



**Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar**

Önálló laboratórium beszámoló

Név: Recskó Ádám	Neptun kód: WPDJG7	Képzés: MI MSC
Dolgozat címe: NoSQL adatbázisok: Gráf adatbázisok bioinformatikai alkalmazása		
Konzulens(ek) neve: Ligeti Balázs		

A hallgató a kitűzött feladatot megfelelő színvonalon és a kiírásnak megfelelően teljesítette.

Az írásbeli beszámoló javasolt érdemjegye (számmal és betűvel): „

Budapest, 2014. május 12.

Konzulens aláírása

NoSQL Adatbázisok: Gráf adatbázisok bioinformatikai alkalmazása

Tartalom

1. A feladat ismertetése.....	3
2. Fehérje interakciós hálózat.....	4
2.1.1 Gráfok és gráfelmélet alapfogalmai.....	4
2.2. Fehérje-fehérje hálózatok.....	4
2.2.1. Fehérje interakciós hálózatok reprezentációja.....	4
2.2.2. Interakciós gráfok pontjai.....	5
2.2.3. Interakciós gráfok élei.....	5
3. STRING: functional protein association networks.....	6
3.1. A hálózat lehetséges tárolási formái.....	6
3.1.1. MySQL.....	7
3.1.2. Neo4J.....	8
4. Benchmark tesztek.....	8
4.1. A környezet kialakítása.....	8
5. A teszt program.....	10
5.1. Az adatbázis feltöltése.....	10
5.2. A tesztek.....	11
5.3. Mért eredményeim.....	13
5.4. Konklúzió.....	15
6. Összefoglaló.....	16
7. Irodalomjegyzék.....	16

1. A feladat ismertetése

A beszámoló célja, hogy fényt derítsen a manapság egyre nagyobb térre nyitó gráf adatbázisok alkalmazhatóságára a bioinformatikában.

A fehérjék egymásra gyakorolt kölcsönhatásai nagy segítséget adnak az egyes fehérjék funkciójának megértéséhez. Ezen kapcsolatok leírására a matematikából ismert gráfok jó eszközt biztosítanak számunkra. A fehérje kapcsolatokat fehérje hálózatok formájában reprezentáljuk, melyek hatalmas méretre nőhetnek. Az informatika nyújt segítséget ezen adathalmazok hatékony feldolgozására. Éveken át a relációs adatbázis kezelők látták el ezt a feladatot, de a NoSQL megjelenésével érdekes alternatívát kaphatunk. A következőkben ezt a lehetőséget próbálom feltárni és leírni saját tapasztalataimat, melyek a munka során jöttek létre.

A rendszerek összehasonlításának alapját nálam a sebességük képezi, mivel úgy gondolom a gyorsaság a legfontosabb szempont, amelynek meg kell felelniük. A benchmarkokat ezen irányelv alapján állítottam össze. A tesztek ugyan valódi bioinformatikai adatokkal dolgoznak, nem oldanak meg valódi problémákat, feladatuk a rendszerek összehasonlítására irányul.

A benchmark tesztekhez a Neo4J-re esett a választás, mivel jelenleg ez a legelterjedtebb a gráf adatbázisok között. Versenytársának a MySQL-t választottam, a relációs adatbázis-kezelők legnépszerűbbikét. Mindkét rendszernek létezik ingyenesen letölthető verziója, a következőkben ezeket használom tesztelési céllal.

Az adatbázisokat a string-db.org-ról letöltött *9606.protein.actions.detailed.v9.1.txt.gz* fájl szolgáltatja. Ennek felépítéséről írok a dokumentum további részében.

Mivel javarészt webbel foglalkozok, ezért a benchmark keretrendszert is ennek fényében állítottam össze. Szébb formába szerettem volna önteni a feladat megoldását, mint pár sor szöveg egy terminál ablakán, így készítettem egy webapplikációt, melyben előre

összeállított méréseket futtat le a rendszer, és grafikonos formában megjeleníti az eredményeket.

Szerettem volna a feladat keretein belül megismerkedni egy számomra új programozási nyelvvel is, így a webapplikációt scala-ban készítettem el, a typesafe által létrehozott play! keretrendszerben. Nem volt a feladat része, de érdekesnek találom a scala megközelítését, a funkcionális programozást ezért választottam eszközként a feladat megoldásához.

2. Fehérje interakciós hálózat

Ebben a fejezetben rövid áttekintést adok a fehérje interakciós hálózatokról megszerzett információimról, és a hálózatokat reprezentáló gráfok tulajdonságairól, ám ezeket megelőzően szót ejtek magáról a gráfokról.

2.1.1 Gráfok és gráfelmélet alapfogalmai

A gráfelmélet alapfogalma a gráf. Egy gráf csúcsokból és élekből áll. Az élek két csúcsot kötnek össze. Lehet olyan csúcs, amit nem köt össze él másik csúcsponttal, ezen csúcsok az izolált pontok.

