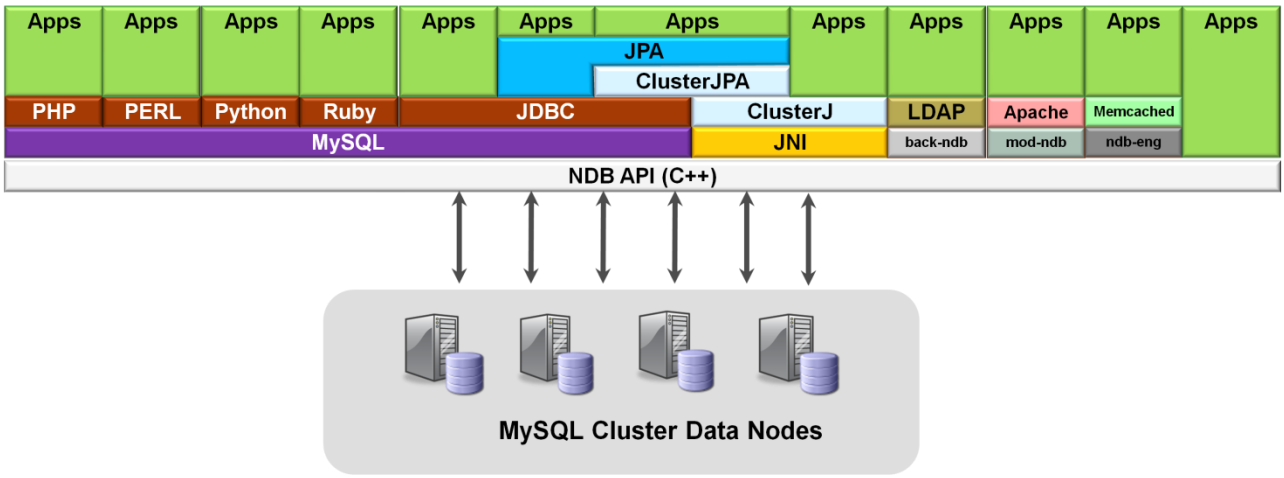

図 3 MySQL Cluster データへのAPIセット

MySQL Cluster 7.2 では、データベースへのアクセスに利用できる多様な API に、ネイティブ Memcached プロトコルが追加されました。 Memcached API を使用すると、SQL に変換することなく Web サービスから直接 MySQL Cluster にアクセスできます。そのため、読み取り/書き込みクエリーにおいて低レイテンシ、高スループットを実現できます。 SQL 解析などの処理が必要なくなるため、サーバーのハードウェア・リソース（CPU、メモリ、I/O）をストレージエンジン内部のクエリー自体に集中的に使用できるようになります。

MySQL Cluster と Memcached をネイティブに統合することには、パフォーマンス面の他にも数々のメリットがあります。たとえば、キャッシュ処理層とデータベース層を単一のレイヤーに統合して、キャッシュ無効化や整合性チェックなどの複雑な処理を削減できます。 このようなメリットの詳細については、次の URL に投稿されている Developer Zone の記事を参照してください。

http://dev.mysql.com/tech-resources/articles/nosql-to-mysql-with-memcached.html

Memcached と同様に、MySQL Cluster ではキャッシングをインメモリで処理する分散ハッシュ・テーブルを提供します。（このハッシュ・テーブルにはシンプルな Memcached API からアクセスします）。 MySQL Cluster ではこの Memcached 機能を拡張し、多くの管理機能や監視機能に加え、書き込み集中型の負荷、永続性を含む ACID 準拠の完全なリレーショナル・モデル、高度なクエリー、スケールアウトのための透過的なパーティショニング、99.999%の可用性をサポートしています。

実装はシンプルです。アプリケーションから Memcached プロセスに、標準 Memcached API を使用して読み取り/書き込みリクエストを送信します。 その結果、NDB の Memcached ドライバ（Memcached プロセスの一部）が呼び出され、さらに NDB API が呼び出されて、MySQL Cluster の各データ・ノードで保持しているデータに非常に高速にアクセスします。

この機能は、アプリケーション・アーキテクトがニーズに最も適合した構成を見つけることを可能にする、非常に柔軟性の高いソリューションとして設計されています。 Memcached API はデータ・ノードとアプリケーション・ノードのいずれかに共同配置することも、専用の Memcached レイヤー内に共同配置することも可能です（図 4 を参照）。

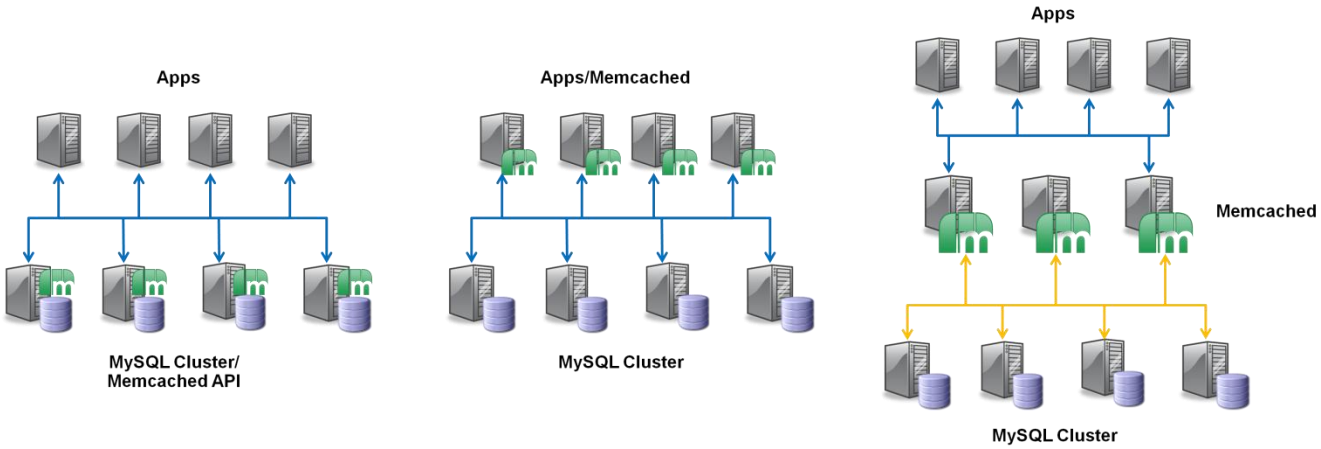

図4 Memcached API配置オプション

Memcached サーバー内でデータの一部をキャッシュするか、すべてのデータをキャッシュするかは開発者が選択できます（さらに、データを MySQL Cluster にも保存するかどうかを指定できます）。そのため、たとえば次のように、異なるデータの処理方法を選択できます。

- 変動しやすい（頻繁に読み取り/書き込みが行われる）データについては、MySQL Cluster のみに保存
- 更新頻度は低く、読み取り頻度は高いデータについては、MySQL Cluster と Memcached メモリの両方に保存
- 生存期間が短く、読み取り頻度は高いが、長期的に保存する必要のないデータについては、Memcached キャッシュのみに保存

この動作は、MySQL Cluster のテーブルを使用してキー・プリフィックスごとに構成できます。アプリケーションで注意を払う必要はなく、Memcached API を使用するだけで済みます。データを適切な場所に保存してデータ全体の同期状態を維持する役割は、MySQL Cluster が担います。

運用オプションの詳細については、次の記事を参照してください。

http://www.clusterdb.com/mysql-cluster/scalabale-persistent-ha-nosql-memcache-storage-using-mysql-cluster/

MySQL Cluster に保存されているキー・バリュー型データに対する変更はすべてバイナリ・ログに記録され、レプリケーションやポイント・イン・タイム・リカバリの実行中に適用されます。

