NoSQL - Quelques principes clés

I. Mougenot isabelle.mougenot@umontpellier.fr

Faculté des Sciences Université Montpellier

2018

Les motivations autour du NoSQL (Not Only SQL)

Recouvre différentes initiatives voulues complémentaires au relationnel et relationnel/objet

1. évolution du Web, projets LOD (Linked Open Data), impulsion Google, Facebook, Amazon, Twitter, ...
2. ↗ volume des données ↗ interconnexion des données
3. limites des BD relationnelles face à de nouveaux besoins :
   - flexibilité : schémas très ouverts : nombreuses entités et nombreuses associations entre ces entités
   - adaptabilité : évolutions très fréquentes des schémas
   - des milliers voire des millions d'usagers

Quand passer par un système NoSQL ?

Forte volumétrie et nécessité de gestion de données distribuées

- collections de données par exemple issues de capteurs de l'ordre du TB ou PB voire plus

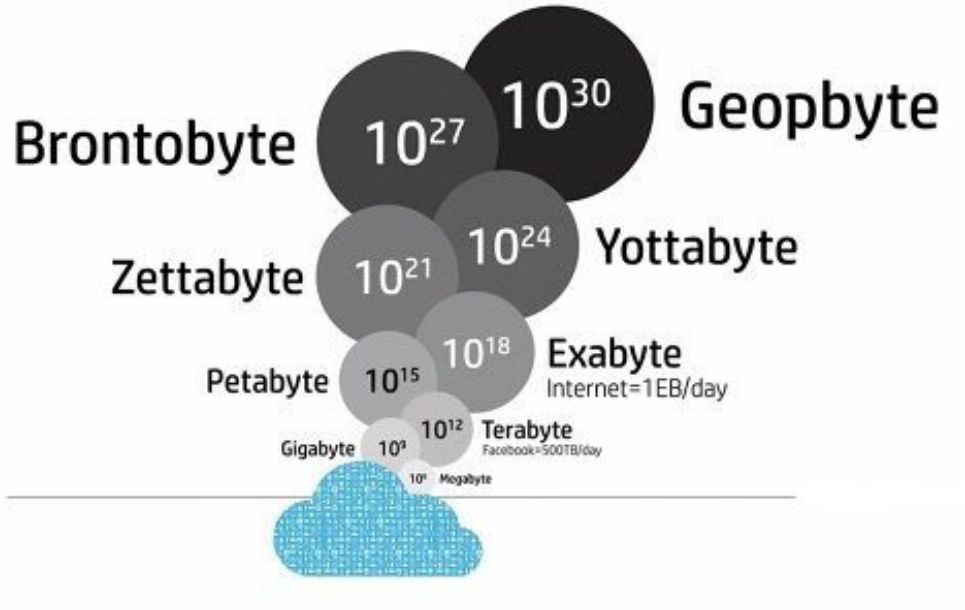

Figure: Ordre de grandeurs

Quand passer par un système NoSQL ?

Alternatives au relationnel dans des cas de figure ciblés

- recours fréquent à de l'évolution de schémas
- entités munies de nombreuses caractéristiques souvent non renseignées
- de multiples associations avec des multiplicités 1..* aux extrémités
- des attributs organisés naturellement sous forme d'arborescences
- un flux transactionnel très élevé

NoSQL : se démarquer des SGBD relationnels

Critiques ouvertes

- prépondérance du schéma et poids fort mis sur la représentation du domaine d'intérêt : s'affranchir de schémas normalisés vus comme des sophistications inutiles au détriment de l'efficacité
- modèle transactionnel et propriétés ACID : proposer une alternative moins exigeante : CAP (comprenant BASE)
- passage à l'échelle ou scalabilité (scalability) par ajout de serveurs au niveau de l'architecture physique : diminuer le temps de réactivité lors de l'afflux de nouveaux usagers, de nouvelles transactions à servir
- systèmes distribués et mécanismes de tolérance aux pannes : fragmentation des schémas et réplication, médiateur, entrepôt de données ...