A gráfok jelölése a $G(V, E)$. Ebből a V a pontthalmaz, az gráf pontjainak halmaza, E pedig a V elemeiből képzett párok halmaza, azaz, az élek halmaza. („Vertex”, „Edge”) [2]

2.2. Fehérje-fehérje hálózatok

A fehérjék fizikai összekapcsolódásával keletkező hálózatokat fehérje-fehérje kölcsönhatási hálózatoknak, avagy interaktómoknak nevezzük. Ez az egyik leggyakrabban vizsgált hálózat típus. Az interaktómok moduljai (csoportjai) a fehérjék komplexeinek felelnek meg. A legtöbb modulnak igen jól definiált funkciója van a sejt életében. [3]

2.2.1. Fehérje interakciós hálózatok reprezentációja

Az alábbi módokon szervezhetjük gráfokba a fehérjéket, későbbi munkavégzés céljából. [2]

2.2.2. Interakciós gráfok pontjai

- Adott faj adott fehérjéi (pl. UniProt azonosítóval címkézhető)
- Adott gén által kódolt összes fehérje gyűjtőneve (génnévvel címkézhető)
- Adott funkciójú fehérjék gyűjtőneve (EC számmal címkézhető)
- Valamilyen adott reakció szempontjából összetartozó fehérjék halmaza

2.2.3. Interakciós gráfok élei

Élt akkor húzunk két pont közé, ha azok valamilyen módon interakcióba léphetnek egymással. Ezeket tároljuk a pontok mellett.

Több fajta adatbázis építhető ezen adatokból.

Ezeknek típusai:

- Metabolikus interakciókat tartalmazó adatbázisok/hálózatok.
- Egyéb interakciókat tartalmazó hálózatok

Metabolikus hálózatok

A metabolikus hálózatok elemeit a sejtben előforduló szerves molekulák, a metabolitok alkotják.

A metabolikus hálózatok vizsgálatával a sejt életciklusának pontos kémiai jellemzői meghatározhatóak.

Egyéb interakciókat tartalmazó hálózatok

Esetemben fizikai interakciós hálózatról van szó, a string-db-ről ebbe a csoportba sorolható adatokat lehet letölteni. [4]

3. STRING: functional protein association networks

A benchmark tesztjeimhez a STRING oldaláról letölthető ingyenes adatbázisokat használtam. Ezek a <http://string-db.org/> oldalán a download menüre kattintva érhetőek el.

Az adatbázisaimat a *protein.links.detailed.v9.1.txt.gz* fájlt feldolgozva töltöttem fel. Mivel ez a fájl, amennyiben az összes adatot tartalmazza, eszközeimmel kezelhetetlen lett volna ezért csak az emberhez tartozó 9606-os taxon_id-jű fehérjék kapcsolatait töltöttem fel. Ez kitömörítve, szöveges fájlban 4850628 sort tartalmaz. Ezzel a későbbiekben még nehézségeim támadtak, de ezt kifejtem a konkrét program ismeretesénél. [5]

A gz fájlt kitömörítve kapjuk a fehérje-fehérje interakcióinkat.

```
protein1 protein2 neighborhood fusion cooccurrence coexpression experimental database textmining
combined_score
9606.ENSP00000000233 9606.ENSP00000020673 0 0 0 0 0 176 176
9606.ENSP00000000233 9606.ENSP00000054666 0 0 0 0 88 0 309 327
9606.ENSP00000000233 9606.ENSP000000158762 0 0 0 0 0 718 718
9606.ENSP00000000233 9606.ENSP000000203407 0 0 0 272 0 0 0 272
9606.ENSP00000000233 9606.ENSP000000203630 0 0 0 241 0 0 0 241
9606.ENSP00000000233 9606.ENSP000000215071 0 0 0 130 0 0 105 170
9606.ENSP00000000233 9606.ENSP000000215115 0 0 0 196 0 0 0 196
```

Az első sor, a fájl fejléce tartalmazza a fájl további adatainak megnevezését. Minden ezt követő sor két fehérje közötti interakció pontszámait írja le. Az első két elem felfogható a gráf élének, hiszen két pontot határoz meg, melyek között meghatározza az interakció mértékét. Tesztjeimben a combined_scoreval dogoztam, ez az alábbi módon számítható ki a meglévő adatok alapján.

$$1 - (1 - \text{nscore}) * (1 - \text{fscore}) * (1 - \text{pscore}) * (1 - \text{cscore}) * (1 - \text{escore}) * (1 - \text{tscore})$$

neighborhood fusion cooccurrence coexpression experimental textmining

[5]

