Sistemes Distribuïts

Practica 1: Models de Comunicació i Middleware

UNIVERSITAT ROVIRA i VIRGILI

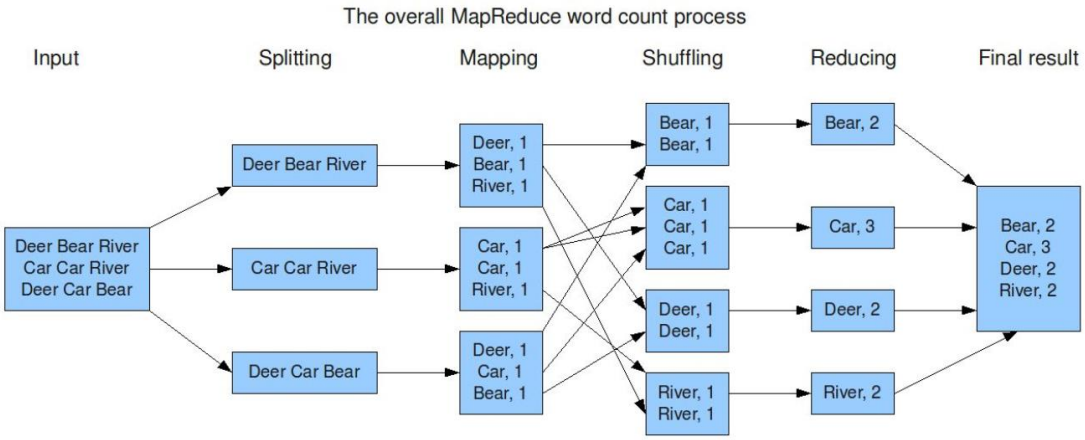
David Ferrer Fernández

Ivan Chernyy

Introducció

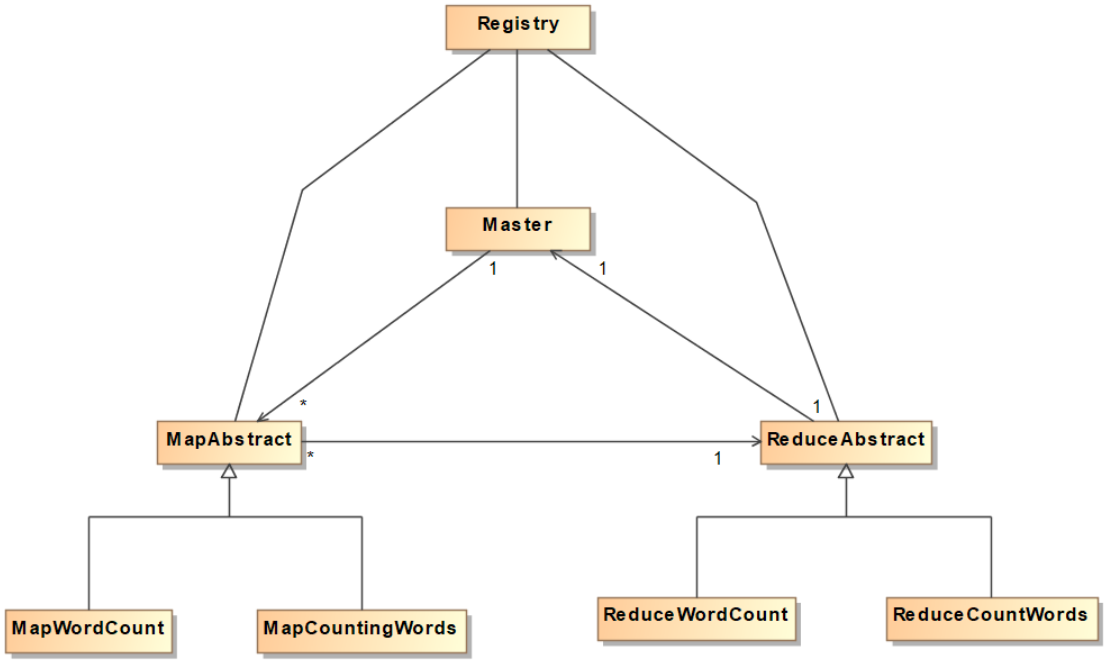
En aquesta pràctica hem fet una implementació simple del model MapReduce, explicada al Paper de Google. Aquest model de programació permet processar de manera paral·lela i distribuïda una gran quantitat d'informació, implementa l'arquitectura master/slave i ho hem programat en Python mitjançant la llibreria pyactor, proporcionada pel professor.

