Algoritmo di clustering k-means in stile MapReduce e in Go

SISTEMI DISTRIBUITI E CLOUD COMPUTING 2021/22

Melissa Petrolo
Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma “Tor Vergata”
emelis.ptr@gmail.com

Indice

Introduzione

Lo scopo del progetto e realizzare nel linguaggio di programmazione Go un'applicazione distribuita che implementi l'algoritmo di clustering k-means in versione distribuita secondo il paradigma di computazione MapReduce.

Algoritmi

- Llyod
- Standard K-means
- K-means++

Tecnologie

- Golang
- Docker
- Docker-compose
- EC2

Architettura

La creazione dell'applicazione è avvenuta sulla piattaforma Docker attraverso docker-compose, che permette di definire ed eseguire le applicazioni:

- master
- worker
- registry
- benchmark

Architettura

```
FROM golang:1.17-alpine
ADD . /code/master-worker/master
WORKDIR /code/master-worker/master
COPY . .
#RUN go mod download
RUN go build -o ./main ./code/master-worker/master
ENTRYPOINT ["./main"]
```

```
FROM golang:1.17-alpine
ADD . /code/master-worker/worker
WORKDIR /code/master-worker/worker
COPY . .
RUN go mod download
RUN go build -o ./main ./code/master-worker/worker
ENTRYPOINT ["./main"]
```

Un Dockerfile è un documento di testo che contiene tutti i comandi che un utente può chiamare dalla riga di comando per assemblare un'immagine.
Docker esegue i comandi in ordine del Dockerfile.

Implementazione

MapReduce

Map/Reduce supporta due modalità operative, sequenziali e distribuite.
La fase di Map e la fase di Reduce.

Map

- ciascun mapper riceve in input un sottoinsieme di punti e per ogni punto determina la distanza euclidea con ognuno dei centroidi, determinando la distanza minima e assegnandolo al cluster più vicino

Reduce

- ciascun reducer determina per ogni cluster il nuovo centroide calcolando la media dei punti che appartengono a quel cluster.

Implementazione

Algoritmi

Sono state analizzate tre diversi algoritmi di clustering k-means:

Llyod

1. *Inizializzare k;*
2. *Suddividere i punti dell'insieme X nei k clusters;*
3. *Scegliere k punti uniformemente dall'insieme dei k clusters, determinando i centroidi di ogni cluster;*
4. *Applicare la fase di Map e Reduce;*
5. *Determinare i nuovi centroidi dal risultato della fase di Reduce*
6. *Ripetere step 4 fino allo step 5 fino alla convergenza;*

