Treball de Fi de Grau
GRAU D'ENGINYERIA INFORMÀTICA

Anàlisi i cerca de subgrafs més densos mitjançant Hadoop MapReduce

Alumne: Oriol Ventura Jariod
Tutor: Elitza Maneva

Universitat de Barcelona
Facultat de Matemàtiques

Índex

Índex de figures

Prefaci

En aquest projecte de final de grau tractarem el tema del "BigData" i de les seves aplicacions. Concretament i amb el tema que ens dona per títol, veurem com resoldre el problema de trobar subgrafs densos dins d'un graf gran, amb un algorisme adaptat per a un framework especial anomenat MapReduce, que conjuntament amb Hadoop són eines que ens permeten treballar amb problemes de BigData de manera distribuïda més fàcil.

També veurem l'anàlisi de resultats dels diferents entorns de proves amb els diferents grafs estudiats i finalment veurem en els annexos un estudi dels costos d'ús dels serveis d'Amazon Cloud i un petit manual de funcionament.

1. Introducció a "BigData"

El terme BigData [PI] fa referència al conjunt d'eines emprades pel tractament massiu de grans quantitats de dades, que per la seva grandària fan molt difícil que siguin tractades de manera tradicional amb les eines de bases de dades relacionals que disposem.

Per entendre millor el problema que ens representa el nou paradigma de gestió de dades, podem donar un cop d'ull a les dades oficials de la plataforma social Facebook que va fer públiques en una conferència l'any 2012:

- 2.500 milions de continguts compartits per dia.
- 300 milions de fotografies pujades per dia.
- Més de 500 TB de noves dades introduïdes a base de dades cada dia.

També podem veure com altres plataformes com Twitter generen més de 400 milions de missatges per dia i amb una mida mitjana de 150-200bytes per missatge ens genera un total de més de 80Gb. A més a més de tot això, en l'actualitat generem moltes dades en altres àmbits, com per exemple: d'indexació de cercadors d'internet, registre de trucades i missatges telefònics i mails, dades d'astronomia, dades meteorològiques i climàtiques, dades mèdiques i biològiques, projectes tecnològics a gran escala com el CERN, etc....

Tot aquests són indicadors clars de la necessitat d'obtenir mecanismes per tractar aquesta quantitat d'informació de manera ràpida, fiable i eficient.

2. Frameworks relacionats

2.1. MapReduce

Davant la necessitat de processar les ingents quantitats de dades descrites anteriorment, la empresa Google va desenvolupar MapReduce. MapReduce és un framework que aborda el problema paral · lelitzant el càlcul i distribuint grans quantitats de dades sobre clústers d'ordinadors. Aquest model és el que fa servir des de fa anys Google en tots els seus projectes com per exemple pageRank entre altres.

El nom MapReduce s'obté d'ajuntar el nom de les dues funcions principals en que consisteix l'algorisme intern del framework. A continuació mostrem l'esquema general d'una seqüència d'execució de MapReduce [DG]:

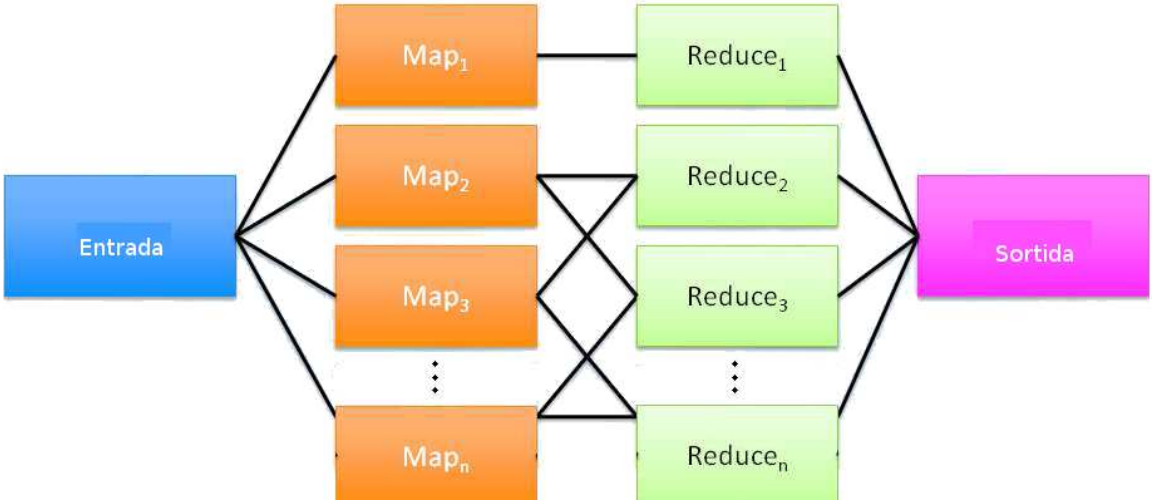


Figure 2.1.: Esquema general MapReduce

Primer de tot, i com a fet diferenciador del framework destaquem que els fluxos de dades dels algorismes MapReduce es basen en parelles de valors *<clau,valor>*. Tal com mostra l'esquema anterior els fitxers d'entrada de dades es parteixen en blocs i cada bloc és processat per un mapper. El sistema ens dóna l'opció d'implementar els nostres mappers amb el codi que considerem adient per tractar els blocs de dades de la mateixa manera. Així doncs en la funció de mapper associem valors llegits del bloc de dades d'entrada a valors que considerem oportuns, això ho veurem en un

exemple senzill més endevant. Un cop hem acabat amb les dades d'entrada de cada mapper el sistema junta totes les parelles de *<clau,valor>* produides pels mappers i les agrupa per la *clau.* Per tant, tenim que s'executarà una funció de reduce per cada clau produida anteriorment amb la llista de valors associats. Les parelles de valors generades en els reducers seran la sortida que produirà l'algorisme.

Per veure-ho més clar descriurem un exemple bàsic de funcionament d'una tasca que compta la freqüència de repetició de paraules en un text [Eng]. Ho farem mostrant els algorismes dels mappers i reducers gràficament i textualment. A continuació tenim els algorismes:

```
1. funció map(text):
2.     per_cada paraula en text fer:
3.      genera <paraula, 1>
4. fi
```

```
1. funció reduce(clau, valors):
2.     enter i ← 0
3.     per_cada valor en valors fer:
4.          i ← i+1
3.     genera <clau, i>
4. fi
```

Primer de tot tenim un fitxer de text amb paraules tant gran com es desitgi. Per cada bloc de dades s'executarà un mapper. Aquest rebra les dades i a continuació per cada paraula del troç de text rebut genrearà la parella *<paraula,1>*. Després el reducer només ha de comptar el nombre d'uns que ha rebut juntament amb la clau (paraula llegida) i generar per la sortida la parella de valors *<clau, nombre calculat>*. Veiem-ho gràficament:

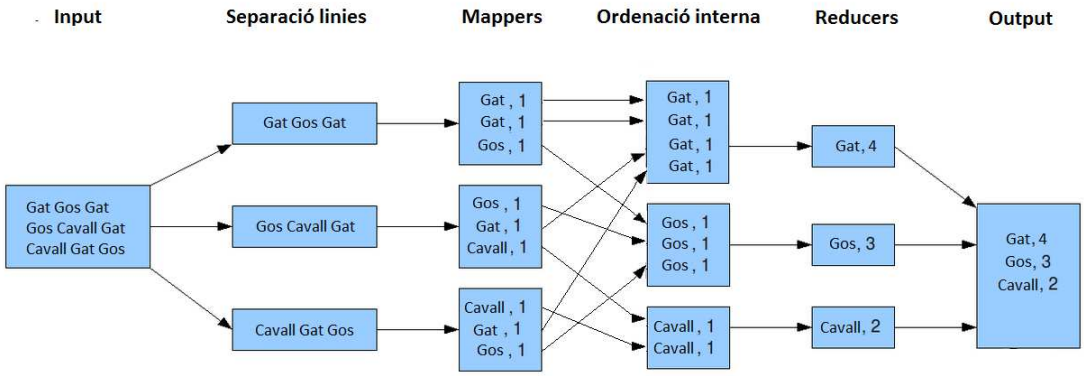


Figure 2.2.: Sortida exemple bàsic MapReudce

En aquest punt cal mencionar que tal com veurem en l'implementació del nostre algorisme, un cop finalitzada una tasca la podem tornar a executar redirigint la sortida d'aquesta a l'entrada d'una nova iterant el nombre de cops que es desitgi.

Després de veure aquest exemple sencill ens endinsarem una mica més en l'aspecte tècnic de funcionament i veurem l"arquitectura general del framework. Primer comencem per descriure els diferents nodes que formen el clúster en el qual s'executaran les tasques. Tenim 2 tipus de nodes: "master" i "slaves". El node master és únic i s'encarrega de distribuir els blocs de dades a la resta de nodes. Els nodes "slaves" que poden ser varis, executen els codis dels mappers i els reducers. El comportament detallat de l'execució d'una tasca es el següent. Primer un cop s'executa el programa per part de l'usuari aquest crea instàncies de nodes "màster" i "slaves". El node "master" com hem descrit abans distribueix blocs de dades d'entrada als diferents mappers que s'executaran en els nodes "slaves". Un cop acabats desen la sortida en una ubicació temporal. En paral · lel quan comencen a executar-se els Reducers en els nodes "slaves" aquests processen l'entrada i desen els seus resultats en els fitxers de sortida. Un cop han acabat tots els processos dels "slaves" es retorna el control al programa principal i s'acaba l'execució del programa.

Per finalitzar cal destacar que el mecanisme de MapReduce és basculant quan un dels nodes queda fora de servei. Això vol dir que cada cert temps el node "master" comprova la disponibilitat dels nodes "slaves" i si un node "slave" no respon en un determinat temps qualsevol bloc de dades tractat en el procés de map torna d'immediat a la cua d'espera tornant a estar disponibles per la resta de nodes. En canvi els blocs reduce no fa falta tornar-los a repetir ja que la seva sortida ja és la global del procés.

2.2. Hadoop

Hadoop [Dum][PI][Had] és una implementació de MapReduce de codi obert escrita completament en Java originalment desenvolupada per Yahoo en el 2006 i posteriorment adquirida per Apache. Aquesta implementació és la que es farà servir durant l'execució de codi d'aquest projecte, tant en clúster, local com en cloud.