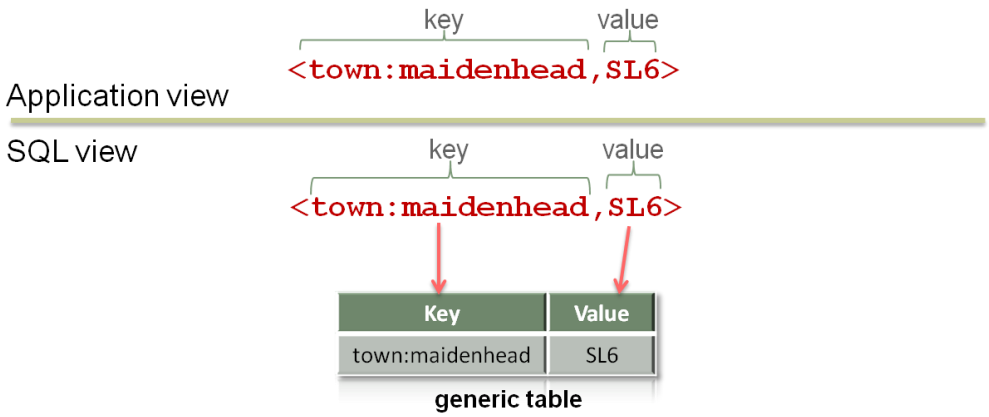

図5 スキーマの制約を受けないキー/値の保存

デフォルトでは、すべてのキー・バリューは同じテーブルに書込まれ、各キー・バリューのペアは 1 行で保存されます。そのため、スキーマレスのデータ保存が可能です。 この方法について、図 5 に示します。この図では、キー・バリューのペア（key="town:maidenhead"、value="SL6"）は単純に、Memcached API によって追加される他のデータと同じ汎用テーブルに保存されます。

これに対して、キー・バリューのそれぞれが特定のテーブルにある事前定義済みの列にリンクされるように、キー・プリフィックスを定義することもできます。 この方法については、図 6 に示します。この図は分かりやすくするために簡略化されていますが、実際は多数のテーブルに対してこの動作が構成されます。実際の構成テーブルをセットアップする方法については、次のセクションで説明します。

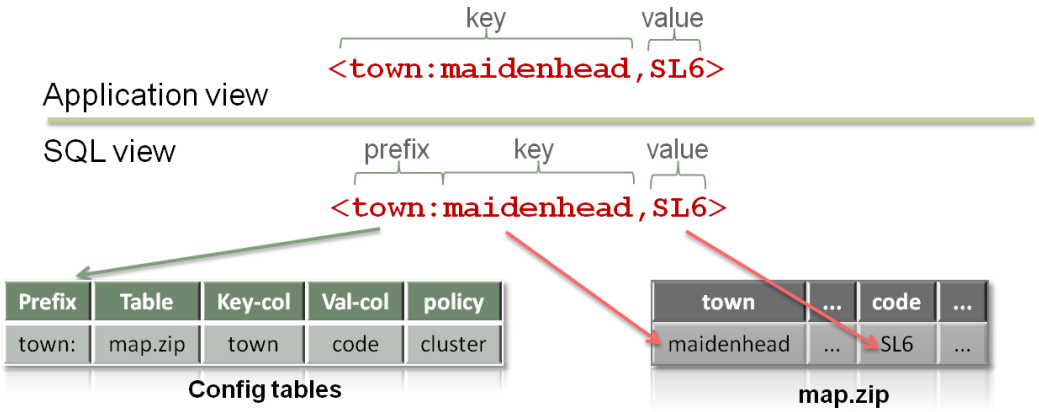

図6 ユーザー定義スキーマにマッピングされたキー/値データ

上の図では、キー・プリフィックス `town:` を使用して、town:プリフィックスで始まるキー（この場合、`town:maidenhead`）のための構成情報セットを識別します。 図の構成データは、キーの残りの部分（`maidenhead`）を `map` データベースの `zip` テーブルの `town` 列にマッピングすること、およびそのキーに関連付けられる値（`SL6`）を同じテーブルの `code` 列に保存することを示しています。

アプリケーションからこの同じデータに SQL 経由でアクセスする必要がある場合は、キー・プリフィックスを既存のテーブルの列にマッピングして、MySQL Cluster にすでに保存されているスキーマ構造化データに Memcached でアクセスできるように設定できます。

Memcached API の構成と利用

このセクションでは、MySQL Cluster データベースがインストール済みであり、実行中であることを前提としています。その状態からいくつかの追加手順を実行する、元の Memcached API ブログ記事[2]の手順に従って説明していきます。

まず、NDB ドライバを有効にして Memcached サーバーを起動します。

```
[billy@ws2 ~]$ cd $INSTALL_PREFIX
[billy@ws2 mysql-memcache]$ bin/memcached -E lib/ndb_engine.so -e
"connectstring=localhost:1186;role=db-only" -vv -c 20
```

例にあるように“connectstring”を指定すると、Memcached API と異なるサーバー上にプライマリの MySQL Cluster を配置できます。 また、実際には同じ Memcached サーバーから複数のクラスタにアクセスできます。この構成は、プライマリ・クラスタの `ndbmemcached` データベース内で行います。

アプリケーションから Memcached API にアクセスするにはさまざまなコネクタ/クライアントを使用できますが、ここは説明を分かりやすくするために、対話型セッションに標準の Telnet クライアントを使用します。 Memcached ポート（デフォルトは `11211`）に接続した後、キー“maidenhead”に対して値“SL6”を保存し、次にその値を取得します。

```
[billy@ws2 ~]$ telnet localhost 11211

set maidenhead 0 0 3 SL6
STORED

get maidenhead
VALUE maidenhead 0 3 SL6 END
```

この同じデータに、標準の SQL を使用して MySQL Cluster データベースからアクセスできます。

```
mysql> SELECT * FROM ndbmemcache.demo_table;
   +------------------+------------+-----------------+--------------+
   | mkey             | math_value | cas_value       | string_value |
   +------------------+------------+-----------------+--------------+
   | maidenhead       |       NULL | 263827761397761 | SL6          |
   +------------------+------------+-----------------+--------------+
```

このデータを SQL で変更し、その直後に Memcached API からその変更後の内容を確認することもできます。

```
mysql> UPDATE ndbmemcache.demo_table SET string_value='sl6 4' WHERE mkey='maidenhead';
```

---

[2] http://www.clusterdb.com/mysql-cluster/scalabale-persistent-ha-nosql-memcache-storage-using-mysql-cluster/

```
[billy@ws2 ~]$ telnet localhost 11211

get maidenhead
VALUE maidenhead 0 5 SL6 4 END
```

これまでの操作は完全にスキーマレスであり、アプリケーションで新しいキー・バリュー・ペアを追加していくこともできます。そのキー・バリュー・ペアはすべてデフォルトのテーブルに追加されます。 プロトタイプ・データベースや比較的小さなサイズのデータベースの場合は、これで十分対応できます。前述のとおり、このデータにはSQL 経由でアクセスできますが、SQL 側でより詳細なスキーマを使用したい場合や、あるいはその他の理由（データの一部のみを 2 つ目のクラスタにレプリケーションして地理的な冗長性を確保する、InnoDB にレプリケーションしてレポートを生成するなど）で複数のテーブルにデータを保存することが必要な場合もあります。

次の手順では、独自のデータベースとテーブルを作成し（作成済みでないことを前提とする）、アプリケーションから Memcached API 経由でデータを取得するための定義を作成します。 まず、いくつかの列を持つテーブルを作成し、それらのテーブルに Memcached API 経由でアクセスできるようにします。

```
mysql> CREATE DATABASE clusterdb; USE clusterdb;
mysql> CREATE TABLE towns_tab (town VARCHAR(30) NOT NULL PRIMARY KEY, zip VARCHAR(10),
population INT, county VARCHAR(10)) ENGINE=NDB;
mysql> INSERT INTO towns_tab VALUES ('Marlow', 'SL7', 14004, 'Berkshire');
```

