Tijd voor een overzicht om een helder beeld te krijgen

NoSQL: producten en hun kenmerken

Roland Bouman

De term NoSQL, die gebruikt wordt om een bonte verzameling database-systemen van recente makelij aan te duiden, duikt steeds vaker op. Deze berichtgeving is vooral afkomstig uit de wereld van grote websites als Google, Amazon en Facebook. Maar met de aanhoudende 'Big Data' trend lijkt NoSQL ook te zijn toegetreden tot de Business Intelligence.

Men kan tevens in de ontwikkeling van de NoSQL databases een evolutie waarnemen die doet vermoeden dat het meer dan een kortstondige hype betreft. Toch hebben veel IT-professionals nog geen helder beeld van de precieze betekenis van de term NoSQL, de producten en hun typerende eigenschappen en specifieke toepassingen. Daarom: tijd voor een overzicht.

Het begrip NoSQL

Het is niet eenvoudig om een definitie te geven van 'de NoSQL databasesystemen'. Wel kan gesteld worden dat het om databasesystemen gaat die wezenlijk anders zijn dan RDBMS'en zoals Oracle, MS SQL Server en DB2. Dit lijkt vanzelfsprekend: bijna alle RDBMS'en ondersteunen SQL, hetgeen de term NoSQL nu juist lijkt uit te sluiten. Toch is het niet zo simpel. NoSQL kan worden gelezen als 'No-spatie-SQL', hetgeen inderdaad het ontbreken van SQL impliceert. Maar de verwijzing naar SQL dient vooral ter onderscheid van RDBMS'en. Hoewel SQL van groot belang is voor applicatieontwikkelaars en de inzet van bijvoorbeeld reporting tools, is het paradoxaal genoeg niet de hoofdzaak tot het begrip van NoSQL databasesystemen.
NoSQL wordt ook wel opgevat als afkorting van 'Not Only SQL': het idee dat men voor databasesystemen moet kiezen die zijn toegesneden op de eisen van de specifieke toepassing, ook als dit tot een oplossing leidt die niet-relationeel is of minder strikte eisen stelt aan gegevensconsistentie.

Producten en Ontwikkelingen

Voordat we dieper ingaan op de kenmerkende eigenschappen van NoSQL databases, is het goed om even een aantal concrete producten en ontwikkelingen te introduceren. Zie afbeelding 1.

- 2002 – Gilbert & Lynch bewijzen het CAP postulaat [1] dat in 2000 door Brewer werd geformuleerd [2]. Meer hierover verderop in dit artikel.
- 2003 – Release van memcached [3]. Deze gedistribueerde in-memory cache wordt meestal bovenop een RDBMS gebruikt om website performance te verhogen en is een NoSQL voorloper.
- 2004 – Google publiceert over MapReduce [4], een algoritme om grote hoeveelheden data clustergewijs te analyseren, bijvoorbeeld om de Google websearch index op te bouwen.
- 2005 – Katz schrijft over een op Lotus Notes geïnspireerde schemaloze document database genaamd CouchDB [5]. Achteraf blijkt dit ook het jaar te zijn waarin Google diens eigen NoSQL platform Bigtable in productie neemt.
- 2006 – Google publiceert over Bigtable [6], een gedistribueerde hashtable voor opslag van grote hoeveelheden data. Verder ziet CouchDB [7] zijn eerste release. Samen met Bigtable kunnen we spreken over de eerste echte NoSQL databases. Tevens lanceert Amazon zijn Simple Storage (S3) [8] en Elastic Compute Cloud (EC2) [9] Services, waarmee Cloud computing binnen ieders handbereik komt.
- 2007 – Amazon publiceert over Dynamo [10], een voor hoge beschikbaarheid geoptimaliseerde key/value store met eventual consistency, en de daarop gebaseerde databasedienst SimpleDB [11]. Open Source implementaties van Bigtable genaamd Hypertable [12] en Hbase [13] komen op de markt. Aankondiging van de Heroku [14] Cloud computing service, waarbinnen steeds meer NoSQL-oplossingen worden aangeboden.
- 2008 – Facebook geeft diens op Dynamo en Bigtable geïnspireerde DBMS Cassandra [15] vrij als Open Source. Yahoo! publiceert over PNUTS [16] en Google over Megastore [17]. PNUTS en Megastore zijn beide NoSQL-systemen met enkele RDBMS-achtige trekjes.
- 2009 – Release van de MongoDB [18] document store. Verder Open Source implementaties naar voorbeeld van Amazon's Dynamo: Riak [19], Redis [20] en Voldemort [21].