Passage à l'échelle ou scalabilité

Capacité de l'architecture à s'adapter à une montée en charge (nouveaux usagers, nouvelles transactions) sans besoin de refonte des applications

- scalabilité horizontale (scaling out) : ajouter des serveurs (noeuds) avec des mécanismes de répartition de charge $\Leftarrow$ NoSQL
- scalabilité verticale (scaling up) : rendre plus performant un serveur : ajout de processeurs (CPU), barrettes mémoire (RAM), disques secondaires, cartes réseaux ...

Scalabilité horizontale

Etablir une relation linéaire entre les ressources ajoutées et l'accroissement des performances

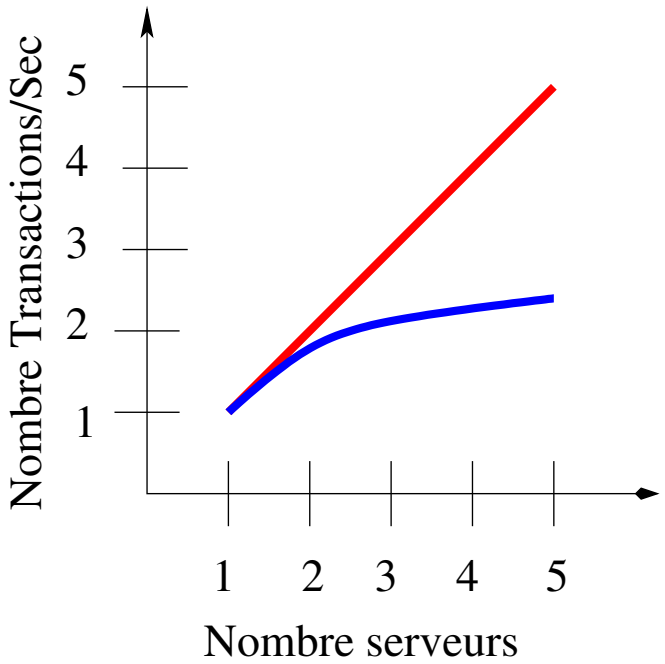

Figure: 1 serveur : 100 transactions/s ; 2 serveurs : 200 transactions/s . . .

SGBDR et besoins applicatifs à large échelle

Limites face aux besoins des applications à large échelle sur le Web (à partir Web 2.0)

- partionnement : les schémas fragmentés (fragmentations horizontale, verticale, hybride) distribués sur l'ensemble des partitions doivent être des fragments d'un seul schéma de données initial
- réplication sur différents noeuds : les fondements OLTP (On Line transactional processing) imposent de maintenir une intégrité forte sur les données, dans une application faisant appel à de nombreux noeuds, la disponibilité des données va être pénalisée (surtout si les transactions impliquent de nombreuses écritures).

systèmes NoSQL : grands principes

Pensés comme des systèmes de données distribués (distributed data stores)

- **Simplicité**
- **Flexibilité**
- **Efficacité**
- **Passage à l'échelle** : gros volumes de données distribués et interconnectés
  - partitionnement dynamique - sharding (partitionnement horizontal + plusieurs co-occurrences de schémas)
  - réplication à large échelle
  - architecture décentralisée

Définitions / scalabilité horizontale

Système distribué : système qui coordonne les activités de plusieurs machines pour accomplir une tâche spécifique

partitionnement : chaque machine héberge une portion des données désignée par fragment ou partition. Le partitionnement nécessite de disposer de mécanismes d'allocation des partitions sur les nœuds, d'équilibrer les charges et de disposer de communications réseaux optimisées. Le sharding correspond à du partitionnement horizontal. Le choix de la clé joue un rôle essential dans le succès du partitionnement

réplication : les mêmes données sont stockées sur plusieurs nœuds du système. On parle alors de réplicas ou de copies. La réplication permet d'accroître la fiabilité et la performance mais ouvre le problème de la synchronisation des copies et de la cohérence des données

