Universitatea “Alexandru Ioan Cuza”
Facultatea de Informatică

Conf. Dr. Lenuța Alboaie
adria@info.uaic.ro

Cuprins

- Patternuri de Aplicatii in Cloud
- *Sharding*
  - Definitie si caracteristici
  - Tipuri
  - Provocari si probleme
  - Usecase: Flicker
- *Cloudbursting*
  - Usecase: eventseer.com

Pattern-uri de aplicatii in cloud

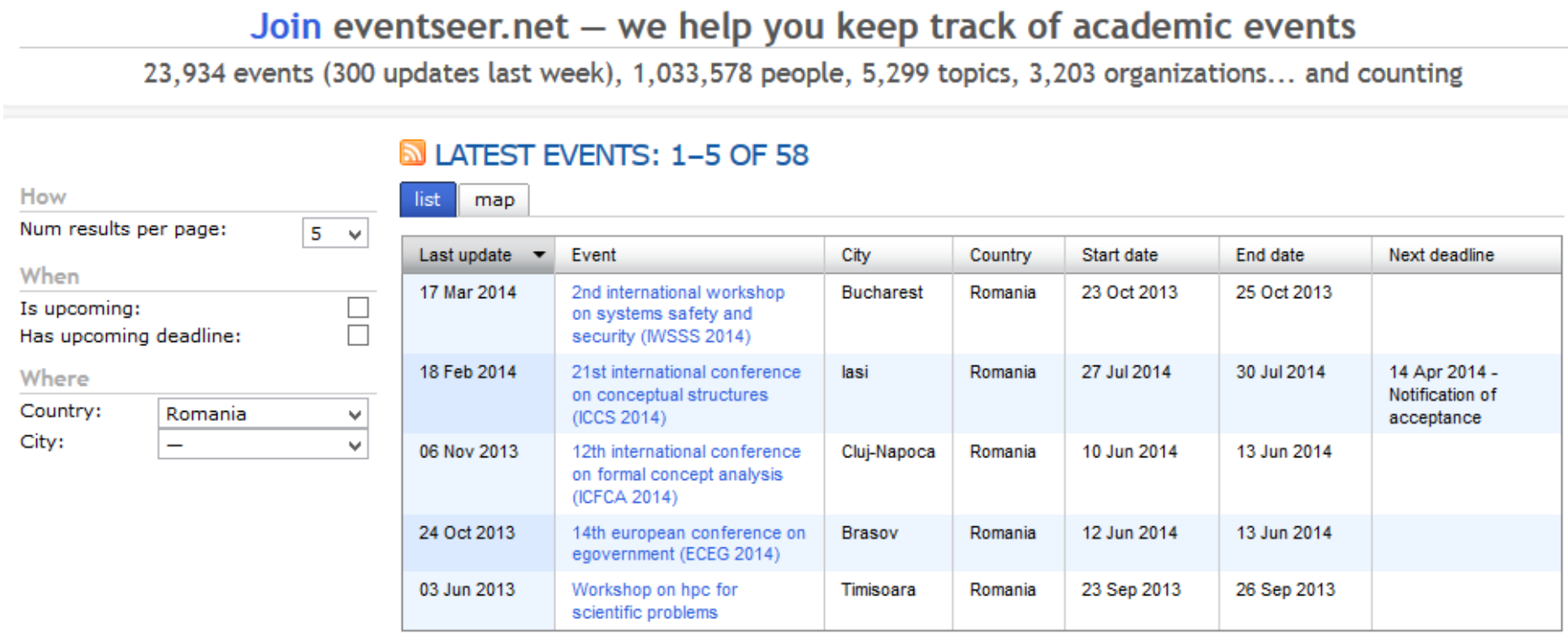
- Presupunere: doriti sa creati o aplicatie in cloud?
  - Care ar trebui sa fie arhitectura?
  - Care sunt componentele arhitecturale?
  - Ce caracteristici sunt obligatorii?
  - Ce nevoi hardware exista?
  - ….
  - Cum sa proiectez a.i. sa nu fie nevoie de *reengineering*?

Pattern-uri de aplicatii in cloud

- *Transference*
  - Mutarea aplicatiilor *on-premise* in cloud
  - Exemplu: email, CRM, …
  - Patternul este de obicei determinat de motive economice
  - Problema: gradul de customizare oferit de furnizorul de cloud
- *Internet-scale*
  - Aplicatii speciale pentru sisteme cloud care au abilitatea de a face mangementul unui numar mare de utilizatori (YouTube, Flicker, Facebook, Twitter)
- *Burst compute*
  - Sunt aplicatii care au abilitatea de a face managmentul de resurse de calcul suplimentare in functie de cerere
  - Exemplu: eventseer.net (site care avea peste 600.000 de pagini)– folosea cloud-ul Amazon pentru a genera paginile care au suferit actualizari/modificari

Pattern-uri de aplicatii in cloud

- Evenseer.net



Pattern-uri de aplicatii in cloud

- *Elastic storage*
  - Potrivit aplicatiilor care au nevoie de crestere exponentiala din perspectiva stocarii
  - Obs. Apelare la mecanism de stocare in cloud dar procesare locala => performante inacceptabile

Centralizare?

