Calcolo Parallelo in Ambiente Distribuito di qualcosa tramite Apache Hadoop

Università degli Studi di Salerno

Diego Avella

17 luglio 2018

Sommario

Ci troviamo in una situazione storica, per quanto riguarda il mondo dell'informatica, dove enormi quantità di dati (nell'ordine dei terabyte o addirittura dei petabyte e in un futuro non molto lontano di zettabyte) vengono processati. Questi dati devono subire analisi e trasformazioni in modo da poter essere utilizzati in settori che non riguardano solamentamente il mondo informatico: ambienti scientifici come l'ingegneria, la biologia, e la medicina sono i maggiori consumatori di queste informazioni e hanno bisogno di risolvere i loro "problemi" in modo semplice e il più velocemente possibile. Proprio per questo il calcolo parallelo in ambiente shared memory non è più grado di poter gestire e sostenere queste enormi moli di dati in input senza sorpassare con facilità i limiti di tempo (CPU) e di spazio (Ram e Disco) per poter rispettivamente calcolare i risultati e memorizzarli. La nascita del paradigma Map Reduce ideato da Google e del suo file system (GFS), reso famoso dalla Apache Foundation grazie rispettivamente alle implementazioni di Hadoop e HDFS, ha risolto una grandissima branca di problemi disparati senza dover ricorrere a soluzioni artificiose che tentano di ottimizzare a livelli estremi perdendone di portabilità ma che inoltre necessitano di dover essere continuamente aggiornati per poter andare incontro ai limiti fisici di cui si è parlato prima. L'obiettivo di questa tesi è quindi quello di dimostrare che un approccio distribuito di calcolo, attraverso il modello basato su Grid Computing, permette di "superare" questi limiti in maniera semplice e con risultati accettabili se non migliori che potenzialmente possono scalare con facilità. La tesina si propone di (inserire l'argomento di tesi). Il primo capitolo tratterà delle basi su cui si fonda il calcolo distribuito e il Grid Computing, il secondo parlerà di come il framework Hadoop permette di programmare in ambiente distribuito tramite il paradigma Map-Reduce e infine si metteranno in pratica le conoscenze acquisite per analizzare qualcosa e misurarne prestazioni e potenziale scalabilità e infine ci saranno le conclusioni finali.

Indice

Capitolo 1

Fondamenti di Calcolo Distribuito

In questo capitolo verranno introdotte le definizioni e i principi generali su cui si fonda il calcolo parallelo moderno. Verranno introdotti concetti fondamentali come granularità e parallelismo, si analizzeranno le infrastrutture di calcolo parallelo in ambiente distribuito più utilizzate ovvero Cloud Computing e Grid Computing, e si presenteranno le limitazioni che il modello shared memory possiede rispetto al modello distribuito ripercorrendo la storia dell'informatica focalizzandoci sul destino che hanno subito le macchine parallele.

1.1 Definizione di Calcolo Distribuito

Il **Calcolo Distribuito** rappresenta un ramo dell'informatica che si occupa di studiare i **Sistemi Distribuiti**. Con quest'ultima definizione, facciamo riferimento ad un insieme di calcolatori che interagiscono tra loro (di solito tramite l'ausilio della rete) al fine di effettuare dei calcoli per concorrere ad un obiettivo comune che può essere un'erogazione di un servizio, il risultato di un algoritmo molto complesso etc. Un programma scritto per un ambiente distribuito è detto **software distribuito** e la programmazione distribuita è il processo di scrittura di questi software. Il protocollo con cui i calcolatori comunicano tra loro può essere di vario tipo come ad esempio HTTP, RPC, o basato su scambio di messsaggi. Il calcolo distribuito è nato sia grazie alla nascita di Internet, che ha permesso a macchine remote di comunicare tra loro, sia dovuto ad altre cause:

1. **Il limite di potenza di calcolo da parte dei processori:** Se fino alla fine degli anni 90 l'incremento della frequenza di clock di un processore aumentava in maniera costante, permettendo così agli algoritmi sequenziali di migliorare le loro prestazioni ogniqualvolta usciva hardware migliore, con l'avvento delle architetture multicore si è raggiunto un limite difficilmente raggiungibile.