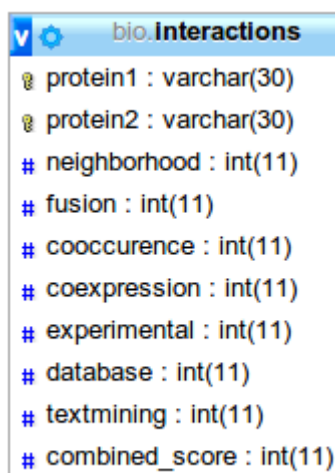
3.1. A hálózat lehetséges tárolási formái

Ebben a fejezetben ismeretem az általam felhasznált adatbázis kezelőkben milyen formában történt az adatok tárolása. Első részében a relációs adatbázis (MySQL) másodikban a NoSQL (Neo4J) tárolás módszerét írom le.

3.1.1. MySQL

Először is létre kell hoznunk egy adatbázist, majd ebben az adatbázisban a fentebb ismertetett szövegfájl, *protein.links.detailed.v9.1.txt* fejlécét felhasználva kaphatjuk meg az interakciós táblánkat.

A táblában letároltam a gráf pontjait, így az interactions táblám a gráf éleit tartalmazza.



bio.Interactions	
protein1	: varchar(30)
protein2	: varchar(30)
neighborhood	: int(11)
fusion	: int(11)
cooccurence	: int(11)
coexpression	: int(11)
experimental	: int(11)
database	: int(11)
textmining	: int(11)
combined_score	: int(11)

Első két oszlopa a *protein1* *protein2*, az interakciós hálózat két pontja. A többi oszlop a vele megegyező nevű adatot tartalmazza, a végén látható a *combined score*, mely szintén a STRING adatbázisból lett betöltve.

A tábla első két oszlopa: *protein1* *protein2* együtt a séma elsődleges kulcsai. Ez azt jelenti, hogy a két kulcs indexelve is van az adatbázisban, és a két kulcs együtt unique, azaz nem lehet két egyező sor a táblában.

A relációs modell soronként tárolja az interakciókat, ahogyan az a txt fájlban is van, de itt már egyszerűbben végezhetők műveletek velük, ám az adatbázis kezelő nem tudja, hogy itt gráfokról van szó. Ha gráf műveleteket szeretnénk vele végezni, azt nekünk kell SQL-ben megírni, és úgy kezelni az adathalmazt, mint egy gráfot.

A műveletek végzéséhez az SQL nyelv nyújt segítséget. Ezzel képesek vagyunk lekérdezéseket futtatni az adatbázisunkon.

3.1.2. Neo4J

A Neo4J előnye, hogy ténylegesen gráfokhoz készült. Adatbázisában explicit módon tároljuk a gráfot, minden csomóponthoz közvetlenül tartozik mutatója a szomszédos csomópontjaira, így a gráf bejárásához nem szükséges az adatok gráffá alakítása, táblák illesztése, indexek használata. [7]

Az adatbázis két alkotóelemből épül fel. Első a csomópontok halmaza. Ezek a proteinek, melyek a *protein.links.detailed.v9.1.txt* fájlban találhatóak. Amint ezek eltárolásra kerültek, utána jöhet a kapcsolatok, az élek felépítése. Az éleknek is lehet típusa, és tartalmazhatnak egyéb adatokat. Jelen esetben csupán nevük van, ami arra utal, hogy itt egy interakcióról van szó.

Ez a felépítés teszi képessé a Neo4J-t gráfokkal végzett műveletek hatékony kivitelezésére. Itt már nem tároltam le az összes adatot, csupán a proteinek nevét, és a *combined_score*-t.

4. Benchmark tesztek

A fejezetben leírom, hogyan alakítottam ki a tesztkörnyezetet, milyen rendszereket használok mely beállításokkal. Írok az adatbázisok beszerzéséről, telepítéséről, valamint a benchmark keretprogram összeállításáról, tulajdonságairól.

4.1. A környezet kialakítása

Operációs rendszernek a Linuxot választottam, ezt használom a mindennapok során, így ez tűnt kézenfekvőnek.