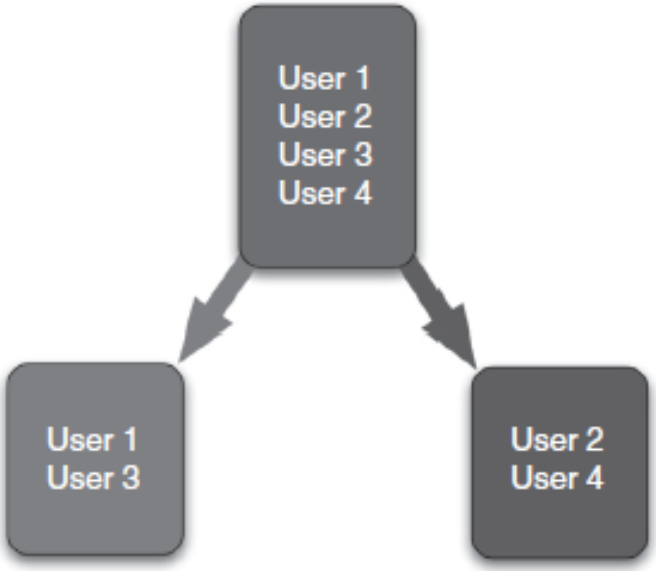
- => sisteme ca eBay nu pot supravietui cu o baza de data centralizata
- Memorie necesara pentru a tine resursele accesate frecvent
  - Nu poate fi furnizata in mod obisnuit folosind *commodity machine*
  - Numarul de operatii de *write* foarte mare duce la “sugrumarea” (eng. *throttling*) aplicatiei
- Solutie:
  - *Sharding* – procesul de partitionare a bazei de date
    - Termen inventat de inginerii Google, dar termenul de *shared-nothing database partitioning* era deja utilizat

> **SHARDING** A decomposition of a database into multiple smaller units (called *shards*) that can handle requests individually. It’s related to a scalability concept called *shared-nothing* that removes dependencies between portions of the application such that they can run completely independently and in parallel for much higher throughput.

  - *E.g.* in Fliker, noua arhitectura a permis un bilion de tranzactii per zi si raspuns la nivel de secunde

Sharding

- Exemplu de model de partitionare bazat pe criterul de sharding: id-uri pare si id-uri impare

