6 Big Data and internet of things

28/5
Registrazione 354

Accenneremo alcune tecnologie per analizzare i big data. Innanzitutto dobbiamo capire cosa sono i BD. IL corso si è comunque fondato sull'analisi dei piccoli dati. Se abbiamo dati dell'ordine dei byte si possono usare i metodi fino ad ora utilizzati, dati dell'ordine dei terabyte possono essere analizzati con SQL e la teoria relazionale. Per dati più grandi, dell'ordine dei petabyte, 10^{15} byte, non basta neanche il modello relazionale. fondamentalmente dobbiamo trovare forse alternative per archiviare e fare le analisi.

Per giungere al concetto di BD la storia è passata per 4 fasi:

1. Nascita del web (1989), che permette a tutti o quasi a tutti di crearsi la pagina web e di mettere in rete una gran quantità di dati di tutti i tipi.
2. L'arrivo dei social network (FB nel 2004, twitter 2006), anche questa è stata una spinta ulteriore a mettere dati sul sistema.
3. L'introduzione degli smartphone (prima generazione dell'iPhone nel 2007). Il fatto di non avere più semplicemente dei pc sulla scrivania, ma anche in tasca ha facilitato l'every where e l'every time della condivisione.
4. L'avvento dell'internet delle cose (Internet of Things), cioè l'introduzione nelle cose di piccoli chip che hanno dei sensori e sono connessi in rete e immettono continuamente dati nella rete. Si pensi ad esempio alla domotica, in cui alcuni oggetti (tapparelle, caldaia, ecc.) possono essere comandati tramite smartphone e cose del genere. Stiamo andando verso un modo in cui non solo le pagine web saranno connesse tra loro ma anche gli oggetti e questo causerà un ennesimo salto verso la condivisione dei dati.

Video - hadoop software , oracle big data, -

Consiglio di leggere rapporto McKinsey (o sennò la sintesi della sintesi su Forbes https://www.forbes.com/sites/louiscolumbus/2016/12/18/mckinseys-2016-analytics-study-defines-the-future-machine-learning/#5ef347d214ebt)

Vedremo 3 cose:

1. Rendere efficiente il codice R: gli algoritmi vanno raffinati per ottimizzare il tempo di processione,
2. Vedremo 3 tecniche di split, apply and combine (map reduce): il problema si spezzetta, ogni pezzo si risolve indipendentemente dagli altri (apply) infine si ricombinano i pezzi. I pacchetti saranno bigmemory, Itools? E sparklire?

6.1 Big data

Entriamo nell'ultima parte di questo corso, accenneremo alcune tecniche per analizzare i **Big Data**. Se non sappiamo analizzare dati piccoli sicuramente non sapremo analizzare dati grandi. Se abbiamo dati che vanno oltre i terabyte allora la base di dati relazionale non è più utilizzabile, ci servirebbe un supporto molto grande. Dobbiamo trovare vie alternative per archiviare il nostro dataset e per fare le analisi. Come mai ci siamo riempiti di big data? 4 passaggi che ci hanno portato ai Big Data: la nascita del web nel 1989, più recentemente c'è l'arrivo dei social network (facebook nel 2004), dopodiché nel 2007 c'è l'arrivo

dei primi smartphone ed infine c'è l'avvento dell'internet delle cose (introduzione nelle cose di piccoli computer che hanno dei sensori e sentono l'ambiente circostante, immettendo continuamente dati nella rete). I Big Data sono dei dataset la cui dimensione va oltre la dimensione tipica dei database. Una delle tecniche per analizzare i big data consiste nel suddividere grossi dati in molte porzioni e poi analizzarli con le tecniche degli small data.

I Big Data sono caratterizzati dalle **3 V**:

- **Volume**: si parla di Big Data quando i dati sono sufficientemente grandi da non poter essere rappresentati da una base di dati
- **Velocità**: questi dati arrivano molto velocemente, quindi il tempo di elaborazione deve essere molto basso. Abbiamo bisogno di metodi efficienti per la loro analisi
- **Varietà**: questi dati sono spesso vari, cioè includono
    - dati strutturati: sono quelli che sono definiti in un modello come quello relazionale
    - dati semi strutturati: testo che è etichettato con dei tag che danno semantica agli elementi)
    - dati non strutturati: dati che non hanno una struttura, come ad esempio il testo ma anche audio, video e immagini)

Ci sono anche dei problemi associati a questa proliferazione di dati:

- Privacy: per esempio i dati relativi alla cartella clinica o al conto corrente sono privati e quindi si vogliono proteggere, d'altra parte sono proprio i dati sensibili che possono essere più utili alla persona stessa. Se la cartella clinica fosse monitorata e analizzata chiaramente questo sarebbe un grosso vantaggio per tutti. (Quindi c'è un trade-off tra privacy e utilità)
- Security: vogliamo proteggere i dati da intrusioni altrui, quindi è connesso all'aspetto della privacy, vogliamo proteggere i dati da intrusioni altrui, anche se abbiamo concesso all'ospedale i dati della nostra cartella clinica, questo non vuol dire che l'ospedale può metterli sul web, deve tenerli ben sicuri e non cederli
- Legal issues (aspetti legali): un dataset può essere copiato facilmente, più persone possono utilizzare un dataset (cosa che non avviene per un bene fisico, ad esempio non posso duplicare esattamente la Gioconda). Se abbiamo costruito un dataset come facciamo a dire che lo abbiamo costruito noi? Questioni legali che sono legate al fatto che un dato può essere replicato con grande facilità e può essere utilizzato da più persone contemporaneamente

È bene affrontare prima i dati piccoli e poi quelli grossi. Ci son due buoni motivi per cui i nostri BD sono dei small data camuffati:

1. Potrebbe essere che abbiamo un DSet molto ampio, ma ci bastano pochi dati di questo insieme di dati per fare la nostra analisi. Quindi possiamo estrapolare i dati necessari per la nostra challenge. Quindi possiamo trovare un campione o un sottoinsieme

2. Oppure il nostro DSet può essere effettivamente grande e non può essere compresso in nessun modo, ma può essere suddiviso in tanti piccoli pezzi e ogni pezzo può essere analizzato separatamente. Quindi se abbiamo un DSet di 10 terabyte lo si divide in 100 parti, quindi ogni parte contiene 100 gb e questi possono essere utilizzati sul disco fisso e quindi possiamo archiviarlo in una base di dati. Questa tecnica è chiamata ***map reduce*** o ***split apply and combine***. Se il nostro obiettivo è calcolare il massimo, si divide il BD in gruppi, si calcola il massimo locale e poi il massimo complessivo. Non sempre è fattibile questa procedura, dipende se la procedura è parallelizzabile. Un esempio di operazione non parallelizzabile è la mediana.

Vedremo tre approcci:

1. la prima cosa che vedremo sono delle tecniche per rendere efficiente il nostro codice R, perché quando lavoriamo con big data abbiamo problemi di tempo. Per questo gli algoritmi usati devono essere molto efficienti.
2. **Fase di split**: alcune volte il nostro problema si può spezzettare, e ogni pezzo può essere analizzato e alla fine tutto può esser messo assieme
3. combine: mette insieme i risultati parziali. Pacchetti in R *BigMemory*, Itools,.., *Spark*.

Sintesi su come trattare database di grandi dimensioni (Big Data)

- dobbiamo avere:
  - macchine performanti,
  - software aggiornati
  - algoritmi furbi ed ottimizzati.
- Applicare poi la tecnica dividi e conquista dividendo il dataset in parti che si possono maneggiare nella memoria centrale della macchina (al massimo qlc Gigabyte) applicando ad ogni parte il metodo di calcolo, anche in *calcolo parallelo* se abbiamo una macchina con più processori o un cluster, infine si riducono i risultati locali ad un risultato globale.

Questa è la strategia più comune per affrontare un problemi di grosse dimensioni.

Fare su Datacamp - Writing efficient R code (solo video1° , 4° , 5°? video)

Pacchetto Microbenchmark

serveper cfr 2 alternative; parametro times dice quante volte devo ripetere la lettura dei file. Morale dell'esercizietto: conviene salvare un file in formato RDS, perché 10 volte più efficiente dell'altro.

Datacampvideo "Writing efficient R code",
Datacamp video "Scalable data processing in R"
Datacamp video "Introduction to Spark in R using sparklyr"

Pacchetto parallel (Pacchetto parallel)

Parallelizzare le operazioni significa suddividere un compito a più macchine, e ha un costo: come quando si deve fare in due un lavoro, ha un indubbio vantaggio e risparmio di tempo, ma i due devono anche integrarsi. In generale non è detto che il risultato venga meglio se si lavora assieme, idem per il computer. Chiamiamo ***overhead*** l'operazione di divisione dei compiti e la aggregazione dei risultati: se l'overhead è superiore al vantaggio che ottengo parallelizzando, allora la parallelizzazione non è stata una buona idea.

datacamp Scalable data processing in R
2 videolezioni

Pacchetto Iotools

Implementa in modo efficiente la lettura della memoria secondaria del pc, che solitamente è un'operazione molto lenta. *readAsRaw* legge byte dopo byte e solo dopo converte i file in formato stringa, booleano o altro. Le funzioni che useremo leggeranno i dati byte dopo byte e faranno come detto sopra.