El funcionament consisteix en un node master que divideix la informació per a N mappers i els crida perquè la processin de forma paral·lela, un cop han acabat tots criden al reducer perquè l'ajunti tota i doni un resultat.



En la nostra implementació no hem fet servir Shuffling i nomes hem utilitzat un reducer, per a fer-ho més simple.

Arquitectura



La nostra arquitectura es basa principalment en un Registry, un Reducer i N Mappers, com a classes (actors); on el numero Mappers són els hosts enllaçats - 1.

Per començar hem d'iniciar el Registry en qualsevol ordinador de la xarxa, aquest disposa d'un diccionari on es van enllaçant tots els hosts que s'executen des de els diferents ordinadors de la xarxa local. Quan el numero de hosts al Registry és 0 enllaçarem un Reducer, la resta de hosts seran Mappers, com que el nom d'aquests es la clau del diccionari que conté el Registry hem decidit que els noms estaran formats per la paraula mapper més el port que se l'hi ha assignat automàticament.

Després d'engegar el Registry, el Reducer i un nombre de Mappers superior a 0 podem procedir a executar el Master, aquest és l'encarregat de dirigir tot el Software mitjançant la seva funció main i una classe actor Master que hem fet amb diverses funcionalitats com, mostrar missatges a la terminal del master des d'altres actors, iniciar el cronòmetre que calcula el temps en segons de l'execució del programa, aturar el cronòmetre i mostrar el temps que ha trigat en finalitzar, a més s'encarrega d'esborrar els arxius creats en el procés de dividir el fitxer principal i atura de manera correcta l'execució del software (funció shutdown). Per altra banda, el main del màster és l'encarregat de demanar a l'usuari el fitxer i la funció que vol executar sobre aquest, també busca al Registry els hosts de reducer i mappers que s'han enllaçat he invoca de manera remota l'actor Reducer i els actors Mapper, un cop invocats divideix per línies el fitxer introduït per l'usuari en N fitxers, segons el número de mappers i inicia un servidor HTTP en un nou procés mitjançant llibreries de Python, per últim, inicia el temps d'execució i fica a treballar a tots els Mappers (funció work).

La classe Map que pot ser MapWordCount o MapCountWords per herència, segons la funció que vulguem executar, rep per paràmetre al constructor el nom del fitxer que li toca processar, el Registry i la adreça IP del servidor HTTP, per mitjà de la llibreria urllib2 fa una petició get al servidor per agafar el contingut del seu fitxer i el guarda en memòria, ja pot cridar a la funció countingWords o wordCount, guarda el resultat i tot seguit busca al Reducer dins del Registry per a enviar-li el resultat.

Un cop el Reducer ha rebut tots els resultats que esperava, ho sap perquè l'hi hem passat el número de mappers que hem ficat a treballar, junta tots els resultats en un de sol i crida a les diverses funcions que conté l'actor Master i que hem mencionat anteriorment, així finalitza l'execució del programa.

¿Com afegiríem noves funcions per a executar en el nostre software distribuït?

Per a executar noves funcions hauríem de crear noves parelles d'actors Mapper i Reducerque heretin de les Abstractes i que s'adaptessin a les necessitats de la nova funció, mitjançant la implementació de work per a Mapper i add_element per a Reducer. D'aquesta manera no ens faria falta modificar el codi proposat, només afegir les noves classes que heretin de les Map i Reduce abstractes, i afegir la nova opció al menú de funcionalitats del master.