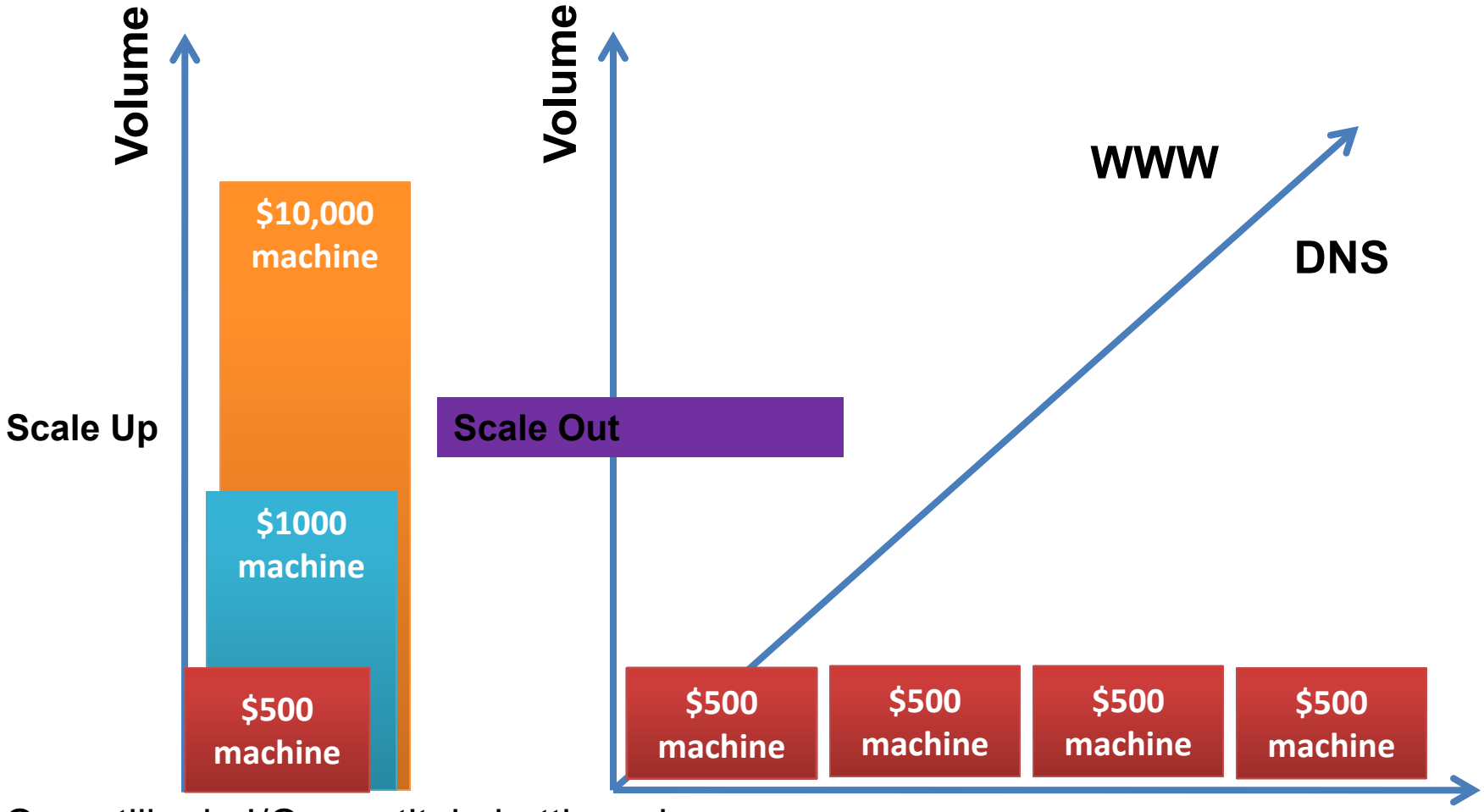


- *Avantaje:*
  - *High availability:* daca o partitie cade, cealalta parte ramane neafectata
    - Obs. In cloud exista replici si a partii afectate
  - *Faster Queries*: cantitati mai mici de date implica raspunsuri mai rapide
  - *More write bandwidth:* operatii in paralel

Sharding

- Facebook:
  - 2004 : folosit ca *online yearbook* -> server centralizat
  - 2008: 5000 de pagini vizualizate per secunda (14 bilioane vizualizari pe luna) => nevoie de CPU, memorie, I/O si storage
  - In acest moment Facebook stocheaza in jur de 40 de bilioane de fisiere => stocare la nivel de petabytes
  - …la un anumit moment a existat atingerea limitei fizice a resurselor disponibile….=> solutii?

Scale-up versus *Scale-out*



- Operatiile de I/O constituie bottleneck

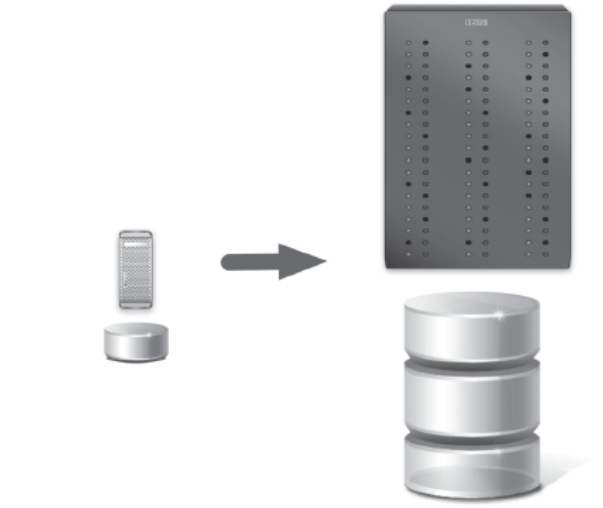
Sharding versus BD traditionale

- In sharding, in general, datele nu sunt normalizate
  - In BD traditionale,
    - normalizarea asigura baze de date cu o structura usor de inteles

    dar cu un nivel de scalabilitate scazut
  - Ex. In sharding mutarea unor date nu implica referinte la alte tabele din BD, partajata de utilizatori multipli si care este un *single point of congestion*
    - Obs. Nu inseamna ca datele unui utilizator nu pot fi separate, e.g. profilul utilizator sa fie stocat separat fata de comentarii, blog, email, media etc.
  - Operatiile pe care le face dezvoltatorul sunt: “get a blob of data”, “store a blob of data”, fara operatii de join

Sharding versus DB traditionale

- Scalabilitatea in modul traditional versus sharding



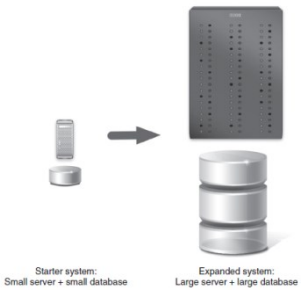
Starter system:
Small server + small database

Expanded system:
Large server + large database

Internet scaled system:
Many commodity servers + many parallel databases

Sharding versus DB traditionale

- Scalabilitatea in modul traditional versus sharding
  - » Cu cat cantitatea de date este mai mare cu atat este mai greu realizarea unui mecanism de cache inteligent => "*data are kept small*" => datele sunt mai usor de salvat, restaurat si se poate face un mangement mai usor