- 2010 – Release van OrientDB [22], een 'document graph store' met expliciete SQL ondersteuning. Verder verschijnt VoltDB [23], een gedistribueerd RDBMS met sterke oriëntatie op hoge beschikbaarheid en lage latency.

Kenmerken NoSQL

NoSQL vormt een klasse van gedistribueerde niet-relationele databasesystemen. Gedistribueerde systemen draaien op clusters van met elkaar samenwerkende servers, nodes genaamd. Nodes zijn met een netwerk onderling verbonden, en kunnen fysiek ver van elkaar verwijderd zijn (denk aan meerdere datacenters op verschillende continenten). Kenmerkend voor NoSQL-systemen is dat zij ontworpen zijn om op basis van goedkope commodity hardware te kunnen voldoen aan extreme eisen voor wat betreft:

- bijna lineaire en dynamische schaalbaarheid (elastic near-linear scalability);
- hoge beschikbaarheid (high availability);
- korte responstijden (low latency).

Gedistribueerde RDBMS'en zoals Oracle RAC en MySQL Cluster zijn ook ontworpen om tegemoet te komen aan dergelijke eisen, en maken daarvoor net als NoSQL databases gebruik van twee fundamentele processen:

- *partitioning*: het verdelen van data over meerdere nodes. Dit leidt tot schaalbaarheid doordat zaken als opslagcapaciteit en hoeveelheid werkgeheugen niet meer gebonden zijn aan hardwarematige limieten. In plaats daarvan kan de capaciteit gewoon worden gereguleerd door het aantal nodes te variëren;
- *replicatie*: data kopiëren naar meerdere nodes. Dit maakt het systeem tolerant tegen uitval van nodes omdat data beschikbaar blijven op andere nodes.

Ondanks deze (oppervlakkige) overeenkomsten tussen gedistribueerde RDBM'en en NoSQL-systemen, zijn er belangrijke verschillen. NoSQL-systemen stellen minder stringente eisen aan gegevensconsistentie en -integriteit, maar vergroten hierdoor hun beschikbaarheid terwijl responstijden verkleind worden. Dit uit zich in:

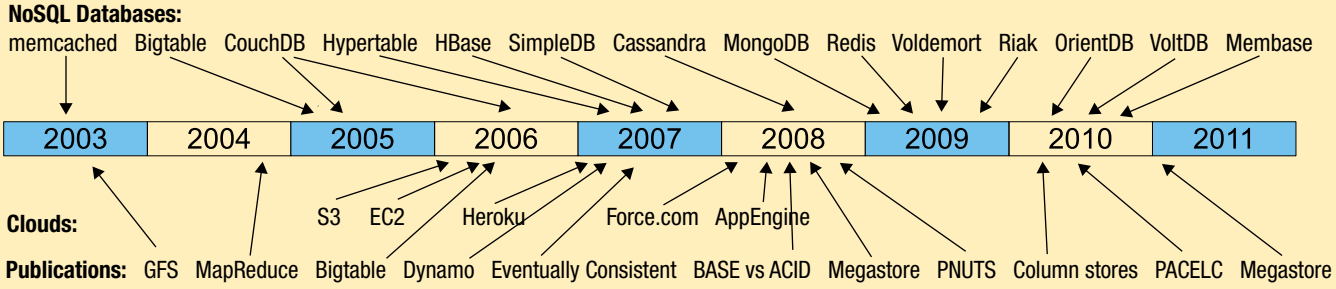
- *schemaloze* datamodellen in plaats van het relationele model;
- simpele API's in plaats van geavanceerde querytalen;
- geen ACID (Atomic, Consistent, Isolated, Durable) transacties over meerdere statements, maar BASE (Basically Available, Soft-state, Eventually consistent) atomaire operaties op enkelvoudige gegevens.