Complémentarité des systèmes NoSQL

One size doesn't fit all

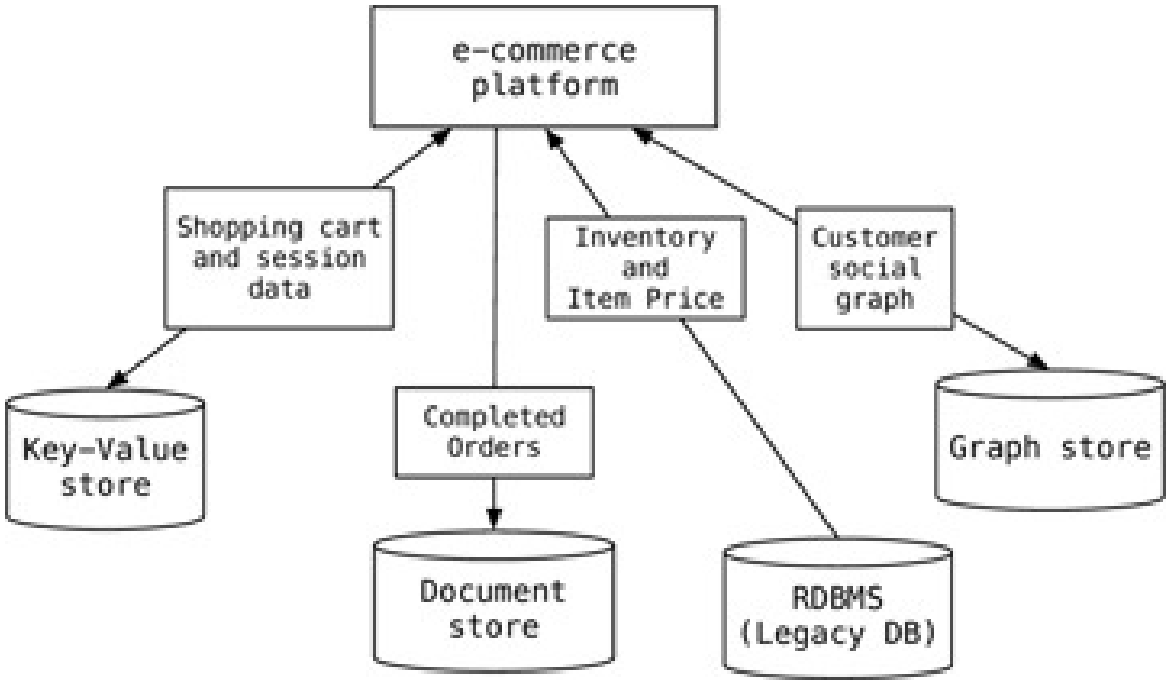

Figure: Persistance dite polyglotte (extrait de NoSQL distilled)

Théorème CAP

Constat de Brewer : aucun des systèmes distribués n'est à même de satisfaire en même temps les principes C, A et P :

1. **Consistency ou cohérence des données** : toute modification de donnée est suivie d'effet pour tous les noeuds du système
2. **Availability ou disponibilité des données** : toute requête émise et traitée par un noeud du système, reçoit une réponse (même en situation d'échec à produire une réponse)
3. **Partition tolerance ou recouvrement des noeuds** assurer une continuité du fonctionnement en cas d'ajout/suppression de noeud (ou partition) du système distribué

Un système distribué va satisfaire deux des trois points mais ne va pouvoir satisfaire les trois - Brewer. Towards robust distributed systems - ACM 2000

Théorème CAP

Considérations SGBDR / Systèmes NoSQL

1. **SGBDR** : Cohérence et haute disponibilité (pas ou peu de P, cad de différents noeuds système)
2. **Systèmes NoSQL** : Choix du P (système naturellement distribué) et sélection soit du C, soit du A
   1. abandon du A $\Leftarrow$ Accepte d'attendre que les données soient cohérentes
   2. abandon du C $\Leftarrow$ Accepte de recevoir des données parfois incohérentes

