解説

NoSQLの世界

松下雅和（(株)オープンストリーム）

NoSQL ～ Eric Evans のブログから～

☆ NoSQL の広まり

「NoSQL」，その言葉は 2009 年に開催された「NOSQL meetup」というイベントをきっかけに広く知れ渡った[1]．さまざまな非リレーショナルモデルのデータベース（以降，非 RDB と略す）の開発者が集まるイベントで，イベント名の名付け親は Rackspace の Eric Evans 氏である．Evans 氏は当時のことを

『Johan Oskarsson が最初のミーティングを企画したときに「何かいい名前ないか？」って尋ねられたんだ．そのときに何も考えずに 45 秒ぐらいで思いついた 3, 4 つの名前の 1 つなんだよ』

と自身のブログで振り返っている．それほど深い意味は持たずに名付けられた NoSQL だったが，このイベントがきっかけで，非 RDB を指す言葉として広く知れ渡るようになる．

一方で，言葉の響きから「SQL 不要」「RDB 否定」といったネガティブなイメージを持たれてしまい，RDB に相対するデータモデルかのような認識も広まってしまった．それに対し，Emil Eifrem 氏の

『「NoSQL」は「Not Only SQL」の略として考えたらどうだろうか』

という Twitter のつぶやきが Evans 氏を含む多くの人から賛同を得られたため，現在は「NoSQL ＝ Not Only SQL（SQL に限らない）」という理解が一般化しつつある．今でも最適な言葉を巡る議論が続いているため，今後別の言葉に置き換えられる可能性もあるが，非 RDB が注目されている点は疑う余地はない．

☆ NoSQL とは?

ではなぜ NoSQL が最近になって注目を浴びだしたのか？

従来は，更新頻度は高いが小規模，あるいは大規模でも更新はあまり行われないシステムがほとんどだった．こうしたシステムでは，RDB を利用して SQL によるデータ操作を行うことで，ほとんどすべての要件を満たすことができた．

しかし近年では，スキーマレス／スケーラビリティ／高可用性／大量データ処理など，RDB には向かないさまざまな要件が出てきた．こうした要件に対応するため，さまざまなデータモデルの DB が開発されたことが，NoSQL に注目が集まった背景にある．

以下は，NoSQL の多くが備えている特徴である．

- 明示的なスキーマを必要としない
- join 操作を必要としない
- 水平スケーラビリティ（スケールアウト）が確保されている
- 単一障害点がない
- ACID を必ずしも要求しない
- 大量データ，分散処理に向いている
- 何らかの処理に特化している

☆具体的なデータモデル

表 -1 は，執筆時点での主要な NoSQL のデータモデルとそのプロダクトである[2]．

このほか，オブジェクト／グリッド／ XML とい

分類	プロダクト	説明
KVS	Amazon SimpleDB	Amazon Web Services のストレージサービス
	Amazon Dynamo	Amazon が自社サービスで利用．非公開
	Azure Table Storage	Windows Azure のストレージサービス
	Tokyo Cabinet/Tyrant	mixi が自社サービスで利用．LGPL
	Voldemort	Amazon Dynamo のオープンソース実装．Apache License 2.0
	Dynomite	Amazon Dynamo のオープンソース実装．独自ライセンス
	KAI	Amazon Dynamo のオープンソース実装．goo ホームで採用
	Oracle Berkeley DB	Oracle が提供するプロダクト
	Membase	memcached 互換のプロダクト．Apache License 2.0
	Roma	楽天が自社サービスで利用．GPLv3
	kumofs	えとらぼが自社サービスで公開．Apache License 2.0
	Flare	GREE が自社サービスで利用．GPLv2
列指向	Google Bigtable	Google が自社サービスで利用．非公開
	Apache Cassandra	Bigtable のデータモデル +Dynamo の分散技術．Apach License 2.0
	HBase	Hadoop に Bigtable に近い機能を提供．Apache License 2.0
	Hypertable	Bigtable 互換のオープンソース実装．GPLv2
ドキュメント指向	Apache CouchDB	Erlang による実装．Apache License 2.0
	MongoDB	C++ による実装．10gen によるサポートあり．AGPLv3
グラフ指向	Neo4J	Java による実装．組込み型．AGPLv3
	HyperGraphDB	Java による実装．組込み型．内部で BerkeleyDB を利用．LGPL