Deze NoSQL kenmerken roepen bij RDBMS professionals vaak weerstand op: het idee dat gegevensconsistentie en -integriteit onderhandelbaar zijn is moeilijk te accepteren. Ook een niet-relationeel datamodel en een API in plaats van een 'echte' querytaal lijken een stap achteruit richting het prerelationele tijdperk. Toch valt deze keuze te verklaren, ten minste als men prioriteit wil geven aan schaalbaarheid en beschikbaarheid.

Het CAP theorema

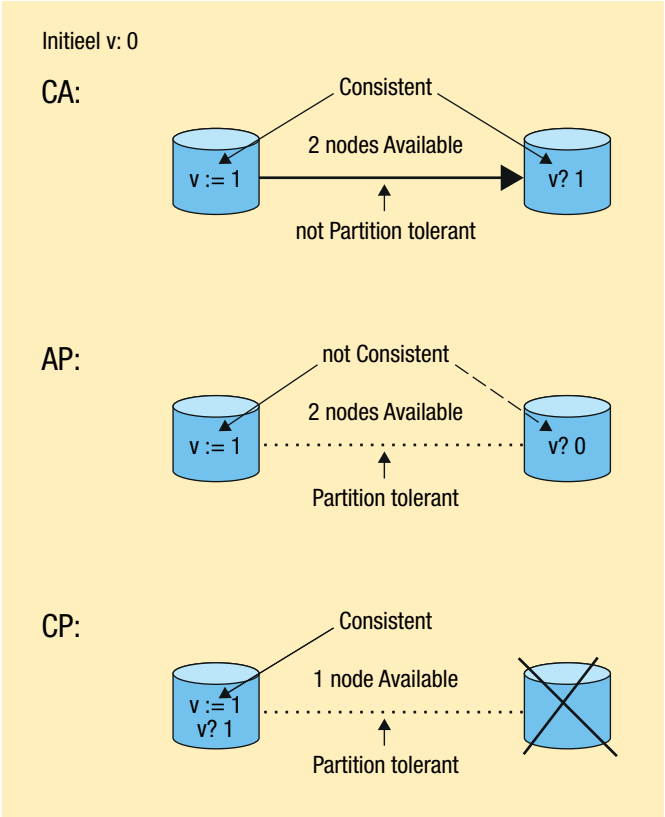
Dat NoSQL databasesystemen minder stringente eisen stellen aan gegevensconsistentie dan RDBMS'en is verklaarbaar vanuit Brewer's CAP theorema (oorspronkelijk gepostuleerd op het 'Principles of Distributed Computing Symposium' in 2000). Dit theorema slaat op gedistribueerde databasesystemen in het algemeen en de letters C, A en P staan voor:

- Consistency: alle nodes zien op elk moment dezelfde data;
- Availability: het systeem is beschikbaar om verzoeken te verwerken;