Positionnement des systèmes / CAP

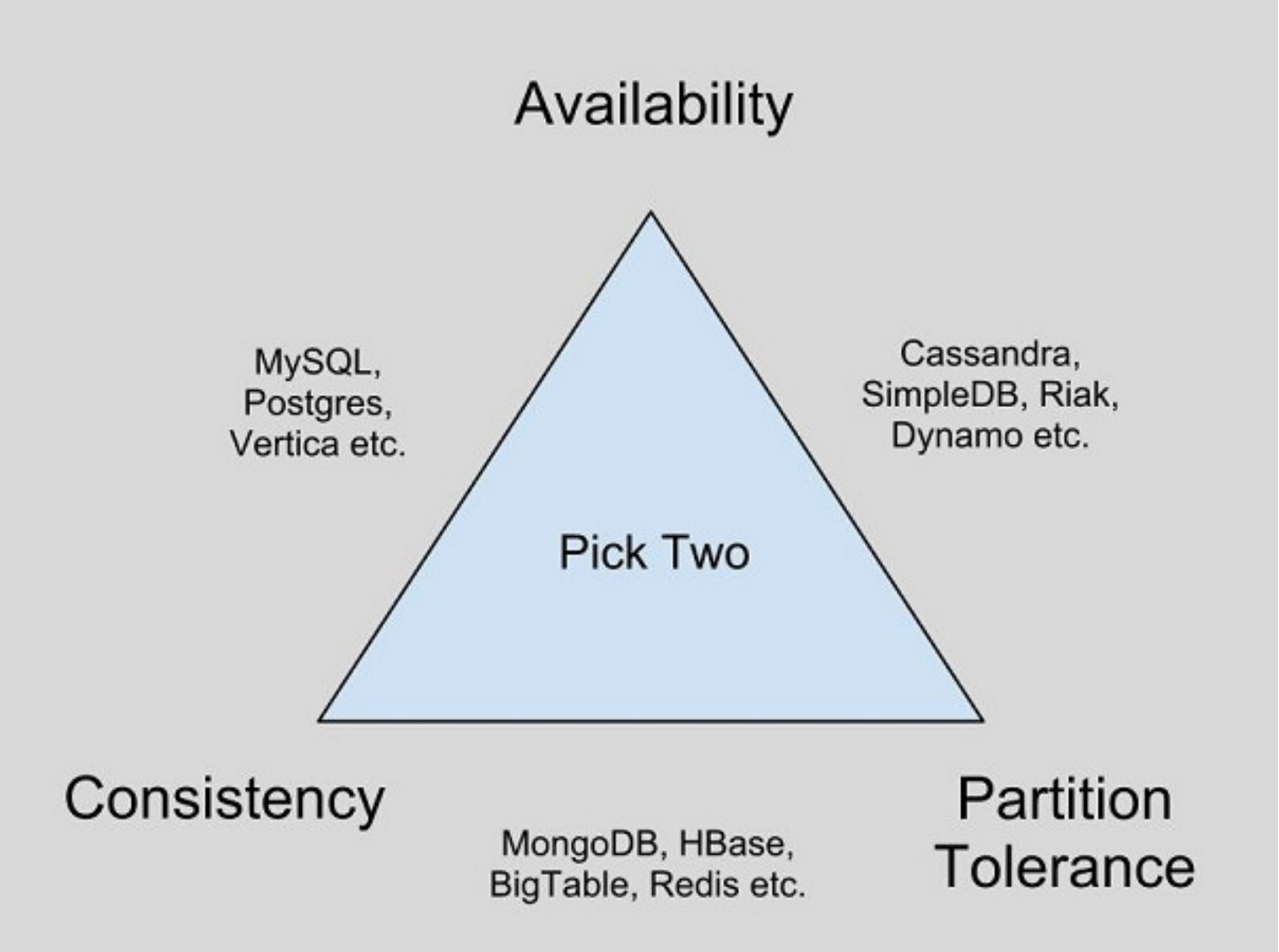

Figure: Synthèse CAP

Parti-pris Base (issu de CAP) versus propriétés ACID

BASE : **B**asically **A**vailable, **S**oft state, **E**ventual consistency

- **Modèle transactionnel** : les propriétés ACID (Atomique, Cohérent, Isolé, Durable) des transactions des SGBDRs ne sont pas complètement respectées au profit des performances et du passage à l'échelle
- **BASE** :
  - réplication et partitionnement horizontal/sharding pour aller vers de la haute disponibilité des données distribuées sur les différents noeuds du système (faire en sorte de diminuer l'impact de pannes éventuelles). Le résultat en est un système hautement disponible même si des sous-ensembles de données peuvent devenir indisponibles sur de de courtes périodes $\Leftarrow$ "disponible à court terme"

