FERMELE WEB. EXTINDEREA NIVELELOR WEB TIER ŞI DATA TIER

BARBARII Veaceslav

Universitatea Tehnică a Moldovei

Abstract: *În acest articol, sunt discutate noţiunea de Fermă Web („Web Farm”), de ce e nevoie să creăm o Fermă Web, caracterul fiabil şi scalabil al acesteia, principalele beneficii ale procesului de management al unei Ferme Web, ce rezidă în flexibilitate şi operativitate, avantajele extinderii nivelului Web Tier faţă de extinderea lui Data Tier, de ce este complicată extinderea nivelului Data Tier şi specificul procesului de extindere în dependenţă de reprezentarea datelor – relaţională sau non-relaţională (NoSQL), avantajele şi dezavantajele fiecărei tehnologii etc.*

Pe lângă aceasta, sunt analizate principalele tipuri de distribuitori de sarcină în universul familiei Microsoft, diferenţele dintre acestea şi, de asemenea, importanţa identităţii serverelor web pe care urmează a fi efectuată derularea (deployment-ul) codului.

Cuvinte cheie: *Web Server, Cluster, Web Farm, Web Farm Framework, Code deployment, Data Tier, Web Tier, Tier scaling, Management, Load Balancing Service, RDBMS, BASE DBMS, NoSQL.*

1. Introducere

Conceptul de „*Fermă Web*” a apărut odată cu apariţia ideii de creare a unor sisteme de calcul puternic încărcate, bazate pe distribuirea resurselor de calcul dintre mai multe servere web. Fermele web sunt utilizate în cazul în care un singur server nu este în stare de a satisface nevoile precum deservirea unui număr imens de oameni în limitele unui website, furnizarea resurselor şi serviciilor în cadrul unui *Cloud* etc. Există nişte cazuri specific când fermele web se utilizează pentru *3D rendering*, modelări şi calcule ştiinţifice, modelări ale condiţiilor meteorologice ş.a.m.d.

În ceea ce priveşte dezvoltarea produselor soft pentru crearea şi gestionarea fermelor web − principalii rivali pe piaţa mondială sunt Corporaţia Microsoft şi Oracle.

2. Caracteristicele de bază ale *Data Tier*, *Web Tier* şi a Fermelor Web la general

În ultimii ani, în contextul cerinţelor crescânde faţă de stabilitate, fiabilitate şi performanţă înaltă a serviciilor şi aplicaţiilor web, apare tendinţa de extindere a capacităţilor de calcul şi a receptivităţii nivelelor de acces la date (*Data Tier*) şi a nivelului de *business computing* (*Web Tier*) a serviciilor web.

Pentru extinderea nivelului *Data Tier*, se potrivesc cel mai mult bazele de date non-relaţionale în contextul *NoSQL,* care au fost concepute pentru o utilizare eficientă, fiind distribuite între mai multe servere. Principalele avantajele ale bazelor de date *NoSQL* sunt scalarea înaltă, calculul distribuit, costurile reduse, flexibilitatea schemei (*scheme-less property*), posibilitatea păstrării datelor nestructurate şi semistructurate, lipsa relaţiilor complexe dintre noduri etc. *NoSQL* urmăreşte principiile *CAP* (*Consistency, Availability, Patition Tolerance*) formulate de Eric Brewer, care afirmă că pentru un sistem computerizat este imposibil de respectat simultan cele trei proprietăţi. *Consistenţa* presupune faptul că orice cerere de citire primeşte cea mai recentă înregistrare actualizată; *Disponibilitatea* rezidă în primirea răspunsului de către fiecare cerere, fără garanţie că răspunsul conţine cea mai recentă versiune a informaţiei; *Toleranţa faţă de partiţionare* denotă că dacă o parte a bazei de date este indisponibilă, celelalte părţi componente sunt neafectate.

Sistemele de gestiune a bazelor de date relaţionale (*RDBMS*) susţin Consistenţa şi Disponibilitatea, pe când sistemele de gestiune a bazelor de date *BASE* (*Basically available, Soft State, Eventual Consistency*) au ales susţinerea Disponibilităţii şi a Toleranţei faţă de partiţionare, în detrimentul Consistenţei, astfel nodurile bazei de date non-relaţionale (*NoSQL*) rămân permanent online, chiar dacă acestea nu pot comunica unul cu altul, şi vor efectua sincronizarea datelor îndată ce problema partiţionării datelor va fi soluţionată, ceea ce, la rândul său, nu ne asigură că toate nodurile vor avea aceleaşi date (în timpul său după procesul de partiţionare).