2002-06-01: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services (paper)
2003-05-22: Eerste release memcached
2003-10-01: The Google File System (paper)
2004-12-01: MapReduce: Simplified Data Processing on Large Clusters (paper)
2005-04-01: Bigtable in productie bij Google
2005-04-25: Eerste werkende versie van CouchDB
2006-03-15: Amazone S3 (Simple Storage Service) beschikbaar
2006-08-25: Amazon EC2 (Elastic Compute Cloud) beschikbaar
2006-08-31: Eerste release CouchDB
2006-11-01: Bigtable: A Distributed Storage System for Structural Data (paper)
2007-06-01: Heroku (PaaS) beschikbaar
2007-07-10: Eerste release Hypertable
2007-10-02: Dynamo: Amazon's Highly Available Key-value Store (paper)
2007-10-29: Eerste release van HBase (in Hadoop 0.15.0)
2007-12-14: Amazon SimpleDB beschikbaar
2007-12-19: Eventually Consistent (blog)
2008-06-09: Megastore: A scalable data system for user facing applications
2008-01-17: Force.com (PaaS) door Salesforce.com)
2008-04-08: Google AppEngine beschikbaar
2008-05-01: BASE, and ACID alternative (paper)
2008-08-01: PNUTS: Yahoo!'s hosted data serving platform (paper)
2008-08-07: Eerste release Cassandra
2008-12-23: Eventually Consistent (blog, gereviseerde versie van 2007-12-19)
2009-02-11: Eerste release MongoDB
2010-03-29: Distinguishing Two Major Types of Column-Stores (blog)
2010-04-02: Eerste release OrientDB
2010-04-23: Problems with CAP, and Yahoo's little known NoSQL system (blog)
2010-05-25: Eerste release VoltDB
2010-06-01: Eerste release Membase
2011-01-09: Megastore: Providing Scalable, Highly Available Storage for Interactive Services

Afbeelding 1: Tijdlijn.



Afbeelding 2: Drie situaties.

- Partition Tolerance: het systeem blijft werken indien verbindingen tussen de nodes verbroken raken (network split).

Het theorema stelt dat een gedistribueerd (database)systeem wel twee maar niet drie van deze eigenschappen tegelijkertijd kan hebben. Deze verschillende situaties zijn geïllustreerd in afbeelding 2. In afbeelding 2 zien we een 'cluster' van twee nodes die via een netwerk verbonden zijn en waarvan ten minste één node beschikbaar is voor lees- en schrijfacties op een variable *v*. Initiëel heeft *v* de waarde 0; daarna wordt via één van de nodes de waarde 1 aan *v* toegekend, waarna *v* wordt uitgelezen.
De eerste situatie is consistent en available (CA): beide nodes zijn beschikbaar (A), en de toekenning van 1 aan *v* is dankzij het netwerk zichtbaar op beide nodes, zodat het systeem consistent (C) is. Omdat consistentie afhankelijk is van het netwerk tussen de nodes is deze situatie niet tolerant tegen een netwerkpartitie (P). De tweede situatie is available en tolerant voor een netwerkpartitie (AP): beide nodes zijn beschikbaar (A), en het netwerk tussen de nodes is afwezig (P). Daardoor is de toekenning van 1 aan *v* via één van de nodes niet zichtbaar op de andere node. Die denkt zodoende dat *v* nog de waarde 0 heeft, hetgeen niet consistent (C) is.
De derde situatie is consistent en tolerant voor een netwerkpartitie (CP). Net als in de vorige situatie is hier het netwerk afwezig. Maar nu is ook de node aan de andere zijde van de netwerkpartitie (P) onbeschikbaar (A) gemaakt. Hierdoor is er nog maar één node waarop zowel de lees- als schrijfactie op *v* kan plaatsvinden, waardoor consistentie (C) gewaarborgd is. Toegepast op concrete gedistribueerde databasesystemen blijkt er een belangrijk verschil tussen gedistribueerde RDBMS'en en NoSQL databases: RDBMS'en gedragen zich als CA-systemen zolang er geen netwerkpartitie optreedt. Treedt er wel een netwerkpartitie op, dan gedragen zij zich als CP-systemen, en wordt de consistentie gewaarborgd doordat geïsoleerde nodes actief ten gronde gaan todat er één cluster overblijft waarvan alle nodes wel met elkaar verbonden zijn. Die vermindering van nodes gaat wel ten koste van de beschikbaarheid van het systeem.