次に、Memcached API 経由でこのデータにアクセスする方法について、NDB ドライバに通知する必要があります。新規作成したテーブルの列のうち、公開するものを識別するための 2 個の‘containers’を作成します。 さらに、Memcached API のユーザーがアクセス対象データ（database/table/column など）を示すためのキー・プリフィックスを定義します。

```
mysql> USE ndbmemcache;
mysql> INSERT INTO containers VALUES ('towns_cnt', 'clusterdb', 'towns_tab', 'town', 'zip',
0, NULL, NULL, NULL);
mysql> INSERT INTO containers VALUES ('pop_cnt', 'clusterdb',
  'towns_tab', 'town', 'population', 0, NULL, NULL, NULL);
mysql> SELECT * FROM containers;
+-----------+------------+----------------+------------+---------------+-------+-----------------+------------+-------------------+
| name      | db_schema  | db_table       | key_columns | value_columns | flags | increment_column | cas_column | expire_time_column |
+-----------+------------+----------------+------------+---------------+-------+-----------------+------------+-------------------+
| demo_table | ndbmemcache | demo_table     | mkey       | string_value  | 0     | math_value      | cas_value  | NULL              |
| pop_cnt    | clusterdb   | towns_tab      | town       | population    | 0     | NULL            | NULL       | NULL              |
| demo_tabs  | ndbmemcache | demo_table_tabs | mkey       | val1,val2,val3 | 0     | NULL            | NULL       | NULL              |
| towns_cnt  | clusterdb   | towns_tab      | town       | zip           | 0     | NULL            | NULL       | NULL              |
+-----------+------------+----------------+------------+---------------+-------+-----------------+------------+-------------------+
mysql> INSERT INTO key_prefixes VALUES (1, 'twn_pr:', 0, 'ndb-only', 'towns_cnt');
mysql> INSERT INTO key_prefixes VALUES (1, 'pop_pr:', 0, 'ndb-only', 'pop_cnt');
mysql> SELECT * FROM key_prefixes;
   +----------------+------------+------------+----------------+------------+
   | server_role_id | key_prefix | cluster_id | policy         | container  |
   +----------------+------------+------------+----------------+------------+
   |              1 | pop_pr:    |          0 | ndb-only       | pop_cnt    |
   |              3 |            |          0 | caching        | demo_table |
   |              0 |            |          0 | caching        | demo_table |
   |              0 | db:        |          0 | ndb-only       | demo_table |
   |              0 | mc:        |          0 | memcache-only  | NULL       |
   |              2 |            |          0 | memcache-only  | NULL       |
   |              1 |            |          0 | ndb-only       | demo_table |
   |              1 | t:         |          0 | ndb-only       | demo_tabs  |
   |              1 | twn_pr:    |          0 | ndb-only       | towns_cnt  |
   +----------------+------------+------------+----------------+------------+
```

これで、これらの列（および SQL 経由ですでに追加されたデータ）に、Memcached API 経由でアクセスできるようになります。

```
[billy@ws2 ~]$ telnet localhost 11211
get twn_pr:Marlow
VALUE twn_pr:Marlow 0 3 SL7 END

set twn_pr:Maidenhead 0 0 3 SL6
STORED

set pop_pr:Maidenhead 0 0 5 42827
STORED
```

さらに、この変更の結果を SQL 経由で確認できます。

```
mysql> SELECT * FROM clusterdb.towns_tab;
   +------------+------+------------+-----------+
   | town       | zip  | population | county    |
   +------------+------+------------+-----------+
   | Maidenhead | SL6  |      42827 | NULL      |
   | Marlow     | SL7  |      14004 | Berkshire |
   +------------+------+------------+-----------+
```

最終テストとして、同じデータにアクセスする 2 つ目の Memcached サーバーを起動します。 すべてのサーバーを同じホスト上で実行しているので、2 つ目のサーバーのリスニング・ポートには異なるポートを指定する必要があります。

```
[billy@ws2 ~] cd /usr/local/mysql-memcache [billy@ws2 mysql-memcached]$ bin/memcached -E
lib/ndb_engine.so -e "connectstring=localhost:1186;role=db-only" -vv -c 20 -p 11212 -U 11212
[billy@ws2 ~]$ telnet localhost 11212

get twn_pr:Marlow
VALUE twn_pr:Marlow 0 3 SL7 END
```

パフォーマンス面の拡張機能

MySQL Cluster 7.2 では、各データ・ノードで利用可能なスレッドをより効果的に活用できるようにすることで、データ・ノードでのノード単位のパフォーマンスが改善されています。 最新のベンチマーク結果については、http://www-jp.mysql.com/why-mysql/benchmarks/mysql-cluster/ を参照してください。

マルチサイト・クラスタリング

マルチサイト・クラスタリングでは、データセンター間のスケーラビリティを確保するための新しいオプションが用意されています。 複数のデータセンター間でデータ・ノードを分割する運用が初めてサポートされるようになりました。 この運用モデルを利用すると、競合処理のためにアプリケーションやスキーマを変更する必要なく、データセンター間で更新を同期的にレプリケーションし、ノード障害の発生時にはそれらのサイト間で自動的にフェイルオーバーさせることができます。

MySQL Cluster で利用されるハートビート・メカニズムが改善された結果、WAN 上での一時的なレイテンシ・スパイクに対する耐性が向上し、クラスタの維持運用が容易になりました。 “ConnectivityCheck”メカニズムが新たに導入されています。このメカニズムを利用するには、明示的に構成する必要があります。 このメカニズムを追加するとメッセージングのオーバーヘッドや障害対応のレイテンシが増加するため、デフォルトではオフになっています。

ハートビート、Connectivity_Check、GCP タイムアウト、トランザクション・デッドロック・タイムアウトなどのさまざまな MySQL Cluster パラメータを構成できます。 これらのパラメータの詳細については、別のドキュメント[3]を参照してください。

マルチサイト・クラスタリングの推奨事項

- 最小限の安定したレイテンシを確保する
- ピーク時の予想負荷にも対応する十分な帯域幅を持つネットワークをプロビジョニングし、ノード・リカバリおよびシステム・リカバリについてテストを実施する
- レイテンシの変動幅を超える安全範囲を確保できるハートビート期間を構成する
- 不要なノード障害を避けるため、ConnectivtyCheckPeriod を構成する
- GCP タイムアウト、トランザクション・デッドロック・タイムアウト、トランザクション非アクティブ・タイムアウトなどの、その他のタイムアウトを構成する

上記の事項の詳細については、次の記事を参照してください。

http://blogs.oracle.com/MySQL/entry/synchronously_replicating_databases_across_data

例

高スループットおよび迅速な障害検知が必要となるアプリケーションで推奨されるレイテンシおよび帯域幅の要件は次のとおりです。

- リモートにあるデータ・ノード間のレイテンシは 20 ミリ秒以内とする
- ネットワーク・リンクの帯域幅は 1 秒あたり 1 ギガビットを超える値とする

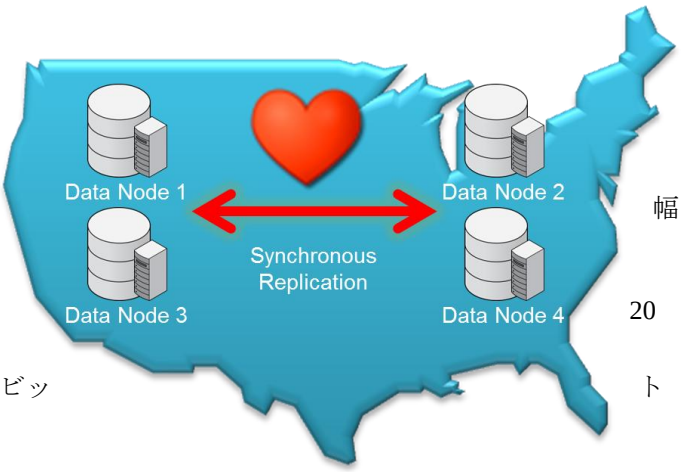

図 7 データセンター間でのデータ・ノード分割

このような厳しい運用環境を必要としないアプリケーションの場合は、上記で推奨したテストの結果に従って、レイテンシと帯域幅の設定を緩めることもできます。

推奨事項で示しているとおり、マルチサイト・クラスタリングを運用する前に多くの要因を検討する必要があります。 地理的な冗長性を確保するため、オラクルでは遠隔地レプリケーションの利用をお勧めしますが、上記で説明した検討事項や制約に則った上で、マルチサイト・クラスタリングでも代替的な運用方法が用意されています。