Pacchetto Spark

Datacamp con introduzione a sparklyr
installare java 8 e spark

Useremo la versione locale, in modalità pseudo distribuita. In realtà la Spark è una tecnologia che permette di condividere dati e computazioni personali su altre macchine sparse sul pianeta (cluster computer). L'interfaccia e interrogazione di dplyr e di SQL qui ritornano utili per interrogare i dataset.
Dopo aver applicato la tecnica delle 3 fasi di prima, ora possiamo affidare la computazione di questi pezzi a diverse macchine.
Simuleremo un cluster sulla nostra macchina. In ordine ci si connette, si fanno i calcoli e poi ci si disconnette.
Rstudio ha una interfaccia dedicata a Spark (in alto sx) ➔ problemi ad installare, quindi guardo la demo.
Video per impostare una variabile come *variabile ambiante* e java home su windows 10.

È buona norma minimizzare il trasferimento da R a Spark (noi spostiamo flights), perché è molto lento.
L'ambiente in alto a sx Spark è il cluster della Sissa (x es, facciamo finta) con il comando seguente

```
flights_tbl <- copy_to(sc, nycflights13::flights, "flights")
```

Quando vedrò le prime righe di un database che sta sulla memoria dei server della Sissa, solo queste poche righe stanno sulla memoria del mio pc.

`dim(flights_tbl)` ➔ mi ricorda che non ho i dati in memoria
`flighstable` → è solo un puntatore\riferimento di qlc che sta in Sissa

Proviamo ad interrogare il DBin Sissa,
Se mi chiedo quel'è la classe della variabile output, scopro che è un "tbl_spark" cioè non stanno sulla mia memoria. CI sono alcune query non valide, come la funzione *median* (che non è dispnbile in SQL)

Quindi ***COPY TO*** muove dati da R a SPARK
Il viceversa è la funzione COLLECT

```
collected_flight_delay <- flight_delay %>%
  collect()
```

Ora i dati sono in memoria e posso graficarli (prima non potevo)

```
ggplot(collected_flight_delay, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth() +
  scale_size_area(max_size = 2)
```

Posso ogni tanto spezzare la pipeline in risultati intermedi. Ma come fare in remoto? Con la funzione ***compute***:

```
computed_flight_delay <- flight_delay %>%
  compute("flight_deleay")
```

crea una tabella nell'ambiente remoto

Così eseguo la query in SQL sul db remoto in Spark:
`df <- dbGetQuery(sc, query)`

a differenza di dyplyr che mantiene I dati in remote, l'esecuzione di SQL trasferisce I dati da remoto alla mia macchina perché li salva in R! Se non lo voglio posso usare *dbSendQuery* e poi recupero il risultato con *dbFetch()*.

`sparklyr also has two **native interfaces*`
Possiamo usare interfacce inattive, lavorando direttamente sul cluster e non sulla macchina mia

