NoSQL: concetti generali

Paolo Atzeni

30/05/2011

Il solito primo lucido ☺

- DataBase Management System (DBMS)
  - Sistema che gestisce collezioni di dati:
    - grandi
    - persistenti
    - condivise

    garantendo
    - privatezza
    - affidabilità
    - efficienza
    - efficacia

DBMS relazionali

- Nati negli anni '70 (più di trent'anni fa)
- Supporto efficace ed efficiente per applicazioni di tipo "gestionale" (contabilità, prenotazioni, gestione personale, …) con requisiti
  - persistenza, condivisione, affidabilità
  - dati a struttura semplice, con dati di tipo numerico/simbolico
  - transazioni concorrenti di breve durata (OLTP)
  - interrogazioni complesse, espresse mediante linguaggi dichiarativi e con accesso di tipo "associativo"

In termini più tecnologici

(Stonebraker & Cattel, CACM June 2011)

- "General-purpose traditional row stores"
  - Disk-oriented storage
  - Tables stored row-by-row on disk (hence, a row store)
  - B-trees as the indexing mechanism
  - Dynamic locking as the concurrency control mechanism
  - A write-ahead log (WAL) for crash recovery
  - SQL as the access language
  - A "row-oriented" query optimizer and executor

Un'altra cosa che si dice da vent'anni ☺

- La rapida evoluzione tecnologica (miglioramento di prestazioni, capacità, e costi dell'hardware) ha fatto emergere nuove esigenze applicative, per le quali la tecnologia relazionale è inadeguata

> **1. New non business-type users are feeling the need for large amounts of reliable, sharable and persistent data (CAD, CASE, office automation and AI). These new customers of database technology bring in new requirements**

(F. Bancilhon: Object-Oriented Database Systems. PODS 1988: 152-162)

Aree applicative emergenti (già nel 1988)

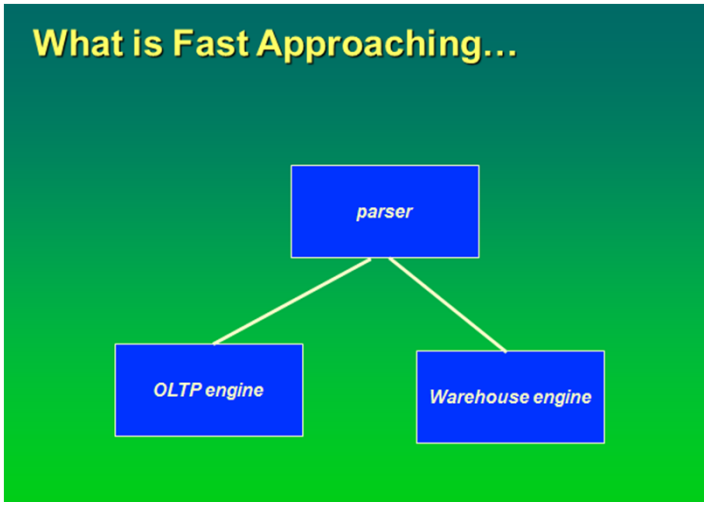
- Progettazione assistita da calcolatore
  - CASE (Computer-Aided Software Engineering)
  - CAD (Computer-Aided Design)
  - CAM (Computer-Aided Manufacturing)
- Gestione di documenti
  - testi e automazione d'ufficio
  - dati multimediali
- Altro
  - scienza e medicina
  - sistemi AI