MySQL

Az 5.5.37 MySQL-t használok, ubuntu rendszeremen a hivatalos repositoryból telepítettem. A repositoryban ez most a legfrissebb verzió. Telepítés után létrehoztam egy adatbázist bio néven, majd ebben az adatbázisban egy táblát, InnoDB tárolómotorral interactions néven.

```
CREATE TABLE IF NOT EXISTS `interactions` (  
  `protein1` varchar(30) NOT NULL,  
  `protein2` varchar(30) NOT NULL,  
  `neighborhood` int(11) DEFAULT NULL,  
  `fusion` int(11) DEFAULT NULL,  
  `cooccurrence` int(11) DEFAULT NULL,  
  `coexpression` int(11) DEFAULT NULL,  
  `experimental` int(11) DEFAULT NULL,  
  `database` int(11) DEFAULT NULL,  
  `textmining` int(11) DEFAULT NULL,  
  `combined_score` int(11) DEFAULT NULL COMMENT 'protein interaction ',  
  PRIMARY KEY (`protein1`,`protein2`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Ezek után feltelepítettem egy *phpmyadmin*t, hogy ne csak a terminálon érjem el a leendő adataimat.

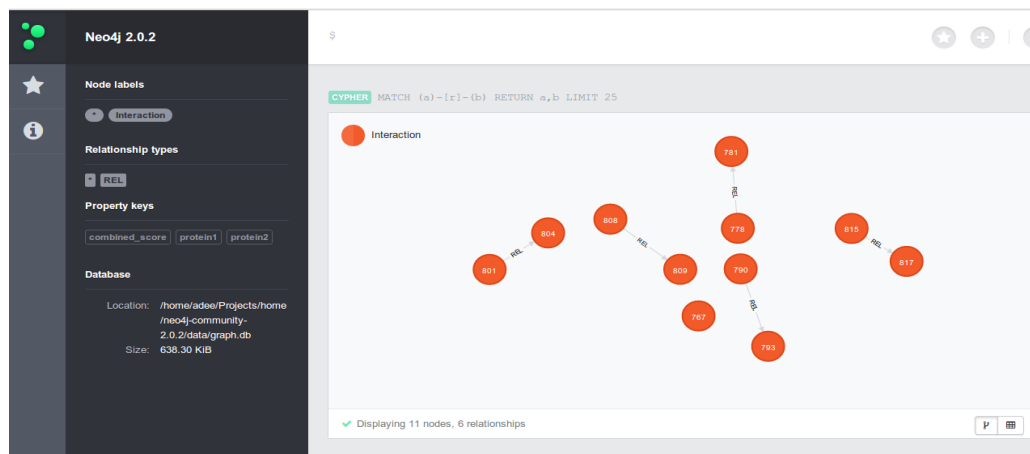
Neo4J

A Neo4J letölthető a <http://www.neo4j.org/> oldalról community edition verzióban. Ebből a 2.0.2-mas verziószámút töltöttem le, ez jelenleg a legfrissebb stabil kiadás.

Nem szükséges telepíteni, a letöltött tar.gz fájl kicsomagolása után a rendszer indítható, azonban szükség van java futtatókörnyezetre a működéshez.

A kicsomagolt könyvtárba lépve, a bin könyvtárban, a neo4j scripttel indítható: *./neo4j start*
Ekkor létrehozza a számára szükséges fájlokat, és elindul egy webes adminisztrációs felület is, mely a <http://localhost:7474/browser/> alatt érhető el.

Itt megtekinthetők az adataink. Az interakciók feltöltése után a következő állapot látható:



Baloldalon a felöltött Node típusok láthatók, alatta pedig a Node-ok között létrehozott kapcsolatok típusai.

A képernyő jobb részén egy lekérdezés eredménye látható.

A Neo4j és a MySQL esetében is a gyártó alapbeállításai használok. Mindkét esetben lehetőségünk van ezeket testre szabni, sőt MySQL esetében rengeteg beállítás áll a rendelkezésünkre.

5. A teszt program