Din acest motiv, nu în toate cazurile ne convine utilizarea unui model non-relaţional al entităţilor ce persistă într-o bază de date. Astfel, există situaţiile când utilizarea modelului relaţional al bazelor de date ne aduce un şir de avantaje, precum urmărirea principiilor *ACID* (*Atomicitate, Consistenţă, Izolare, Durabilitate*), reducerea duplicării datelor în BD prin procesul de normalizare, existenţa unui asortiment bogat de *ORM*-uri orientate pe modelul relaţional al datelor etc. Dar *ACID*-ul implică şi unele restricţii asupra scalării bazei de

date prin faptul că *atomicitatea* – impune indivizibilitatea tranzacțiilor care trebuie efectuate ca un tot întreg sau rebutate (după care urmează un *rollback* la starea inițială a datelor), ceea ce aduce un șir de probleme de izolare, în timpul blocării tabelelor sau tuplurilor din aceste tabele care se află pe diferite servere unde sunt localizate *SGBD*-urile; Constrîngerile de *consistență* impune identitatea datelor a tuturor grupurilor de noduri din cadrul clusterului, ceea ce înseamnă că realizarea procesului de scriere și actualizare a datelor este destul de complicată și îngreunează scalarea lui *Data Tier*; *Durabilitatea* înseamnă că pentru a nu pierde niciodată datele, fiecare proces de scriere trebuie să fie urmat de persistarea obligatorie a datelor înaintea de a trimite un răspuns (*response*) clientului.

Din această cauză, în cazul utilizării sistemelor de gestiune a bazelor de date relaționale, suntem limitați în scalarea lui *Data Tier*. Migrarea datelor unui proiect existent, bazat pe utilizarea a BD relaționale, spre BD non-relaționale (*NoSQL*) este, de asemenea, destul de anevoioasă, deoarece implică necesitatea de a avea o bună experiență practică de proiectare a BD non-relaționale, necesitatea de concepere preventivă a modelului orientat pe documente în cadrul colecțiilor, rescrierea codului aplicației pentru interogările specifice pentru SGBD-ul *NoSQL* ales, prin instrucțiunile precum *insert()* sau *find()*, testarea ulterioară a funcționării *API*-lui de acces la date ș.a.m.d. De aceea, în majoritatea cazurilor se procedează în modul următor: se procură un server mai mare și mai puternic pentru *Data Tier* sau se recurge la divizarea logică a bazei de date.

Divizarea logică a BD constă în următorul procedeu:

− Dacă o BD poate fi separată pe diferite scheme de acces independente pentru diferite grupuri de utilizatori, atunci relațiile (tabelele), ce aparțin fiecărei scheme pot fi trecute în diferite BD și distribuite pe diferite servere din clusterul *Data Tier*;

− Urmărind principiul de grupare a datelor după anumite criterii comune, precum cele regionale. Spre exemplu, structura unei baze de date ale unui magazin online ce deservește comenzile internaționale (regionale sau continentale) precum eBay, AliExpress etc., poate fi clonată, creând baze de date specializate pe deservirea clienților amplasate în țările concrete (*NameDB*_France, *NameDB*_Germany, etc) sau pe continente concrete (*NameDB_NA, NameDB_EU, NameDB_ASIA*, etc.) ceea ce favorizează distribuirea bazelor de date pe diferite servere din cadrul *cluster*-ului ce formează *Data Tier*.

În practică, efectuarea operațiilor de divizare logică a BD (descrise mai sus) implică mult efort și, în majoritatea cazurilor pot fi acceptabile numai la etapa de proiectare a serviciului web. Dar, în cazul în care dispunem de o bază de date indivizibilă/extrem de mare și pretențioasă (*performance demanding DB*) și necesită mult RAM adițional sau capacitățile mari ale CPU pentru o bună funcționare, atunci suntem nevoiți să mărim chiar potențialul serverului prin multiplicarea procesoarelor sau înlocuirea acestora cu altele de frecvență mai mare și/sau un număr mai mare de procesoare, să mărim capacitatea RAM, să adăugăm mai multe discuri rigide pentru scriere/citire (I/O) sau, în cel mai rău caz, să procurăm un server mai puternic, ceea ce este destul de costisitor.

