Erlang/OTP: magas rendelkezésre állású webalkalmazások fejlesztése

Czinkos Zsolt

2012-04-28

Absztrakt

Az Erlang programozási nyelvet az Ericssonnál hozták létre hálózati eszközök, telefonrendszerek programozására. A magas telekom elvárások tükröződnek az Erlangban fejlesztett rendszerek architektúrájában, amely az Open Telecom Platform szoftverkönyvtárban ölt testet. A dolgozat bemutatja az Erlang programozási nyelvet, a fejlesztési elveket és alkalmazási lehetőségeit a webes technológiára épülő alkalmazások területén.

Mindenkinek, aki szereti

Tartalomjegyzék

1. fejezet

Bevezetés

Az Internet – azon belül különösen a Web – terjedésével párhuzamosan nőtt az igény kiszámítható, jó minőségű szolgáltatásokra. A szolgáltatóknak egyre magasabb elvárásoknak kell megfelelnie – nem utolsó sorban azért, mert a felhasználók fejében a web és az ingyenes tartalom összeforrt. Még színvonalas termékekért is nehezen adnak ki pénzt, nemhogy hibás, elavult tartalomért, akadozó és kiszámíthatatlanul működő szolgáltatásokért. Ma már a hálózat nem csupán mérnököknek, kutatóknak érdekes kábelezést jelent, amely így-úgy hasznos a tudományos kutatásaik során, hanem a mindennapi élet részét képező társadalmi kapcsolatok *reprezentációját* is. A felhasználó egyre aktívabb *cselekvője* ezeknek a valós vagy virtuális világban létrejött hálózatoknak, egyre inkább itt keresi (és többnyire találja meg) azt a teret, ahol ismerik, és ő is ismer, ahol ura annak az eszköztárnak, amelynek birtokában különböző – rövid, prompt, aszinkron, szöveg, hang vagy videó – *üzenetek* segítségével ápolni tudja kapcsolatait. Ez a kapcsolatrendszer és eszköztár jelenti azt az új mikroikozmoszt, amelyben a felhasználó – cselekvő- és befolyásolóképessége tudatában – kényelemben és biztonságban érzi magát.

Ez a kényelem és biztonság *függővé* tesz: világunk megszokott működésének zavarait nehezen vagy egyáltalán nem tudjuk tolerálni, kiszolgáltatottnak és tehetetlennek érezzük magunkat. Ilyenkor derül ki, hogy bár mikrokozmoszunkat ismerni véljük, az azt működtető rendszer elemeit meg sem tudjuk nevezni, csak azt tudjuk, hogy „van” (ez a valami pillanatnyilag a legtöbb ember számára néhány cég szolgáltatásában ölt testet: Facebook, Google, Twitter). A szoftverfejlesztőknek, tervezőknek ennek a világnak a működtetéséhez szükséges rendszert kell tudniuk megépíteni és üzemeltetni úgy, hogy a felhasználók a lehető legkevesebb alkalommal szembesüljenek azzal, hogy kihúzták alóluk a talajt. Nem emberbaráti, hanem üzleti megfontolások miatt.

A weben a sikerhez elengedhetetlen a folyamatos és megbízható szolgáltatás, nagyon alacsony az ingerküszöb, ha egy oldal betöltődése tovább tart mint 4 másodperc, már odébb is állt a felhasználó (Akamai felmérés, 2006). Ha túl sokszor

kap hibaüzenetet – amitől jobb esetben ingerült lesz, rosszabb esetben halálra rémül, hátha ő rontott el valamit –, keres mást. Éppen ezért nagyon fontos, hogy olyan rendszert építsünk, amely

1. folyamatosan, megszakítás nélkül működik;
2. megfelelő válaszidővel, sebességgel működik;
3. funkcionálisan jól működik;
4. a felmerülő hibák nyomon követhetők, kezelhetők.

A fentebb már említett vezető webes cégek mind megfelelnek ezeknek a követelményeknek, persze nem kevés munka és pénz árán. A felhasználót azonban a legkevésbé sem érdekli, hogy a szolgáltatást nyújtó üzleti vállalkozás hogyan tudja működtetni rendszerét, hány embert alkalmaz, stb. Őt az érdekli, hogy neki ingyen vagy elérhető áron a lehető legtöbbet nyújtsa. Ez az elvárása sajnos (vagy szerencsére) nem csak a mammutcégekkel szemben áll fent, hanem minden webes céggel szemben, mindenhol szeretné megkapni azt a minőséget, amihez hozzászokott. Azt a céget tekinti profinak, jónak, amely ugyanazt tudja nyújtani. Ha egy cég sikert akar, akkor már induláskor fel kell készülnie arra, hogy ha elsül a kapanyél, és özönlenek a felhasználók, akkor tartani tudja az iramot, ki tudja szolgálni ugrásszerűen megnőtt ügyfélkörét; miközben egy kezdő vállalkozás nem engedhet meg magának földrajzilag diverzifikált többtízezer gépes szerverparkot: kicsiből indulva kell képesnek lennie a növekedésre.