A fejezetben a tesztprogram felépítését részletezem, bemutatom, hogy milyen lekérdezéseket támogat, hogyan helyezhető üzembe, valamint írok a tapasztalt nehézségeimről az adatbázis feltöltésével kapcsolatban.

A teszt programom alapvetően egy web applikáció. A play! keretrendszerre épül, scala nyelven íródott. Úgy állítottam össze, hogy egyszerűen fel lehessen venni új teszteket, amik futásának leméri az idejét.

Tartalmaz egy segédosztályt, amivel fel lehet olvasni a string-db-ről beszerzett adatbázist és képes azt továbbítani a MySQL és a Neo4J felé.

5.1. Az adatbázis feltöltése

A megírt alkalmazással kezdtem el az adatok feltöltését, először a MySQL, majd a Neo4J adatbázisokba.

A MySQL gond nélkül betöltődött, az én gépemén 4 óra alatt sikerült a teljes 9606 `taxon_id`-jű adatokkal feltöltenem az `interactions` nevű táblát.

Ezzel szemben a Neo4J-nek gondjai akadtak ezzel az adatmennyiséggel. Első körben több mint 2 napig ment a feltöltés mire sikerült eredményt elérnem, de ekkor még a `node`-ok közötti kapcsolatok nem voltak létrehozva.

A kapcsolatok létrehozása ezzel az elemszámmal végül nem sikerült. Elindítás után, egy nappal *Java.Lang.OutOfMemoryError: Java Heap Space*-vel vége szakadt a folyamatnak. Az újbóli próbálkozás szintén erre a végeredményre vezetett.

Ekkor döntöttem úgy, hogy csökkentem az elemszámot, és egészen 100000 sorig voltam kénytelen lemenni, a majd 5 milliós adatbázisból.

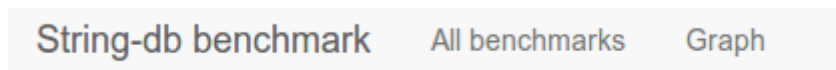
Megjegyezném, hogy nem áll rendelkezésemre nagyobb erőforrás a saját laptopomnál, ami 8 gigabájt RAM-ot tartalmaz. Ha bővíthetem volna az elérhető memória mennyiséget, illetve magasabbra állítom az `heapsize`-t, akkor talán sikerült volna létrehoznom a kapcsolatok is a Neo4J adatbázisomban.

5.2. A tesztek

A benchmark alkalmazás a `play run`, parancssal indítható. Ezután az automatikusan csatlakozik mind a MySQL-hez, mind pedig az elérhető Neo4J-hez.

Az alkalmazás indítása után a <http://localhost:9000/> címen érhető el, amennyiben saját rendszerünkön indítjuk.