Una de les principals característiques d'aquest framework és la incorporació del sistema de fitxers distribuït HDFS. Aquest sistema de fitxers fa servir automàticament tots els nodes per emmagatzemar dades i els enllaça com si fossin un sistema de fitxers més gran. A més a més, com que els nodes poden quedar fora de servei, replica les dades entre diferents nodes per no perdre informació.

Per facilitar el seguiment de les tasques incorpora uns serveis web anomenats jobtracker i namenode per fer seguiment de les tasques en execució i la navegació per el sistema de fitxers distribuït respectivament.

Altres característiques a destacar de Hadoop [IBM]són les següents:

- Escalabilitat: Es poden afegir o treure nodes del sistema sense haver de retocar cap configuració ni modificar el codi de les tasques.
- Cost reduït: Hadoop porta el model de computació paral · lela a servidors de baix cost, el que representa una disminució del cost de emmagatzemament per Tb, i s'adapta a la majoria de necessitats.
- Flexibilitat: Hadoop no té un esquema en concret i pot treballar amb una gran quantitat de tipus de dades des de fonts diverses. A més pot fer unions de diferents fonts i combinar-les de diferents maneres com veurem més endavant en l'algorisme estudiat.
- A prova d'errades: De la mateixa manera que proposa el model MapReduce Hadoop també balanceja la carrega dels diferents nodes per si algun d'ells queda fora de servei.

3. Algorisme per trobar el subgraf més dens

En aquest apartat abordarem el problema de trobar subgrafs densos amb Hadoop MapReduce. Trobar subgrafs densos és un problema important de l'anàlisi de dades que comprèn diversos àmbits d'estudi que abarquen des d'aplicacions de mineria de dades a detecció de spam o aplicacions científiques/mèdiques.

Concretament en aquest projecte ens centrarem en la cerca de subgrafs densos amb l'anàlisi de grafs no dirigits on els nodes corresponen majoritàriament a comunitats establertes dins de xarxes socials d'usuaris o objectes i les arestes les relacions que mantenen entre ells.

Aquests problemes es poden resoldre òptimament amb altres algorismes d'anàlisi de grafs clàssics com "Max Flow" o matemàtics com Programació lineal amb relaxació. Aquestes tècniques garanteixen resultats correctes, però no s'adapten bé a grafs molt grans, degut a la seva complexitat computacional i ús de memòria. Per aquests motius, adaptarem un algorisme [BBV] per poder distribuir-lo amb Hadoop i que garanteixi uns resultats propers a l'òptim, que satisfacin les necessitats d'estudi que proposarem.

Aquest algorisme en qüestió té 3 parts diferenciades, una primera que compta nodes i arestes, seguit d'una altra que calcula el grau de cada node i per acabar una que elimina els nodes amb un grau menor d'un llindar determinat. Mostrem l'algorisme a continuació:

Requisits: G=(V,E): *conjunt de nodes i arestes*

1: S, S2 $\leftarrow$ V
2: **mentre** S $\neq \emptyset$ **fer**
3: C(S) $\leftarrow$ { i $\in$ S | grau(i) $\leq$ 2$\times$densitat(S) }
4: S $\leftarrow$ S $\setminus$ C(S)
5: **si** densitat(S) $>$ densitat(S2) **fer**
6: S2 $\leftarrow$ S
7: **fi_si**
8: **fi_mentre**
9: **retorna** S2

Comencem copiant els nodes del graf original en dos conjunts S i S2 *(1)*. Mentre queden nodes en el conjunt S obtenim els nodes a eliminar amb grau inferior al

doble de la densitat del conjunt de nodes, i els esborrem del conjunt S *(3 i 4)*. Si la densitat del conjunt S és més gran que la del conjunt S2 reescrivim S sobre S2 *(5 i 6)*. Aquest procediment es repeteix fins que no queden nodes per tractar en S i finalment la sortida de l'algorisme és S2 amb la informació del graf més dens.

Les tres seccions diferenciades de l'algorisme que acabem de descriure es corresponen a tres tasques de MapReduce per iteració *(bloc mentre)*. A continuació passarem a descriure-les.

La primera tasca simplement compta els nodes i les arestes del conjunt S, a l'hora que calcula el grau de cada node. Això ho aconsegueix de la següent manera:

-Mappers:

1. Separen per cada línia rebuda els nodes origen i destí.
2. Emeten un parell de valors del tipus <node_origen, node_destí> i <node_destí, node_origen>.
3. Emeten un altre parell de valors amb la clau "#n" per comptar nodes de la manera <#n, node_origen> i <#n, node_destí>.
4. Emeten un darrer parell de valors del tipus <#e, 1>, per comptar les arestes.

-Reducers:

1. S'executa un reducer per cada clau emesa.
2. Per la clau "#n" el reducer guarda la llista de valors que va associada en una HashMap per evitar duplicats. *(calcula el nombre de nodes mirant la mida de la taula)*
3. Per la clau "#e" guarda la mida de la llista de valors que va associada *(calcula el nombre d'arestes)*.
4. Fa el mateix que el punt anterior, per cada clau que rep en format enter *(calcula el grau de cada node)*.

Tenint aquestes dades aquesta primera tasca emet per dues sortides diferents, una primera amb el nombre de nodes i el nombre de arestes, i la segona sortida emetent el node clau seguit de la mida de la llista de valors associat que hem calculat prèviament.

La segona tasca és conceptualment una mica més complexa, ja que per a la part dels mapper realitza una unió entre dos fonts de fitxers d'entrada on cada font utilitza un tipus de mapper diferent. Aquesta segona tasca realitza un marcatge per a tots els nodes d'origen candidats a ser eliminats. La feina dels mappers i reducers és la següent:

-Mapper1:

1. Llegeix el fitxer amb l'informació dels graus de cada node calculat anteriorment.

2. Emet un parell de valors del tipus <node, -1> si el grau de cada node de la llista llegida anteriorment és inferior a 2×(densitat del graf actual).

-Mapper2:

1. El segon mapper rep d'entrada el graf original.
2. Emet un parell de valors del tipus <node_origen, node_destí>.

-Reducers:

1. Guarda la llista de valors associats a cada clau en una llista.
2. Si en la llista hi ha un -1 (valor creat al mapper 1) marca per a esborrar el node.
3. Si el node esta marcat, emet per la sortida de nodes a esborrar el node en qüestió.
4. Si no per la sortida de graf intermedi emet la clau amb totes els seus valors, o sigui una emissió per cada node amb tots els seus nodes_destins.

En aquest punt ja tenim 2 llistes una amb els nodes origen a esborrar i l'altra amb el graf intermedi. Ara ja podem acabar de fer la selecció per als nodes de destí i ja tindrem el graf resultant final d'iteració. Descrivim la feina dels mappers i reducers:

-Mapper1:

1. Llegeix el fitxer amb l'informació dels graus de cada node calculat anteriorment.
2. Emet un parell de valors del tipus <node, -1> si el grau de cada node de la llista llegida anteriorment és inferior a 2×(densitat del graf actual).

-Mapper2:

1. El segon mapper rep d'entrada el graf intermedi.
2. Emet un parell de valors del tipus <node_destí, node_origen>.

-Reducers:

1. Guarda la llista de valors associats a cada clau en una llista.
2. Si en la llista hi ha un -1 (valor creat al mapper 1) marca per a esborrar el node.
3. Si el node no està marcat, emet la clau amb totes els seus valors, o sigui una emissió per cada node amb tots els seus nodes_destins.

Amb la sortida produida del reducer anterior ja tenim el graf complet amb els nodes de grau inferior esborrats. Així doncs aquestes tres tasques descrites s'executaran un nombre de vegades determinat, fins que la sortida d'algun dels reducers intermedis sigui un conjunt buit. Un fet important que succeeix en el codi principal entre l'execució de la primera tasca i la segona, és que si la densitat del graf calculat en

la primera tasca és més gran que la que teniem en l'iteració anterior es copiarà el graf *(S)* sobre el graf resultat *(S2)*.

Tal com hem especificat anteriorment aquest algorisme és una aproximació que extreu un subgraf dens com a mínim la meitat de la densitat del subgraf més dens, i matemáticament amb la següent demostració ho veurem més clar:

A mesura que el codi s'executa, la densitat del graf restant és no monotònica. Així doncs, mostrarem que un dels grafs intermedis és una aproximació de com a mínim la meitat de la solució optima.

Primer definim la densitat d'un graf G tal que $G = (V, E)$ on V és la llista de nodes i E són les arestes, tenim que la densitat p és:

$p(S) = \frac{|E(S)|}{|S|}$, $S \subseteq V$

i la desnitat máxima p^* d'aquest mateix graf es:

$p^*(G) = max\{p(S)\}$

Entrant en materia defininm una solució S^* tal que $p(S^*)$ sigui igual a $p^*(G)$. Ens adonem que per cada $i \in S^*$ el grau de $i \geq p(S^*)$ i de fet, per optimalitat de S^* per cada $i \in S^*$ tenim que:

$\frac{|E(S^*)|}{|S^*|} = p(S^*) \geq p(S^* \backslash \{i\}) = \frac{|E(S^*) - grau_{S*}(i)|}{|S^*|-1}$

Com que el $\sum_{i \in S} grau_S(i) = 2|S|p(S)$, com a mínim un node serà esborrat en cada passada. Ara considerem el moment de la primera passada quan el node i de la solució optima S^* és esborrat, en aquest moment es garanteix ja que eventualment S quedara buit. Clarament $S \supseteq S^*$, i si tenim $i \in C(S) \cap S^*$llavors:

$$\begin{aligned} p(S^*) &\leq grau_{S*}(i) \\ &\leq grau_S(i) \\ &\leq 2p(S). \end{aligned}$$

Aixo implica que $p(S) \geq \frac{p(S*)}{2}$ i per tant l'algoritme dóna com a resultat l'aproximació anunciada.

4. Entorns de test de l'Algorisme

4.1. Entorn local

Per provar l'algorisme en local primer hem de deixar les dades disponibles al sistema de fitxers distribuït (HDFS), per tal de fer-ho podem executar la comanda. En aquest punt amb tots els serveis engegats ja podem executar la nostra tasca. Per tal de fer-ho prèviament s'ha de crear un jar amb el codi font compilat i mitjançant línia de comandes executar-lo.