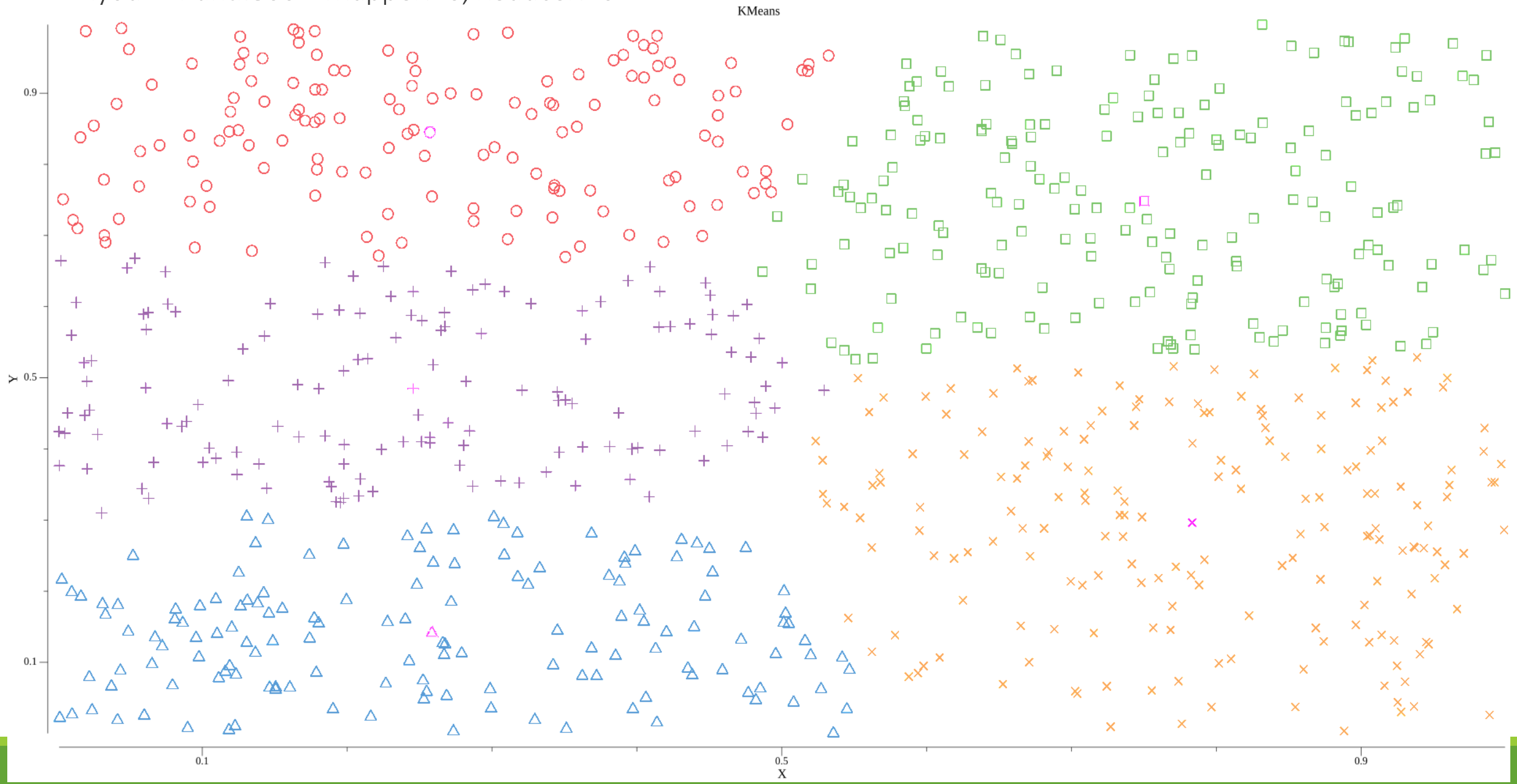
Standard K-means

1. *Inizializzare k;*
2. *Scegliere k punti uniformemente dall'insieme dei punti X determinando i centroidi di ogni cluster;*
3. *Applicare la fase di Map e Reduce;*
4. *Determinare i nuovi centroidi dal risultato della fase di Reduce;*
5. *Ripetere step 3 fino allo step 4 fino alla convergenza;*

K-means++

1. *Inizializzare k;*
2. *Scegliere un punto M scelto uniformemente dall'insieme dei punti X;*
3. *Applicare la fase di Map e Reduce per determinare un nuovo centroide;*
4. *Aggiungere il punto x all'insieme M;*
5. *Ripetere step 3 fino a 4 per un totale di k-1 centroidi;*
6. *Applicare la fase di Map e Reduce dell'algoritmo Llyod con questi centroidi.*

Llyod -> Punti:800 – Mapper:10, Reducer:10



Implementazione

Master-worker

Master	Worker
Prende in input i punti e suddivide il carico di lavoro ai worker.	Esegue la funzione di Map e Reduce.

RPC

Server	Client
rpc.Register(API) rpc.HandleHTTP() lis, e := net.Listen(tcp, «port») http.Serve(lis, nil)	client, err := rpc.DialHTTP(tcp, «IP»:«port») err = client.Call(API.<name>, &args, &reply)

Tecnologie e Librerie

Tecnologie

Go: linguaggio di programmazione usato per l'implementazione dell'applicazione;

- Docker: supporto alla virtualizzazione;
- Docker-compose: orchestrazione dei container, per creare una rete che permette di far comunicare i singoli container;
- Ec2: sviluppo dell'applicazione su un'istanza della piattaforma aws.
- go-RPC: per la comunicazione dei diversi container

Librerie

- Net/rpc: Il pacchetto rpc consente di accedere ai metodi esportati di un oggetto attraverso una rete. Un server registra un oggetto, rendendolo visibile come servizio con il nome del tipo dell'oggetto.
- Testing: Il test dei pacchetti fornisce supporto per il test automatizzato dei pacchetti Go;
- Errorgroup: Il gruppo errgroup del pacchetto fornisce la sincronizzazione, la propagazione degli errori e l'annullamento del contesto per gruppi di routine che lavorano su sotto attività di un'attività comune.
- Godotenv: utilizzato per caricare in env
- Plot: package che fornisce un'API per disegnare grafici.