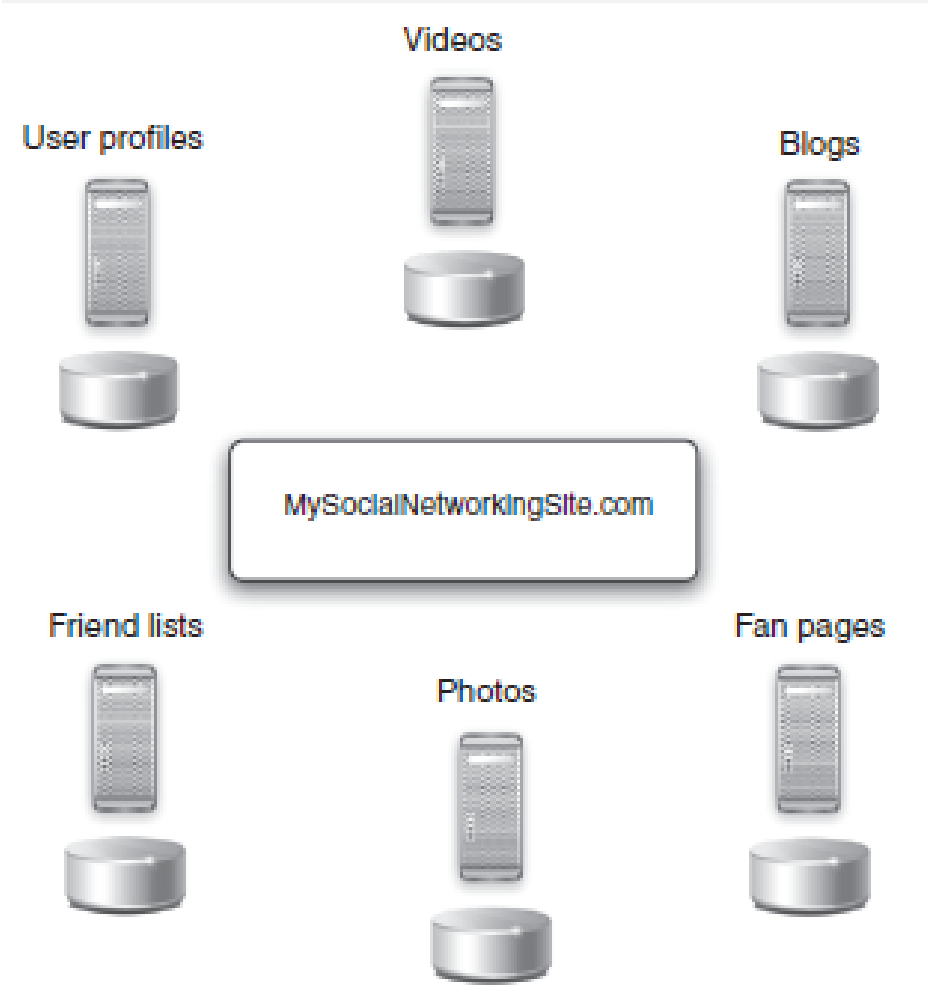
Internet scaled system:
Many commodity servers + many parallel databases

  - In sharding datele nu sunt replicate
  - In metoda traditionala: pentru asigurarea scalabilitatii, datele sunt scrise intr-un nod master si sunt replicate la noduri slave
    - => nodul master este "bottleneck" la operatii de scriere
    - Replicarea implica costuri de CPU, latime de banda, disk
    - Probleme de consistenta

Sharding

- Tipuri de sharding
  - **Partitionare verticala**

  Exemplu: mutarea tabelelor legate de o anumita facilitate pe propriul server
  - In exemplul alaturat avem un caz de partitionare verticala
  - Avantaj: impact minim asupra implementarii aplicatiei
  - Dezavantaj: in caz de cresteri masive, va fi nevoie de un nou nivel de sharding



Sharding

- Tipuri de sharding
  - **Partitionare *range-based***
    - Daca datele sunt asociate unei anumite facilitati (e.g. Photos, Video, …) au nevoie de un nou nivel de sharding, impartirea se face intr-o maniera predictibila
    - Ex: Impartirea tranzactiilor dupa anul in care au fost facute sau impartirea utilizatorilor dupa codul postal
    - Probleme: daca valoarea aleasa pentru partitionare nu este aleasa corespunzator => *unbalanced sharding*
      - serverul ce detine anul curent se poate confrunta cu operatii de read/write mult mai mari
      - daca aplicatia este populara in anumite zone geografice si mai putin in altele