Tècnicament l'entorn de test local s'ha format amb un ordinador domèstic amb la següent configuració:

Processador QuadCore9300: 4 nuclis a 2.5Ghz

- 4Gb de memòria RAM
- SO: Ubuntu 12.04
- Versió Hadoop: 1.04
- Plataforma 64bits

4.2. Distribució en clúster local

Aquesta configuració s'ha realitzat en la sala d'ordinadors IA de la UB. La configuració és d'un ordinador "master" i dos "slaves" amb les mateixes característiques tècniques. Pel que fa a l'execució de les tasques és molt similar a la de la configuració local.

4.3. Cloud Amazon ElasticMapReduce (EMR)[1]

Per realitzar les proves en aquest entorn hem descrit 5 tipus de instàncies que s'anomenen "Small", "Medium", "HighCPU", "Large" i "ExtraLarge" en l'entorn

[1]Amazon Elastic MapReduce (Amazon EMR) és un servei web desenvolupat per Amazon, que permet a empreses, investigadors, analistes de dades i desenvolupadors processar grans qantitats de dades de forma eficient. Utilitza un framework Hadoop sobre l'infraestructura web d'Amazon Elastic Compute Cloud (Amazon EC2) i Amazon Simple Storage Service (Amazon S3).

d'Amazon EMR [AWS]. A continuació detallarem les especificacions de cadascuna d'elles:

- **"Small"**: Memòria de 1,7 GiB 1 unitat informàtica EC2 (1 nucli virtual amb 1 unitat informàtica EC2) Plataforma de 32/64 bits Rendiment de E/S: moderat
- **"Medium"**: Memòria de 3,75 GiB 2 unitats informàtiques EC2 (1 nucli virtual amb 2 unitats informàtiques EC2) Plataforma de 32/64 bits Rendiment de E/S: moderat
- **"HighCPU"**: Memòria de 1,7 GiB 5 unitats informàtiques EC2 (2 nuclis virtuals amb 2,5 unitats informàtiques) Plataforma de 32/64 bits Rendiment de E/S: moderat
- **"Large"**: Memòria de 7,5 GiB 4 unitats informàtiques EC2 (2 nuclis virtuals amb 2 unitats informàtiques) Plataforma de 64 bits Rendiment de E/S: moderat
- **"ExtraLarge"**: Memòria de 15 GiB 8 unitats informàtiques EC2 (4 nuclis virtuals amb 2 unitats informàtiques) Plataforma de 64 bits Rendiment de E/S: alt

Segons pròpies instruccions d'Amazon, una unitat de sistemes EC2 proporciona la capacitat de CPU equivalent d'un processador Opteron del 2007 o Xeon del 2007 de 1,0-1,5GHz. També es equivalent a un processador Xeon del 2006 de 1,6GHz.

Totes aquestes instàncies es provaran finalment amb 4, 8 i 16 nodes "slave".

5. Dades a analitzar

Per realitzar els tests hem obtingut una sèrie de grafs no dirigits publicats en la universitat de Stanford [Uni]. El primer graf que usarem per fer proves és una petita porció de la xarxa d'ítems que estan disponibles a la venta en la web d'amazon. Cada node representa un objecte dins de la xarxa i cada aresta és una de les suggerències que fa Amazon sobre un altre producte. Les dades tècniques del graf són les següents:

- Nodes: 334,863
- Arestes: 925,872
- Mida: 12,291Kb

El segon graf és una porció dels usuaris de YouTube. on cada node és un usuari únic, i cada aresta és una subscripció d'un usuari a un altre. Les dades tècniques són les següents:

- Nodes: 1,134,890
- Arestes: 2,987,624
- Mida: 37,814Kb

El tercer graf és també una porció de la xarxa social LiveJournal que s'orienta a comunitats d'interessos i com la resta de xarxes socials, els nodes són els usuaris i les arestes són les connexions d'amistat entre ells. Les dades del graf són les següents:

- Nodes: 3,997,962
- Arestes: 34,682,189
- Mida: 489,799Kb

El darrer graf és el d'Orkut una xarxa social d'origen brasiler, molt semblant a Facebook.Ccom la resta, els nodes són els usuaris i les arestes les connexions existents entre ells. Les dades tècniques són les següents:

- Nodes: 3,072,441
- Arestes: 117,185,083
- Mida: 1,728,293Kb

6. Resultats obtinguts

Les primeres proves que es realitzaran a continuació són els tres primers grafs (Amazon, YouTube, LiveJournal) en local i en Amazon EMR, després es compararà el comportament amb el graf de LiveJournal a Amazon EMR i en clúster i finalment es mostrarà el rendiment de la configuració en clúster amb el graf més gran d'aprox 1.7Gb.

Així doncs comencem veient els resultats de les proves amb el graf d'amazon (figura 6.1).

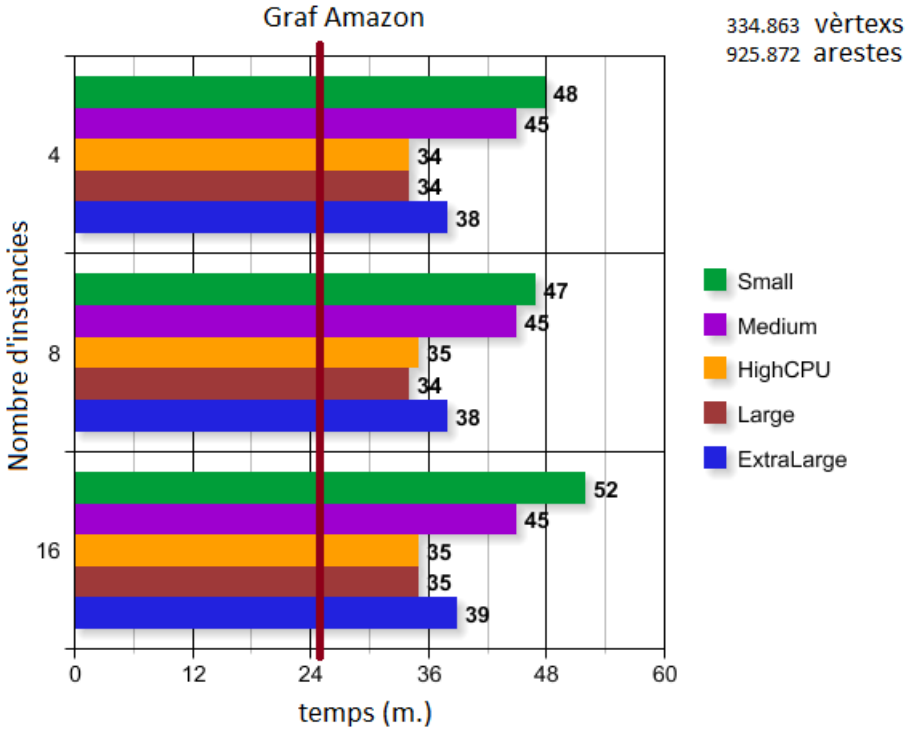


Figure 6.1.: Comparativa graf Amazon

Cada bloc (eix y), representa el nombre d'instàncies o de "slaves" que han executat la tasca en Amazon EMR i en l'eix de les x tenim el temps total en determinar el subgraf més dens expressat en minuts. A més a més la barra vermella vertical és un marcador per determinar el temps d'execució de l'algorisme en l'entorn local.

Podem observar com l'ordinador local guanya per bastant a qualsevol configuració d'Amazon EMR per una simple raó. Cada mapper parteix d'un bloc de dades

d'entrada de 64Mb i com que el graf total és de 12Mb aproximadament només fa servir un sol mapper o el que és el mateix un sol "slave". Aquest fet implica que realment no es paral · lelitza gens el procés i el overhead de comunicacions entre nodes a Amazon i la comunicació amb el sistema de fitxers S3 és molt més costós. Aquest mateix fet també implica que no hi ha gaire diferència entre les configuracions amb diferents nombre de nodes, per això veiem que no canvia gaire el resultat de temps final. Ara podem veure els resultats amb el graf de YouTube (figura 6.2).

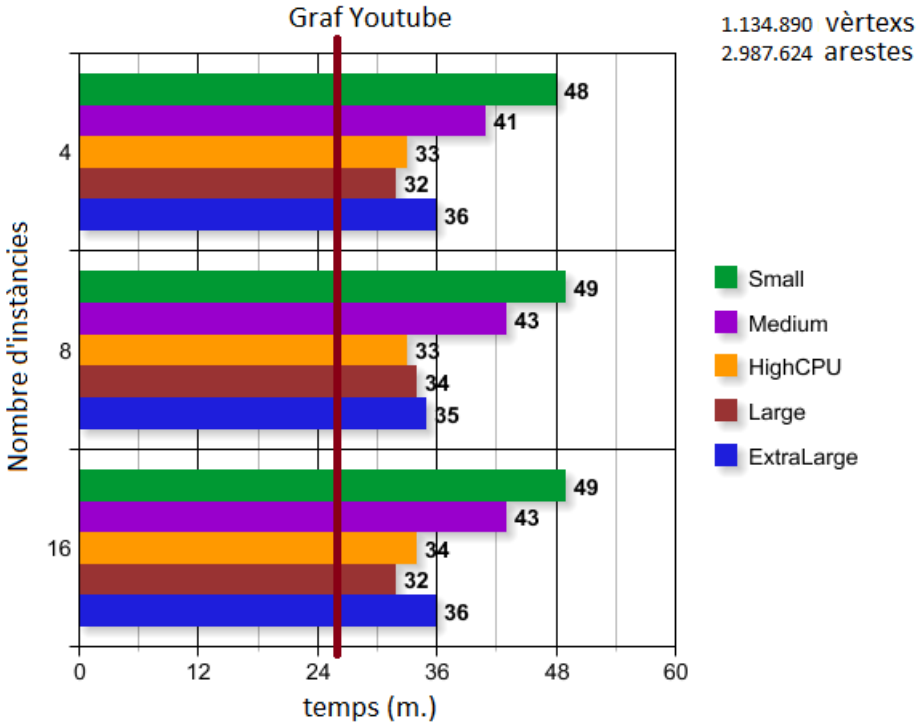


Figure 6.2.: Comparativa graf Youtube

Com en el cas anterior la mida del graf fa que no puguem extreure gaires dades interessants, només destacar que encara que el graf s'ha quasi-bé triplicat en mida, els temps són pràcticament els mateixos. Això ens indica que el temps que no és exclusivament de càlcul de l'algorisme és molt gran amb Hadoop i per feines petites consumeix la majoria del temps.

Ara passem a veure el cas més interessant (figura 6.3) on ja tenim un graf considerablement més gran de més de 500Mb. En aquest cas doncs ja tenim més d'un mapper, concretament en particions de 64Mb en la primera iteració en tenim uns 8. Aquesta dada ens indica que més de 8 nodes ja no obtindran cap resultat útil.