Din această cauză, scalarea lui *Data Tier* este dificilă, spre deosebire de *Web Tier*, care poate fie extins cu ușurință datorită proprietății de independență de stare (*stateless*). De aici apare idea de dezvoltare a unei Ferme Web (*Web Farm*).

Ferma Web (*Web Farm*) reprezintă un grup de 2 sau mai multe servere web care furnizează (îndeplinesc) același serviciu. Conceptual, putem multiplica același server web de mai multe ori. Cu alte cuvinte, o fermă web reprezintă varianta simplificată a clusterului (conglomerație de servere accesibile prin Internet) având aceeași IP adresă (prin intermediul căreia clientul are acces la serviciul furnizat de fermă). În continuare, acest grup de servere trebuie gestionat utilizând un dispozitiv sau un serviciu de echilibrare a sarcinii (*Load Balancing Service, LBS*). *LBS*-ul, la rândul său, efectuează rutarea traficului spre serverele web individuale, în dependență de algoritmul de rutare stabilit.

Fermele web au soluționat problema serverelor puternic încărcate, găsindu-și aplicarea practică la deservirea portalurilor web de dimensiuni mari, site-urilor guvernamentale mari, portalurilor de știri, rețelelor sociale, a operatorilor de e-mail etc.

Crearea fermelor web are un șir de avantaje. Iată câteva dintre acestea:

- Fiabilitatea și scalabilitatea;
- Când de la un utilizator vine o cerere, serviciul de echilibrarea a sarcinii o rutează spre serverul web individual care procesează această cerere. Dacă are loc un accident, problemele tehnice ordinare sau cele stârnite de violarea integrității serverului, eroarea de procesare a produs o excepție (*Fatal Error*) și alți factori ce pot duce la funcționarea perturbată a sistemului local (la nivelul unei instanțe de server), atunci *LBS*-ul poate detecta prezența problemei și redirecționa cererile spre alte servere ce lucrează corect și sunt neafectate (*healthy*). Ca urmare, dacă dorim să instalăm careva pachete de corecție (*patches* / *hotfixes*), *update*-uri, să lansăm modulele noi ale serviciului (*module deployment*), avem posibilitatea de a exclude un anumit server web (sau chiar mai multe) din Fermă (din circuitul

de funcționare a procesului de prelucrare a cererilor) sau, pur și simplu, serverul respectiv poate fi oprit. Astfel pot fi efectuate lucrările de mentenanță asupra serverului și apoi reincluderea lui în Fermă;

- Dacă există un singur server web pe care rulează un serviciu și marea cantitate de cereri (traficul enorm) produce probleme de performanță, este posibilitatea de a adăuga mai multe server web în cadrul Fermei. Ca urmare, traficul va fi divizat între serverele web din *Web Tier*. Deci, fiecare server va primi o anumită cotă de cereri (sarcină). Serverele web adăugate pot fi destul de mici și reduse în performanță și, respectiv, de un preț redus. Aici se regăsește principiul de scalabilitate a Fermelor Web.

Totuși, „în fruntea" unei Ferme Web se află distribuitorul de sarcină (*Load Balancer*). Astfel, în continuare, vor fi descrise cele trei tipuri de distribuitori de sarcină (în baza produselor Microsoft)*:*

- HLB (*Hardware Load Balancers*)
- ARR (*Aplication Request Routing*)
- NLB (*Network Load Balancer*)

HLB reprezintă un dispozitiv fizic ce se află pe primul nivel al Fermei Web. Astfel de dispozitive sunt foarte scumpe. Prețul lor variază între 2,000 – 21,000 de dolari.

ARR este o extensie a lui IIS. Se livrează cu titlul gratuit. ARR se configurează în cadrul IIS și este utilizat în conjuncție cu *Web Farm Framework* (dezvoltat de Microsoft Corporation).

NLB este partea componentă a Windows Server OS. Se configurează cu ajutorul aplicației *Network Load Balancing Manager*. NLB nu dispune de *Health Checks*, de aceea, pentru a întrerupe fluxul de cereri spre serverul web ieșit din funcție, acesta necesită excluderea din Fermă și efectuarea ulterioară a lucrărilor de mentenanță.