Un'altra frase che si ripete spesso

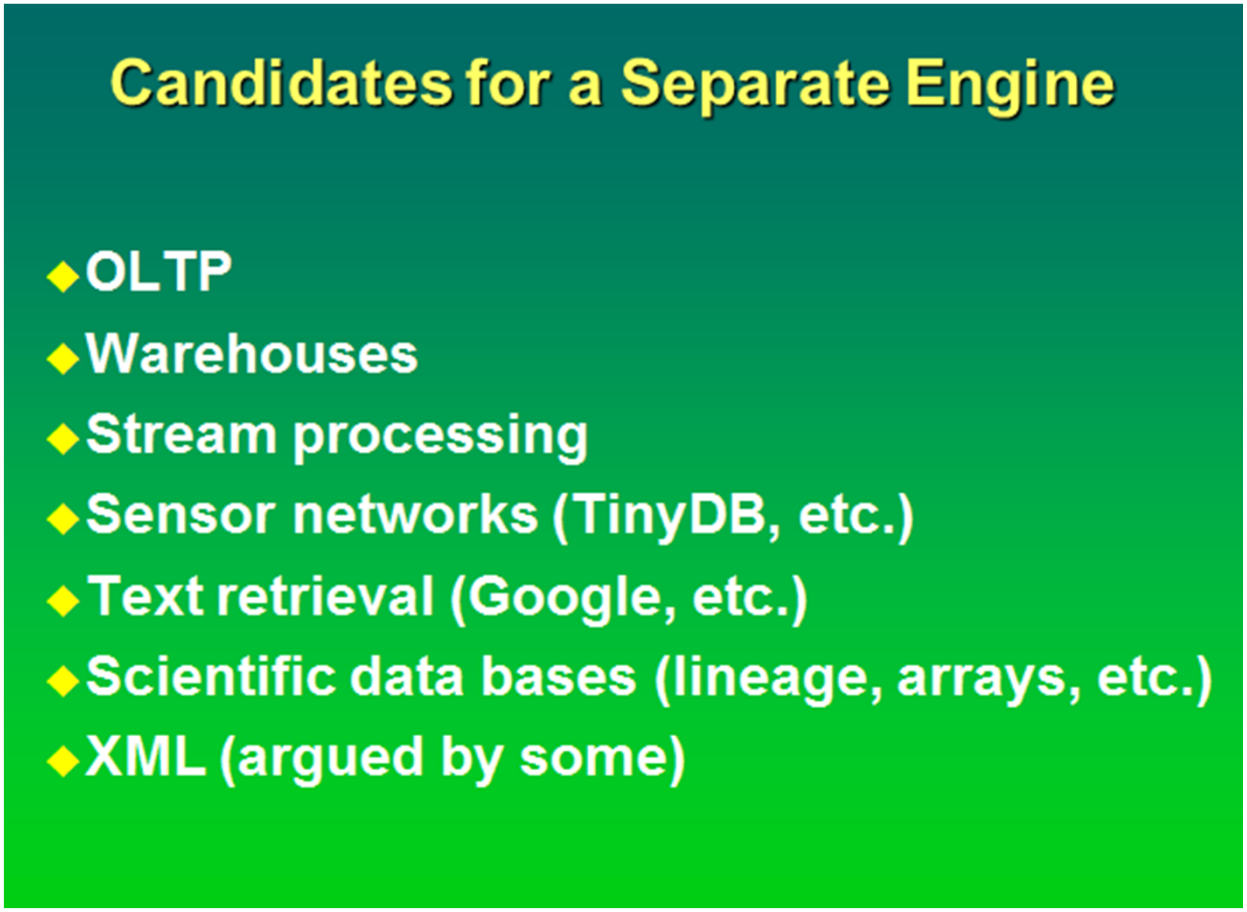
(Michael Stonebraker, Ugur Çetintemel: "One Size Fits All": An Idea Whose Time Has Come and Gone. ICDE 2005: 2-11)

"One-size-fits-all"

- I sistemi relazionali, secondo Stonebraker, vengono proposti come soluzione universale
- In effetti, anche nel mondo relazionale, ciò non è più vero da tempo
  - due grandi famiglie di applicazioni, con requisiti molto diversi
    - OLTP
    - OLAP



Più in generale (nel 2005 …)



e … dopo il 2005

- Motivi di insoddisfazione, nel mondo Web:
  - la rigidità del modello relazionale (questione già emersa più volte, dagli anni '90, soddisfatta solo parzialmente dalle estensioni XML)
  - esigenza di grande scalabilità per operazioni molto semplici in quantità enorme
- Anche in altri contesti:
  - "pesantezza" dei sistemi relazionali, sia in termini di prestazioni sia di costo di gestione/amministrazione

Web company, tipico problema

- Una start-up di successo, con crescita esplosiva
  - Utilizzo di un DBMS open source (perché gratis o perché noto)
  - Versione 1 su una macchina, presto insufficiente
  - Versioni successive: partizionamento orizzontale (sharding), con la logica applicativa responsabile di dirigere le richieste ai nodi di competenza (di solito uno, vista la semplicità delle operazioni). Difficoltà:
    - Interrogazioni internodo da reimplementare
    - Transazioni internodo da gestire
    - Fallimenti di nodo sempre più probabili, con conseguenze su affidabilità e disponibilità
    - Ristrutturazioni dei dati e redistribuzioni "a caldo" molto impegnative