Validació de la solució

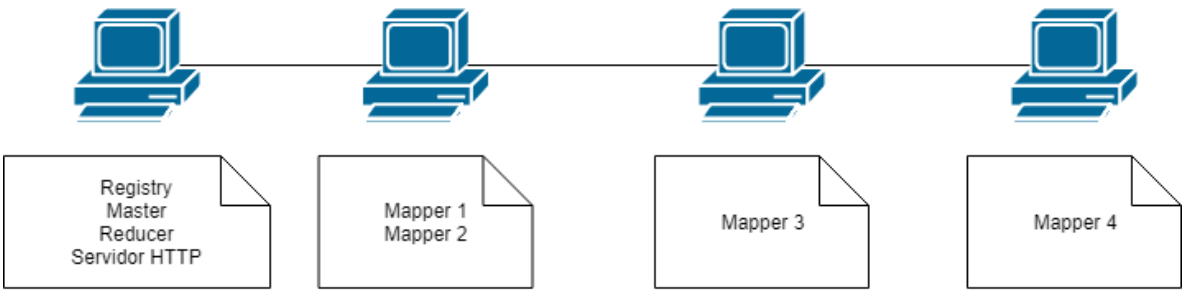
Hem validat la nostra solució de dos maneres diferents, amb diversos ordinadors de l'aula 115 i amb un únic d'aquests ordinadors, aprofitant els 4 nuclis, que disposaven del següent hardware:

Processador: Intel i5-6400, 2,70 GHz, 4 Nuclis

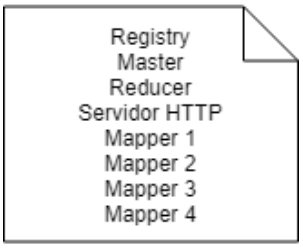
Memòria RAM: 6,7GB reals

Sistema Operatiu: GNU Linux Debian

Validació amb 4 ordinadors.



Validació amb 1 únic ordinador amb 4 nuclis.

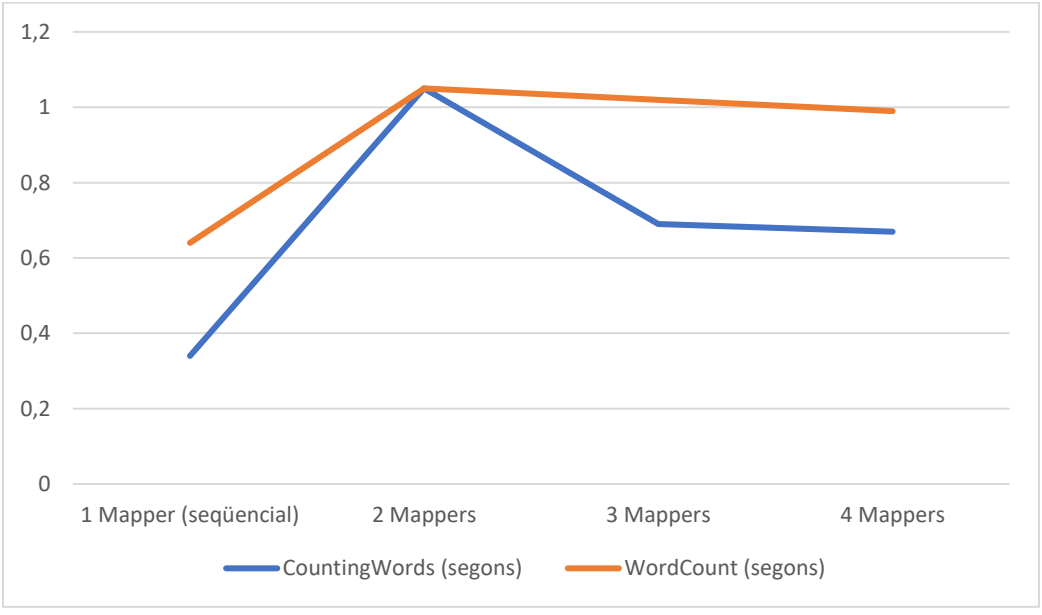


Speedup

Resultats amb 4 ordinadors reals, laboratori 115.