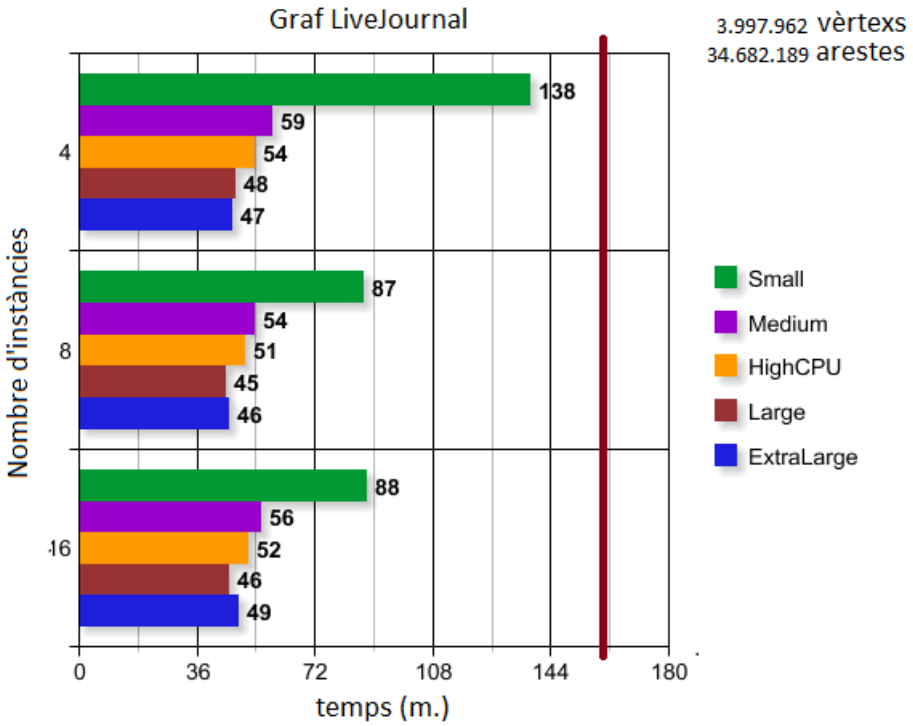


Figure 6.3.: Comparativa graf LiveJournal

Efectivament, veient els resultats (figura 6.3), ara podem confirmar les observacions anteriors i veiem com totes les configuracions d'Amazon EMR passen per davant de la configuració local. També podem observar com la configuració "Small", queda molt despenjada de la resta, mentre que amb aquest exemple no es nota molta diferència entre les altres. Com hem comentat abans més de 8 nodes en aquest exemple no aporta res fins i tot fan enrrederir uns minuts el temps total.

El gràfic situat en (figura 6.4) mostra el temps total entre iteracions de l'algorisme (dades extretes de la configuració més ràpida, que és Amazon EMR Large 8).

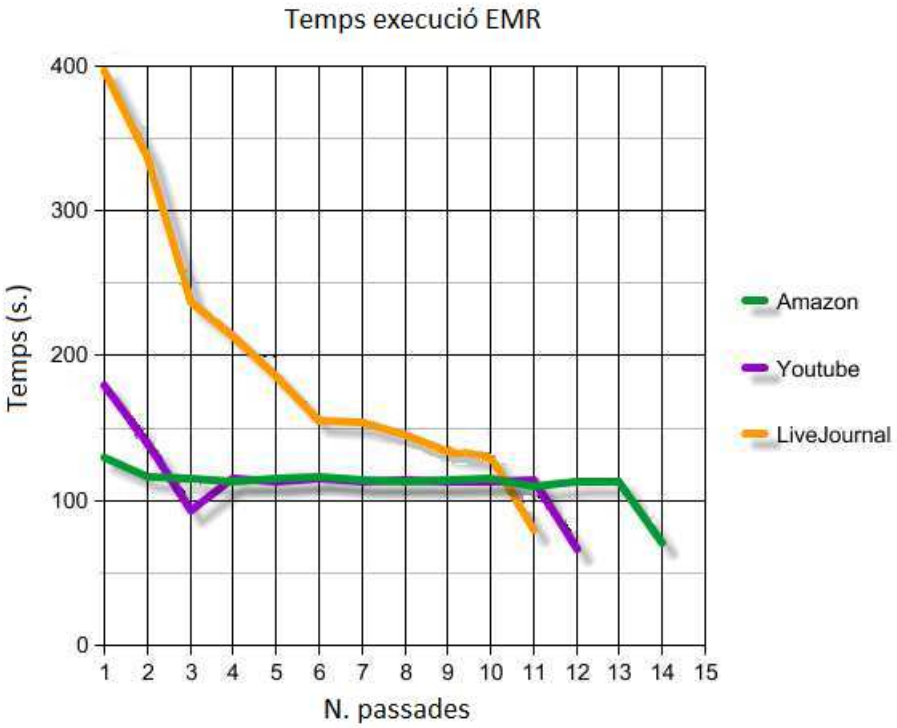


Figure 6.4.: Temps execució a EMR

Podem veure com a mesura que s'eliminen dades del graf i aquest es fa més petit, el temps entre iteracions es va estabilitzant. Aquest fet està relacionat amb el que comentàvem anteriorment, quantes menys dades menys blocs a repartir entre els mappers i menys paral · lelització. Amb caràcter merament informatiu, per mostrar els resultats de l'algorisme veiem quin grau del subgraf ha anat calculant a mesura que passen iteracions, amb els 3 grafs en qüestió (figura 6.5).

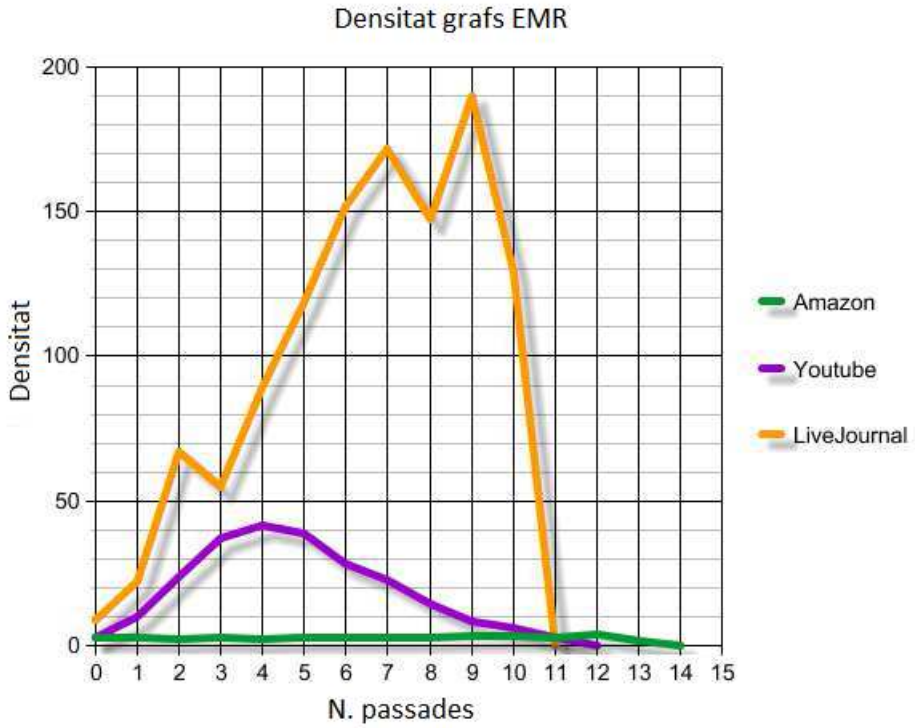


Figure 6.5.: Densitat dels grafs a EMR

Tal com indica la figura, en el cas d'Amazon el subgraf més dens es troba en la iteració 12 amb una densitat de 4,08. En el cas de YouTube aquest es troba en la iteració 4 amb una densitat de 41,416 i finalment en el cas de LiveJournal el troba en la iteració 9 amb una densitat de 189,959.

Ara passem a veure les diferències entre l'execució amb Amazon EMR i la configuració en clúster local del mateix graf de LiveJournal (figura 6.6).

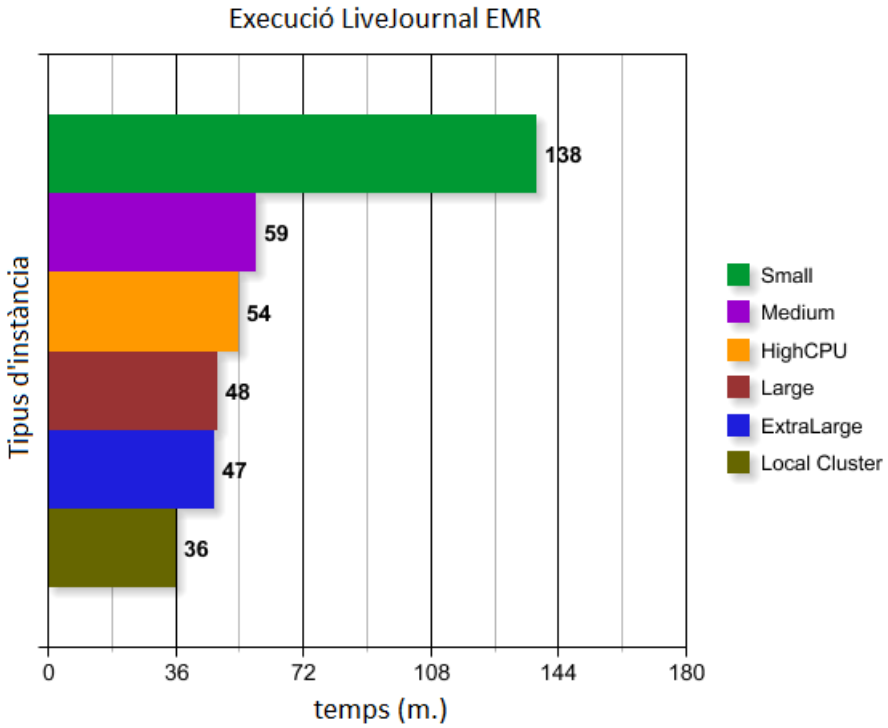


Figure 6.6.: Comparativa graf LiveJournal a EMR

Per realitzar la comparació s'ha agafat la configuració d'Amazon EMR amb 4 "slaves", que és la que més s'assembla en nombre a la configuració en clúster que és de 2. Tot i així veiem com és significativament més rapida que la resta de configuracions en cloud, concretament un 30% més rapida que la més potent i un 283% més rapida que la "Small", la més lenta. Com hem comentat en casos anteriors el fet que el clúster local les comunicacions siguin més rapides que en cloud amb el seu sistema de fitxers S3 fa que els resultats siguin molt millors. Pel que fa al temps per iteració veiem com amb les primeres iteracions la configuració d'Amazon EMR guanya a la de clúster perquè al haver més dades i més blocs a repartir, conjuntament amb el fet que la configuració d'Amazon EMR tingui dos nodes "slave" més aporta un valor en temps significatiu. Al final però amb menys quantitats de dades la configuració en local no ha de suportar tant overhead en comunicacions i s'avança (figura 6.7).