2. **L'enorme mole di dati da dover processare:** Da quando si è integrato l'ausilio dei computer anche in settori estranei all'informatica, gli addetti al mestiere si sono trovati a dover risolvere problemi che avevano come input enormi quantità di dati che non potevano nemmeno lontanamente entrare nella memoria di un unico computer oltre a non avere la potenza computazionale per poter restituire risultati in tempi umanamente accettabili. Se a questo aggiungiamo l'avvento dei Big Data e di tutte quelle branche di analisi dei dati che si sono diffuse a causa di quest'ultimo, possiamo tranquillamente affermare che risulta impossibile per un singolo computer poter processare il tutto.

1.2 Granularità

La granularità rappresenta "l'intervallo che occorre a due processi per sincronizzarsi misurato in numero di instruzioni macchina"[1]. Da questa definizione possiamo sicuramente dedurre che la granularità è inversamente proporzionale ai dati su cui vogliamo operare: più il partizionamento dei dati è piccolo, più istruzioni saranno necessarie per sincronizzarsi e di conseguenza la granularità dei programmi sarà finissima, viceversa se i dati su cui operare è grande sono necessarie meno sincronizzazioni e quindi la granularità è più grossa (coarsed). Quello che offre, ad esempio, un'achitettura basata su Grid-Computing è quella di operare su una granularità più grossa al fine di poter usufruire al meglio dell'hardware di ogni singolo nodo sfruttando la **data-locality** e sincronizzarsi quando è solo strettamente necessario.

1.3 Parallelismo

Il **parallelismo** rappresenta la capacità con cui è possibile eseguire del calcolo in parallelo. È una caratteristica che dipende strettamente dalla granularità

ed entrambe permettono di classificare le architetture di calcolo tramite la tassonomia di Flynn:

SISD: La sigla sta per *Single Instruction Single Data* e non è presente nessuna forma di parallelismo in quanto le operazioni sono svolte in maniera sequenziale basandosi sulla classica architettura di **Von Neumann**. Un limite di questa architettura è sicuramente quello della singola connessione tra processore e memoria che rappresenta un collo di bottiglia per le prestazioni.

SIMD: La sigla sta per *Single Instruction Multiple Data* e il parallelismo è ottenuto dal fatto che ci sono più CPU che processano più flussi di dati in parallelo;è molto utilizzata per il calcolo scientifico ed è stata l'architettura che ha caratterizzato le grandi macchine parallele come Cry e Connection Machine.

MISD: La sigla sta per *Multiple Instruction Single Data* e il parallelismo è dovuto al fatto che più processi o task lavorano su un unico flusso di dati.

MIMD: La sigla sta per *Multiple Instruction Multiple Data* e il parallelismo è dovuto al fatto che più processori lavorano su più flussi di dati ed è l'architettura su cui si basano i cluster moderni.

Se invece vediamo il parallelismo dal punto di vista del programmatore è possibile distinguere due paradigmi principali:

Parallelismo Esplicito: con questo tipo di approccio il programmatore deve esplicitamente tenere conto di come suddividere i task e decidere come e quando sincronizzare questi ultimi. I vantaggi che ne derivano sono sicuramente un maggior controllo da parte del programmatore di quale parti del codice sia necessario parallelizzare oltre ad essere più efficiente ma questo comporta come svantaggio un maggiore sforzo da parte sua nel progettare la soluzione al problema. Un esempio di parallelismo esplicito è quando si usano librerie come MPI.

Parallelismo Implicito: con questo tipo di approccio il programmatore scrive implicitamente del codice parallelo senza che lui ne sia effettivamente consapevole in quanto c'è chi si occupa di farlo per lui (un compilatore, un framework, un middleware). Gli svantaggi sono una

perdita di controllo da parte del programmatore sul proprio codice ma col vantaggio che può focalizzarsi sulla sua risoluzione aumentando di molto la produttività. Un esempio di approccio implicito è quando si usano framework come Hadoop o Spark.

1.4 Modelli di Calcolo Distribuito

Per parlare di Calcolo distribuito e di Sistemi distribuiti dobbiamo sicuramente tenere conto dell'infrastruttura sottostante che sorregge l'intero sistema affinchè possa funzionare al meglio. Ad oggi sono in uso tantissimo due tipologie di infrastrutture: **Cloud-Computing** e **Grid-Computing**.