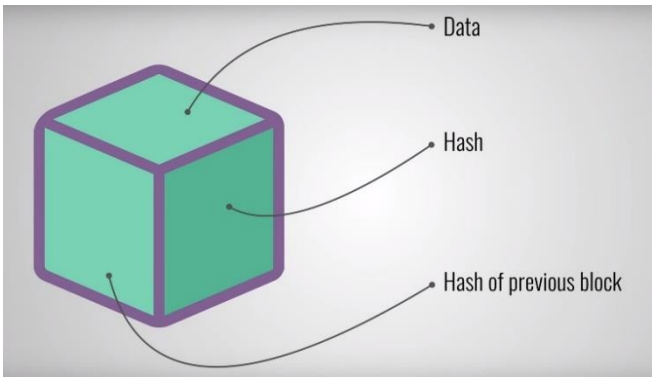
Appunti 29/5
registrazione 358

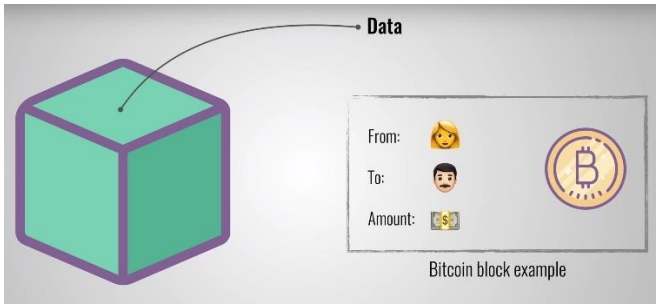
6.1.8 Blockchain

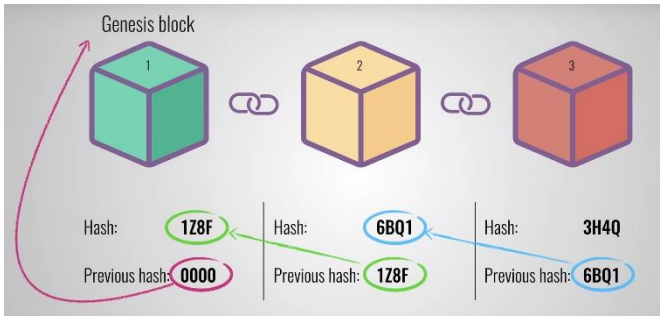
La Blockchain Una delle reti P2P più famose al momento. È un esempio di base di dati distribuita che sfrutta la computazione distribuita: mette insieme le basi di dati e il paradigma della distributed computing. Inoltre la BC utilizza metodi criptografici per rendere sicure le transazioni. È nata per le criptovalute, da Nakamoto con un white paper. Lo scopo era risolvere il *double spending problem*, cioè evitare la possibilità di spendere la stessa moneta due volte. BC è un **libromastro digitale**, idea già presente nel 1800. Dal pdv tecnologico è interessante perché mette insieme diverse cose: base di dati, computazione distribuita, le reti (tra pari) e crittografia (funzioni hash).
Link su YouTube da guardare (a parte il video di 2 h sono 70 minuti di video)

- How does a blockchain work

- What is blockchain?
- Blockchain explained
- Blockchain technology explained
- The Bitcoin blockchain explained
- How the blockchain will radically transform the economy
- L'evoluzione della fiducia
- Smart contracts
- Proof-of-Stake
- Proof of Work vs Proof of Stake
- Blockchains: how can they be used?
- Asymmetric encryption
- ERC20 tokens
- Cardano







In R-Studio:

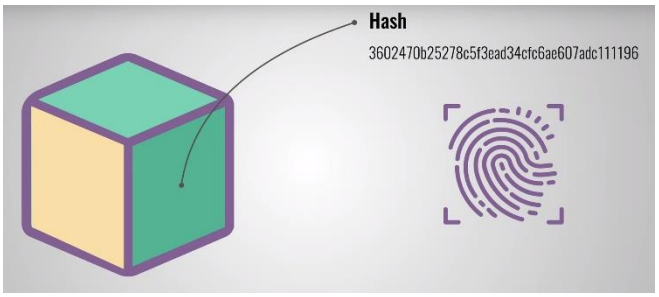
http://users.dimi.uniud.it/~massimo.franceschet/ds/r4ds/syllabus/learn/blockchain/blockchain.html

Un **blocco** è la una lista di alcuni elementi. Più blocchi formato una **catena**.

```
block_example <- list(index = 3, # numero del blocco
                      timestamp = "2018-05-09 19:48:18 CEST",
#data e h immissione blocco nella catena
                      data = "Move 1.5 ether from A to B",
# dati\contenuto: qui una transazione dunque: tender, receiver e amount
                      previous_hash = "310a5f2a1fa3bc815ebc11a717c5b8415d07570956
934c25e57463b69919b1cf",
# codice hash blocco precedente
                      proof = 198, # numero corrispondente alla Proof of Work
                      new_hash = "e94576baba834b7e57798e114436dec02c3c44561374405
d995be7635181aab2")
#codice hash del blocco 3
```

Hash

Le funzioni hash sono funzioni di dati. Qui abbiamo funzioni hash crittografiche: sono algoritmi matematici che sfruttano la teoria dei numeri che, dato un insieme di dati di dimensione arbitraria, servono a creare una sorta di riassunto (*digest*). Noi useremo un digest di 256 bit.



Lo hash è l'impronta digitale di un blocco, intesa come stringa con tutte le sue informazioni, quindi dato un oggetto in input la funzione hash restituisce una stringa.

La particolarità della funzione è che mentre la complessità della funziona è bassa, quella della funzione inversa è alta: dato un oggetto in input e la funzione hash è facile calcolare la stringa finale, ma data la stringa finale e la funzione inversa non riesco a calcolare l'input, come se la funzione non fosse invertibile (*one way function*), ma in realtà è invertibile con un costo computazionale molto elevato rispetto alla funzione originaria. L'unico modo per calcolare lo hash è usando la forza bruta (come trovare bancomat e cercare di prelevare i soldi provando tutte le combinazioni del pin).