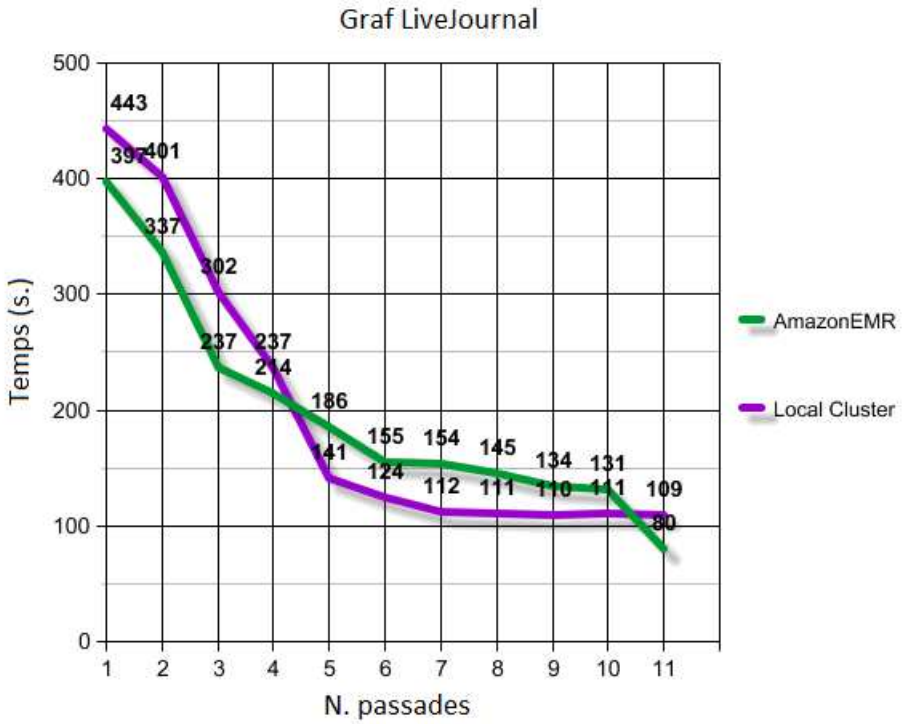


Figure 6.7.: Temps iteracions graf LiveJournal

El darrer graf només s'ha pogut provar en clúster donada la gran mida del graf i les males condicions per pujar dades als "buckets" del sistema de fitxers S3. Tot i així serà interessant veure com s'adapta l'entorn en clúster per treballar amb un graf més del triple de gran que el de LiveJournal. Passem a veure el temps d'execució total en (figura 6.8).

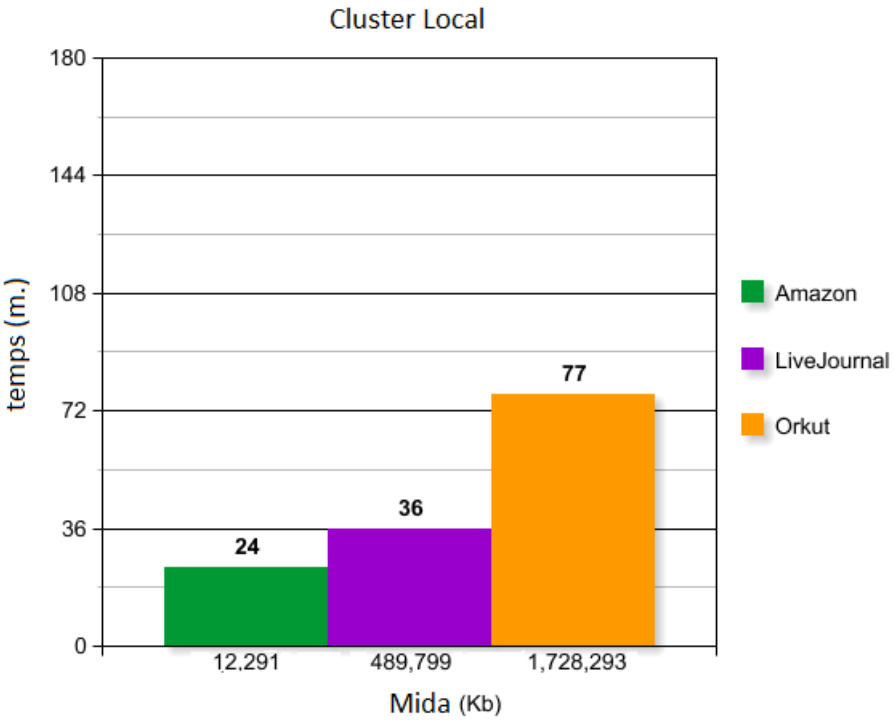


Figure 6.8.: Resultats execució clúster local

Podem observar com el clúster local s'adapta molt bé a les mides més grans de grafs, només duplicant el temps d'execució total i no triplicant com caldria esperar al tindre un graf més de tres cops més gran. Una ullada al graf de temps per iteració ens dirà per que (figura 6.9).

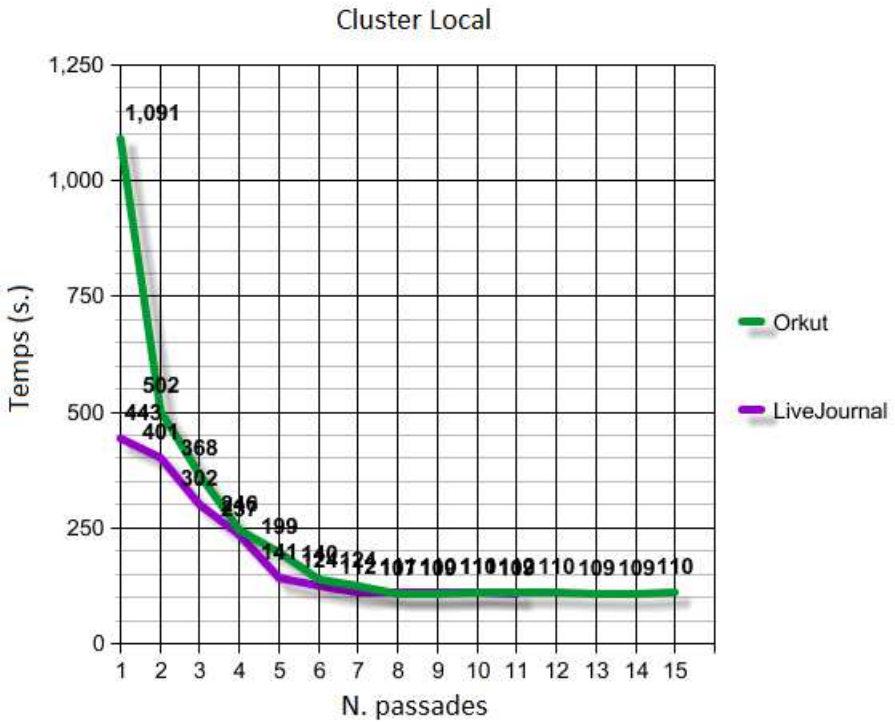


Figure 6.9.: Temps iteracions clúster local

Com podem observar el moment que requereix més temps són les primeres iteracions on el graf s'ha d'analitzar per complet, després quan tenim un graf reduït el temps d'execució és similar. Per acabar aquest apartat mostrarem gràficament quins són els resultats obtinguts amb l'algorisme i si realment podem parlar de subgraf dens com a comunitat. Com que l'únic graf que representa objectes que podem mentalment agrupar per noms i continguts és el d'amazon, mostrarem els dos primers subgrafs densos que s'han trobat. L'únic que canvia en l'algorisme per treure el segon subgraf dens, és que en acabar la primera tasca s'han d'eliminar els nodes obtinguts i les seves connexions i tornar a començar l'execució.

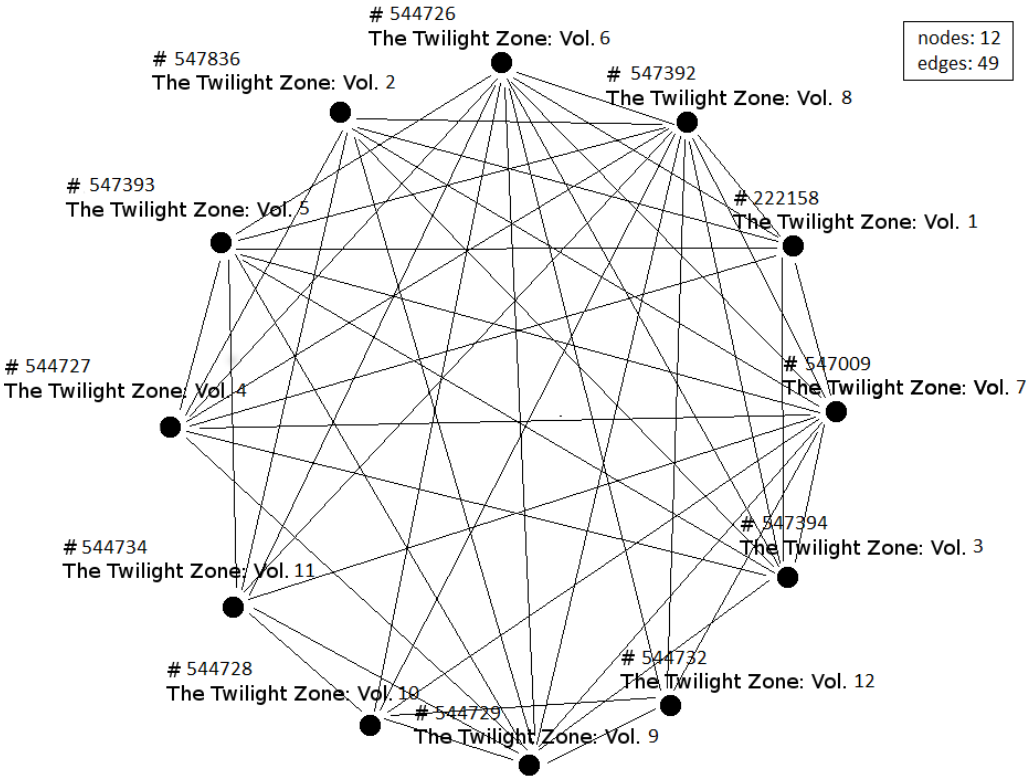


Figure 6.10.: Primer subgraf més dens Amazon

És evident que en aquest cas (figura 6.10) tots els volums de l'edició de "The Twilight Zone" formen la comunitat més densa. El segon graf més dens és molt més gran i ens donara una mostra més significativa del funcionament de l'algorisme.