1.4.1 Cloud-Computing

In un'architettura basata sul Cloud-Computing, le risorse informatiche, di qualunque tipo e non solo computazionali, vengono fornite all'utente quando richiesto. In un' infrastruttura come il Cloud i computer che erogano i servizi non è detto che siano situati sulla medesima rete locale (anzi è un caso raro) e questo comporta comunque un sovraccarico per quanto riguarda la comunicazione remota tra i vari nodi. Esistono diversi tipi di servizi che il Cloud Computing mette a disposizione:

SaaS: (Software as a Service) l'infrastruttura cloud mette a disposizione tutto il necessario per accedere ed usufruitre di un software installato su di una macchina remota.

Paas: (Platform as a Service)l'infrastruttura cloud mette a disposizione una piattaforma su cui è possibile installare eseguibili, librerire, programmi ed è a carico dell'utente gestire tutto il sistema. Questa tipologia di servizio è molto famosa e messa a disposizione da grandi big come Amazon (AWS) e Microsoft (Azure).

Iaas: (Infrastructure as a Service)l'infrastruttura cloud mette a disposizione dell'utente oltre alle risorse virtuali anche quelle hardware che si dovrà gestire.

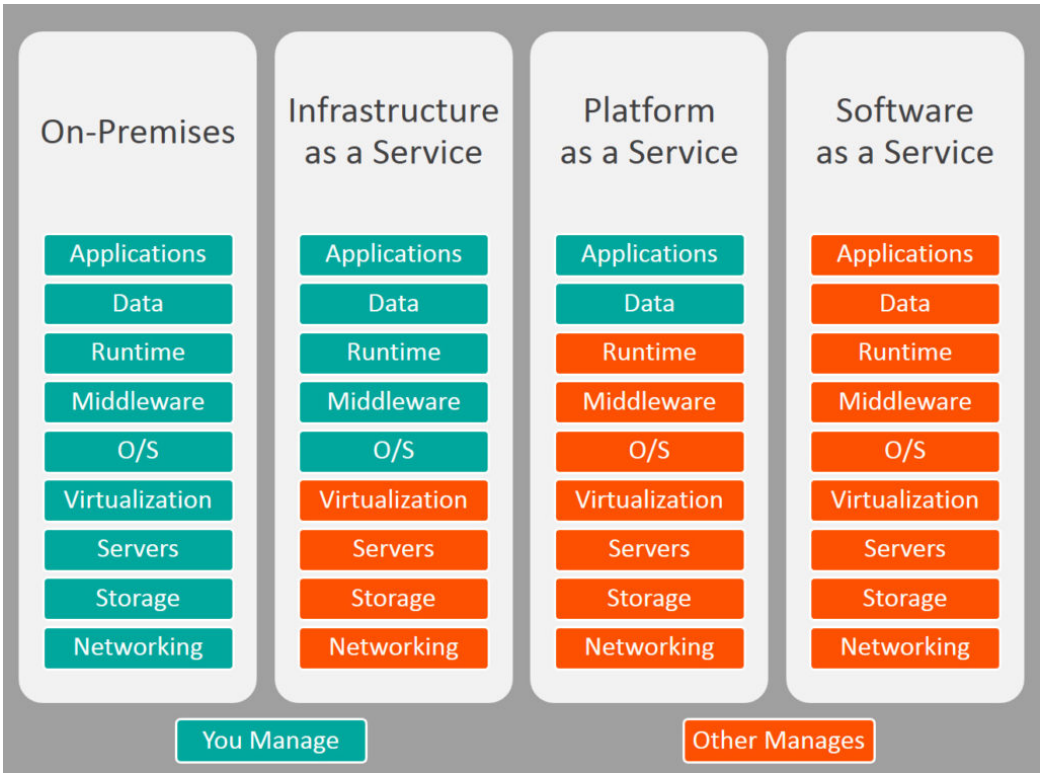

Figura 1.1: I Servizi Cloud

1.4.2 Grid-Computing

In un'architettura basata sul Grid-Computing, le risorse informatiche(anche qui di qualunque tipo ma soprattuto di calcolo e memorizzazione) possono essere messe a disposizione dell'utente per risolvere determinati problemi (di solito di natura molto complessa dal punto di vista del tempo e dello spazio). La maggiore differenza con il Cloud la si evince su come è costruita l'infrastruttura dei nodi. I nodi come dice stesso la parola formano una specia di "griglia" ma in quest'ultima la rete che collega i vari nodi è formata da una rete locale e questo comporta un migliore aumento delle prestazioni a causa del basso consumo a livello comunicativo che i nodi devono effettuare per spedire i dati attraverso la rete. Questa infrastruttura è quella che viene comunemente chiamata **datacenter** e vari sono stati negli anni le possibili soluzioni architetturali che potessero permettere di aggiungengere nuovo

hardware con facilità (quindi facilitare la scalabilità delle risorse), facili da gestire in fase di manutenzione e installazione del sistema operativo il tutto senza compromettere le prestazioni.

1.5 Modello Distribuito e Modello Shared Memory

La domanda che ci si pone immediatamente è la seguente: Perchè eseguire calcoli in ambiente distribuito piuttosto che in shared memory? La risposta la si ottiene se si guarda indietro alla storia dell'informatica e del calcolo parallelo. I primi modelli di macchine parallele sono state ideate tra gli anni cinquanta e sessanta per poi raggiungere il loro periodo d'oro tra gli anni settanta e ottanta con l'avvento d macchine come Cry-1, Cry-2, Encore che avevano un enorme potenza oltre all'enorme quantità di denaro necessario per comprarle e manutenerle. Queste macchine ormai sono un ricordo lontano e la causa del loro fallimento è riconducibile a vari fattori:

Fisico: Come ci dice la "Legge" di Moore il numero di transistor all'interno di un processore raddoppia all'incira ogni 18 mesi. Sappiamo però che questo incremento non sarà, se non è già, più possibile perchè non si può ridurre la grandezza dei transistor oltre una certa soglia (14nm anche se sappiamo che Intel è riuscita ad arrivare fino ad 11nm) per non provocare problemi legati al ***thermal noise***.

Sviluppo: Aumentare il grado di multiprogrammazione tramite l'ausilio di thread ha sicuramente arginato in parte il problema ma non è sicuramente una soluzione in quanto i limiti imposti dalle architetture shared memory rappresentano comunque un collo di bottiglia che inficia sulle prestazioni. Per tentare di risolvere questo problema allora si è cercato di sfruttare modelli di calcolo parallelo tramite l'ausilio di standard di comunicazione (come ad esempio MPI) che sono una possibile soluzione ma rappresentano uno sforzo non da poco da parte del programmatore che deve esplicitamente decidere quali parti del proprio programma parallelizzare o meno.

Portabilità: Macchine del genere avevano sistemi operativi dedicati ed era impossibile effettuare porting dei programmi paralleli su altri architetture dovuto anche alla grande quantità di famiglie di processori che,

prima che Intel ed AMD conquistassero il mercato, esistevano e il passaggio ad architetture multicore peggiorò la situazione portando all'estinzione non soltando queste macchine ma anche un intera famiglia di sistemi operativi Unix-like

Capitolo 2

Architettura di Calcolo Distribuito

2.1 Datacenter

Il **Datacenter** è una struttura composta da calcolatori che comunicano in rete, che aziende ed altre organizzazioni usano per organizzare, processare, memorizzare e disseminare enormi ammontare di dati. Essi rappresentano il cuore del business di un'organizzazione ed è necessario che soddisfino caratteristiche come scalabilità, sicurezza ed affidabilità oltre a possedere tutta l'attrezzatura necessaria come ad esempio la ventilazione, gruppi di continuità, sistemi di raffreddamento e naturalmente lo spazio necessario per installare il tutto. L'infrastruttura di un datacenter è cambiata molto durante gli anni e possiamo classificarla in tre categorie:

Infrastruttura Legacy: In questa disposizione l'hardware è formato da tante commodity machines ma con l'obbligo di dover gestire ogni macchina separatamente ovvero si è costretti a dover installare su ogni macchina il sistema operativo, i programmi e le librerie necessarie. Da un primo impatto si può facilmente capire che con un numero molto grande di macchine la cosa risulta praticamente ingestibile e infatti questo modello è stato subito sostituito a favore delle architetture convergenti.