Hogyan lehet olyan rendszert építeni, amellyel neki lehet vágni egy webes vállalkozásnak úgy, hogy ne kelljen attól félni, mi lesz, ha holnap regisztrál még 10 ezer felhasználó (4 másodperc!), vagy ha tönkremegy az egyik gép?

Számos programozási nyelv és környezet közül lehet ma már választani, amely alkalmas erre a feladatra, ez a dolgozat az Erlang programozási nyelvet és a hozzá kapcsolódó Open Telecom Platform-ot (OTP) mutatja be. Az Erlang egy funkcionális programozási nyelv, amelyet az Ericsson fejlesztett ki mintegy 20 évvel ezelőtt telefonrendszerek, szoftveres kapcsolóközpontok programozásához, a telekommunikációs iparban szokásos rendkívül magas elvárásoknak megfelelően.

Az Erlang megalkotásánál az elsődleges cél magas rendelkezésre állású (*highly available*), hibatűrő (*fault tolerant*) redundáns rendszerek építése volt. Ez az, amire az Erlang igazán alkalmas, ez az a terület, ahol az Erlangnak évtizedes múltja van: akár 99,99999%-os rendelkezésre állás biztosításában. Az hozzávetőleg 5 perc kiesés évente.

1998-ban open source-szá vált a nyelv és a platformot adó szoftverkönyvtárak, azóta bárki használhatja bármilyen feladatra, számos önkéntes és cég teszi be a közösbe a maga alkalmazását: HTTP szervert, NoSQL adatbáziskezelőt, stb.

2. fejezet

Elméleti alapok

2.1. Erlang

2.2. Open Telecom Platform

2.3. Funkcionális programozás

2.4. Aktor modell

2.5. distributed systems, cap theorem

3. fejezet

Gyakorlati probléma tágabb kontextusának bemutatása

3.1. bevezető rizsa jön ide + a példaprogram, ami egy sima messaging alkalmazás, chat vagy ilyesmi, esetleg egyszerű map reduce példa

4. fejezet

Erlang application

4.1. Programozás

4.2. Fault tolerance

4.3. Elosztott rendszer – kell a redundancia

5. fejezet

Web

5.1. RESTful webservices

5.2. HTML5 – websockets

6. fejezet

Üzemeltetés, karbantartás

6.1. Naplózás

6.2. Hibakeresés

6.3. Hibajavítás, verzióléptetés

7. fejezet

Alkalmazási lehetőségek, kitekintés

8. fejezet

Összegzés

Listing 1. Map reduce module

```
-module(mapreduce).

start() ->
  spawn(fun() -> init() end).

init() ->
  loop().

loop() ->
  receive
    {From, To, What} ->
      io:format("~p sent to ~p a message ~p~n", [From,To,What]),
      loop();
    _ -> % avoid full msg box
      io:format("Nothing. Finish.")
  end.
```

Irodalomjegyzék

Armstrong, Joe (2003): *Making reliable distributed systems in the presence of software errors*. PhD. thesis, The Royal Institute of Technology Stockholm, Sweden. Web: `http://www.erlang.org/download/armstrong_thesis_2003.pdf`, letöltés dátuma: 2012-04-01

Armstrong, Joe (2007): *Programming Erlang: Software for a Concurrent World*. USA: The Pragmatic Bookshelf.

Cesarini, Francesco – Thomson, Simon (2009): *Erlang programming*. USA: O'Reilly Media.

Logan, Martin – Merritt, Eric – Carlsson, Richard (2011): *Erlang and OTP in Action*. USA: Manning Publications.

Fielding, Roy Thomas (2001): *Architectural Styles and the Design of Network-based Software Architectures*. PhD. thesis, University of California, Irvine. Web: `http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf`, letöltés dátuma: 2012-04- 01.

Erlang documentation... (2011) Web: `http://www.erlang.org`

Akamai felmérés (2006): *Retail web site performance: Consumer Reaction to a Poor Online Shopping Experience* `http://www.akamai.com/dl/reports/Site_Abandonment_Final_Report.pdf`