Big.txt (6,5MB)

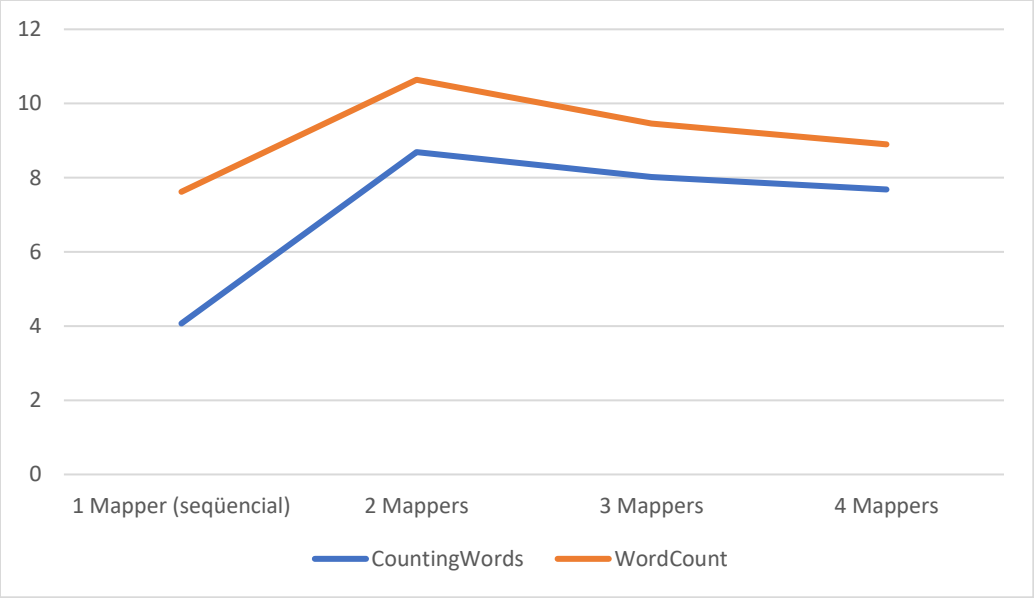
	CountingWords (segons)	WordCount (segons)
1 Mapper (seqüencial)	0,34	0,64
2 Mappers	1,05	1,05
3 Mappers	0,69	1,02
4 Mappers	0,67	0,99



Biggest.txt (77,9MB)

	CountingWords	WordCount
1 Mapper (seqüencial)	4,07	7,62

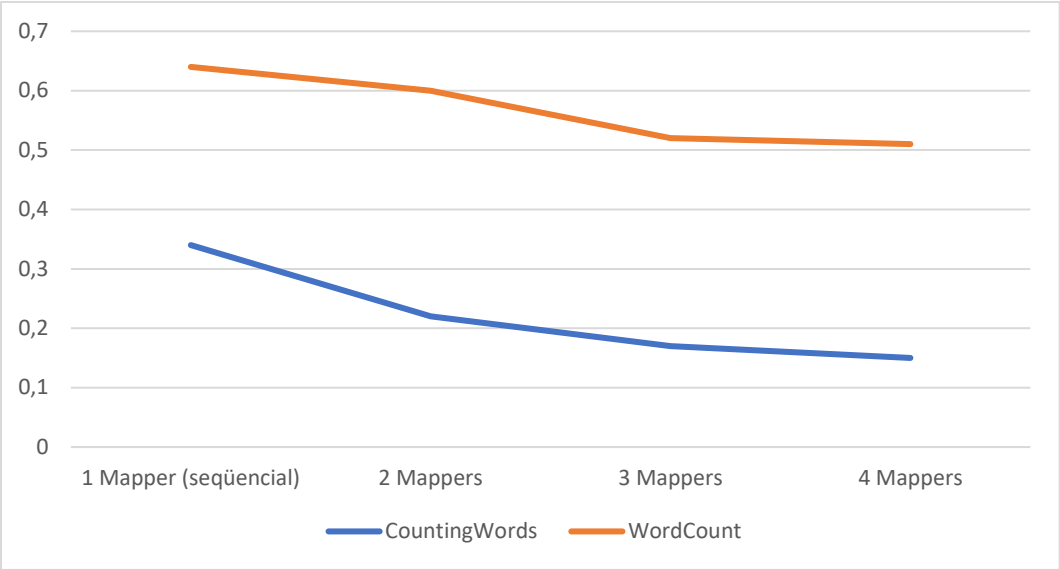
2 Mappers	8,69	10,64
3 Mappers	8,02	9,46
4 Mappers	7,68	8,90



Resultats amb 1 únic ordinador amb 4 nuclis.