Infrastruttura Convergente: In questa disposizione l'hardware è formato da enormi centri di calcolo che a differenza della struttura legacy sono gestiti tramite l'ausilio di un hypervisor quindi favorendo la gestione

delle macchine ma con dei limiti dal punto di vista delle prestazioni in quanto la comunicazione con i dispositivi di memorizzazione avviene tramite una SAN[1] che rappresenta un collo di bottiglia nell'accedere e scrivere ai dati su disco.

Infrastruttura Iperconvergente: Rappresenta la soluzione moderna per offrire un centro di calcolo performante. Questa architettura riprende il meglio delle due precedenti in quanto fa uso di commodity hardware, che costa poco, utilizza un hypervisor per gestire il datacenter ma la memorizzazione dei dati non avviene tramite una SAN. Quello che accade è che ogni rack possiede un insieme di piani che permette di aggiungere una motherboard formata da cpu, ram, e disco in modo che i dati delle singole motherboard possono accedere in maniera locale ai propri dischi aumentando le prestazioni e inoltre favorisce la scalabilità in quanto basta aggiungere al rack nuove motherboard quando è strettamente necessario.

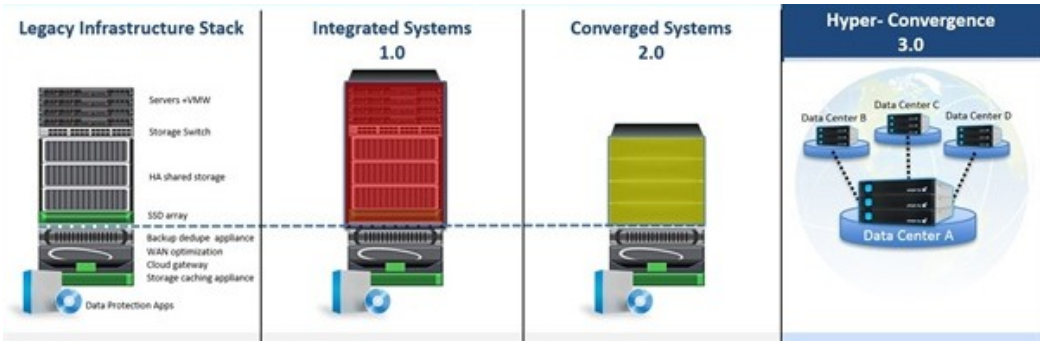

Figura 2.1: Le tre architetture

2.2 Hypervisor

Ogni commodity hardware che forma il moderno datacenter non si limita all'esecuzione di un unico sistema operativo in quanto questo rappresenterebbe uno spreco di risorse di memoria e di CPU che possono essede dedicate per

[1]Storage Area Network

eseguire altro calcolo. La virtualizzazione è stata la tecnologia fondamentale che ha permesso alle architetture dei datacenter di evolversi da legacy a convergenti. L'hypervisor rappresenta il componente software o hardware necessario per ottenere virtualizzazione e quando Bugnon nel 97 presentò Disco[2] e il VMM (sta per virtual machine monitor ed è un sinonimo di hypervisor) nessuno avrebbe mai pensato che la sua intuizione lo avrebbe portato al successo fondando VMware.

2.3 Gestione dello storage

Essendo che un datacenter deve processare e memorizzare ordini di petabyte, quindi che non entrano in un disco commerciale, è stato necessario trovare delle soluzioni adeguate. La storia ci insegna che inizialmente i dati venivano memorizzati in enormi dischi rigidi (ricordiamo ad esempio SLED di IBM) che non solo costavano tantissimo ma occupavano tanto spazio con altezze che arrivavano addirittura a 14 pollici. La soluzione è arrivata nel 1988 grazie a Patterson e alla sua architettura **RAID** che ha evidenziato con i suoi esperimenti[3] che è possibile ottenere, mettendo insieme un array di dischi economici con appositi controller, più memoria, migliori prestazioni e tolleranza ai guasti.