La libreria *digest* implementa metodi crittografici (algoritmo SHA-2 , che noi useremo, ma oggi sta per essere soppiantanto da SHA-512 che usa 512 bit).

```
# Creates hash digests of arbitrary R objects
library(digest)


# add hash value to the current block
hash_block <- function(block)


# hash blocco corrente: passiamo alla funzione una stringa formata da oggetti co
ncatenati: indice, timestamp, dati e (soprattutto) lo hash del blocco precedente


 block$new_hash <- digest(c(block$index,
                            block$timestamp,
                            block$data,
                            block$previous_hash,
                            block$nonce), "sha256")
  return(block)
}
```

Se voglio manomettere lo hash di un certo blocco, perché sono un **farabutto**, devo anche ricalcolare tutti gli hash dei blocchi successivi al blocco con hash manomesso, in quanto gli hash seguenti saranno tutti invalidi. (un esempio è ricevere per errore 2 bitcoin anziché uno, ciò implica rifare tutti gli hash seguenti)

Proof-of-Work (pow)

Lo hash non è l'unico elemento di sicurezza della blockchain. L'introduzione così come la modifica di un nuovo blocco nella catena è stato voluto difficile e costoso per motivi di sicurezza. Ad esempio, nei bitcon i blocchi corrispondono alle monete bitcoin. Una Proof-of-Work è un algoritmo che controlla la difficoltà di inserire un nuovo blocco, si dice che è un algoritmo *parametrico rispetto alla difficoltà*: ciò perché nuove macchine potrebbero risolvere più facilmente i problemi crittografici, così l'algoritmo si complica all'aumentare della potenza di calcolo.

Vediamo il codice: dice che va cercato un numero proof divisibile sia per 99 che per il precedente proof, facciamo un ciclo, partendo da proof =1. Di per se i primi numeri sono facili da trovare ma in termini della dimensione dell'inut il tempo di esecuzione cresce in modo esponenziale all'aumentare die blocchi.

```
### Simple Proof of Work algorithm


proof_of_work <- function(last_proof){
  proof <- last_proof + 1
    # afficnhé sia divisibile per last_proof parto da quello +1.
    # Sotto: incremento proof fino a che non trovo un numero divisibile per 99 e
per last proof (dato)
  while (!(proof %% 99 == 0 & proof %% last_proof == 0 )){
    proof <- proof + 1
  }
   return(proof)
}
```

```
## $hash
## [1] "0006d041d3fb2127ec60c9a47a3a818a8335fab330998516625dcebbb8447a1d"
##
## $nonce
## [1] 2562
```

Questa proof-of-Work è facilmente manomissibile, non è un buon esempio. Quello che viene utilizzato realmente è: dato un blocco, trovare la codifica hash di un certo numero di *digit* iniziali; ciò ha difficoltà esponenziale in funzione della difficulty assegnata.

In sintesi: creare nuovi blocchi è difficile per 2 motivi:

- per la ricodifica gli hash dei blocchi successivi
- perché va rifatta la proof-of-Work per tutti i blocchi seguenti

Building the blockchain

Ora sai come appare un blocco, come i blocchi sono incatenati usando gli hash e come il ritmo di creazione di nuovi blocchi viene regolato dai PoW. Vediamo una funzione che genera un nuovo blocco.

```
gen_new_block <- function(previous_block){

  #Create new Block
  new_block <- list(index = previous_block$index + 1,
                    timestamp = Sys.time(),   # tempo corrente
                    data = paste0("this is block ", previous_block$index + 1),

# dati del blocco corrente ,
```

```
                    previous_hash = previous_block$new_hash, #hash blocco preced
                    proof = proof_of_work(previous_block$proof))

 # proof è n° uscito da mining del blocco precedente: per creare nuovo bloco dev
o fare un mining dl blocco precedente, il primo miner che lo risolve crea nuovo
blocco e lo distribuisce a tutti i nodi della rete che ne verificano la corrette
zza (almeno 50% votanti)

  # Add the hash of the current block
  new_block_hashed <- hash_block(new_block)

    return(new_block_hashed)
}
```

Affinché un nuovo blocco sia aggiunto alla rete almeno 50% dei nodi della rete devono concordare sul fatto che l'hash del nuovo blocco possiede l'hash del blocco precedente.