De NoSQL-systemen kiezen voor de AP-strategie, en offeren consistentie op ten gunste van beschikbaarheid. Deze uitspraak behoeft wel wat nuance: de definitie van consistentie in het CAP theorema is vrij sterk en vereist immediate consistency: alle nodes zien op elk moment dezelfde data. In de praktijk wordt er zowel bij NoSQL-systemen als bij sommige RDBMS'en gebruik gemaakt van eventual consistency, hetgeen inhoudt dat er gegeven enige tijd na het repareren van netwerkpartities uiteindelijk een consistente situatie ontstaat. Tevens zijn de NoSQL-producten uitgerust met mechanismes om inconsistenties die ontstaan na een netwerkpartitie te consolideren (conflict resolution).

Waarom niet Relationeel?

Om te begrijpen waarom de NoSQL databasesystemen geen gebruik maken van het relationele model, moeten we in gedachten houden dat schaalbaarheid een van de belangrijke doelstellingen is van de NoSQL-toepassingen, en dat dit bereikt wordt door data te verdelen over meerdere fysiek uiteengelegen nodes (partitioning).
Het relationele model brengt een logische scheiding aan in data door haar over meerdere tabellen te verdelen. De tabellen zijn niet allemaal even vol, en niet allemaal even breed, en deze natuurlijke 'partitioning' van de data is minder schaalbaar dan een kunstmatige opdeling: eigenlijk zouden de tabellen moeten worden opgehakt, bijvoorbeeld door de rijen over de nodes in het cluster te verdelen.

Nu kunnen tabellen wel opgedeeld worden, maar applicaties moeten data juist uit verschillende tabellen samenbrengen (bijvoorbeeld met JOIN's, subquery's en UNION operaties). Hier wringt de schoen: in een niet-gedistribueerde database bevinden deze tabellen zich op één node, en kunnen deze operaties efficiënt worden uitgevoerd. Maar als gegevens vanaf meerdere nodes moeten worden opgehaald neemt de responstijd snel toe. De oplossing voor dit probleem is eenvoudig: door data domweg niet in types op te splitsen maar gewoon bij elkaar te houden, en ongenormaliseerd met herhalende groepen en al op te slaan verdwijnt dit probleem. Een ander vaak geciteerd voordeel van dergelijke schemaloze modellen is dat zij flexibel zijn, waardoor aanpassingen in applicaties gemakkelijk zijn aan te brengen. Hier zitten natuurlijk twee kanten aan. Enerzijds vormt deze flexibiliteit net zo goed een risico, en levert zij ballast op voor de

applicaties, die immers meer verantwoordelijkheid moeten nemen om de data te beschermen. Anderzijds is het wel zo dat in een groot RDBMS schemawijzingen zoals het afsplitsen of toevoegen van kolommen veel tijd kunnen kosten, gedurende welke het systeem niet of minder beschikbaar is. Dus opnieuw geldt dat de keuze wel te rechtvaardigen is indien vaststaat dat beschikbaarheid een hoofddoel is.

NoSQL datamodellen

Er zijn drie belangrijke soorten datamodellen in het huidige NoSQL landschap te onderscheiden: key/value stores; sparse hashtable; document stores, zie afbeelding 3.
(De zogenaamde graph databases worden soms ook tot het NoSQL kamp gerekend, maar eigenlijk vormt dit een geheel aparte klasse, welke hier dan ook niet besproken wordt.)
Deze modellen hebben met elkaar gemeen dat zij associaties opslaan tussen een unieke sleutel en een verzameling gegevens (die zelf geen deel uitmaken van die sleutel). De gegevens zijn meestal ongenormaliseerd, en worden primair door de applicatie geïnterpreteerd en bewerkt, en niet door het DBMS. Ook is het de verantwoordelijkheid van de applicatie om de sleutels zo te kiezen dat het de data weet terug te vinden. Omdat het DBMS geen of weinig regels oplegt aan de data, spreekt men van schemaloze datamodellen.