表 -1　NoSQL プロダクト一覧

ったデータモデルも存在するが，前述した NoSQL の特徴を備えていないものもあり，NoSQL に含めるかどうかは今でも議論が分かれている．

ここで紹介したデータモデルやプロダクトは執筆時点のものであり，今後もますますその種類は増えていくだろう．

NoSQL の適用分野

NoSQL はすでにさまざまなシステムで利用されている．

たとえば，Google の Bigtable は Google の分散ストレージ・システムを支える重要な技術であり，Google 検索はもちろん，YouTube/Google Maps/Google App Engine などでも利用されている．数 PB（ペタバイト）にも達するデータを数万～数十万のサーバに分散格納しており，他社には真似できない圧倒的なスケーラビリティと高可用性を実現している．国内でも，mixi の Tokyo Tyrant，GREE の Flare，楽天の ROMA のように，インターネット関連企業を中心にそれぞれ独自の NoSQL を開発し，積極的に採用を進めている．

ただし，すべてのシステムに NoSQL が適用できるわけではない．プロダクトやデータモデルにもよるが，NoSQL の多くは ACID トランザクションを持たない．そのため，厳密なトランザクションを必要とするシステム（金銭の移動，受発注処理など）には不向きとされる．また RDB のように複雑なデータ操作は行えないため，NoSQL に格納したデータを多角的に分析／集計することも難しい．

そもそも，NoSQL は RDB を置き換えるものではない．それぞれの長所／短所，メリット／デメリットを把握した上で，対象のシステムに合わせて適材適所，場合によっては併用して利用するものである．先述した事例の多くも，主に大量データを扱いながら検索性能を高めるために，RDB を補完する形で NoSQL を採用している．

では，NoSQL の中でも今，最も注目されているデータモデルはご存知だろうか？　KVS である．先述した事例もすべて，KVS あるいは KVS をベースとした列指向の NoSQL に分類される．KVS は，「Key（キー）」と「Value（値）」のペアでデータを保持するシンプルなデータモデルであり，シンプルであるがゆえに拡張性／可用性が高く，運用コストも低い点などが注目されている．

NoSQL に対する理解を深めるために，次章では列指向に分類される Cassandra を紹介する．

Apache Cassandra の紹介

Cassandra は元々 Facebook 社で開発されていたプロダクトで，2008 年に OSS として公開，2009 年に Apache ファウンデーションに寄贈され，2010 年に Apache のトッププロジェクトに昇格した．執筆時点では，Facebook のほかに Digg や Twitter でも採用されている．Google Bigtable のような列指向のデータモデルと，Amazon Dynamo の分散の仕組みを合わせ持った NoSQL として，特に注目されているプロダクトである．先述した「NoSQL の多くが備えている特徴」のほとんどを備えている上に，すべて Java で実装されているためあらゆる環境で実行可能であり，クライアントとの通信も Thrift というフレームワークで主要な言語はすべてサポートされている．

Cassandra を利用する場合のセットアップ方法は，以下の通りである．

ダウンロードは The Apache Cassandra Project[3] から行う．執筆時点での最新バージョンは 0.6.3 である．JVM がインストールされた環境であれば，バイナリ版を展開後，ログやデータの保存先を設定すればすぐに起動できる．詳細は Cassandra の公式サイトの「Getting Started」を参照していただきたい．

分散環境を構築する場合は，複数のマシン上に Cassandra を用意（1 つずつをノードと呼ぶ）し，環境に合わせて各ノードの設定ファイルに以下の項目を設定する．