Sharding

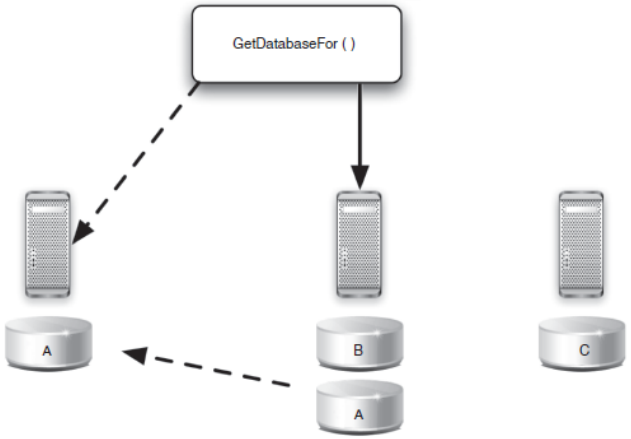
- Tipuri de sharding
  - **Partitionare *hash-based* sau *key partitioning***
    - Fiecare entitate are o valoare care poate fi utilizata ca input pentru o functie hash al carei output identifica ce server de baza de date trebuie utilizat
    - Exemplu: Consideram 10 servere de BD ce stocheaza utilizatori. Utilizatorii au un ID numeric si cand sunt adaugati ID-ul este incrementat ; functia hash face o operatie de *modulo* asupra ID-ul utilizatorului si identifica serverul => o alocare uniforma a datelor pe server
    - Problema: Numarul de servere este fixat, adaugarea de noi servere implica schimbarea functiei hash ⇔ "*asked to change the tires on a moving car*"

Sharding

- Tipuri de sharding
  - ***Directory-based partitioning***
    - Folosirea unui nivel intermediar de abstractizare intre codul aplicatiei si schema de partitionare ⇔ un serviciu de *lookup* care stocheaza/returneaza maparea intre cheia (key) entitatii si serverul de baze de date
      - => aceasta solutie *loosed coupled* permite adaugarea de servere sau schimbarea schemei de partitionare fara afectarea aplicatiei
    - Daca in exemplul anterior adaugam inca 5 servere de baze de date?
      - Solutie: rularea unui script care face mutarea de pe 10 pe 15 servere conform unei noi functii de hash care face operatia de modulo asupra ID-urilor utilizatorilor dar facand distributia la 15 servere
      - Obs. Utilizatorii isi actualizeaza permanent datele….
    - => abordarea este complicata
  - Sharding este o solutie puternica dar principiul este: "*sharding shouldn't be used too early or too often*"

Sharding| Provocari si probleme

- *Rebalancing data*
  - Ce se intampla cand un shard necesita mai mult storage si este nevoie de impartire?
    - => inchiderea serviciului? ☹
  - Mecanismul de *rebalancing* trebuie proiectat de la inceput
    - Se creaza un serviciu de numire, care tine referinte la date si permite realocarea lor



    - Partitioneara *directory-based* face ca rebalansarea sa fie relansabila (chiar daca avem un nou single *point of failure* – serviciul de *lookup*)

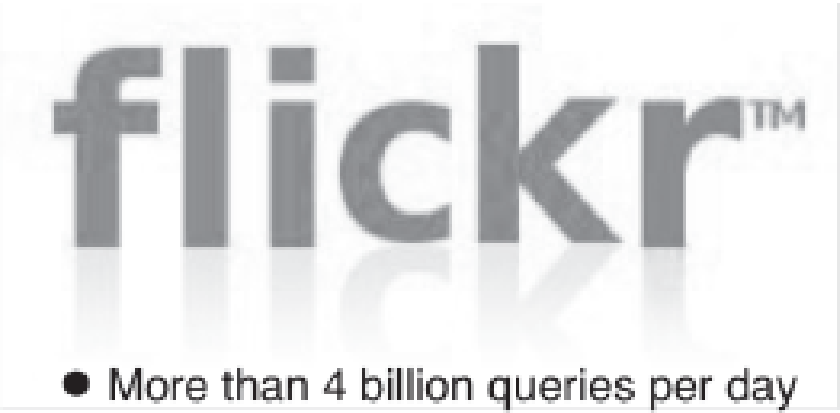
Sharding| Provocari si probleme

- Obtinerea datelor de la shard-uri multiple
  - E.g crearea unei pagini complexe necesita interogarea de resurse multiple
    - Mecanismele de cache si retelele de viteza mare asigura un timp de raspuns bun
- Suport – exemple:
  - *Hibernate ofera librarii pentru sharding*

  Sharding with Hibernate:
  http://aws.amazon.com/articles/Amazon-RDS/004302286264415
  - MySQL a adaugat suport pentru partitionare
  - …
  - *"sharding is something you must implement yourself"*

Sharding

Usecase: Flicker



- More than 4 billion queries per day
- ~35 million photos in Squid cache (an open source Web delivery system)
- ~2 million photos in Squid's RAM
- ~470 million photos, 4 or 5 sizes each
- 38 thousand requests/second to memcached (open source distributed memory system)
- 2 petabytes raw storage
- ~400,000 photos added every day