2.3.1 RAID

L'acronimo sta per *Redundant Array of Inexpensive Disks* e rappresenta l'innovazione che ha permesso di raggruppare diversi dischi dando la sensazione al sistema che possano essere utilizzati come se fossero un unico volume. L'implementazione del RAID può essere effettuata o hardware o software dove nel primo caso si usano controller hardware fatti a hoc molto costosi (più di tutti i dischi messi assieme), nel secondo caso il tutto viene gestito dal sistema operativo con un normale controller (che può essere SATA, ATI, SCSI, in fibra). Il tipo di array viene identificato dal livello RAID che determina il numero di dischi minimo necessario per poter essere configurato. La caratteristica fondamentale di questa tecnologia è quella della ridondanza che permette di individuare e correggere errori ed è ottenuta con differenti tecniche che variano a seconda del livello di RAID utilizzato:

RAID 0: Questo livello non possiede ridondanza e utilizza lo **striping** (unità minima in cui viene diviso ogni file per la scrittura) per distribuire i

file nei dischi facendo si che le letture e scritture avvengano in parallelo. Ha come svantaggio naturalmente la perdita totale dei dati in caso di rottura del disco.

RAID 1: Questo livello dedica fa sì che alcuni dischi vengano usati come copia per i dati in modo da intervenire in caso di guasto. Dal punto di vista delle prestazioni siamo pari a quelle di un singolo disco ma è presente la fault tolerance.

RAID 2: Questo livello presenta delle caratteristiche simile al livello 1 ma con l'aggiunta di codici di correzione ECC dei dati. Questa configurazione è caduta in disuso a causa del fatto che ora i dischi attuali implementano di suo questa tipologia di correzione.

RAID 3: Questo livello utilizza sia lo striping che il controllo della parità. Lo striping viene applicato a livello di segmenti e la parità mantiene le informazioni necessarie per poter recuperare i dati persi. In questo livello le scritture peggiorano poichè ad ogni scrittura si affianca il calcolo della parità e inolre la scrittura di essa avviene in un unico disco causando un collo di bottiglia sulle prestazioni totali.

RAID 4: Questo livello utilizza le stesse funzionalità del livello 3 con la differenza che lo striping non viene effettuato con i segmenti ma a livello di blocchi.

RAID 5: Questo livello utilizza le stesse funzionalità del livello 4 ma con la differenza che in questa configrazione non esiste un unico disco per la scrittura della parità ma su tutti vengono scritti dati o calcolo di parità (da notare che la parità non viene scritta sullo stesso disco dei dati). RAID 5 ha delle buone prestazioni che tendono a migliorare con l'aumento dei dischi installati.

RAID 6: Questo livello ha le medesime caratteristiche del livello 5 con la differenza che effettua un doppio calcolo della parità (tramite codici di Solomon). Le prestazioni sono le medesime di RAID 5 con la presenza di una ridondanza aggiuntiva dei dati di controllo a causa della parità.

RAID Annidati: Sono combinazioni di configurazioni di RAID permettendo così di accorpare le caratteristiche dei livelli. Le due combinazioni più diffuse sono la 01 e la 10. La prima prende due configurazioni RAID

0 e le combina in un RAID 1 e questo comporta che ogni gruppo di dischi conterrà la copia speculare dell'altro gruppo. Il secondo prende gruppi di dischi in RAID 1 e si combinano in RAID 0 permettendo così di vedere il tutto come se fosse un unico disco e inoltre permette la tolleranza del guasto di sue dischi.

2.3.2 Software Defined Storage

Ceph è una piattaforma di storage distribuito creata da Red Hat recentemente (2016) e rappresenta quella che potrebbe essere in futuro una valida alternativa a RAID per gestire la memorizzazione dei dati. Questa piattaforma fornisce intefacce di storage di diverse tipologie (a oggetti, a file) quindi permettendo una grande flessibilità ma soprattutto riesce a scalare nell'ordine degli exabyte. La replicazione dei dati avviene al livello software rendendo così Ceph una piattafroma indipendente dall'hardware. La ragione del suo successo risiede nel fatto che sia possibile accedere ai dati in maniera completamente trasparente e diversificata permettendo di adattarsi alle esigenze delle organizzazioni. Un'altra caratteristica a fvore è che un sistema Ceph può essere costruito con commodity hardware riducendo di molto i costi a discapito di un controller RAID hardware.