- **Seed**
  自分以外のノードのホスト名または IP アドレスを，少なくとも 1 つ追加（すべてのノードを追加しなくても，Gossip プロトコルにより最終的にすべてのノードがお互いに認識される）
- **ListenAddress ／ StoragePort**
  他のノードと通信するための自ノードの IP アドレス／ポート

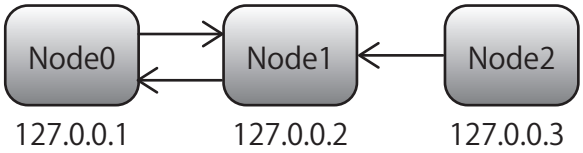

図 -1　Cassandra のノード構成例

- **ThriftAddress ／ ThriftPort**
  クライアントと通信するための自ノードの IP アドレス／ポート

テスト時などに複数のマシンの用意が大変な場合は，ローカル環境で複数ノードを立てることも可能である．詳細は筆者のブログ[4]（0.7 をベースとした内容のため，設定ファイルの記述などは適宜置き換える必要がある）で公開しているので，そちらをご参照いただきたい．

Apache Cassandra で NoSQL 体験

この章では，Cassandra を通じて NoSQL を体験していただく．Cassandra の特徴を理解するために，まず**図 -1** の構成で複数ノードを準備する．ここでは，Node0/Node1 は双方向で認識し合っているが，Node1/Node2 は片方向（Node2 → Node1）からのみ認識している．

初めに，Node0 と Node1 を起動する．Node0/Node1 はお互い認識し合っているため，この時点で Node0/Node1 のクラスタが構築される．

続いて，Node0 に対してデータを挿入する．ここでは，Cassandra に内包されている CLI ツールを利用する．

Keyspace	ColumnFamily	Key	Column	Value
Keyspace1	Standard1	matsukaz	name	matsukaz
			blog	http://d.hatena.ne.jp/matsukaz/
		hoge	name	hoge
			age	20

表 -2　挿入されたデータ(イメージ)

```
cassandra> set Keyspace1.
 Standard1['matsukaz']['name'] =
 'matsukaz'
cassandra> set Keyspace1.
 Standard1['matsukaz']['blog'] =
 'http://d.hatena.ne.jp/matsukaz/'
cassandra> set Keyspace1.
 Standard1['hoge']['name'] = 'hoge'
cassandra> set Keyspace1.
 Standard1['hoge']['age'] = '20'
```

上記の操作を行うと，Node0 に対して**表 -2** の多次元ハッシュのようなデータ構造でデータが挿入される．

このように，Cassandra では RDB のテーブルとは根本的に異なる，列指向と呼ばれるデータモデルが利用されている．KVS よりもリッチなデータモデルを提供しながら，明示的なスキーマを必要とせず，動的にデータ構造の追加／削除ができる点が，列指向に分類される NoSQL の大きな特徴である．ただし，同じ列指向の NoSQL であっても，プロダクトによってデータモデルが異なる点（名称／階層／API など）は注意が必要である．

ここで Node1 に対して検索してみると，Node0 と同じ結果が取得できる．Node0/Node1 のクラスタが構築されているためである．また，この状態で Node2 を起動すると，Gossip プロトコルと呼ばれる仕組みを利用することで，自動的に Node1 → Node0 の順に Node2 のノード情報が伝播され，Node0/Node1/Node2 のクラスタが構築される．以降は，どのノードに対して検索しても，同じ結果が取得できる．

ただし，このままではデータが一部のノードにしか存在しないため，データを保持しているノードが停止してしまうとデータが消滅してしまう．そこで，以下の項目が重要となる．

- **データの冗長性**：同じデータをいくつのノードに重複保持させるか．デフォルトは 1.
- **データの分割ルール**：各ノードにデータを分ける際の分割ルール．デフォルトはランダム．