[3] http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-params-ndbd.html

シンプルなアクティブ / アクティブ遠隔地レプリケーションと競合の解消

MySQL Cluster は長年にわたって、遠隔地レプリケーション機能を提供してきました。これは、リモートにある複数のデータセンターにクラスタを分散して、ユーザーに近い場所にデータを配置することで地理的なレイテンシの影響を抑え、さらに地理的な冗長性と障害時リカバリを実現するものです。

遠隔地レプリケーションはこれまで常にアクティブ/アクティブ・テクノロジーに基づいて設計されてきました。そのため、アプリケーションで異なるクラスタ上の同じ行を同時に更新しようとすると、競合が検出され解消されます。 この結果、各サイトで積極的に読み取り/書き込みリクエストに対応しながら、クラスタ間のデータ整合性を維持できます。 また、リモート・サイトに待機系のハードウェアをプロビジョニングして実行することによるオーバーヘッドもありません。

MySQL Cluster 7.2 リリースでは、アクティブ/アクティブ・レプリケーションの実装が格段に容易になりました。開発者がアプリケーション内部でタイムスタンプ列の実装や管理を行う必要がなくなります。 また、個々の処理だけではなく、トランザクション全体に対するロールバックを実行できるようになります。

これらの拡張機能によって、複数のデータセンターにまたがるグローバル規模のサービスを非常に運用しやすくなります。

競合とは

MySQL Cluster では、2 クラスタ（またはそれ以上）の間での双方向レプリケーションが可能です。 各クラスタ内でのレプリケーションは同期的に実行されますが、クラスタ間では非同期的に実行されます。そのため、表 1 に示すようなことが起こり得ます。

サイト A	レプリケーション	サイト B
x == 10		x == 10
x == 11		x == 20
	-- x=11 -->	x == 11
x==20	<-- x=20 --	

表1 競合につながるレプリケーションの流れ

この例では、値（あるテーブルのある行の列）がサイト A で 11 に設定され、その変更内容がサイト B に対するレプリケーションのキューに入れられます。 また、その間にサイト B でアプリケーションによって値が 20 に設定され、その変更内容がサイト A に対するレプリケーションのキューに入れられます。 どちらのサイトでも、もう一方のクラスタからレプリケーションされた変更内容を受信して適用し、その結果サイト A では値 20 が、サイト B では値 11 が設定されます。つまり、データベースが一貫性のない状態となります。

MySQL Cluster 7.2 で最終的に整合性を確保する方法

一貫性のない状態となった後で双方のクラスタ間で整合性を確立するために、次の 2 つの手順を実行します。

1. 発生した競合を検出する
2. 一貫性のない状態を解消する

競合の検出

通常は 2 つのクラスタがアクティブ/アクティブ・レプリケーションを構成することが想定されますが、ここでは一方のクラスタをプライマリに、もう一方のクラスタをセカンダリに指定します。 読み取り操作と書き込み操作はどちらのクラスタにも送信できますが、競合の発生を特定し、一貫性のない状態を解消するのはプライマリの責務となります。

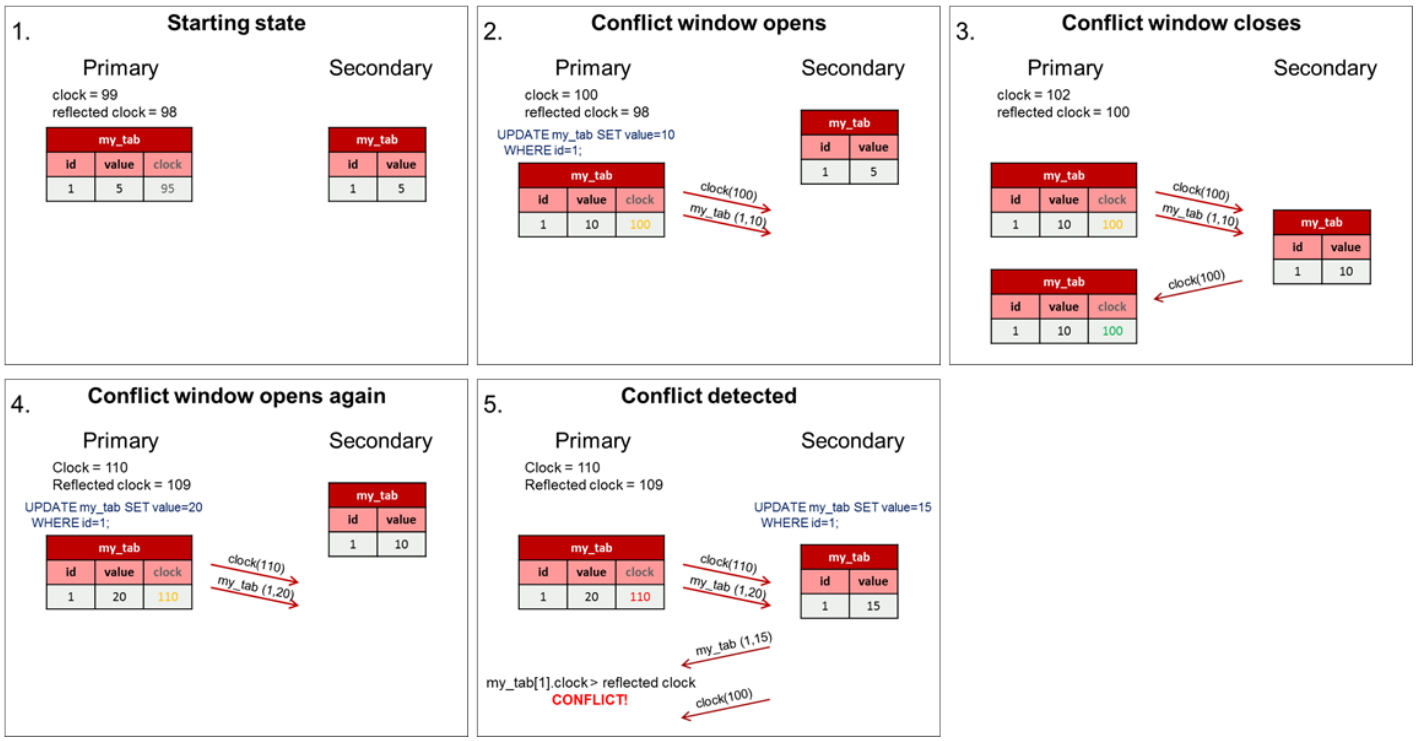

図8 アクティブ/アクティブ非同期レプリケーションでデータの一貫性が失われる状況

論理クロックを使用して、変更がプライマリ上で行われたことを（相対的な時間で）識別します。MySQL Cluster の内部仕様に精通しているユーザー向けにさらに説明すると、このクロック値には更新内容が含まれるグローバル・チェックポイントのインデックスが使用されます。 この機能がオンになっているすべてのテーブルで、非表示列がプライマリ上で自動的に追加されます。この非表示列は、変更時の論理クロック値を表します。

プライマリ上で変更が行われると、影響を受ける行に対して"競合期間"が発生します。この期間にセカンダリ上の同じ行に対して異なる変更が行われれば、一貫性のない状態となります。 セカンダリ上のスレーブでプライマリから送信された変更が適用されると、セカンダリ上のスレーブはプライマリ上のスレーブにレプリケーション・イベントを送り返します。このイベントには、セカンダリ上で適用された変更内容と関連付けられているプライマリのクロック値が含まれます（クロックは実際にはグローバル・チェックポイントのインデックスであるため、この機能は Reflected GCI（反映済み GCI）とも呼ばれます）。 プライマリ上のスレーブでこのイベントを受信すると、スレーブは、クロック値のタグが付けられたすべての変更について、そのクロック値が Reflected GCI の値よりも小さいため安全である、つまり競合期間が終了していることを認識します。