Parti-pris Base (issu de CAP) versus propriétés ACID

BASE : **B**asically **A**vailable, **S**oft state, **E**ventual consistency

- dans l'idée les systèmes NoSQL garantissent que les données deviennent cohérentes non pas en instantané mais au travers du temps. Les propriétés ACID nécessitent un verrouillage pessimiste et obligent à vérifier la cohérence des données à chaque fin de transaction. BASE propose une vision optimiste en reportant à plus tard la vérification de la cohérence de la base de données $\Leftarrow$ "cohérente à terme".
- L'état du système peut changer au travers du temps et cela sans nouvelle mise à jour en raison du modèle "cohérence à court terme" $\Leftarrow$ "Etat lâche"

Typologie des systèmes NoSQL

Au regard du mode de représentation choisi

- principe de base : clé/valeur
  - **Systèmes clé/valeur distribués**
  - **Systèmes orientés colonne**
  - **Systèmes orientés document**
- **Systèmes orientés graphe**
- dans un certaine mesure les triples stores et les SGBDOO

Illustration typologie NoSQL

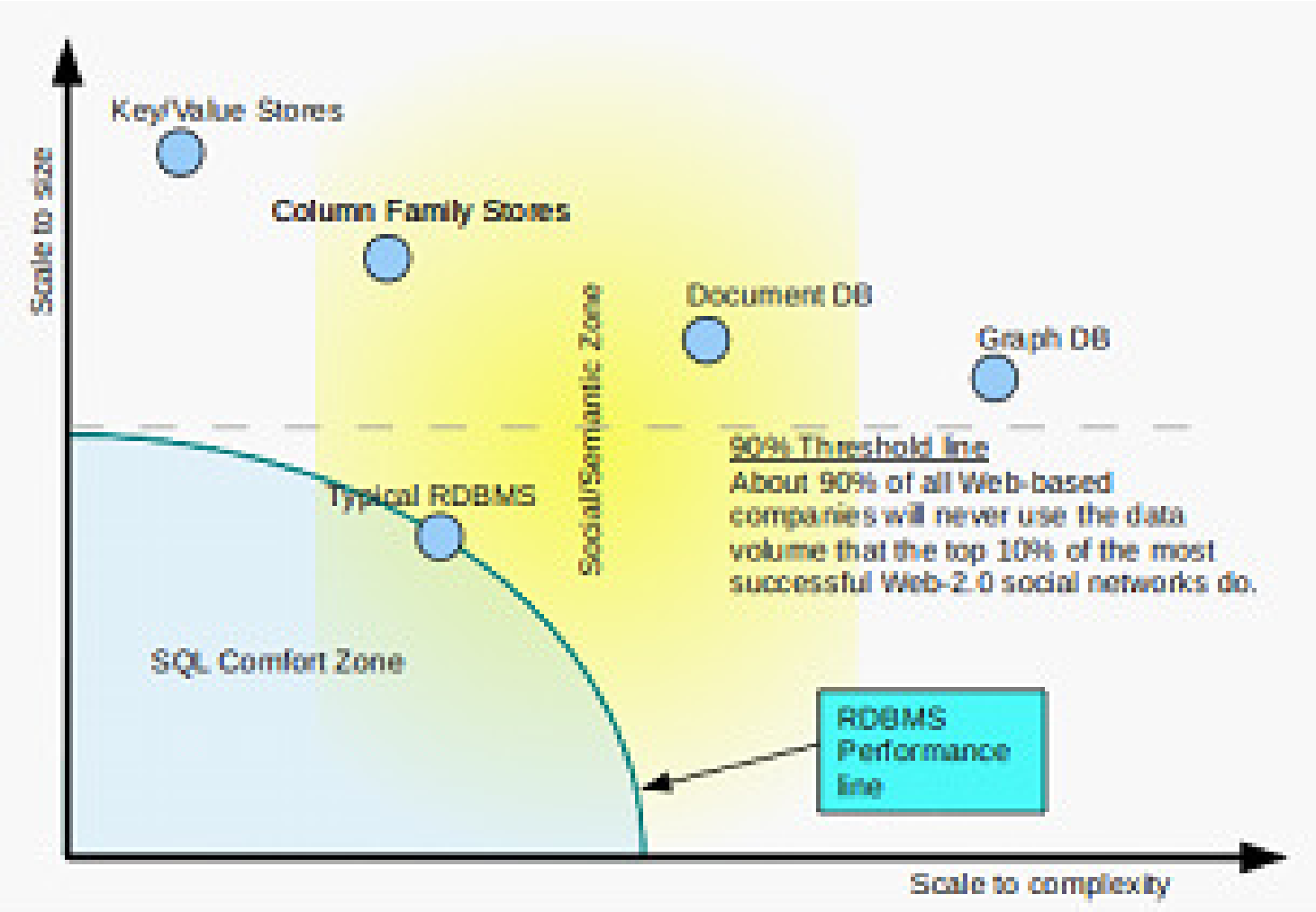

Figure:

Difficulté : absence de standards

Au regard du mode de représentation comme du système choisis

1. **APIs spécifiques**
2. **Terminologies propriétaires**
3. **Mécanismes de requêtage à géométrie variable**

1. **Systèmes ayant fait école (”proofs of concept”)**
   1. BigTable
   2. Memcached
   3. Amazon’s Dynamo

Systèmes existants

Table: Quelques systèmes et leurs modes de représentation

Name	Mode représentation	CAP
CouchDB	Document	AP
MongoDB	Document	CP
Neo4j	Graph	CA
GraphDB	Graph	unknown
Hbase	Column	CP
Memcachedb	Key-Value	unknown
Riak	Key-Value	CP
Project Voldemort	Key-Value	AP
Cassandra	Column	AP
Hypertable	Column	unknown

Systèmes existants

Table: Applications communautaires sur le Web

Name	Système NoSQL	Mode
Google	BigTable, LevelDB	Column
LinkedIn	Voldemort	Key-Value
Facebook	Cassandra	Column
Twitter	Hadoop/Hbase, Cassandra	Column
Netflix	SimpleDB, Hadoop/HBase, Cassandra	Column
CERN	CouchDB	Document
Amazon	Dynamo	Key-Value

NewSQL (M. Stonebraker), une alternative à NoSQL

Alléger les SGBDR pour les rendre à nouveau performants

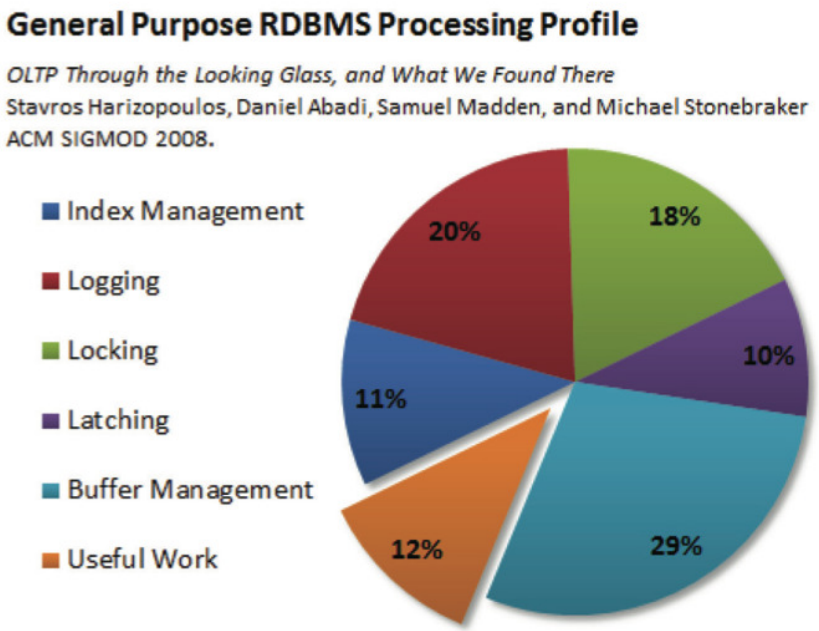

Figure: Trop de tâches complémentaires pour les SGBDR

Fondamentaux NeWSQL

Caractéristiques essentielles

1. Langage SQL
2. Transactions ACID
3. contrôle de concurrence sans verrouillage (ex. MVCC)
4. Architecture distribuée performante pour chacun des noeuds
5. Architecture ”shared-nothing”

voir M. Stonebraker, 2011: New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps

Quelques exemples de systèmes

1. VoltDB
2. Citus (au dessus de PostgreSQL)
3. CockroachDB
4. Spanner
5. MemSQL
6. ...

voir db-engines.com