A felső menüsávban navigálhatunk az elérhető funkciók között.



Az alkalmazás nevére kattintva a kezdőképernyője jön be, amiben írok a feladatról és megjelenítek néhány adatot, amik érdekesek lehetnek a használnak.

String-db benchmark

PÁZMÁNY PÉTER CATHOLIC UNIVERSITY - Faculty of Information Technology and Bionics

Self Project: Benchmark the MySQL and the Neo4J queries.

The database loaded from <http://string-db.org/protein.links.detailed.v9.1.txt.gz> - Homo sapiens

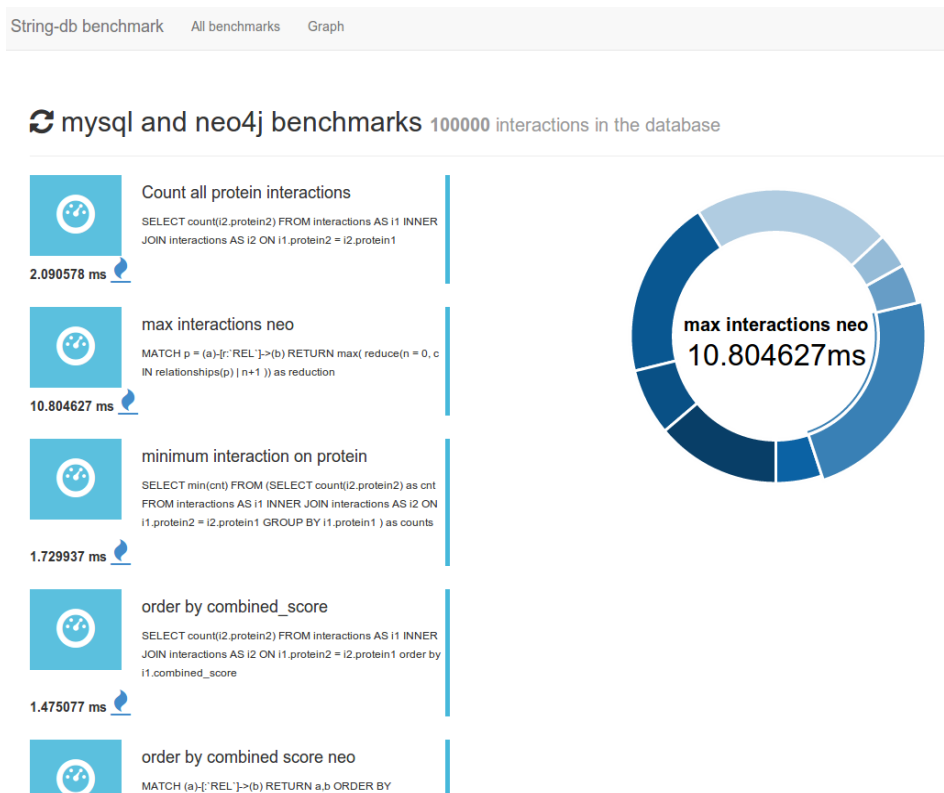
Backend is MySQL

Server version: 5.5.35-0ubuntu0.12.04.2
and Play Framework 2.2

My name is Adam Recsko

Source code located at: <https://github.com/AdeeLNx/stringdb-benchmark>

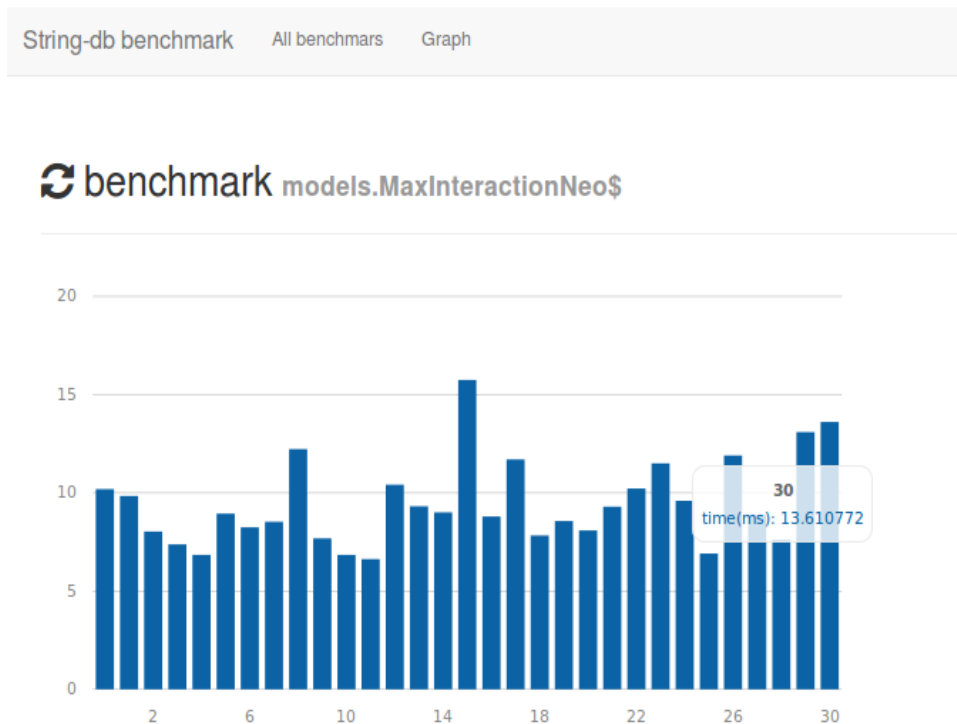
Az all benchmarks fülre kattintva, a rendszer lefuttatja az összes tárolt lekérdezést. Jelenleg 4-4 lekérdezés áll rendelkezésre mind a MySQL mind Neo4J oldalon. Amint végzett az összes tesztel, megjeleníti az oldalon a tesztek eredményét.



A kördiagramon látható, hogy a teljes időből hogyan oszlott meg az egyes lekérdezésekre fordított idő. A kördiagram mellett szerepel a futtatott tesztek eredménye egyenként, valamint annak rövid leírása, ami jelen esetben maga a lekérdezés, SQL és Cypher. Látható a lekérdezések időtartama is.

A cím mellett látható refresh gomb hatására a rendszer újra futtatja a lekérdezéseket, és újraépíti a diagramot.

Minden lekérdezés eredményben látható egy kék tűz ikon. Kattintásra a rendszer alapbeállításként az adott lekérdezést 15x egymás utána lefuttatja. Ez állítható, megnövelhető nagyobb számra is. Kattintás után az alábbi ablak jelenik meg, attól függően, hogy mely



lekérdezésre esett a választás.

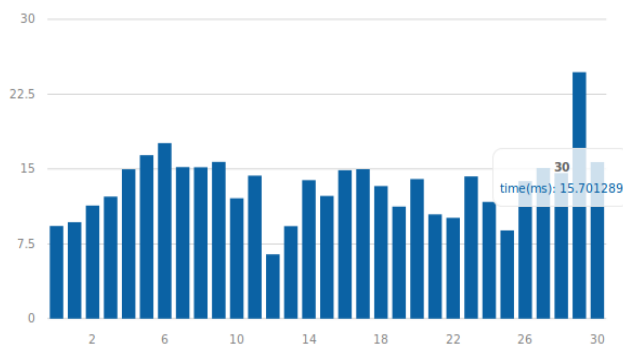
A grafikon x tengelye a lekérdezés indexe, itt 1-től 30-ig, azaz 30 lekérdezést hajtott végre a rendszer. Y tengelyen pedig az idő ezredmásodpercben, ameddig a lekérdezés tartott.

Itt felhívnom a figyelmet arra, hogy bár webes felületet kapott a tesztkörnyezet, nem a HTTP kérések idejét méri. A szerver oldalon futó lekérdezés időtartamát adja vissza az alkalmazás.