- *http://highscalability.com*

Sharding

Usecase: Flicker

- Schema de partitionare:
  - Flicker - asigneaza un numar random pentru noile conturi si foloseste acest numar ca index pentru shard-ul corespunzator acelui utilizator
  - *ShardToUse = RandomNumber mod NumberofShards;*
  - Un shard este proiectat sa tina date de la aproximativ 400.000+ de utilizatori, separat de imagini
  - Din cand in cand apar utilizatori care distrug balanta intre sharduri -> are loc migrarea lor, in mod manual in alta parte a bazei de date
  - Anumite date sunt stocate de doua ori: e.g. comentariile (= relatie intre cel care a comentat si cel care a primit comentariul) sunt stocate in ambele parti => compromis intre performanta si utilizarea disk-ului

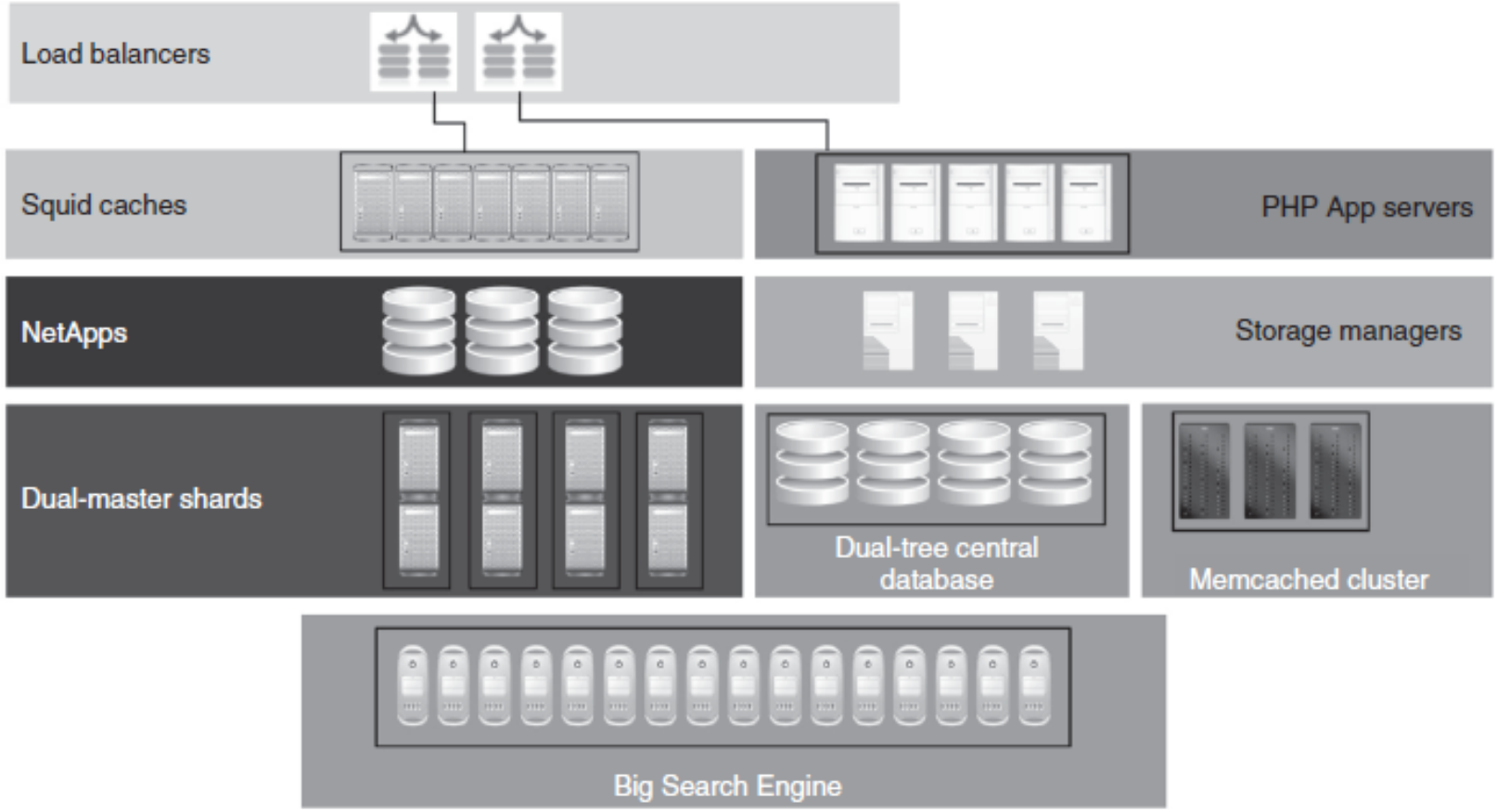
Sharding

Usecase: Flicker

- Anumite operatii (e.g.click pe un favorit) implica cateva shard-uri
  - Contul detinatorului acelei poze este preluat din cache pentru a se obtine locatia shard-ului acestui utilizator
  - Se preiau informatiile despre “mine” pentru a se obtine locatia shard-ului
  - Se porneste o tranzactie distribuita pentru a raspunde la interogari