La blockchain può dunque fallire se:

- manometto blocco ricalcolando tutti gli hash dei blocchi successivi,
- calcolare tutte le proof-of work dei blocchi successivi
- convincere almeno 50% dei miners

Il primo blocco della rete è ***blocco genesi***, che è il primo blocco della catena ed ha hash precedente 0.

```
# Define Genesis Block (index 1 and arbitrary previous hash)
block_genesis <-  list(index = 1,
                       timestamp = Sys.time(),
                       data = "Genesis Block",
                       previous_hash = "0",
                       proof = 1)

block_genesis <- hash_block(block_genesis)
```

Aggiungo 5 nuovi blocchi che andranno nella blockchain (list)

```
# First block is the genesis block
blockchain <- list(block_genesis)
previous_block <- block_genesis  # riassegno

# How many blocks should we add to the chain after the genesis block
num_of_blocks_to_add <- 5

# Add blocks to the chain
for (i in 1: num_of_blocks_to_add){
  block_to_add <- gen_new_block(previous_block)  #funzione(blocco precedente)
  blockchain[[i+1]] <- block_to_add
  previous_block <- block_to_add
}

blockchain
```

```
## [[1]]
## [[1]]$index
## [1] 1
```

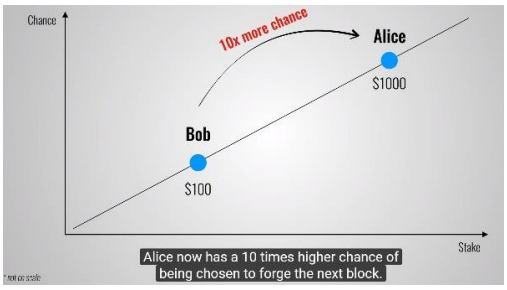
```
##
## [[1]]$timestamp
## [1] "2018-05-29 19:36:45 CEST"
##
## [[1]]$data
## [1] "Genesis Block"
##
## [[1]]$previous_hash
## [1] "0"
##
## [[1]]$nonce
## [1] 0
##
## [[1]]$new_hash
## [1] "e2b87bf6b2ba5bfa298081b401bb49e06659ad853f1e5f22c77791f4608d31e1"
##
##
## [[2]]
## [[2]]$index
## [1] 2
##
## [[2]]$timestamp
## [1] "2018-05-29 19:36:45 CEST"
##
## [[2]]$data
## [1] "this is block 2"
##
## [[2]]$previous_hash
## [1] "e2b87bf6b2ba5bfa298081b401bb49e06659ad853f1e5f22c77791f4608d31e1"
##
## [[2]]$nonce
## [1] 1752
##
## [[2]]$new_hash
## [1] "ebe0d116e4651cb6e3c09016e8361dcb0340296dfff5995889861eb8e7730037"
##
##
## [[3]]
## [[3]]$index
## [1] 3
##
## [[3]]$timestamp
## [1] "2018-05-29 19:36:45 CEST"
##
## [[3]]$data
## [1] "this is block 3"
##
## [[3]]$previous_hash
## [1] "ebe0d116e4651cb6e3c09016e8361dcb0340296dfff5995889861eb8e7730037"
##
## [[3]]$nonce
## [1] 6594
##
## [[3]]$new_hash
## [1] "ae339ea3436de0339d021de462c65af49e5d83eeb60995f18623e1e51fc686fd"
##
##
## [[4]]
## [[4]]$index
## [1] 4
##
## [[4]]$timestamp
## [1] "2018-05-29 19:36:45 CEST"
##
## [[4]]$data
## [1] "this is block 4"
##
## [[4]]$previous_hash
## [1] "ae339ea3436de0339d021de462c65af49e5d83eeb60995f18623e1e51fc686fd"
##
```

```
## [[4]]$nonce
## [1] 14442
##
## [[4]]$new_hash
## [1] "a0ecdb2bdcb5ab7eea825e4ee41d729e77c9ba26c05f4166f2cf1f864cc94e3a"
##
##
## [[5]]
## [[5]]$index
## [1] 5
##
## [[5]]$timestamp
## [1] "2018-05-29 19:36:46 CEST"
##
## [[5]]$data
## [1] "this is block 5"
##
## [[5]]$previous_hash
## [1] "a0ecdb2bdcb5ab7eea825e4ee41d729e77c9ba26c05f4166f2cf1f864cc94e3a"
##
## [[5]]$nonce
## [1] 15067
##
## [[5]]$new_hash
## [1] "0fb92488b6a7463ab6453af9c89e10ac214e9163434c43fa834e4e40f9754a44"
##
##
## [[6]]
## [[6]]$index
## [1] 6
##
## [[6]]$timestamp
## [1] "2018-05-29 19:36:46 CEST"
##
## [[6]]$data
## [1] "this is block 6"
##
## [[6]]$previous_hash
## [1] "0fb92488b6a7463ab6453af9c89e10ac214e9163434c43fa834e4e40f9754a44"
##
## [[6]]$nonce
## [1] 23442
##
## [[6]]$new_hash
## [1] "61ef60a29edce9b79fc48f62310258b1f16b5fce1fc08da7cbdf9110e6039658"
```