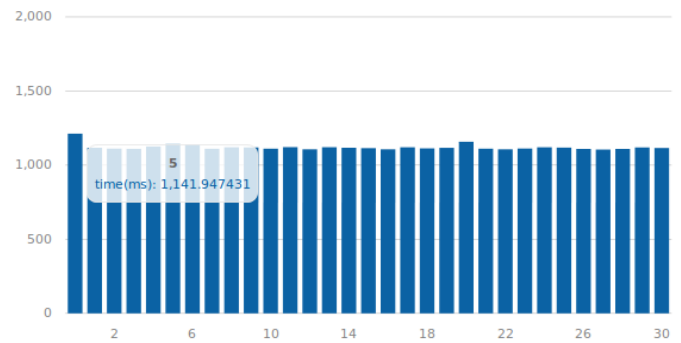
5.3. Mért eredményeim

Legnagyobb élszámmal rendelkező node.

benchmark models.MaxInteractionNeo\$



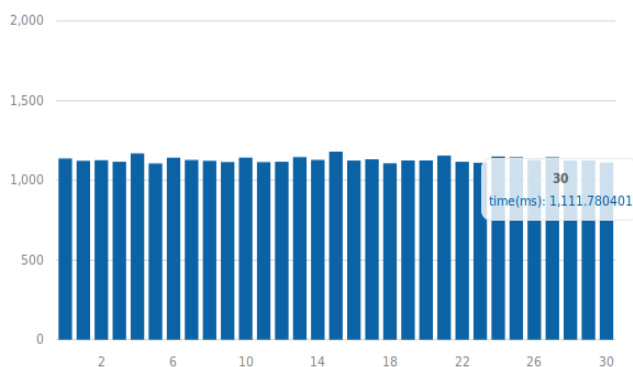
benchmark models.MaxInteractionMysql\$



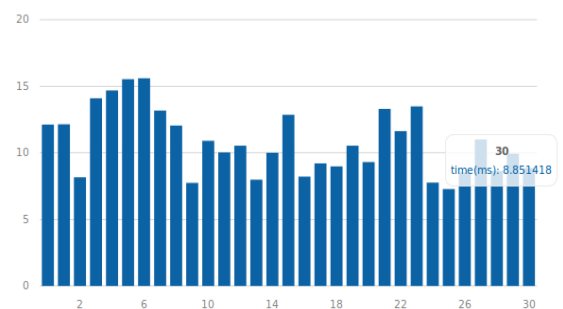
Ami a grafikonokról leolvasható, az az, hogy esetemben a mysql lekérdezés, körül belül 1 másodpercet vesz igénybe, míg a neo4J átlagban 15 ms alatt végez. Ebben az esetben az 66X-os gyorsulást jelent.

Legkisebb él számmal rendelkező node.

benchmark models.MinInteractionMysql\$



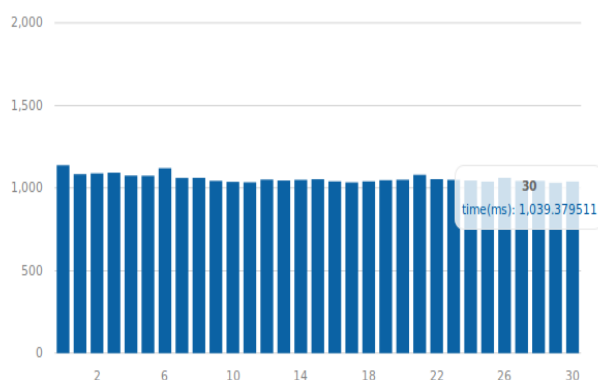
benchmark models.MinInteractionNeo\$



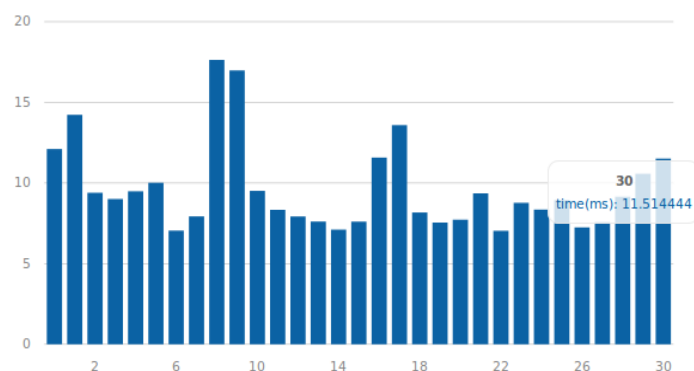
A legkisebb elemszámmal rendelkező elemnél is hasonló eredményt kapunk, itt átlagban 10ms a neo4J ideje, míg a mysql itt is 1 másodperc felett teljesít.

Az összes kapcsolattal rendelkező elem megszámlálása

benchmark models.CountProteinMysql\$



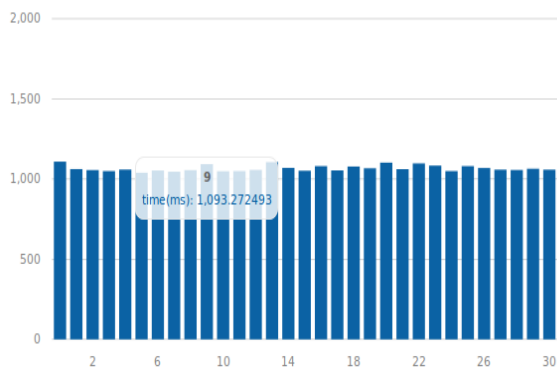
benchmark models.CountProteinNeo\$



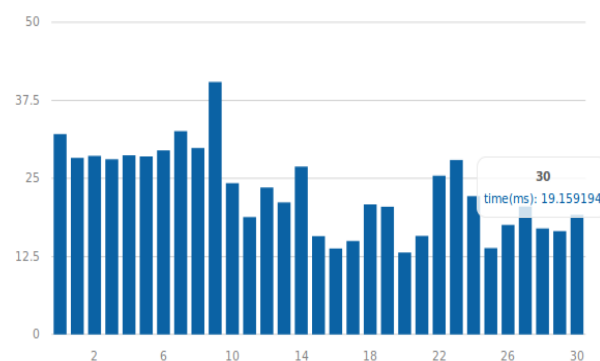
Bár a neo4j-nél tapasztalhatók kiugrások a teljesítményben, mégis 20 ms alatt marad egy-egy lekérdezés ideje, a mysql itt is 1 másodperc felett hozza a lekérdezéseket.

Rendezés a combined score alapján, a kapcsolatban álló elemek között.

benchmark models.OrderByCombinedScoreMysql\$



benchmark models.OrderByCombinedScoreNeo\$



Rendezésnél a neo4j is lassabban teljesít csaknem duplája az eddigi idejének, végig 20ms felett marad. A mysql hasonló eredményt produkál, az előző lekérdezésekhez, átlagban 1 másodperc felett marad.

5.4. Konklúzió

A mért lekérdezésekben a neo4j sokkal gyorsabb a mysql-tól. Az én gépemén 50X-es sebesség növekedéstől, esetenként egészen 100X-os sebesség növekedést is lehet tapasztalni.

6. Összefoglaló

Sikeresen létrehoztam egy kisebb keretrendszert, amivel összehasonlíthatók különféle adatbázis rendszerek. Amire használtam, hogy a feladat céljaként kitűzött MySQL-t és Neo4J-t összehasonlítsam olyan módon, hogy bioinformatikai adatbázist építsek mindkettő felhasználásával, és lekérdezéseket futtatok.