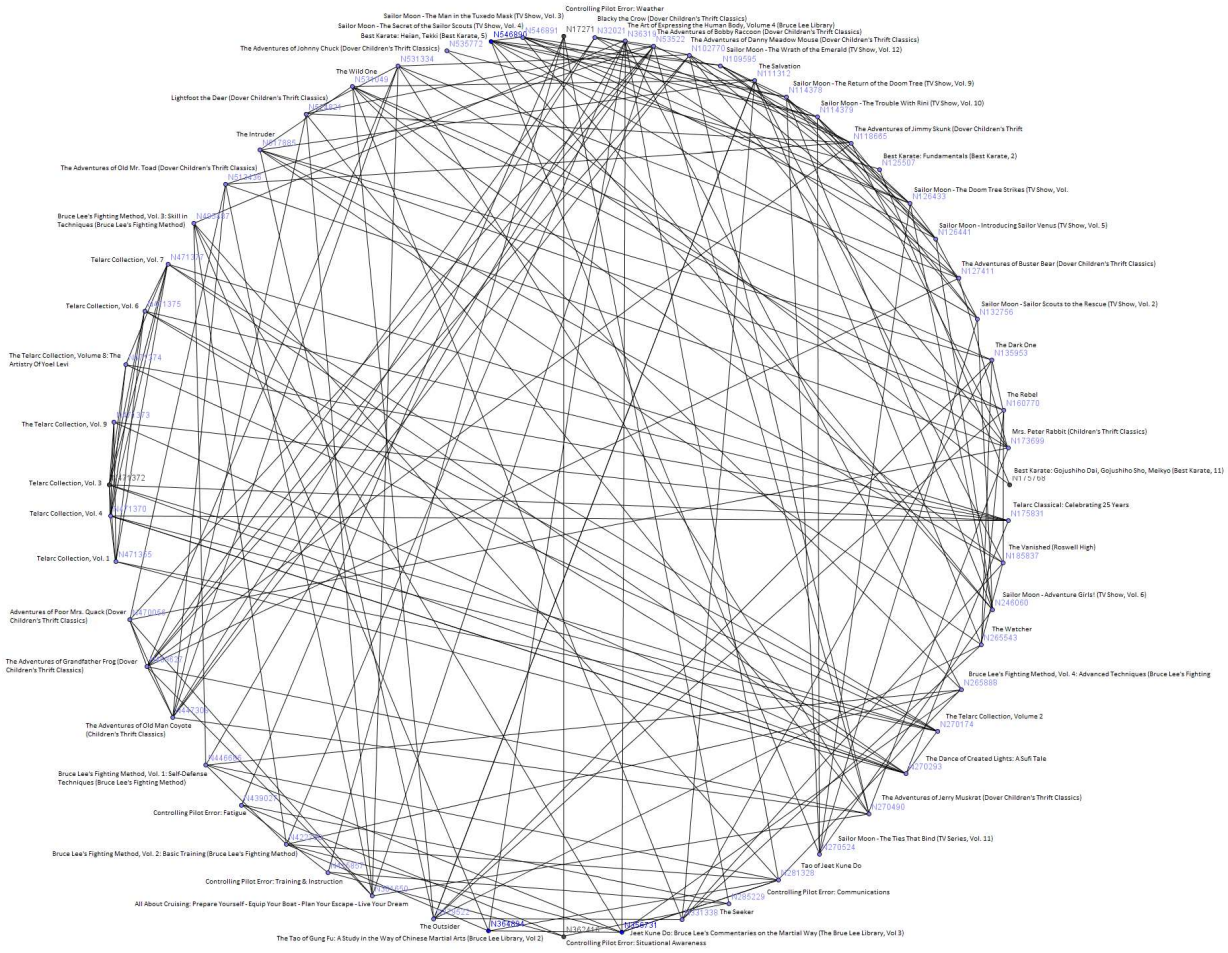


Figure 6.11.: Segon subgraf més dens Amazon

En aquest segon graf (figura 6.11) ja podem trobar diferents col · leccions enllaçades en una gran comunitat, entre d'altres una col · lecció de pilotatge d'avions, arts marcials, animé, obres infantils, etc....

7. Conclusions

Hem vist com canviar la mentalitat per tal d'adaptar algorismes a la manera de treballar amb paral · lelització pot dificultar una mica més la programació al principi, però un cop habituat, amb poques línies de codi i amb tota la feina que fa per darrera Hadoop i MapReduce es pot aconseguir resoldre problemes complexos, que d'altra manera serien molt més difícils d'abordar.

Pel que fa a les diferents configuracions també es pot veure que les opcions de cloud d'Amazon EMR més econòmiques són inferiors en capacitats a la feina que es podria fer amb un clúster d'ordinadors local i tot i sent més econòmiques resulten força costoses, ja que com s'explica en l'annex d'anàlisi de costos si no es té un control d'hores els preus es disparen.

Si finalment s'opta per l'ús d'Amazon EMR també s'ha de tindre en compte que com hem vist en l'anàlisi de resultats s'han d'escollir les configuracions més adients per la nostra tasca. Primer hem de veure que depenent de la mida de les dades d'entrada, tindrem mappers (que es paral · lelitzen) per cada bloc de 64Mb per defecte, aixi doncs és inútil escollir un nombre de nodes "slave" més gran que el nombre de mappers o blocs que tindrem. En aquest sentit si cada mapper no realitza gaires càlculs és millor escollir instàncies més petites ja que com hem vist al final per emetre valors i llegir-los, a partir de les instàncies "Medium" ja no es noten diferències significatives en quant a temps.

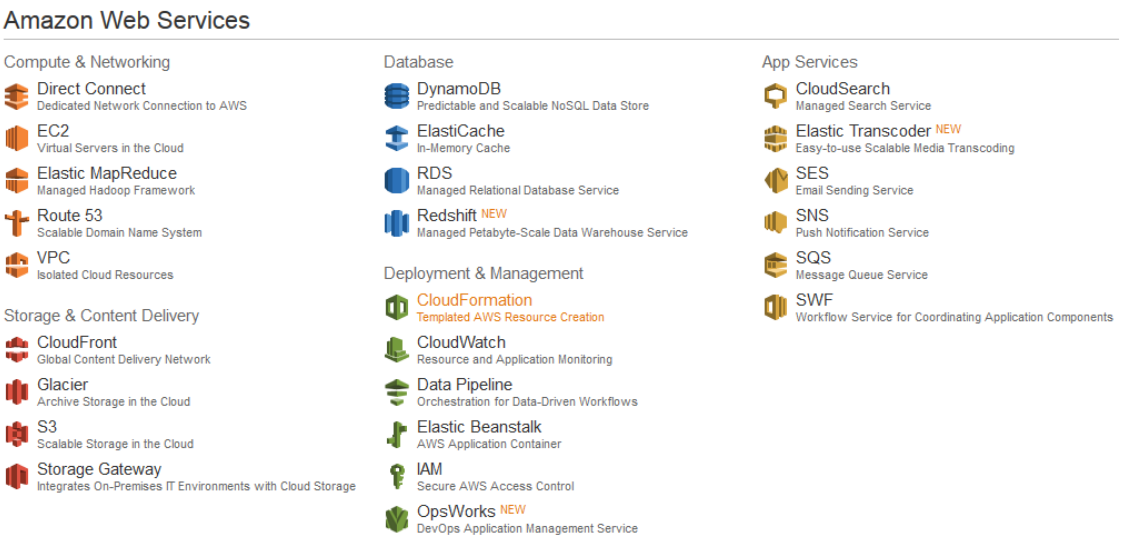
Com a punt final remarcar que la gran quantitat de camins que s'obren amb aquest paradigma de programació és immensa i que és el futur immediat pel que fa al tractament de totes les dades que generem en un mon totalment connectat i en constant creixement. Aixo ho constata el fet que les empreses més grans de tecnologia (Google, Facebook, ..) ja fa temps que usen aquestes tecnologies.

A. Annexos

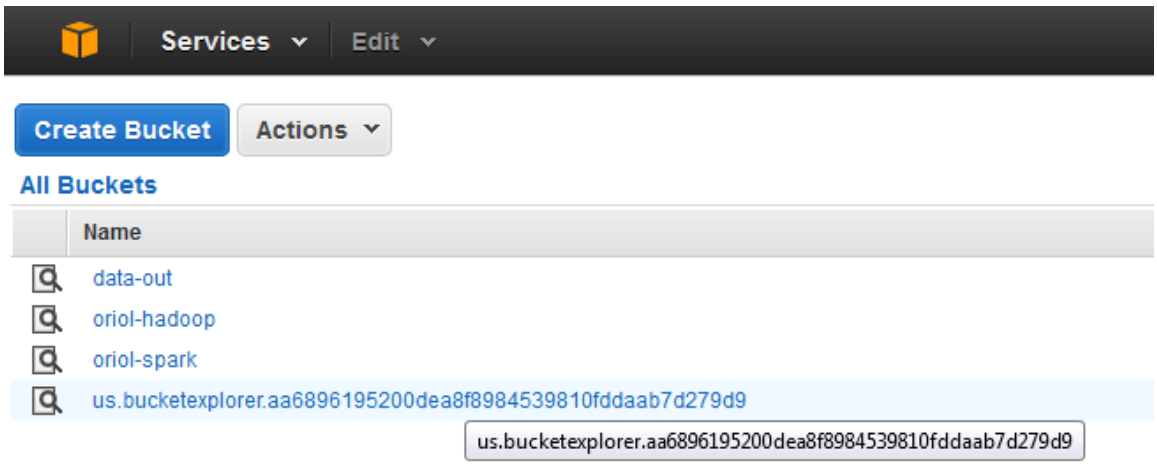
A.1. Breu explicació de l'ús dels serveis d'Amazon EMR

Els serveis en cloud d'Amazon són realment senzills d'usar i per tant en aquest annex abordarem el seu ús de manera breu. Primer començarem explicant com deixar les dades necessàries en el sistema de fitxers en cloud S3.