アプリケーションでセカンダリ上の同じ行を変更し、その後プライマリからのレプリケーション・イベントが適用される場合には、セカンダリは関連するレプリケーション・イベントをプライマリ上のスレーブに送信し、その後新しい GCI を反映します。プライマリ上のスレーブはこのレプリケーション・イベントを処理し、影響を受ける行

に記録されているクロック値を、最新の Reflected GCI と比較します。この場合、競合する行のクロック値の方が大きいため、プライマリは競合の発生を認識し、一貫性のない状態を解消するためのアルゴリズムを起動します。

一貫性のない状態の解消

MySQL Cluster の以前のリリースでは（または、MySQL Cluster 7.2 で従来のアルゴリズムを使用する場合は）、単純に競合する行の主キーにフラグを付ける、または競合する行に対する変更の 1 つをバックアウト（取り消す）するという処理が実行されました。 新機能の NDB$EPOCH_TRANS 関数を使用すると、プライマリによって、セカンダリ内の影響を受けた行、および同じトランザクションで更新されたその他の行のデータ（競合の検知がオフになっているテーブルにあるデータも含む）が上書きされるようになります。

実際には、競合解消アルゴリズムはさらに一歩踏み込んで、競合している行に対してセカンダリ上の後続トランザクションが書き込みを行っている場合に、セカンダリ上のこのような依存トランザクションによる変更のすべてが同様にロールバックされます。

高度な競合解消ソリューションの実装方法

このセクションでは、図 9 に示すように 2 クラスタ間でレプリケーションがセットアップされていることを前提としています。 この構成をセットアップする方法の詳細については、ブログ記事の『Enhanced conflict resolution with MySQL Cluster active-active replication』[4]を参照してください。

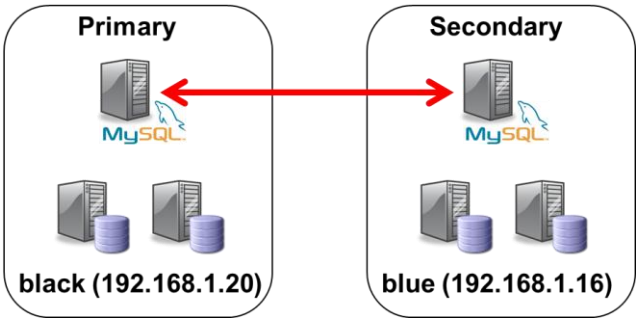

図9 レプリケーション構成

説明を分かりやすくするために、ここでは 2 つのホストのみを使用しています。"black"にはプライマリ・クラスタの全ノードが、また、"blue"にはセカンダリ・クラスタの全ノードが含まれます。 さらに分かりやすくするため、各クラスタでは単一の MySQL サーバーがマスターとしてもスレーブとしても動作するものとします。

最初の手順として、競合の検出を有効にする必要があるテーブルを識別します。 次に、識別したそれぞれのテーブルのエントリを mysql.ndb_replication テーブルに作成する必要があります。各エントリで、新機能の NDB$EPOCH_TRANS()関数を使用するようにタグ付けします。また、トランザクション全体ではなく競合する行に対する変更のみを対象とする場合は、NDB$EPOCH()関数を使用できます。 次の点に注意してください。

- この操作は、アプリケーションのテーブル自体を作成する前に実行する必要がある
- プライマリのみで実行する
- デフォルトではこのテーブルは存在していないため、最初にこのテーブルを作成する必要がある

```
black-mysql> CREATE TABLE mysql.ndb_replication  (
    ->     db VARBINARY(63),
    ->     table_name VARBINARY(63),
    ->     server_id INT UNSIGNED,
```

[4] http://www.clusterdb.com/mysql-cluster/enhanced-conflict-resolution-with-mysql-cluster-active-active-replication/

```
    ->     binlog_type INT UNSIGNED,
    ->     conflict_fn VARBINARY(128),
    ->     PRIMARY KEY USING HASH (db, table_name, server_id)
    -> )   ENGINE=NDB
    -> PARTITION BY KEY(db,table_name);
black-mysql> INSERT INTO mysql.ndb_replication VALUES ('clusterdb', 'simple1', 8, 0,
'NDB$EPOCH_TRANS()');
black-mysql> INSERT INTO mysql.ndb_replication VALUES ('clusterdb', 'simple2', 8, 0,
'NDB$EPOCH_TRANS()');
black-mysql> INSERT INTO mysql.ndb_replication VALUES ('clusterdb', 'simple3', 8, 0,
'NDB$EPOCH_TRANS()');
```

識別したテーブルのそれぞれに対して、例外テーブルも作成する必要があります。例外テーブルには、変更のロールバックにつながった競合が記録されます。例外テーブルの形式は厳密に定義されているため、データ型を正確にコピーするように注意してください。この例外テーブルも、プライマリのみで実行します。

```
black-mysql> CREATE DATABASE clusterdb;USE clusterdb;
black-mysql> CREATE TABLE <strong>simple1$EX</strong> (server_id INT UNSIGNED,
               master_server_id INT UNSIGNED, master_epoch BIGINT UNSIGNED,
               count INT UNSIGNED, id INT NOT NULL, PRIMARY KEY(server_id,
               master_server_id, master_epoch, count)) ENGINE=NDB;
black-mysql> CREATE TABLE <strong>simple2$EX</strong> (server_id INT UNSIGNED,
               master_server_id INT UNSIGNED, master_epoch BIGINT UNSIGNED,
               count INT UNSIGNED, id INT NOT NULL, PRIMARY KEY(server_id,
               master_server_id, master_epoch, count)) ENGINE=NDB;
black-mysql> CREATE TABLE <strong>simple3$EX</strong> (server_id INT UNSIGNED,
               master_server_id INT UNSIGNED, master_epoch BIGINT UNSIGNED,
               count INT UNSIGNED, id INT NOT NULL, PRIMARY KEY(server_id,
               master_server_id, master_epoch, count)) ENGINE=NDB;
```

これで、アプリケーション・テーブル自体を作成できます（セカンダリにはレプリケーションされるため、プライマリのみで実行します）。

```
black-mysql> CREATE TABLE simple1 (id INT NOT NULL PRIMARY KEY, value INT) ENGINE=ndb;
black-mysql> CREATE TABLE simple2 (id INT NOT NULL PRIMARY KEY, value INT) ENGINE=ndb;
black-mysql> CREATE TABLE simple3 (id INT NOT NULL PRIMARY KEY, value INT) ENGINE=ndb;
```

セットアップはこれで完了です。この新しい構成に対してテストを実施して、競合が検出され、適切な更新対象がロールバックされることを確認できます。

高度な競合の検出/解消のテスト

最初の手順として、新しく作成したテーブルにデータを追加します（この時点でレプリケーションが実行されるため、プライマリのみで作成します）。次に、1行更新して、セカンダリにその行がレプリケーションされることを確認します。

```
black-mysql> INSERT INTO simple1 VALUES (1,10);
black-mysql> INSERT INTO simple2 VALUES (1,10);
black-mysql> INSERT INTO simple3 VALUES (1,10);
black-mysql> UPDATE simple1 SET value=12 WHERE id=1;
```

```
blue-mysql> USE clusterdb;
blue-mysql> SELECT * FROM simple1;
+----+-------+
| id | value |
+----+-------+
|  1 |    12 |
+----+-------+
```