Eredményeimben a NoSQL győzedelmeskedett a relációs adattárolás felett, de csak a lekérdezéseket illetően. A MySQL-t használva sokkal gyorsabban készült el az adatbázis, és sokkal kisebb az erőforrás igénye. A Neo használatával a felhasznált RAM mennyiség a tárolt adatok méretével folyamatosan nő, és egy bizonyos szint után, kénytelenek vagyunk a beállításaiába is belenyúlni, ha sikert akarunk elérni. Az adatok feltöltése nálam megdöbbentően lassan ment a Neo4J esetében. MySQL használatával, ez órák alatt kivitelezhető, a használt számítógép függvényében, míg az én rendszeremen a Neo4J napok alatt sem végzett ilyen adatmennyiséggel.

A Neo4J fejlesztői szemmel előnyös tulajdonságokat hordoz, a beépített webes adminisztrációs felület, és az automatikusan elinduló REST service. Véleményem szerint modern megoldások, és jól használhatók a munka során. Elképzelhető, hogy egy testre szabott konfigurációval nem támadtak volna nehézségeim a feltöltést illetően sem.

7. Irodalomjegyzék

[1]

<http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full>

[2] Chatr-Aryamontri A(1), Breitkreutz BJ, Heinicke S, Boucher L, Winter A, Stark C, Nixon J, Ramage L, Kolas N, O'Donnell L, Reguly T, Breitkreutz A, Sellam A, Chen D, Chang C, Rust J, Livstone M, Oughtred R, Dolinski K, Tyers M. The BioGRID interaction database: 2013 update.

[3] Björn H. Junker, Falk Schreiber Analysis of Biological Networks 2008

[4] Ian Robinson, Jim Webber, Emil Eifrem Graph Databases

[5] D. Szklarczyk, A. Franceschini, M. Kuhn, M. Simonovic, A. Roth, P. Minguéz, T. Doerks, M. Stark, J. Muller, P. Bork, L. J. Jensen, and C. von Mering, “The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored,” *Nucleic Acids Research*, vol. 39 (Database issue), p. 561–8, 2010.

[6] Franceschini A, et al. STRING v9.1: protein-protein interaction networks, with increased coverage and integration. *Nucleic Acids Res.* 2013;41:D808-D815.

[7] Webber J, et al. *Graph Databases*. Sebastopol: O’Reilly Media; 2013.