Una risposta al problema …

- Nuovi sistemi con
  - Capacità di scalare facilmente operazioni semplici su moltissimi nodi
  - Capacità di replicare e distribuire i dati su molti nodi
  - Flessibilità nella struttura dei dati
  - Nuove tecniche di indicizzazione e gestione della RAM
- Rinunciando a qualcosa
  - Interfaccia molto più semplice di SQL
  - Gestione delle transazioni meno rigorosa

- Commentiamo alcuni aspetti

- Nuovi sistemi con
  - Capacità di scalare facilmente operazioni semplici su moltissimi nodi
  - Capacità di replicare e distribuire i dati su molti nodi
  - **Flessibilità nella struttura dei dati**
  - Nuove tecniche di indicizzazione e gestione della RAM
- Rinunciando a qualcosa
  - Interfaccia molto più semplice di SQL
  - Gestione delle transazioni meno rigorosa

Flessibilità nella struttura dei dati

- Dati semistrutturati, concetto di interesse dagli anni '90, con l'avvento del Web

A Definition of Semistructured Data?

As a partial definition, a semistructured data model is a syntax for data with *no separate syntax for types*.

That is, no schema language or data definition language.

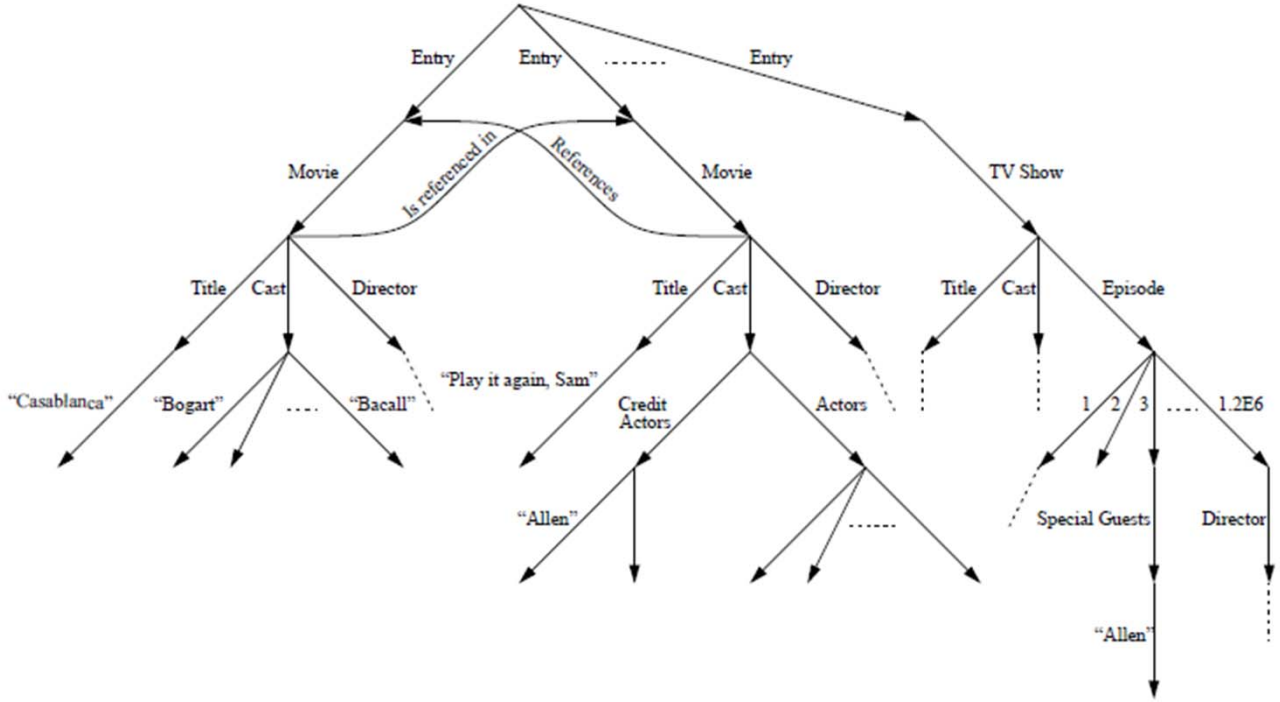
P. Buneman, tutorial, PODS 1997

http://db.cis.upenn.edu/DL/97/Tutorial-Peter/slides.ps.gz

Dati semistrutturati, esempio

Semistructured data is usually "mostly structured". We are typically trying to capture data that has only minor deviations from relational / nested relational / object-oriented data. For example...