Key/Value Stores

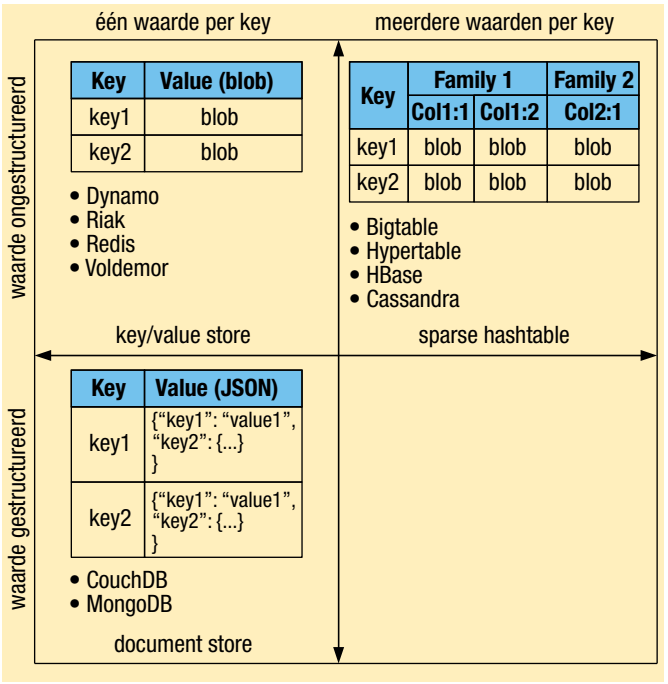
In een key/value store is een lijst waar de lijstregels bestaan uit een unieke sleutel (key) en een waarde (value). Voor de meeste key/value stores is de value vanuit het oogpunt van het DBMS een structuurloze reeks bytes die door de applicatie (niet het DBMS) wordt geïnterpreteerd en gemanipuleerd. De key is meestal net als de value een arbitraire binaire waarde, die voor het DBMS geen inherente betekenis heeft. Toch bevat de sleutel op systeemniveau wel nuttige informatie, omdat het DBMS deze via een partitioningfunctie gebruikt om te bepalen op welke nodes de bijbehorende waarde opgeslagen wordt. Hierdoor kan snel op grond van een enkele keywaarde de bijbehorende waarde worden gevonden zonder het hele cluster af te lopen.
Het canonieke voorbeeld van een key/value store is het Dynamo systeem, hetgeen ontwikkeld en gebruikt wordt door Amazon. Amazon heeft belangrijke details over de architectuur en de implementatie van Dynamo gepubliceerd, en er is inmiddels ook een flink aantal open source implementaties van dergelijke key/value stores verkrijgbaar, zoals Voldemort, Riak en Membase.

Sparse Hashtable

Een sparse hashtable heeft oppervlakkig bezien een tabelstructuur: de tabel bevat rijen, en elke rij heeft voor elke tevoren gedefinieerde kolom plaats om een waarde op te slaan. Maar toch heeft dit datamodel meer weg van de zojuist besproken key/value store dan van een tabel uit een RDBMS: eigenlijk is de value in dit geval onderverdeeld in kolommen. Overigens zijn kolommen op een hoger niveau weer gegroepeerd in kolomfamilies. Het verschil met een relationele tabel is dat de kolomwaarden zelf weer opnieuw vanuit het oogpunt van het DBMS betekenisloos zijn.
In fysiek opzicht zijn er ook wat verschillen. Zo worden in een traditioneel rij-georiënteerd RDBMS de kolomwaarden van een rij fysiek dicht bij elkaar opgeslagen. In de NoSQL sparse hashtable worden kolomwaarden juist dicht bij elkaar opgeslagen, en op hun beurt zijn de kolommen binnen een kolomfamilie ook vaak fysiek nabij. Deze zogenaamde lokaliteit is van belang, omdat dit in belangrijke mate bepaalt hoe snel gegevens kunnen worden opgehaald. Zo zullen gegevens die vaak tegelijkertijd nodig zijn in een familie van kolommen worden opgeslagen.
Het canonieke voorbeeld van een sparse hashtable is Bigtable, hetgeen ontwikkeld en gebruikt wordt door Google. Ook Google heeft details over Bigtable gepubliceerd in een artikel, en ook daarvan zijn inmiddels meerdere Open Source varianten verkrijgbaar, zoals Cassandra, Hypertable en HBase.