2.4 Sistema Operativo

Qui si parla dell'importanza del sistema operativo in ambiente distribuito.

2.4.1 Kernel

Qui si parla delle varie architetture kernel.

2.5 File System

In un data-center è indispensabile avere un filesystem che sia trasparente e che nasconda all'utente che usufruisce del cluster l'effettiva locazione dei file sui dischi. L'informatica ha conosciuto due diverse filosofie di implementazioni di filesystem su reti: I primi tentativi concreti sono stati ottenuti con l'introduzione del protocollo **SMB** (Server Message Block) ed è ancora tuttora utilizzato da Windows e da Linux (Samba), successivamente negli verso

la metà degli anni 80 alla Carnagie Mellow University venne prototipizzato **AFS** (Andrew File System) che nonostante le implementazioni artificiose che i ricercatori dovettero fare per sistemare alcuni problemi di scalabilità e gestione dei fault, fu la base per la nascita negli anni 90 di CODA. Una prospettiva completamente ortoginale ad AFS è stata NFS ideato da Sun che.

2.5.1 AFS

2.5.2 NFS

2.5.3 Coda

2.5.4 GFS

Capitolo 3

Calcolo Distribuito con Hadoop e Map-Reduce

In questo capitolo si presenteranno i concetti e il funzionamento su cui si basa il framework Hadoop. Il capitolo si concentrerà inizialmente su una breve ricapitolazione della storia del framework, le sue origini e le cause del suo successo. Successivamente si analizzeranno nel dettaglio le sue componenti chiave ovvero HDFS e YARN e infine si introdurrà il modello Map Reduce spiegandone il funzionamento.

3.1 Storia di Hadoop

Il framework Hadoop fu creato da **Doug Cutting**, creatore di **Apache Lucene**, la libreria più utilizzata per quanto riguarda la ricerca di tipo testuale. Questo framework fonda le sue radici da **Apache Nutch**, una motore di ricerca per il web open source a sua volta parte integrante del progetto Lucene. Scrivere un intero motore di ricerca da zero era un obiettivo molto ambizioso. Il progetto Nutch partì nel 2002 ed ebbe un buon successo tuttavia i creatori si resero immediatamente conto che la loro architettura non sarebbe stata in grado di scalare a sufficienza a causa dell'alto numero di pagine web da indicizzare (già allora ce ne erano più di un miliardo). Nel 2003 Google pubblicava il paper in cui introduceva il GFS e Nutch intuì che questo filesystem avrebbe risolto i problemi di memorizzazione e amministrazione dei dati cheavevano e decisero di implementarne una versione open source che venne chiamata NDFS (Nutch Distributed File System). Un anno dopo smepre

Google pubblicò il paper che introdusse il paradigma Map-Reduce e ancora una volta il creatore capì che questa idea lo avrebbe aiutato a risolvere i problemi di Apache Nutch e nel 2005 il suo team aveva già implementato una sua versione open source e nel giro di 6 mesi tutti gli algoritmi del motore di ricerca furono adattati per essere eseguiti su NDFS e usando Map-Reduce. Questa "accoppiata" ebbe così tanto successo che nel Febbraio 2006 Nutch decise di spostarlo in un progetto indipendente chiamato Hadoop e nello stesso periodo Doug Cutting si unì a Yahoo! che forni un team dedicato e risorse finanziarie che trasformare Hadoop in un sistema che potesse essere eseguito per scalare anche sul web e ci riuscirono nel 2008 quando l'azienda annunciò che il suo indice di ricerca era stato generato da un cluster Hadopo da 10000 core. Nel Gennaio 2008 Hadoop divenne il progetto di punta della fondazione Apache ed è tuttora utilizzato da grandi compagnie come Facebook, New York Times e finnziato dai big dell'informatica come IBM, Microsoft e dalla stessa Google.