重要なのは、NDB$EPOCH_TRANS()関数によって、競合に関係するセカンダリ上のトランザクション（さらに、それに後続する、同じ行を変更する依存トランザクション）がロールバックされるという点です。これを手動で行うための最も簡単な方法は、競合期間を長くするためにセカンダリ上のスレーブ IO スレッドを停止することです（この操作を行わなければ、競合期間は非常に短くなります）。 スレーブ IO スレッドの停止後は、プライマリ上でテーブル simple1 に変更を行い、セカンダリ上で同じ行に対して（競合する）変更を行い、さらに同じトランザクションでテーブル simple2 に対しても変更を行います。 次に、プライマリ上の 2 つ目のトランザクションで simple3 の行に変更を加えます。このトランザクションでは競合に関係する行に対する変更は行われないため、simple3 に対する変更内容はそのまま残るはずです。

```
blue-mysql> STOP SLAVE IO_THREAD;


black-mysql> UPDATE simple1 SET value=13 WHERE id=1;


blue-mysql> BEGIN; # 競合するトランザクション
blue-mysql> UPDATE simple1 SET value=20 WHERE id=1;
blue-mysql> UPDATE simple2 SET value=20 WHERE id=1;
blue-mysql> COMMIT;
blue-mysql> UPDATE simple3 SET value=20 WHERE id=1; # 競合なし
blue-mysql> SELECT * FROM simple1;
+----+-------+
| id | value |
+----+-------+
|  1 |    20 |
+----+-------+
blue-mysql> SELECT * FROM simple2;
+----+-------+
| id | value |
+----+-------+
|  1 |    20 |
+----+-------+
blue-mysql> SELECT * FROM simple3;
+----+-------+
| id | value |
+----+-------+
|  1 |    20 |
+----+-------+
```

ここで例外テーブルを確認すると、プライマリ（black）でセカンダリ（blue）による変更を受信しており、最初のトランザクションが競合期間中に simple1 の同じ行を更新しているため、変更をロールバックする必要があることが記録されています。このロールバックは、セカンダリ上でレプリケーション・スレッドが再起動された直後に行われることになります。

```
black-mysql> SELECT * FROM simple1$EX;
+-----------+------------------+---------------+-------+----+
| server_id | master_server_id | master_epoch  | count | id |
+-----------+------------------+---------------+-------+----+
|         8 |                9 | 1494648619009 |     3 |  1 |
+-----------+------------------+---------------+-------+----+

black-mysql> SELECT * FROM simple2$EX;
+-----------+------------------+---------------+-------+----+
| server_id | master_server_id | master_epoch  | count | id |
+-----------+------------------+---------------+-------+----+
|         8 |                9 | 1494648619009 |     1 |  1 |
+-----------+------------------+---------------+-------+----+

black-mysql> SELECT * FROM simple3$EX;
Empty set (0.05 sec)


blue-mysql> START SLAVE IO_THREAD;
blue-mysql> SELECT * FROM simple1;
+----+-------+
| id | value |
+----+-------+
|  1 |    13 |
+----+-------+

blue-mysql> SELECT * FROM simple2;
+----+-------+
| id | value |
+----+-------+
|  1 |    10 |
+----+-------+

blue-mysql> SELECT * FROM simple3;
+----+-------+
| id | value |
+----+-------+
|  1 |    20 |
+----+-------+
```

これは期待していたとおりの結果です。つまり、simple1 にはプライマリによって設定された値が含まれ、セカンダリでの後続の変更はロールバックされています。また、simple2 はプライマリによって更新されておらず、セカンダリ上の変更は simple1 に対する競合する更新と同じトランザクションで行われているためにロールバックされています。 セカンダリでの simple3 に対する変更は、競合するトランザクションの外部で実行されており、競合する変更には依存していないため、そのまま残っています。 最後に、プライマリ上でも同じデータであることを確認します。

```
black-mysql> SELECT * FROM simple1;
+----+-------+
| id | value |
+----+-------+
|  1 |    13 |
+----+-------+

black-mysql> SELECT * FROM simple2;
+----+-------+
| id | value |
+----+-------+
|  1 |    10 |
+----+-------+
```

```
black-mysql> SELECT * FROM simple3;
+----+-------+
| id | value |
+----+-------+
|  1 |    20 |
+----+-------+
```

プライマリ上の統計情報には、1 個の競合が検出され、1 個のトランザクションに影響し、その結果 2 行に対する変更がロールバックされたことが記録されています。

```
black-mysql> SHOW STATUS LIKE 'ndb_conflict%';
+----------------------------------------+-------+
| Variable_name                          | Value |
+----------------------------------------+-------+
| Ndb_conflict_fn_max                    | 0     |
| Ndb_conflict_fn_old                    | 0     |
| Ndb_conflict_fn_max_del_win            | 0     |
| Ndb_conflict_fn_epoch                  | 0     |
| Ndb_conflict_fn_epoch_trans            | 1     |
| Ndb_conflict_trans_row_conflict_count  | 1     |
| Ndb_conflict_trans_row_reject_count    | 2     |
| Ndb_conflict_trans_reject_count        | 1     |
| Ndb_conflict_trans_detect_iter_count   | 1     |
| Ndb_conflict_trans_conflict_commit_count | 1   |
+----------------------------------------+-------+
```

ユーザー権限の統合

MySQL Cluster 7.2 では、ユーザー・アクセスに必要なシステム・データをすべての MySQL サーバーと共有できる新しい機能が追加されました。デフォルトでは、これらの証明情報を保持しているすべてのテーブルは MyISAM、つまり MySQL サーバーのローカルに格納されています。

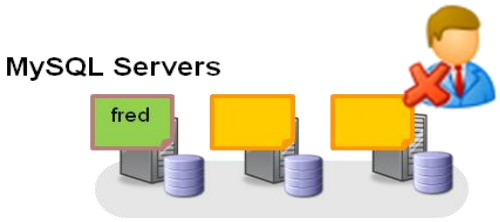

図 10 サーバーごとの権限

この場合、管理が困難になります。つまり、新しいユーザーを作成およびそれらの権限を変更する場合、それをすべてのサーバーで繰り返さなければなりません。1 つでも設定を省いてしまうと、ユーザーはそのサーバーにアクセスすることができません（または、それらの権限を消去すれば引き続きアクセスできます）。

これは図 10 に示されています。 ユーザー「fred」が 1 つの MySQL サーバーで作成されましたが、Fred が他の MySQL サーバーにアクセスしようと試みたときにブロックされました。これは意図的な場合もあるかもしれませんが、通常 DBA は、すべての MySQL サーバーに変更を反映したいはずです。

このセクションでは、変更をすべての MySQL サーバーで共有する方法を説明する前に、このデフォルトの動作について説明します。より詳しい要件については、ブログ記事[5] をご覧ください。

[5] http://www.clusterdb.com/mysql-cluster/sharing-user-credentials-between-mysql-servers-with-cluster/

ユーザー「fred」が server1 で作成され、test テーブルが root ユーザーによって作成されます。

```
$ mysql -h 192.168.1.1 -P3306 -u root --prompt 'server1-root> '
server1-root> GRANT ALL ON *.* TO 'fred'@'192.168.1.1';
server1-root> CREATE DATABASE clusterdb; USE clusterdb;
server1-root> CREATE TABLE towns (id INT NOT NULL PRIMARY KEY, town VARCHAR(20))
ENGINE=NDBCLUSTER;
server1-root> INSERT INTO towns VALUES (1,'Maidenhead'),(2, 'Reading');
```

次に、新規ユーザー「fred」が server1 を介して接続し、データにアクセスできることを確認します。

```
$ mysql -h 192.168.1.1 -P3306 -u fred --prompt 'server1-fred> '
server1-fred> SELECT * FROM clusterdb.towns;
+----+------------+
| id | town       |
+----+------------+
|  1 | Maidenhead |
|  2 | Reading    |
+----+------------+
```

同じ認証情報が 2 番目のサーバー (server 2)に使用された場合、接続できません。

$ mysql -h 192.168.1.2 -P3306 -u fred --prompt 'server2> '

server2-fred> SELECT * FROM clusterdb.towns;

ERROR 1142 (42000): SELECT command denied to user ''@'ws2.localdomain' for table 'towns'

次に必要な作業は、スクリプトを実行し（MySQL root として）、そしてストアド・プロシージャを実行して、mysql データベースの 5 つのテーブル(“user”、“db”、“tables_priv”、“columns_priv” および “procs_priv”)を MyISAM から ndbcluster ストレージエンジンに変更します。

```
server1-root> SOURCE /usr/local/mysql/share/ndb_dist_priv.sql;
server1-root> CALL mysql.mysql_cluster_move_privileges();
```