Document Store

Document stores zijn eigenlijk ook key/value stores, maar in dit geval is de value niet geheel betekenisloos vanuit het oogpunt van het DBMS. In plaats daarvan is de value gestructureerd volgens een dataformaat dat ook door het DBMS begrepen wordt, zodat het mogelijk wordt om het DBMS te laten zoeken naar documenten met bepaalde kenmerken.
Twee populaire document stores zijn CouchDB en MongoDB, die allebei het JSON (JavaScript Object Notation, een subset van JavaScript) formaat gebruiken voor documenten. Maar op grond van het concept zijn er ook goede redenen om producten als bijvoorbeeld Lotus Notes en MarkLogic XML server tot deze groep te rekenen.



Afbeelding 3: Datamodellen.

NoSQL 'Querytalen'

In de meeste gevallen is datatoegang tot de NoSQL-systemen beperkt tot simpele API's die ondersteuning bieden voor low-level acties als: put(key, value); get(key); delete(key). Voor de meeste NoSQL-systemen zijn dergelijke API's voor de meest populaire programmeer- en scripttalen beschikbaar. Tevens is het meestal mogelijk om alle sleutels te scannen, en vaak zijn de sleutels geordend, hetgeen ook range scans mogelijk maakt.
Een belangrijk verschil met het RDBMS is dus de afwezigheid van een querytaal, en daarmee complexe operaties als JOIN's. Deze operaties zijn natuurlijk wel mogelijk, maar worden voor zover zij nodig zijn dan door de applicatie geïmplementeerd.

Map/Reduce

Een echte querytaal veronderstelt het vergelijken en transformeren van gegevens. Schemaloze NoSQL-systemen kunnen de gegevens zelf niet interpreteren en bieden hooguit een nauwelijks nuttige binaire vergelijking. Gegevensanalyse is wel mogelijk maar wordt dan in de toepassingslaag uitgevoerd, meestal volgens het Map/Reduce algoritme. Map/Reduce is een gedistribueerd algoritme met twee fases:

- Map, waar zogenaamde 'mapper nodes' parallel een partitie van de keys volledig scannen, om er de interessante values uit te filteren;
- Reduce. In deze fase aggregeren zogenaamde 'reducer nodes' gegevens van meerdere mapper nodes, hetgeen het query-resultaat vormt.

Eventueel kunnen meerdere Map/Reduce jobs elkaar opvolgen. Een belangrijk punt om te onthouden is dat Map/Reduce geen querytaal is, maar een manier om een taak te distribueren. Het eigenlijke filteren en aggregeren moet afhankelijk van de zoekvraag geprogrammeerd worden.

Gestructureerde querytalen

De NoSQL-systemen die een document store datamodel hanteren bieden vaak wel een querytaal. Dat kan, omdat de gegevens niet ongestructureerd zijn. Vanwege diezelfde reden kunnen deze systemen de gegevens ook indiceren, terwijl er met andere datamodellen alleen indexes op de keys mogelijk zijn.
Ook zijn er inmiddels enkele SQL-achtige talen beschikbaar voor de NoSQL databases. Zo biedt Hbase Hive aan, een SQL-achtige taal die op de achtergrond vertaald wordt in een Map/Reduce job. Zo kan men kan zich voorstellen dat WHERE clauses naar Mappers worden vertaald, en GROUP BY clauses naar Reducers.