El primer pas és seleccionar l'opció S3 en la consola d'administració:

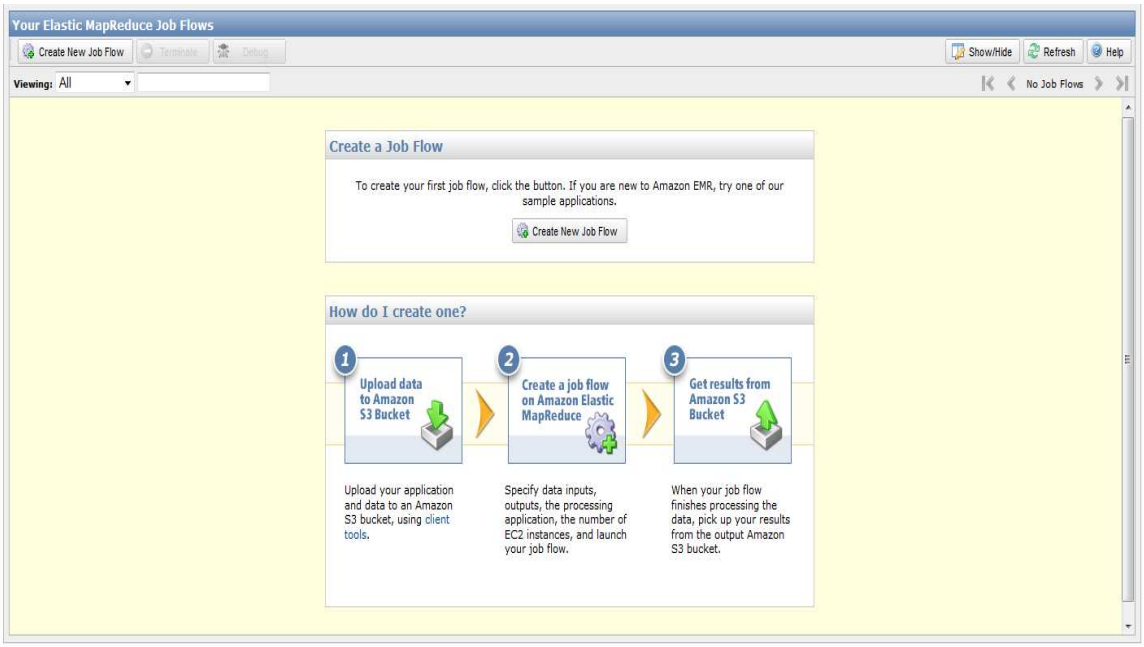


Això ens obrirà la pantalla principal per administrar les nostres dades:



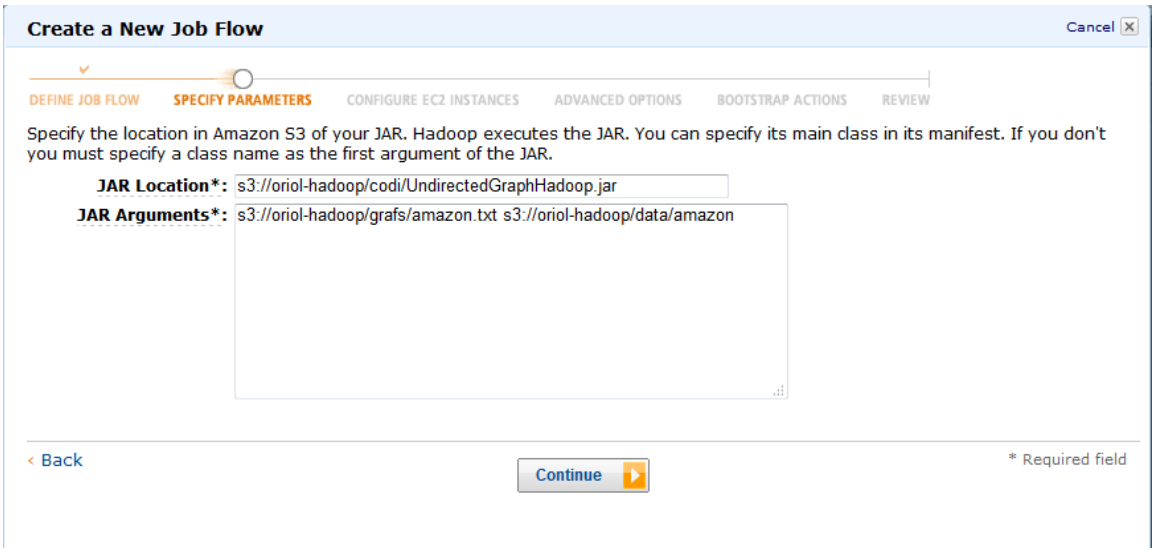
Ara ja podem crear nous "buckets" que són com carpetes per pujar els jars executables i els grafs font, i on també aniran a parar els fitxers de sortida.

Un cop tenim les dades que necessitem pujades, des de la consola d'administració hem d'anar a Amazon EMR i se'ns obrirà la següent aplicació de gestió de tasques amb instruccions d'ús:

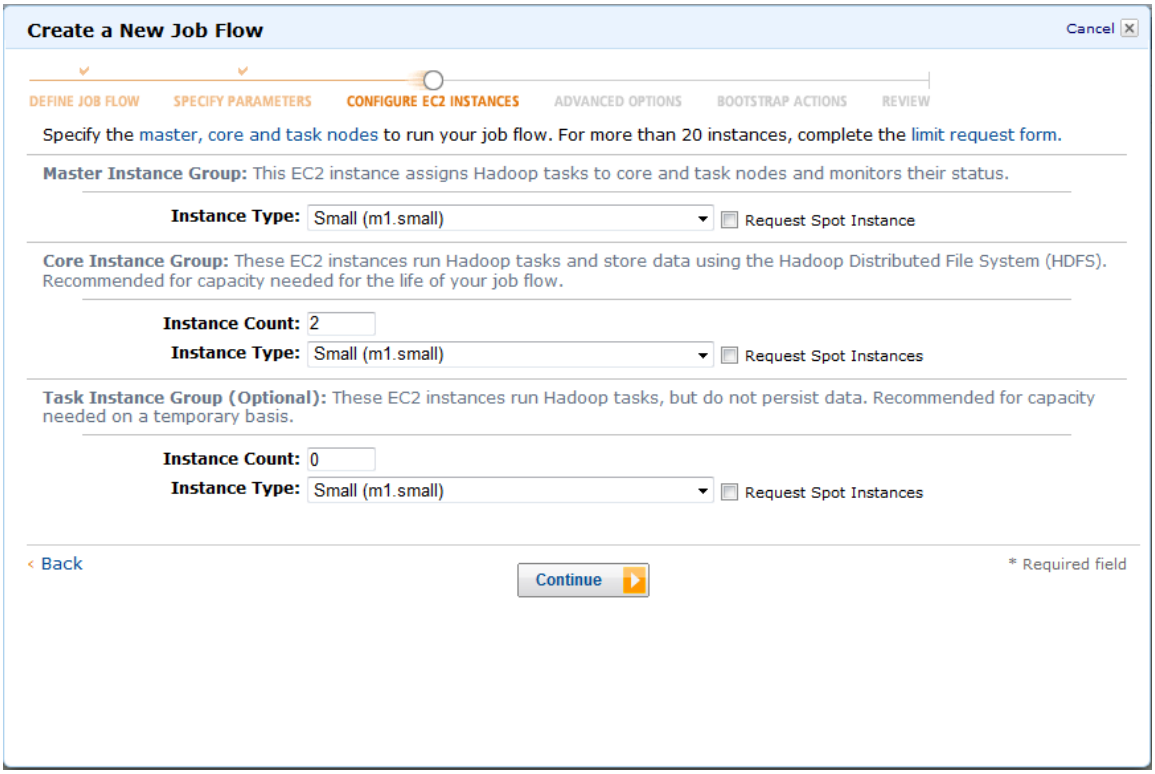


Els passos per executar la tasca són molt senzills primer hem de prémer sobre el botó de la part superior "Create New Job Flow" i ens apareixerà un diàleg on omplir el nom de la nostra tasca, el tipus de distribució desitjat (en el nostre cas Distribució Hadoop normal) i el tipus d'aplicació(en aquest cas hem d'escollir "Run your own application" i en el desplegable inferior "Custom JAR". Després de prémer

"Continue", el següent pas és omplir el path del nostre jar i els arguments tal com s'indica en la següent figura el nostre cas és path:
-s3://oriol-hadoop/codi/UndirectedGraphHadoop.jar i els arguments:
-s3://oriol-hadoop/grafs/amazon.txt s3://oriol-hadoop/data/amazon