これでユーザー権限テーブルのストレージエンジンは ndbcluster に変更されました。これは次のように確認できます。

```
server1-root> SHOW CREATE TABLE mysql.user\G
*************************** 1. row ***************************
       Table: userCreate Table: CREATE TABLE `user` (
  `Host` char(60) COLLATE utf8_bin NOT NULL DEFAULT '',
....
....
 ) ENGINE=ndbcluster DEFAULT CHARSET=utf8
COLLATE=utf8_bin COMMENT='Users and global privileges'
```

これらのテーブルは MySQL Cluster に格納されており、全ての MySQL サーバーから見ることができます。 どの MySQL サーバーにユーザーがアクセスを試みても、MySQL サーバーは権限データをローカル情報でなく共有データ・ノードから取得するため、ユーザーは同じアクセス権限を得ます。clusterdb.towns は ndbcluster を使用して作成されてい

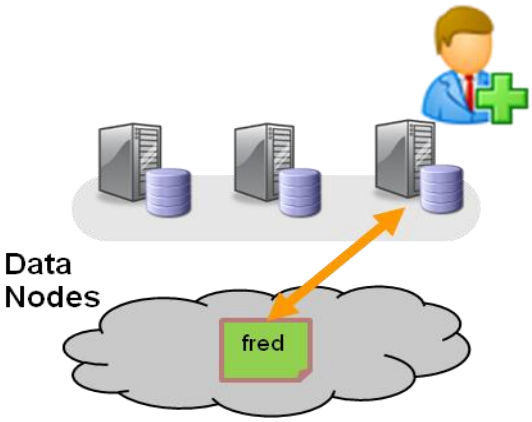

図 11 サーバー間で共有されるデータ

るため、全てのサーバーからアクセス可能で、server 2 の権限からも許可されるため、ユーザーもテーブルの内容を server 2 から見ることができます。これらの 5 つの mysql テーブルに格納済みのデータは、MyISAM から MySQL Cluster への移行時も維持されます。

最後のテストは、新規ユーザーが、server 2 からこのデータにアクセスできるかを確認することです。

```
$ mysql -h 192.168.1.2 -P3306 -u fred --prompt 'server2-fred>
server2-fred> SELECT * FROM clusterdb.towns;
+----+------------+
| id | town       |
+----+------------+
|  1 | Maidenhead |
|  2 | Reading    |
+----+------------+
```

ユーザーがすでにサーバーに接続していて、権限が他のサーバーによって変更された場合、ユーザーは一旦接続を切り、再接続して新しい権限が適用されるようにする必要があります。

MySQL 5.5サーバーとの統合

MySQL Cluster 7.2 は MySQL 5.5 と統合され、既存の MySQL サーバーの運用とバイナリ互換性があります。ユーザーは、 InnoDB および MySQL Cluster ストレージエンジン両方の最新の機能を、1 つのアプリケーションでフルに利用可能です。

ユーザーは、MySQL 5.5 リリースを含む MySQL Cluster バイナリをインストールし、再起動するだけで、すぐに InnoDB および MySQL Cluster の両方へアクセスできます。

Virtual Machines（仮想環境）

MySQL Cluster は、オンプレミスおよびクラウド・プラットフォーム上の両方の仮想マシン（Virtual Machine）環境での運用がさサポートされ[6] 、運用環境オプションが広がりました。

MySQL Cluster は、Xen-ベースの Oracle VM Server[7]での運用が保障されています。

MySQL Enterprise Monitor - MySQL Cluster向け機能拡張

MySQL Enterprise Monitor (MEM) は、OS やソフトウェアのリソースの状況を表示するグラフや、警告を発生するためのルールからなるアドバイザを、MySQL Cluster 向けに多く含んでいます。オリジナルのグラフやルールについては、 “Monitoring MySQL Cluster with MySQL Enterprise Monitor”[8] に詳く書かれていますが、さらに機能が拡張されています。

---

[6] VM 環境のサポートは、関連 VM ベンダーまたは承認されているパートナーから受ける必要があります。

[7] http://www.oracle.com/us/technologies/virtualization/oraclevm/index.html

[8] http://www.clusterdb.com/mysql-cluster/monitoring-mysql-cluster-with-mysql-enterprise-monitor/

Cluster データ・ノードが再起動しました

この新しい警告は、データ・ノードが再起動したときに発せられます（デフォルトでは、過去 10 分以内に開始した任意のデータ・ノードについて警告を発しますが、このインターバルは必要に応じて変更可能です）。再起動を手動で行なった場合（たとえばローリング・アップグレードの一部として）この警告を無視できます（または一時的にこのスケジュールを解除可能です）。この再起動が自発的なものである場合、問題が大きくなる前に、エラーログを確認して対処すべきという警告である可能性があります。

Cluster DiskPageBuffer ヒット率が低い（および 関連グラフ）

Disk Page Buffer は、ディスクベース・テーブルと共に使用される各データ・ノードのキャッシュです。他のキャッシュ同様、ヒット率が高いほどパフォーマンスが高いことを示します。このキャッシュのサイズをチューニングすることでシステムに大きな効果があります。グラフで変更結果を容易に確認可能で、ヒット率が許容範囲より低くなると、警告が発せられます。たとえば、データ・ノードが再起動したあと一時的に、または長期間アクティブ・データセットが増加した場合に発生します。

ndbinfo データベースに新しいテーブル “diskpagebuffer”が追加されましたが、これはヒット率を計算するために必要な基本情報で、新しいグラフおよびアドバイザの基となるものです。このテーブルから手動でキャッシュのヒット率を計算するには、次のクエリーを使用します。

```
mysql> SELECT node_id, page_requests_direct_return AS hit,
 page_requests_wait_io AS miss,  100*page_requests_direct_return/
 (page_requests_direct_return+page_requests_wait_io) AS hit_rate
  FROM ndbinfo.diskpagebuffer;

+---------+------+------+----------+
| node_id | hit  | miss | hit_rate |
+---------+------+------+----------+
| 3       | 6    | 3    | 66.6667  |
| 4       | 10   | 3    | 76.9231  |
+---------+------+------+----------+
```

まず最初に、ヒット率が 97%を下回ると「情報」レベルの警告が発せられ、90%まで下がると「警告」、80%になると「重要」にまでレベルが上がります。これらの閾値は変更可能です。

グラフには、単純に時間的に変化するヒット率が表示されますので、その傾向を確認できます。

MySQL Cluster Carrier-Grade Edition - 主要コンポーネント

MySQL Cluster Community Server は、無償で自由にダウンロードして利用できる MySQL データベースのバージョンで、ライセンスは GPL です。

Oracle では、商用版のサブスクリプションおよびライセンスである MySQL Cluster Carrier Grade Edition (CGE) も提供しており、これは稼働環境用データベースと、バックアップ、監視およびモデリング/開発/管理ツール、および技術サポートなどのサービスを包括的に含んだセットです。MySQL Cluster CGE を使用することで、最高レベルのパフォーマンス、信頼性、そしてアップタイムを達成することが可能になります。 MySQL Cluster CGE には、主要なコンポーネントとして MySQL Cluster Manager および Oracle Premier Support が含まれます。

MySQL Cluster Manager

MySQL Cluster Manager (MCM)[9] は、基本的な共通管理タスクを自動化するため、MySQL Cluster データベースの構築と管理が大幅に簡略化されます。その結果、データベース管理者およびシステム・アドミニストレータの生産性が向上し、戦略的な IT 構想により注力できます。これまで手動による構成エラーが原因で発生していたダウンタイムのリスクを大幅に軽減することもできます。

以下で、MCM の 2 つの機能について説明します。

1 ステップ Cluster 作成 (bootstrap オプション)