A Semistructured Movie Database



- Nuovi sistemi con
  - **Capacità di scalare facilmente operazioni semplici su moltissimi nodi**
  - **Capacità di replicare e distribuire i dati su molti nodi**
  - Flessibilità nella struttura dei dati
  - Nuove tecniche di indicizzazione e gestione della RAM
- Rinunciando a qualcosa
  - Interfaccia molto più semplice di SQL
  - Gestione delle transazioni meno rigorosa

Scalabilità: un requisito fondamentale

- Tre tipi di architetture hardware parallele per DB
  - Shared-memory:
    - più processori condividono RAM e dischi

    Scalano in modo limitato, perché l'accesso alla memoria è un collo di bottiglia
  - Shared-disk
    - più processori (con RAM proprie) condividono dischi

    I buffer e i lock vanno coordinati, con limiti alla scalabilità
  - Shared-nothing
    - più processori ciascuno con RAM e dischi propri

    Scalano meglio (se i dati sono partizionabili e distribuibili automaticamente in modo da avere accessi bilanciati e le operazioni sono "localizzabili")

Esempi e sistemi

(Stonebraker & Cattel, CACM June 2011)

- Shared-memory:
  - MySQL, PostgreSQL, SQL server
- Shared-disk
  - Oracle RAC
- Shared-nothing
  - Sistemi specializzati per DW, DB2, molti sistemi "NoSQL"

- Scalabilità necessaria: migliaia di nodi (Facebook, al momento, utilizza 4000 nodi MySQL, coordinati da una logica applicativa apposita)

La scalabilità richiede operazioni locali

- Scalabilità delle letture è ottenuta (anche) attraverso la replicazione
- Scalabilità delle scritture è ottenuta attraverso il partizionamento ("sharding"); se una scrittura coinvolge più nodi, è necessario sincronizzare (vediamo poi le conseguenze delle repliche)
- La localizzazione delle transazioni che scrivono può essere favorita dalla replicazione dei dati di sola lettura (ad esempio anagrafiche)

- Nuovi sistemi con
  - Capacità di scalare facilmente operazioni semplici su moltissimi nodi
  - Capacità di replicare e distribuire i dati su molti nodi
  - Flessibilità nella struttura dei dati
  - Nuove tecniche di indicizzazione e gestione della RAM
- Rinunciando a qualcosa
  - Interfaccia molto più semplice di SQL
  - **Gestione delle transazioni meno rigorosa**

E le transazioni?

- Abbiamo sistemi distribuiti, con replicazione. In teoria, l'obiettivo è una trasparenza completa:
  - gli utenti vedono la base di dati come se fosse un unico corpus e tutti gli utenti vedono gli stessi valori
- Però:
  - la replicazione serve per aumentare le prestazioni
  - se le repliche debbono essere aggiornate, gli aggiornamenti distribuiti vanno gestiti con 2PC e questo può rallentare

Un inciso, il "CAP theorem"

- Un sistema distribuito non può soddisfare contemporaneamente le tre proprietà:
  - **Consistency** (consistenza o, meglio, coerenza)
  - **Availability** (disponibilità)
  - **Partition-tolerance**

CAP Theorem

- Consistency
  - In un sistema che abbia repliche, queste debbono essere aggiornate tutte prima di permettere accessi (gli accessi sono soggetti a controllo di concorrenza, con lock o mv)
- Availability
  - Un sistema disponibile deve avere repliche (visto che i guasti sono possibili)
- Partition-tolerance
  - Se le repliche sono su nodi diversi, questi possono essere isolati l'uno dall'altro

- "Dimostrazione": se abbiamo partition-tolerance e in ciascuna partizione si può leggere senza interruzione (availability), allora non possiamo avere consistency, perché gli aggiornamenti non possono propagarsi,

Un'osservazione sul CAP Theorem

(Stonebraker & Cattel, CACM June 2011)

- Distinguiamo reti locali (LAN) e geogragiche (WAN)
  - In ambiente LAN, il rischio di partizionamenti è basso e quindi si può rinunciare alla partition tolerance
  - Le repliche WAN sono usate soprattutto per disaster recovery, ma i disastri sono rari, nel qual caso diventa (quasi) accettabile la rinuncia alla availability

Forme di "consistency"