上記の設定を適切に行えば，データが消滅するリスクを大幅に軽減できる．手軽に水平スケーラビリティの確保と単一障害点の回避ができる点も，Cassandra の大きな特徴と言える．

NoSQL の世界 ～システム開発に訪れる変化とは?～

前述した通り，NoSQL は RDB に置き換わる「銀の弾丸」ではない．まず対象システムの要件を洗い出し，RDB で実現できない要件なのかどうかで NoSQL の利用を判断する．NoSQL を利用すると判断した場合は，要件や CAP 定理（一貫性／可用性／分割耐性の 3 つの要素は同時に満たすことはできないという定理）で重視する要素をポイントに，データモデルやプロダクト，利用範囲（RDB との併用など）を検討する．

またシステム設計時も，以下の点を考慮する必要がある．

- **スキーマ設計**：データモデルに合わせた設計．
- **CRUD 分析**：RDB にはない，CAP 定理や BASE トランザクションといった概念．
- **セキュリティ**：RDB のプロダクトに比べて不十

分なものが多く，扱うデータによって対策が必要.

- **バックアップ**：データの保持形式に合わせたバックアップ方法．単一障害点がないプロダクトはバックアップしないという判断も.
- **障害対応**：多くのプロダクトはRDBとはリカバリ方法が異なる.
- **性能見積もり**：スケールアウトでリニアに性能向上するプロダクトが多く，データ量などによる従来の性能見積もりとは異なる.

そもそもフォーマットから作らなければならない設計書もあり，システム設計に与える影響は非常に大きい．実装やテストについても，RDBと同様のやり方はできないため，事前に検証が必要である.

ここまで挙げてきたポイントをすべて考慮することは難しい．プロダクト自体も未成熟な今，NoSQLを正式採用するには時期尚早かもしれない.

しかし，NoSQLを利用するメリットも確実にある．まずはデータモデルとプロダクトの特徴を押さえよう．そして，社内システムやバックエンドシステムなどの限られた環境で利用してみてはいかがだろうか？　システム開発に必要なスキーマ設計やCRUD分析，運用で必要なバックアップ方法など，どのようなストレージであっても最低限必要なポイントを押さえておけば，スモールスタートで利用することは可能である．そして徐々に利用範囲を広げることで，ノウハウの蓄積とともに上記で挙げたポイントも考慮できるようになるだろう.

最後に，こちらからNoSQLの利用方法を2点提案したい.

（1）Hadoopなどを利用した分散処理時（データ分析／ログ解析／データ収集など）のストレージとして

大量データを扱う場合にメリットを活かしやすい．バックエンドでの利用のため，要求されるサービスレベルは比較的低く，稼働中のシステムに対する影響も少ない.

（2）更新頻度が低い参照系データのストレージとして

memcachedなどのキャッシュの仕組みが一般化しつつあるが，NoSQLを利用するだけで十分な性能を提供できる可能性がある．機能や利用方法を限定することで，導入コストやリスクを低減できる.

今後NoSQLが広く利用されるようになれば，さらなる利用方法も考えられるだろう.

NoSQLはバズワードのようにも考えられているが，そこには確かなデータモデルとさまざまなストレージ技術や分散技術が取り入れられている．今後のシステム開発には，RDBとNoSQL，それぞれの良さを活かしたシステム構成が求められるようになるはずだ.

参考文献，引用元

1) Publickey -「NoSQL」は「Not Only SQL」である，と定着するか？，http://www.publickey1.jp/blog/09/nosqlnot_only_sql.html
2) NoSQL Databases，http://nosql-database.org/
3) http://cassandra.apache.org
4) http://d.hatena.ne.jp/matsukaz/

（平成22年7月31日受付）

松下雅和（正会員）matsukaz@gmail.com

2001年早稲田大学社会科学部卒業．中堅SIerを経て，2005年に（株）オープンストリーム入社．現在に至る.