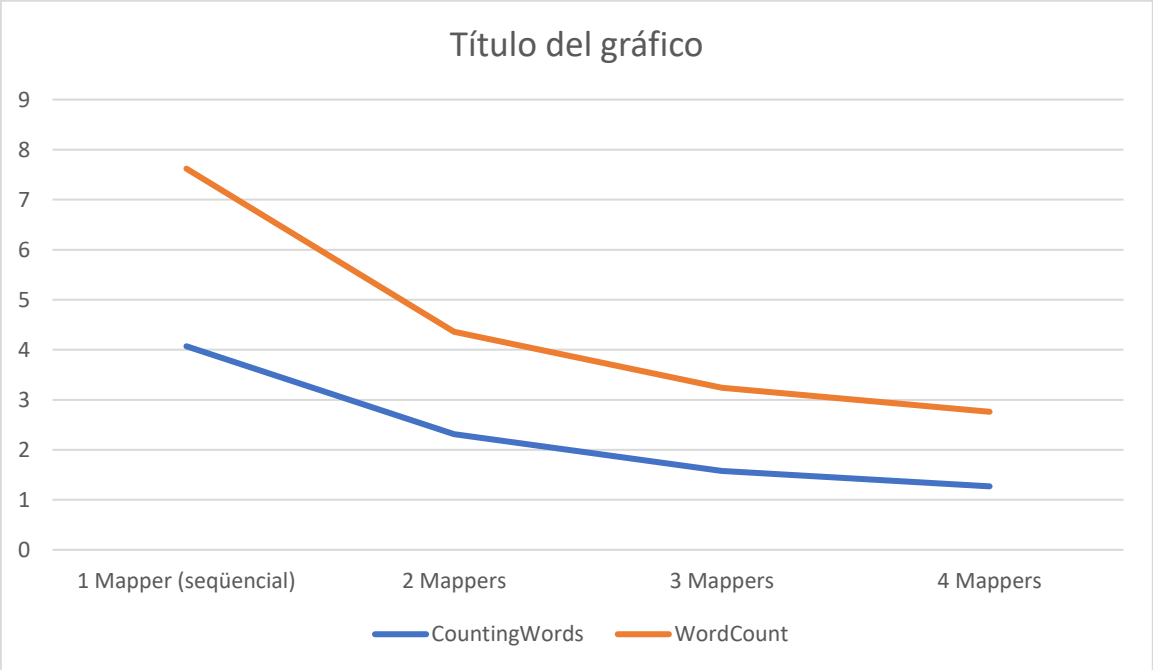
Big.txt (6,5MB)

	CountingWords	**WordCount**
1 Mapper (seqüencial)	0,34	0,64
2 Mappers	0,22	0,6
3 Mappers	0,17	0,52
4 Mappers	0,15	0,51



Biggest.txt (77,9MB)

	CountingWords	WordCount
1 Mapper (seqüencial)	4,07	7,62
2 Mappers	2,31	4,36
3 Mappers	1,58	3,24
4 Mappers	1,27	2,76



Observacions

Amb els resultats obtinguts podem comprovar que en seqüencial el nostre programa es més ràpid, però te una gran penalització en el cas de ordinadors connectats per xarxa, degut a que és molt més lent transferir els arxius entre els diversos ordinadors que treballen en el nostre sistema distribuït, això fa que en aquest cas sigui més lent.

També hem pogut comprovar que contra més gran es l'arxiu en el que volem treballar de forma distribuïda, major és l'Speedup que obtenim.