MySQL Cluster Manager は、Oracle Software Delivery Cloud[10] からダウンロードでき、スタンド・アロン・パッケージと MySQL Cluster を含んだパッケージがあります。全てを含んだセットをダウンロードすれば、非常に容易に 1 ホストの MySQL Cluster 運用環境を構築して、機能を評価できます。

クラスタを作成して起動するには、ダウンロードしたアーカイブを解凍し、MCM daemon に“bootstrap” オプションを使用します。

```
D:\Andrew\Documents\MySQL\mcm> bin\mcmd --bootstrap
MySQL Cluster Manager 1.1.2 started
Connect to MySQL Cluster Manager by running "D:\Andrew\Documents\MySQL\mcm\bin\mcm" -a
NOVA:1862
Configuring default cluster 'mycluster'...
Starting default cluster 'mycluster'...
Cluster 'mycluster' started successfully
 ndb_mgmd NOVA:1186
 ndbd NOVA
 ndbd NOVA
 mysqld NOVA:3306
 mysqld NOVA:3307
 ndbapi *
Connect to the database by running "D:\Andrew\Documents\MySQL\mcm\cluster\bin\mysql" -h NOVA
-P 3306 -u root
```

詳しくはこちらのブロクを参照ください。[11]

[9] http://www.mysql.com/products/cluster/mcm/

[10] https://edelivery.oracle.com/

[11] http://www.clusterdb.com/mysql-cluster/mysql-cluster-manager-1-1-2-creating-a-cluster-is-now-trivial/

オンラインでのノードの自動追加

MySQL Cluster 7.0 以降、稼働中の Cluster への新たなノードの追加が可能になりました。これには多くのステップがあり、オンラインでのアップグレードと同じく、管理者がミスをしてしまうと、システム停止になる可能性があります。

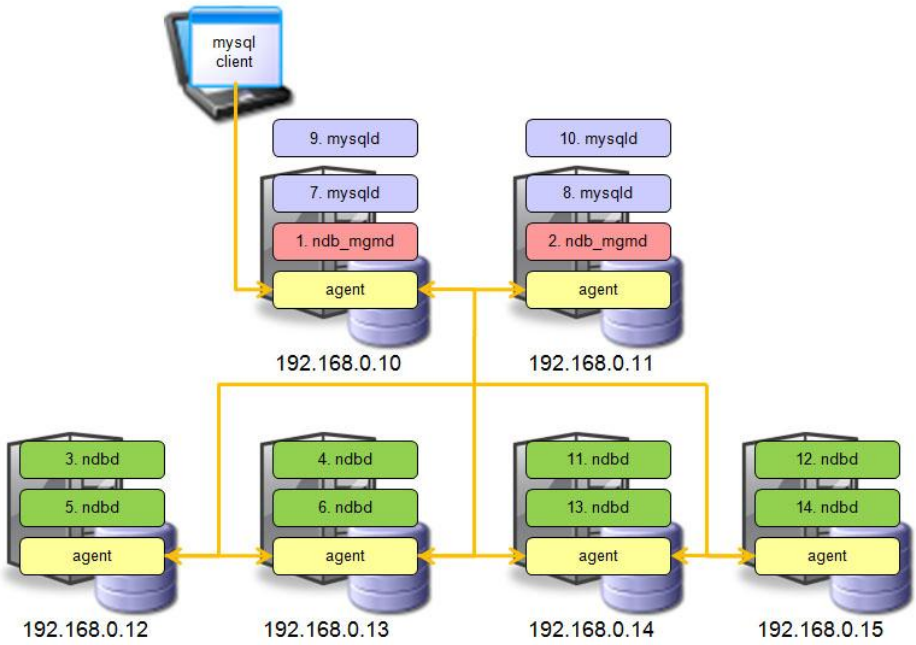

図 12 実行中のクラスタへの 2 つのデータ・ノードの追加

MySQL Cluster Manager を使用すると、このプロセスが自動化されます。最初のステップとして、任意の新しいホスト（サーバー）をサイトに追加し、これらのホストがどこで Cluster ソフトウェアを見つけられるかを示します。

```
mcm> add hosts --hosts=192.168.0.14,192.168.0.15 mysite;
mcm> add package --basedir=/usr/local/mysql_7_2_1
  --hosts=192.168.0.14,192.168.0.15 7_2_1;
```

そして、新規ノードを Cluster に追加し再起動します。

```
mcm> add process --processhosts=mysqld@192.168.0.10,mysqld@192.168.0.11,ndbd@192.168.0.14,
  ndbd@192.168.0.15,ndbd@192.168.0.14,ndbd@192.168.0.15 mycluster;
mcm> start process --added mycluster;
```

Cluster は拡張されましたが、最後のステップとして、任意の MySQL サーバーより既存の Custer テーブルの再パーティショニングを行い、新規データ・ノードを使用するようにする必要があります。

```
mysql> ALTER ONLINE TABLE <table-name> REORGANIZE PARTITION;
mysql> OPTIMIZE TABLE <table-name>;
```

Oracle Premier Support

オラクルは、MySQL のグローバル・サポートを 24 時間 365 日提供します。MySQL サポート・チームは、様々な問題や課題を解決してきた豊富な経験をもつ熟練したエンジニアから構成されるため、お客様が直面する問題もすばやく理解して対応することが可能です。

MySQL の Oracle Premier Support には以下のサービスが含まれます。

- 24 時間 365 日プロダクション・サポート（日本語対応は 9:00-17:00）
- 無制限サポート・インシデント
- ナレッジベース
- メンテナンス・リリース、バグ修正、パッチ、アップデートの提供
- MySQL コンサルティング・サポート

まとめ

このホワイトペーパーでは、MySQL Cluster データベースの主要な向上機能について、詳しく説明しました。

- アダプティブ・クエリー・ローカライゼーション（AQL）
- ネイティブ Memchached API NoSQL インタフェース
- パフォーマンスの向上
- マルチサイト・クラスタリング
- アクティブ/アクティブ・レプリケーションの簡素化
- ユーザー権限管理の統合
- MySQL 5.5 サーバーとの統合
- 仮想マシン環境のサポート
- MySQL Enterprise Monitor – MySQL Cluster 向け機能拡張
- オンライン・ノード追加の自動化
- 1 ステップ Cluster 作成(bootstrap)

新規特徴および機能の完全なリストについては、こちらのドキュメントの MySQL Cluster 変更ログを参照してください。

http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-news.html

これらの新しい機能により、MySQL Cluster は、コモディティ・ハードウェアを利用した高スケーラビリティ、99.999%のアップタイムおよびリアルタイム・パフォーマンス必要とする事例およびアプリケーションなど、より広範囲に対応できるようになりました。

関連情報

MySQL Cluster ダウンロード:
http://www-jp.mysql.com/downloads/cluster/

MySQL Cluster Manager トライアル (関連情報セクションを参照ください):
http://www-jp.mysql.com/products/cluster/mcm/

オンライン・デモ - MySQL Cluster in Action:
http://www-jp.mysql.com/products/cluster/cluster_demo.html

ホワイトペーパー:

MySQL Cluster スケーリング Web データベース・ガイド
http://mysql.com/why-mysql/white-papers/mysql_wp_scaling_web_databases.php

MySQL Cluster 評価ガイド
http://www-jp..mysql.com/why-mysql/white-papers/mysql_cluster_eval_guide.php

MySQL による NoSQL ガイド
http://www-jp..mysql.com/why-mysql/white-papers/mysql-wp-guide-to-nosql.php

MySQL Cluster Manager ガイド
http://www-jp..mysql.com/why-mysql/white-papers/mysql_wp_cluster_manager.php

MySQL Cluster データベース・パフォーマンス最適化ガイド
http://www-jp..mysql.com/why-mysql/white-papers/mysql_wp_cluster_perfomance.php

日本オラクル株式会社

〒107-0061 東京都港区北青山 2-5-8 オラクル青山センター
oracle.com/jp
http://www-jp.mysql.com (MySQL)

[お問い合わせ窓口]
Oracle Direct
0120-155-096
oracle.co.jp/direct
mysql-sales_jp@oracle.com