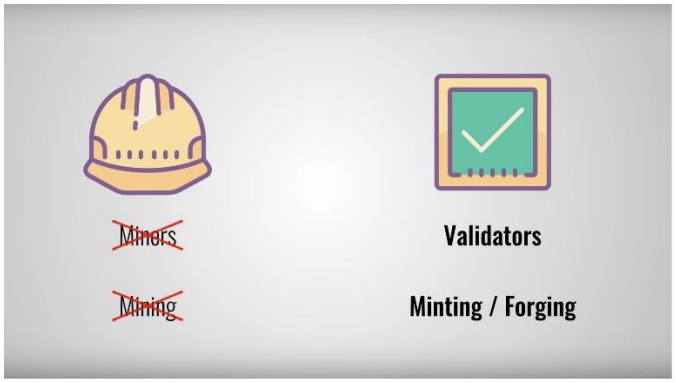
Proof-of-Stake

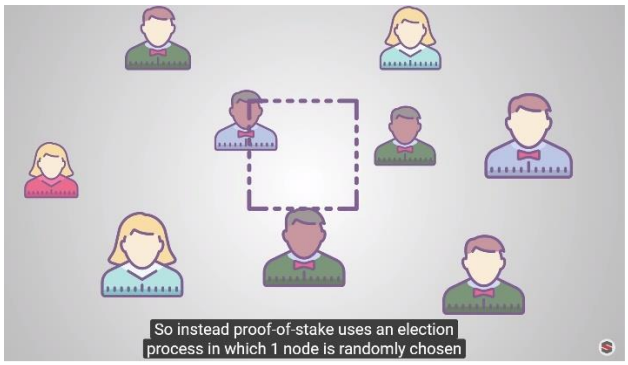
La risoluzione del problema algoritmico ha un forte impatto ecologico, la risoluzione del puzzle crittografico non risolve un problema concreto, ma è fine a se stesso. Sarebbe bello se si risolvessero problemi pratici, tipo problemi del Cern. Esiste una *Proof-of-Stake*, altra tecnica di lavoro rispetto alla *proof of work*: il blocco della catena è creato da un miner selezionato che è scelto casualmente fra i tanti miners, ha il compito di risolvere il problema. Riceverà un compenso proporzionale alla somma depositata nello stake.

Problemi Proof-of-Work:



Confronto fra Proof-of-Work e Proof-of-Stake:





Blockchain e arte

https://www.artnome.com/news/2017/12/22/the-blockchain-art-market-is-here

Artnome. C'è un mercato delle gif. Attraverso la blockchain è possibile rendere unica l'opera d'arte. Su ethereum ci sono aste per queste opere. Inoltre in caso di rivendita l'artista viene remunerato in quota parte. Ciò non avviene nelle opere d'arte reali. In sostanza la blockchain è un luogo dove si vendono beni (tipo opere d'arte) senza bisogno di un intermediario.

Esercizio –implementare vera proof of work

http://users.dimi.uniud.it/~massimo.franceschet/ds/r4ds/syllabus/make/blockchain/pow.html

dalla funzione *mineblock*: restituisce hash e iter: ha fatto 121129 attacchi volendo un hash con 0000 iniziali

6.2 Internet of things

reg 360

Quando abbiamo introdotto i Big Data c'erano 4 elementi che hanno portato a tele entità. Una di queste era l'internet delle cose, cioè una rete che connette oggetti che si parlano tra di loro. Questo è un argomento abbastanza vasto. Noi ci focalizziamo su alcune cose:

1. **Processing**, con cui è possibile molto facilmente creare delle visualizzazioni dinamiche e iterative, che potrebbe essere ortogonale a ggplot. Si interfaccia molto bene con Arduino. Processing è un linguaggio di programmazione che consente di sviluppare diverse applicazioni come giochi, animazioni, contenuti interattivi e opere d'arte generativa.

2. **Arduino** è un'interfaccia che permette di sentire l'ambiente (inclinazione tavolo, luminosità stanza, temperatura aula) accumulano dati che possono essere analizzati statisticamente o visualizzati. In pratica è una scheda, cioè un piccolo computer che può essere programmato attraverso un cavo seriale (giallo), che può trasmettere i dati ad un pc il quale li può leggere ad esempio mediante processing in tempo reale. (es luminosità stanza)

Processing

Le applicazioni di processing si chiamano **skech**. Sono sfere colorate che si muovono nella finestra, quando c'è una piccola sovrapposizione allora appare un segmento che segna la distanza dai loro centri.