Esecuzione

È possibile eseguire l'applicazione sia in locale sia in un'istanza EC2.

- Locale
  - Eseguire lo script run.cmd
- EC2
  - Creare una chiave privata su aws ed avviare lo script create_ec2.sh:

```
aws ec2 run-instances --image-id ${AMI} --count 1 \
--instance-type t2.micro \
--security-group-ids ${sg} \
--key-name key-sdcc --associate-public-ip-address \
--tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=SDCC-$1}]"
```

  Connettersi all'istanza tramite comando:

  ssh -i key-sdcc.pem ec2-user@<IndirizzoIP dell'istanza>

  E avviare run.sh

Esecuzione

Nel docker i diversi servizi sono stati creati con profili differenti; in questo modo è possibile avviare sul docker la creazione di container necessari per eseguire l'applicazione o il benchmark.

Per eseguire l'applicazione:
docker-compose up -d master_s --scale worker_s=%NUMWORKER%

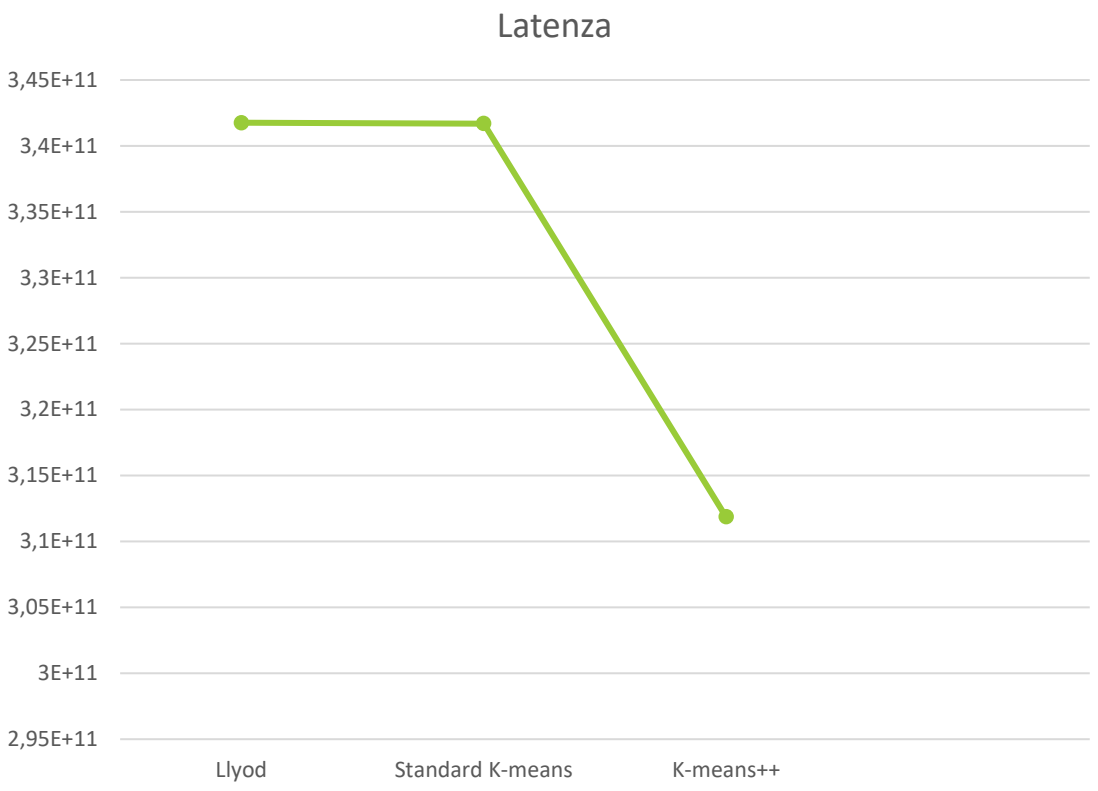
Per eseguire il benchmark:
docker compose --profile app up benchmark_s --scale worker_s=%NUMWORKER%