3.2 Architettura di Hadoop

qui si parla in breve dell'architettura di hadoop

3.2.1 HDFS

HDFS è il filesystem distribuito che utilizza il framework Hadoop per processare i dati. I punti di forza su cui si basa sono i seguenti:

Memorizzazione di grandi file: Dove in questo contesto indichiamo file che sono centinaia di megabyte, gigabyte o terabyte.

Accesso dati: HDFS è costruito intorno all'idea che il migiore pattern per processare i dati è "una scrittura e tante letture". Un dataset è tipicamente generato o copiato da una sorgente e successivamente varie analisi sono effettuate sul dataset nel tempo. Ogni analisi richiede grandi porzioni se non l'intero dataset quindi il tempo per leggere l'intero dataset è più importante della latenza di leggere il primo record.

Commodity Hardware: Hadoop non richiede hardware costoso. È strutturato per essere eseguito su cluster basato su commodity hardware (hardware comunemente disponibile che può essere ottenuto da diversi fornitori).

Blocchi

Sappiamo che i dischi hanno una *block-size* che rappresenta l'ammontare minimo di dati che possono leggere o scrivere. Di solito la grandezza di questi blocchi è nell'ordine dei kilobyte ma in HDFS questa grandezza è pari a 128MB e una caratteristica importante è che file più piccoli del blocco non occupano quest'ultimo interamente (ad esempio se avessimo un file da 1MB esso utilizzerà 1MB di disco e non 128). Il motivo per cui questi blocchi sono così grandi è per minimizzare i tempi di ricerca nel disco. Avere questa astrazioni a blocchi permette di memorizzare file più grandi di un singolo disco, e inoltre semplifica la gestione della memorizzazione dei dati in quanto avere blocchi di taglia fissa permette di calcolare facilmente quanti ne possono essere memorizzati du di un disco ed elimina la gestione dei metadati (che vengono gestiti a parte).

Namenode e Datanode

Un cluster HDFS ha due tipologie di nodi che segue il paradigma master-slave: un **namenode** (master) ed un numero di **datanode** (slave). Il primo gestisce il namespace del filesystem e i metadati per tutte le direcrory e i file e memorizza queste informazioni sul proprio disco locale oltre a conoscere locazione dei blocchi di ogni file. I secondi invece memorizzano e processano i dati riportando periodicamente al namenode quale lista di blocchi memorizzano. La perdita del namenode comporta l'inutilizzabilità del filesystem ed è quindi importante avere un *secondary-namenode* che periodicamente si sincronizza con il primary per fungere da backup in caso di guasti o malfunzionamento.

3.2.2 Yarn

Apache YARN (Yet Another Resource Negotiator) è il sistema della gestione delle risorse in Hadoop. Introdotto dalla versione 2 di Hadoop ha migliorato di molto le prestazioni del framework. Yarn fornisce i suoi servizi attraverso due processi manager, un *resource-manager* (uno per cluster) per gestire l'uso delle risorse e un *node-manager* che è eseguito su tutti i nodi nel cluster per lanciare e monitorare i container (Esegue i task con un insieme ristretto di risorse). Per eseguire un applicativo su YARN il client contatta il resource-manager e gli chiede di lanciare un *application-master*.

3.3 Paradigma Map-Reduce

3.3.1 Map

Qui si parla del metodo map

3.3.2 Reduce

qui si parla del metodo reduce

3.4 Come funziona un Job Map Reduce

Capitolo 4

Utilizzo di Hadoop per Object Detection

4.1 Descrizione del problema

Viene analizzato il problema

4.2 Versione Sequenziale

Descrizione della risoluzione del problema in ambiente shared memory.

4.3 Versione Distibuita con Hadoop

Descrizione della versione distribuita con Hadoop

4.4 Analisi delle prestazioni

Vengono forniti i benchmark con rispettive spiegazioni

Bibliografia

[1] G.Cattaneo, *The basic Hardware & software to medium grain parallelism*, 1988.

[2] E.Bugnion, S.Devine, K.Govil, M.Rosenblum,
Disco: Running Commodity Operating Systems on Scalable Multiprocessors, 1997

[3] D.Patterson, G.Gibson, K.H.Randy, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, 1988.

[4] David G. Lowe
Distinctive Image Features from Scale-Invariant Keypoints, 2004.