Spre deosebire de NLB, HLB și ARR amândoi dispun de *Health Checks* ceea ce reprezintă un avantaj colosal. Pe lângă aceasta, cei doi asigură *Caching* și *SSL Offloading,* ceea ce, la rândul său, asigură o perceptibilă îmbunătățire de performanță. *Cache*-ul distribuitorului de sarcină poate reține careva *content,* ceea ce anihilează necesitatea de a apela din nou *Web Tier* pentru extragerea acelorași date. Tehnologia *SSL* este destul de costisitoare din punct de vedere al performanței, de aceea, în practică, adeseori se mută *SSL* de pe *Web Tier* la *LBS Tier* (se prezumă utilizarea unui dispozitiv specializat/un server destul de puternic).

HLB și NLB sunt înzestrați cu Geo Locația (*Geo Location*), care îi oferă posibilitatea de a adresa serverele web și diferite data centre distribuite geografic. Aceștia doi (HLB și NLB) de asemenea pot opera cu traficul *Non-HTTP*, ARR e doar un HTTP serviciu de echilibrare a sarcinii. De aceea, traficul de date precum SMTP, nu poate fi preluat și gestionat de către un ARR.

Toți trei (HLB, NLB și ARR) furnizează posibilitatea de a crea și de a opera cu *sticky sessions.* Atunci când utilizatorul se întoarce înapoi cu următoarea cerere, *LBS*-ul, utilizând mecanismul *sticky sessions*, poate să redirecționeze cererea spre același server web pe care l-a utilizat clientul pentru prima dată. Pentru realizarea acestui tip de sesiuni, adeseori se folosesc *cookies* pentru identificarea și legarea utilizatorului sau a aplicației client de un server web concret. Cu părere de rău, NLB poate să realizeze *sticky sessions* doar utilizând adresele IP (nu poate opera cu *cookies*). Pentru a obține un avantaj din punct de vedere al performanței, se recomandă deconectarea mecanismului *sticky sessions* prin decuplarea afinității serverelor *(server affinity)*.

Codul aplicației web/serviciului web, de obicei, se păstrează într-un *development environment (DV)* izolat. Testarea și configurarea serviciului/aplicației are loc într-o zonă specializată numită *staging environment (ST)*. După ce serviciul/aplicația a fost configurat(-ă) cu succes, are loc copierea integrală a acestuia pe serverele web ce fac parte din *production environment* (*PE*). Serverele din *PE* nu sunt obligatoriu fizice, adeseori, aceștia sunt niște mașini virtuale, cărora i-au fost repartizate resursele fizice ale serverului gazdă.

Microsoft oferă câteva instrumente specializate care ușurează procesul de deployment a aplicațiilor/serviciilor web pe production environment (PE), de exemplu MSDeploy, Web Deploy etc. ce pot fi instalate în mod automatizat de către Microsoft Web Platform Installer.

Trebuie de menționat că este foarte important ca serverele web să fie absolut identice, atât din punct de vedere fizic, cât și la nivel de software; în caz contrar comportamentul Fermei Web ar putea deveni instabil, ceea poate să ducă la un rezultat imprevizibil.

Concluzii

În contextul cerințelor crescânde față de fiabilitatea și stabilitatea sistemelor și serviciilor web, formarea unei ferme web pentru proiectele mari, ce necesită o funcționare continuă, este inevitabilă. Astfel, crearea Fermelor Web, cu siguranță, va fi solicitată pe parcursul viitorilor zeci de ani, atât timp, cât vor fi actuale sistemele bazate pe calculul distribuit.

Bibliografie

1. Build a web farm with iis servers. [resursă electronică]. -regim de acces: https://msdn.microsoft.com/en-us/library/jj129543(v=ws.11).aspx
2. Веб-ферма − что это? [resursă electronică]. -regim de acces: http://www.vladtime.ru/solntse/381797-veb-ferma-chto-eto.html
3. A short history of databases: from rdbms to nosql & beyond. [resursă electronică]. -regim de acces: https://www.3pillarglobal.com/insights/short-history-databases-rdbms-nosql-beyond
4. Web farm framework. [resursă electronică]. -regim de acces: https://www.iis.net/downloads/microsoft/web-farm-framework
5. Overview of network load balancing. [resursă electronică]. -regim de acces: https://technet.microsoft.com/en-us/library/cc725691(v=ws.11).aspx
6. Investigating nosql for ehr systems: mongodb. [resursă electronică]. -regim de acces: http://www.ehrscience.com/2013/04/15/investigating-nosql-for-ehr-systems-mongodb/
7. Cap theorem. [resursă electronică]. -regim de acces: https://en.wikipedia.org/wiki/cap_theorem
8. Web farm. [resursă electronică]. -regim de acces: https://www.techopedia.com/definition/13493/web-farm