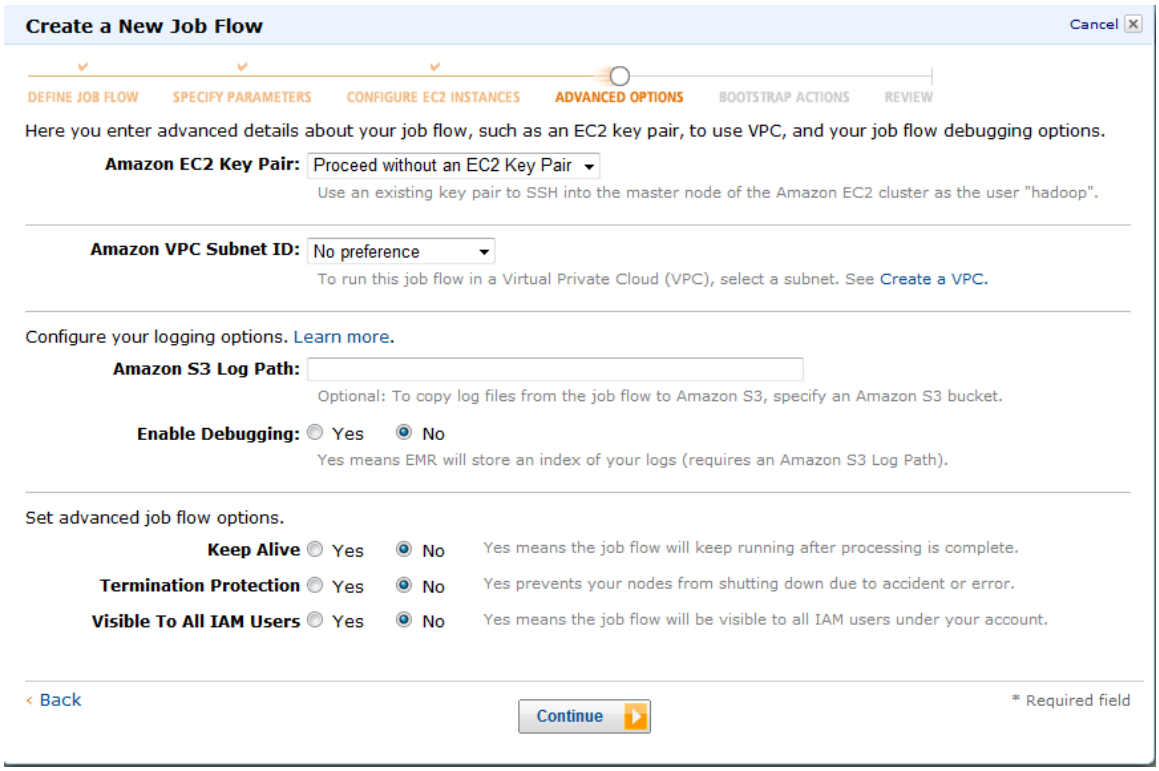


El següent pas és definir quin tipus d'instància escollim i quants nodes "master" i "slave" escollim, en el següent exemple escollirem instancia "Small" amb 1 node "master" i 2 nodes "slave".

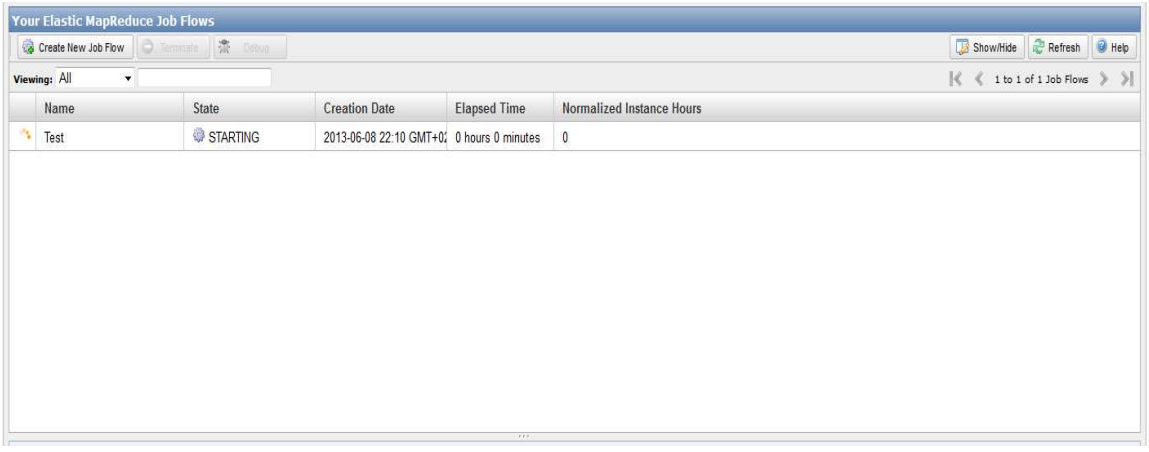


La darrera pantalla d'opcions és on podem escollir si volem engegar la tasca en

mode debugging (penalitza en temps l'execució) el path dels logs, i comportaments diversos.



Seguint amb el pas següent tenim la configuració de "Bootstrap Actions". Que son accions que podem predefinir en moments determinats de l'execució de la nostra tasca, en aquest exemple no en tenim i premem "Continue". Després de revisar totes les opcions podem prémer sobre "Create Job Flow" i ja podem donar per creada la tasca. Ho podem veure en l'aplicació de gestió de tasques d'aquesta manera:



Normalment i depenent del estat de les xarxes i la carrega de feina dels servidors d'amazon trigarà uns minuts en canviar d'estat "STARTING" a "RUNNING". En aquest punt encara podem cancel · lar la tasca, però si passa a estar en estat "RUNNING" ja ens aplicaran el preu del servei. Un cop ha finalitzat la tasca veurem com

canvia de “RUNNING” a “FINISHED” o “FAILED” si és que hi ha hagut algun error en l’execució.

Per acabar cal destacar la informació de la taula on tenim la data de creació, el temps transcorregut de la tasca i per últim les hores bàsiques de càlcul fetes servir (ens ajudarà a calcular el preu total d’execució en curs seguint l’annex 2 d’aquest document). Finalment la part inferior de la consola d’administració tenim un detall de la tasca seleccionada en la taula de tasques on podem veure els passos que esta executant, les gràfiques de monitoreig, etc....

A.2. Anàlisis de costos

L’anàlisi de costos és un factor determinant per poder escollir entre les diferents tecnologies i configuracions, ja sigui en clúster local o en cloud, per trobar la millor estratègia possible per abordar el problema que estiguem tractant. En aquest annex veurem els diferents preus de les configuracions en cloud d’Amazon EMR. Veiem directament el gràfic:

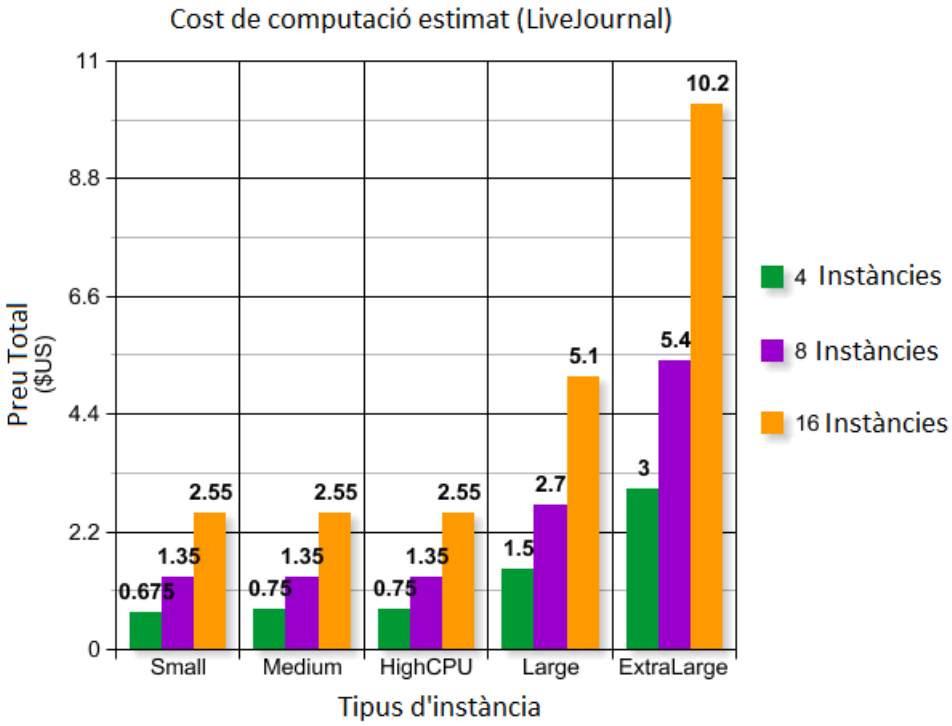


Figure A.1.: Anàlisi de costos

El gràfic anterior representa el preu, expressat en dòlars americans, de l’execució completa de l’algorisme per trobar el graf més dens amb el graf de LiveJournal. Aquests preus estan calculats en base al preu hora bàsic que ofereix Amazon. Cada

configuració amb el nombre de nodes que el componen suposen una quantitat de hores bàsiques de càlcul que es correspon de la següent manera:

- **Configuració "Small":**
  -4 "slaves" → 3 hores bàsiques.
  -8 "slaves" → 9 hores bàsiques.
  -16 "slaves" → 17 hores bàsiques.
- **Configuració "Medium":**
  -4 "slaves" → 10 hores bàsiques.
  -8 "slaves" → 9 hores bàsiques.
  -16 "slaves" → 17 hores bàsiques.
- **Configuració "HighCPU":**
  -4 "slaves" → 10 hores bàsiques.
  -8 "slaves" → 18 hores bàsiques.
  -16 "slaves" → 34 hores bàsiques.
- **Configuració "Large":**
  -4 "slaves" → 20 hores bàsiques.
  -8 "slaves" → 36 hores bàsiques.
  -16 "slaves" → 68 hores bàsiques.
- **Configuració "ExtraLarge":**
  -4 "slaves" → 40 hores bàsiques.
  -8 "slaves" → 72 hores bàsiques.
  -16 "slaves" → 136 hores bàsiques.

Només queda especificar que el preu hora bàsica és la suma de dos serveis que hem contractat, en primer terme el us del servei bàsic EC2 que és de 0,015$ més la configuració predeterminada de EMR que és de 0,060$, fent en total 0,075$/hora.

Aplicant les dades anteriors al nostre cas la configuració "Small" amb 4 "slaves" ens dona un total de 3 hores bàsiques de computació multiplicat per 3 hores que ha durat la tasca fent un total de 0,675$. (Nota: el tram tarifari és d'hores senceres, això vol dir que si una tasca dura 5 minuts es tarificarà com si hagués durat una hora sencera).

Podem observar com les configuracions a partir de "Large" es disparen en preus i més si afegim més de 8 nodes i depenent de la tasca s'orienten més a àmbits professionals. Hem de tindre en compte aquest sistema de tarificació per que per exemple si tinguéssim una tasca de 10 hores amb instancies "ExtraLarge" i 16 nodes el preu pujaria a (0,075×136×10)=102$!.

Bibliography

[AWS] Amazon AWS. Amazon ec2 instances.

[BBV] Ravi Kumar Bahman Bahmani and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce.

[DG] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters.

[Dum] Edd Dumbill. A look at the components and functions of the hadoop ecosystem.

[Eng] Google App Engine. Mapreduce overview.

[Had] Apache Hadoop. Current documentation.

[IBM] IBM. What is hadoop?

[PI] Rob Peglar and EMC Isilon. Introduction to analytics and big data - hadoop.

[Uni] Stanford University. Stanford large network dataset collection.