Conclusie

Hoewel de voordelen van NoSQL ten opzichte van een RDBMS voornamelijk van toepassing zijn op webreuzen als Amazon, Google en Facebook, is het goed om te beseffen dat er wel degelijk een afweging mogelijk is tussen enerzijds strikte gegevensconsistentie en integriteit, en anderzijds elastische schaalbaarheid en hoge beschikbaarheid. Kennelijk belemmert het gebrek aan een relationele database deze bedrijven niet om uiterst succesvol te zijn binnen hun business.
Anderzijds is er ook een kentering te bemerken: na de explosie van NoSQL releases in de periode 2007-2009 (Hypertable, Cassandra, MongoDB, Voldemort en nog veel meer) lijkt er nu een trend zichtbaar te worden om de NoSQL databases uit te rusten met RDBMS features (Megastore en PNUTS, OrientDB) en anderzijds om relationele databases te construeren die de idealen van elastische schaalbaarheid, hoge beschikbaarheid en lage latency combineren (VoltDB). Uiteindelijk zal het zijn weg wel weer gaan vinden in de traditionele producten van Oracle, Microsoft en IBM, zoals we al eerder hebben gezien met Object Oriented en XML databases.

Bronnen

1. *Gilbert & Lynch, 'Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services', http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf*
2. *Brewer, 'Towards Robust Distributed Systems', www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf*
3. *memcached, 'Free & open source, high-performance, distributed memory object caching system', http://memcached.org/*
4. *Dean & Ghemawat, 'MapReduce: Simplified Data Processing on Large Clusters', http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/en//papers/mapreduce-osdi04.pdf*
5. *Katz, 'CouchDb Architecture', http://damienkatz.net/2005/04/couchdb_archite.html*
6. *Chang et al., 'Bigtable: A Distributed Storage System for Structured Data', http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/en//papers/bigtable-osdi06.pdf*
7. *Apache foundation, 'The CouchDB Project', http://couchdb.apache.org/*
8. *Amazon, 'Amazon Simple Storage Service', http://aws.amazon.com/s3/*
9. *Amazon, 'Amazon Elastic Compute Cloud', http://aws.amazon.com/ec2/*
10. *DeCandia et al., 'Dynamo: Amazon's Highly Available Key-value Store', http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf*
11. *Amazon, 'SimpleDB', http://aws.amazon.com/simpledb/*
12. *Hypertable, 'An Open Source, High Performance, Scalable Database', http://hypertable.org/*
13. *Apache foundation, 'HBase is the Hadoop database', http://hbase.apache.org/*
14. *Heroku, 'Rock-solid Ruby Platform', http://heroku.com/*
15. *Lakshman & Malik, 'The Apache Cassandra Project', www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf*
16. *Cooper et al., 'PNUTS: Yahoo!'s Hosted Data Serving Platform', http://research.yahoo.com/files/pnuts.pdf*
17. *Baker et al., 'Megastore: Providing Scalable, Highly Available Storage for Interactive Services', www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf*
18. *mongoDB, {name: 'mongo', type: 'DB'}, www.mongodb.org/*
19. *basho, 'Basho Riak: An Open Source Scalable Data Store', www.basho.com/Riak.html*
20. *Redis, 'an open source, advanced key-value store', http://redis.io*
21. *Project Voldemort, 'A distributed Database', http://project-voldemort.com/*
22. *OrientDB, 'The Document-Graph database with the support of ACID Transactions, SQL and Native Queries, Asynchronous Commands, Intents, and much more', www.orientechnologies.com/orient-db.htm*
23. *VoltDB, 'next-generation SQL RDBMS with ACID for fast-scaling OLTP applications', http://voltdb.com/*

Roland Bouman (roland.bouman@gmail.com) is zelfstandig consultant en auteur. Met dank aan Jos van Dongen (review).