Con // si commenta

```
int nCircles = 10;
Circle[] circles = {};

void setup() {
  size(800, 500);
  background(255);
  smooth();
  strokeWeight(1);
  fill(150, 50);
  newCircles();
}

void draw() { // chiamo la funzione draw
  background(255);
  for (int i=0; i < circles.length; i++) {
    // move bubbles
    circles[i].moveMe();
    // draw bubbles
    circles[i].drawMe();
    // draw links
    circles[i].linkMe();
  }
}

void newCircles() { // chiamo funzione che crea 10 nuovi cerchi al sistema
  for (int i=0; i < nCircles; i++) {
    circles = (Circle[]) append(circles, new Circle(circles.length));
  }
}

void mouseReleased() { // Processing + anche orientato agli eventi: quando rilascio il
mouse venogono aggiunge 10 nuove sfere, infatti richiama la funzione lo fa
  newCircles();
}


class Circle {n  // classe oggetti Circle con attributi

  int id; // identificatore univoco
  float x, y; // coordinate del centro. Risp la finestra
  float radius; // raggio
  color fillcol; // colori del contenuto
  float alpha; // trasparenza
  float xmove, ymove; // vettore di movimento (ha direzione e modulo\velocità),
                      // ultimo comando dice ove si troverà la particella

  Circle(int _id) {  //costruttore della classe istanza il tutto in modo randomico

    id = _id;
    x = random(width);
    y = random(height);
```

```
    radius = random(10, 100);
    fillcol = color(random(255), random(255), random(255));
    alpha = random(255);
    xmove = random(-2, 2);
    ymove = random(-2, 2);
  }

  void drawMe() {
    noStroke();
    fill(fillcol, alpha);
    ellipse(x, y, radius*2, radius*2);
  }

  void moveMe() { //metodo che muove la sfera incrementa la posizione della sfera
    x += xmove;
    y += ymove;
    if (x > (width + radius))    x = 0 - radius;  // controlla se la sfera è uscita dal..
    if (x < (0 - radius))        x = width + radius;  //.. bordo dx rientradal bordo sx
                            rientra in posizione - radius
    if (y > (height + radius))   y = 0 - radius;
    if (y < (0 - radius))        y = height + radius;
    // bouncing
    //if ( (x > (width - radius))  | (x < radius) )  xmove *= -1;
    //if ( (y > (height - radius)) | (y < radius) )  ymove *= -1;
  }

  void linkMe() {   //funzione che crea l'interazione quando due sfere si sovrappongono
    for (int i = id + 1; i < circles.length; i++) {
      float dis = dist(x, y, circles[i].x, circles[i].y);
      float overlap = dis - radius - circles[i].radius;

// distanza far cerchi <   distanza fra raggi. Ovelap = distanza meno somma raggi

      if (overlap < 0) {
        stroke(0);
        line(x, y, circles[i].x, circles[i].y);
        //float control = 50;
        //bezier(x, y, x+control, y+control, circles[i].x-control, circles[i].y-control,
circles[i].x, circles[i].y);
      }
    }
  }

}
```

Vediamo ora interazione fra processing e Arduino

Arduino

Arduino è una piccola sheda elettronica che ha dei **sensori** e degli **attuatori** sono componenti che attuano qualcosa: è come una fotocellula che è sensibile alla luce. In base a ciò che la scheda sente gli attuatori eseguono. Linguaggio di programmazione è simile a processing (C e C++) e si chiama ***wiring***.
Video https://www.youtube.com/watch?time_continue=2&v=M48uQVJ6AFA
Rif http://users.dimi.uniud.it/~massimo.franceschet/ns/syllabus/make/processing/arduino/index.html

Ci sono 4 componenti:

1. L'**hardware**: una manopola (potenziometro) che si può girare come se fosse la manopola del volume, una fotocellula, un bottone (on/off) e un pezzo che vibra in base al tono e alla frequenza che riceve ed emette dei suoni

2. **Protocollo di comunicazione fra Arduino e Processi**ng: Ar manda byte a Pr, Pr dice sono pronto, Ar rileva qnt di luce che poi Pr utilizza per visualizzare i dati, e vanno avanti così a scambiarsi dati in tempo reale.

3. **Firmware**: è il software incluso nella componente hardware (è dentro sistemi embedded); è il codice scritto in Wiring che fa funzionare la parte di Arduino.

4. **Software**: elabora i dati. è la parte scritta in Processing

Sintesi:

Processing come metodo, software, linguaggio sia per **visualizzare** dati in formato non standard sia per interagire con Arduino, sia per interagire con Arduino allo scopo di visualizzare in tempo reale dati.