- Strategia pentru asigurarea *reliability?*
  - Fiecare server intr-un shard suporta o incarcare de 50%
  - =>Flicker poate face *shut down* la jumatate din servere din shard
  - => un server poate suporta incarcare maxima daca un alt server este in mentenanta sau este cazut
  - => pentru upgrade, compania inchide jumatate din shard-uri , face upgrade la jumatate si apoi repeta procesul cu cealalta jumatate

Sharding

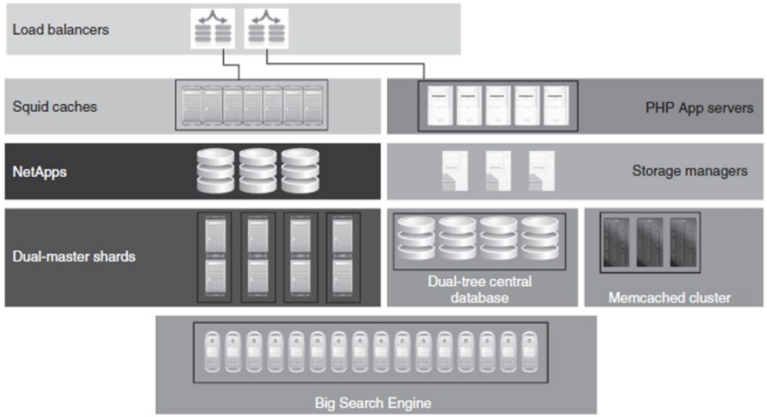
Usecase: Flicker



Sharding

Usecase: Flicker

- *Squid caches* – sunt cash proxy care suporta HTTP, HTTPS pentru livrarea de pagini Web sau imagini
- *PHP App Servers* – se conecteaza la shard-uri si mentin datele consistente
- *Storage managers* – realizeaza maparea index – shardul potrivit
- *NetApps* – utilizat pentru stocarea in masa a pozelor
- *Dual- master shard* – este specific arhitecturii Flicker pentru asigurarea rezilientei in caz de esec
- *Dual-tree central database* – include date de tipul tabele de utilizatori si pointeri la shard-urile care contin datele acestora
  - permite scalarea incrementala prin adaugarea de noduri master;
- *Big Search Engine* – este o replica a bazei de date Flicker