- Strong consistency
  - Gli esiti di una scrittura sono correttamente visibili da tutte le letture successive
- Weak consistency
  - Non è garantito che gli esiti di una scrittura siano visibili dalle letture (immediatamente successive)
- Eventual consistency (una forma di weak consistency)
  - "Prima o poi" gli esiti di una scrittura sono visibili a tutte le letture

"Implementazione"

- In un sistema con repliche, letture e scritture si possono gestire "a maggioranza"
- Supponiamo
  - N repliche
  - W numero di repliche che debbono essere scritte (e confermate) per garantire il completamento di una scrittura
  - R numero di copie (uguali) che debbono essere consultate prima di rispondere ad una lettura
- La "strong consistency" richiede
  - W+R>N (così è sicuro che le letture concludano almeno un dato aggiornato e quindi che siano tutti aggiornati)

"Implementazione", strong consistency

- W+R>N:
  - L'insieme degli elementi letti e quello delle repliche aggiornate hanno elementi in comune; se le letture sono coerenti fra loro, si ha la certezza di avere letto il valore corrente

Sistemi replicati "tradizionali"

- Sistemi ad alta affidabilità (repliche sincrone)
  - N = 2, W=2, R=1, ok
- Sistemi con repliche asincrone che permettono di leggere dal backup:
  - N = 2, W=1, R=1, strong consistency non garantita
- Sistemi ad alta affidabilità e un po' di attenzione alle prestazioni
  - N= 3, W=2, R=2

- Se non si riesce a scrivere su W nodi, l'operazione fallisce (in un certo senso si perde la availability, almeno in scrittura)

Sistemi replicati per grande carico di lettura

- R molto piccolo (di solito 1)
- N molto grande
- Se si vuole consistenza
  - W=N , con rischio di fallimenti delle scritture
- All'estremo opposto
  - W=1 con replicazione asincrona

- Altro punto di vista, con consistency
  - "read-optimized" R=1, W=N
  - "write-optimized" W=1, R=N (ma non garantisce persistenza e c'è il rischio di scritture inconsistenti, cosa che vale sempre se W<(N+1)/2)

Weak consistency

- W+R ≤ N
  - l'insieme degli elementi letti e quello delle repliche aggiornate possono non avere elementi in comune
- In pratica, si usa quasi solo per R=1, per avere alta scalabilità e disponibilità anche in presenza di partizionamento

Eventual consistency, un inciso

- In molte applicazioni reali, gli aggiornamenti sono spesso propagati con un certo ritardo:
  - Le operazioni bancarie (ad esempio i prelevamenti bancomat) vengono spesso registrate dopo qualche giorno.
  - Il saldo, di solito non è aggiornato rispetto alle transazioni eseguite e confermate!

(Dan Pritchett BASE: An Acid Alternative ACM Queue July 28, 2008)

Transazioni nei sistemi noSQL

- Vari approcci
  - Nuove versioni per ogni scrittura con versioni incomparabili in caso di scritture multiple
  - Scrittura preceduta da lettura, con successo se al momento della scrittura il dato non è cambiato
  - ACID, ma solo con transazioni piccole e locali
  - Quorum

To SQL or not to SQL, that is the question

Cattel (2010), Shakespeare (1601)

- SQL
  - A parità di operazioni (piccole, locali, …) le prestazioni e la scalabilità potrebbero essere paragonabili
  - Soluzioni relazionali valide sono state trovate per carichi applicativi specifici (OLTP, OLAP, in-memory, distibuiti,…)
  - "One-size-fits-all" non regge, ma un'interfaccia comune può
  - Il linguaggio ad alto livello facilita lo sviluppo (e non penalizza le prestazioni)
  - La tecnologia relazionale è solida, ha rimpiazzato le precedenti e ha resistito alle successive (OO, XML)
- noSQL
  - Non è stato dimostrato che le prestazioni sono paragonabili
  - Sistemi con modelli semplici hanno una curva d'apprendimento più sostenibile
  - La struttura relazionale è rigida
  - Le operazioni multi-nodo/-tabella sono scoraggiate o impossibili
  - Alcuni sistemi si stanno diffondendo in modo significativo

Sistemi "noSQL"

- Orientati alle applicazioni con operazioni semplici e locali
- Categorie
  - Key-value stores
  - Document stores
  - Extensible record stores

- Esistono anche sistemi SQL per operazioni semplici e locali, con implementazioni diverse da quelle tradizionali