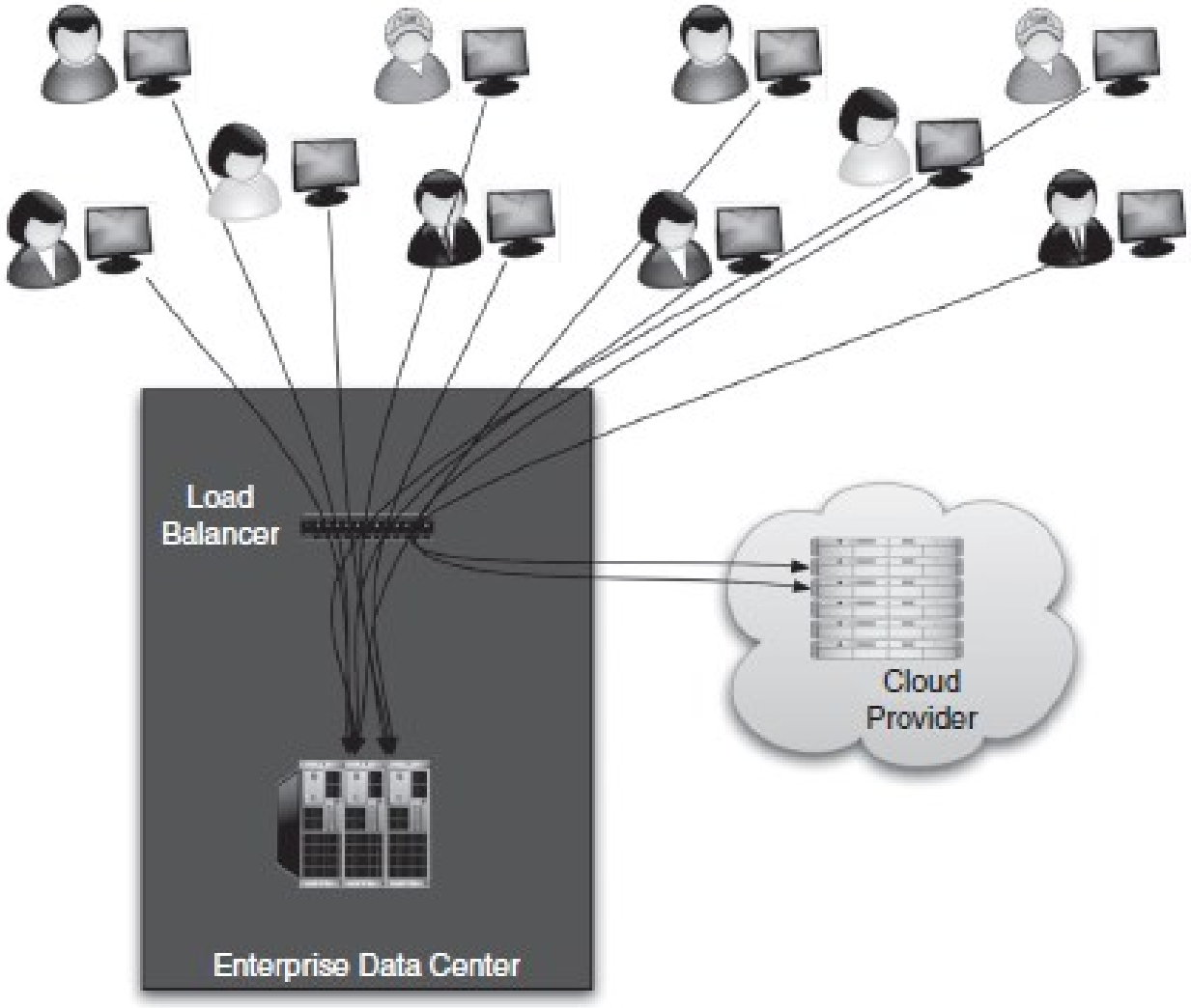
Cloudbursting

- Proiectarea aplicatiei a.i. in caz de necesitatea unor resurse suplimentare acestea sa poata apela in mod dinamic la resurse din cloud in maniera *pay-as-you-go*

CLOUDBURSTING In 2009, the National Institute of Standards and Technology (NIST) published its formal definition of cloud computing, which included the concept of *cloudbursting* as cloud infrastructure that is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

- Utilitate: cresteri sezoniere a traficului care depasesc puterea infrastructurii existente

Cloudbursting



Cloudbursting

Cloudbursting– aspecte economice

- Exemplu:
  - Compania X achizitioneaza un server de $500
  - Aplicatia furnizata de X este solicitata de doua ori pe zi
    - Solutii:
      - Cumpararea a 10 servere care sa satisfaca cererea ($5000) , dar acestea vor fi *idle* o mare parte de timp
      - Apelarea la 10 AMI din EC2 (~ $0.20 per ora) -> $48
- => ne mutam cu totul in cloud?
  - Daca avem nevoie de cloud pe termen lung atunci costurile sunt mai mari
  - Problema *lock-in*

Cloudbursting

Use case: eventseer.com

- ~600.000 de pagini interconectate, si fiecare adaugare de eveniment implica modificari in cascada in paginile existente
- Traficul creste => Eventseer este mai lent (chiar si folosind cashing si reducand numarul de interogari la BD)
- Motoarele de cautare accesand cele 600 de pagini generate dinamic au constituit de asemenea o problema

Decizia: generarea de pagini statice in fiecare noapte care vor fi deservite motoarelor de cautare, utilizatorilor neautentificati => eliberarea CPU

Problema: generarea celor 600.000 de pagini se face in 7 zile pe un singur server….☺

=> *Cloudbursting architecture*

Cloudbursting

Use case: eventseer.com

- Eventseer.com foloseste servicii: Amazon EC2 si S3, Simple Queue Service (SQS)
  - Pagini generate intr-o singura noapte <= IaaS este solutia (25 de instante Amazon (~$12.50)) + asigurarea ca instantele folosesc datele cele mai noi
    - Baza de date din Eventseer este sincronizata in mod regulat cu Amazon Simple Storage Service (S3)
- => problema oferirii de resurse actualizate este rezolvata in parametri acceptabili
- Obs. Eventseer mentine o coada locala in care inregistreaza modificarile care au loc, apoi aceasta este sincronizata cu SQS din Amazon
- Cand instantele EC2 isi finalizeaza treaba sunt inchise automat

Bibliografie

- Implementing and Developing Cloud Computing Applications, DAVID E.Y. SARNA, CRC Press, Taylor&Francis Group, 2011
- Cloud Computing, Software Engineering Fundamentals, J. Heinzelreiter, W. Kurschl, www.fh-hagenberg.at
- The Cloud at Your Service, Jothy Rosenberg, Arthur Mateos, 2011, Manning Publications

Rezumat

- Patternuri de Aplicatii in Cloud
- Sharding
  - Definitie si caracteristici
  - Tipuri
  - Provocari si probleme
  - Usecase: Flicker
- Cloudbursting
  - Usecase: eventseer.com

Universitatea “Alexandru Ioan Cuza”
Facultatea de Informatică

Întrebări?