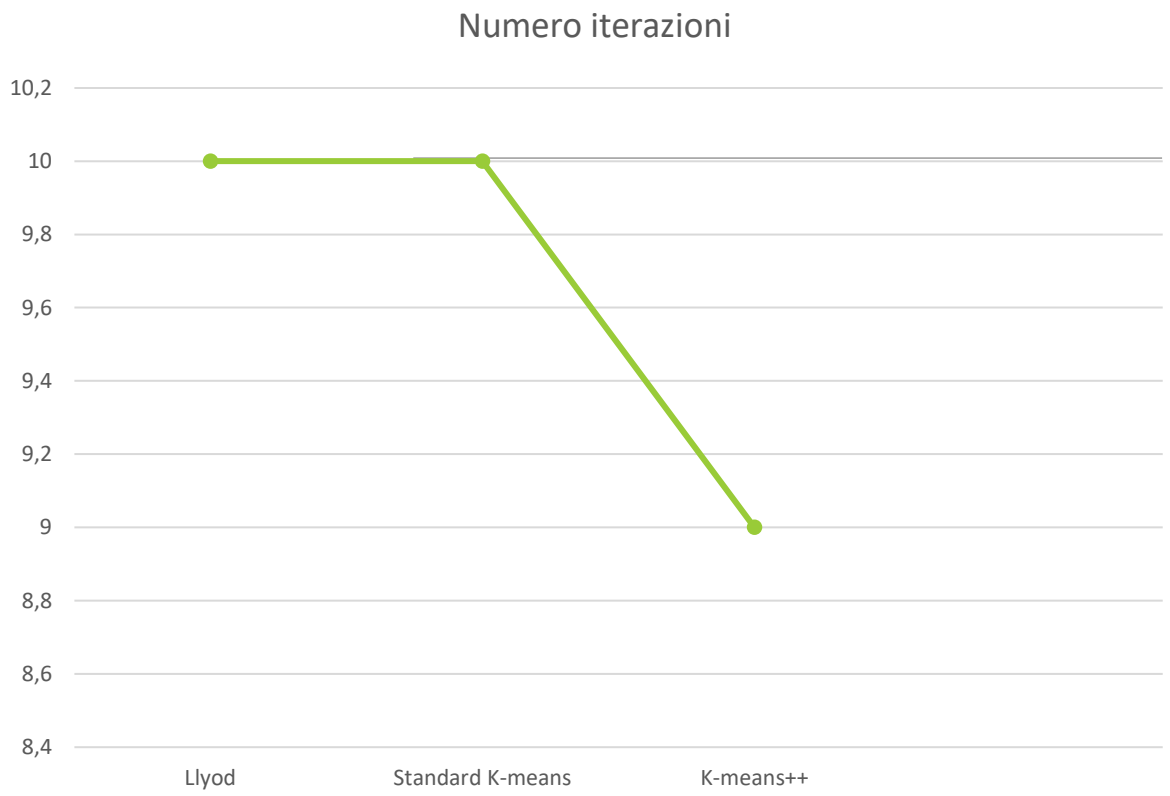
Analisi delle prestazioni

Llyod							
Latenza		Memoria allocata		Bytes allocati		Numero iterazioni	
Cluster	5	Punti	200	Mapper	2	Reducer	2
1m50.5651021s		10446957 allocs/op		171802056 bytes/op		17	
Cluster	5	Punti	200	Mapper	5	Reducer	2
4m2.9829753s		1046328 allocs/op		170171360 bytes/op		15	
Cluster	5	Punti	500	Mapper	5	Reducer	2
2m51.4353833s		6030391 alloc/op		922528416 bytes/op		10	
Cluster	5	Punti	500	Mapper	10	Reducer	5
5m7.0041064s		5938643 alloc/op		903445720 bytes/op		9	
Cluster	5	Punti	800	Mapper	10	Reducer	10
5m41.6744888s		14772226 alloc/op		2144435096 bytes/op		10	

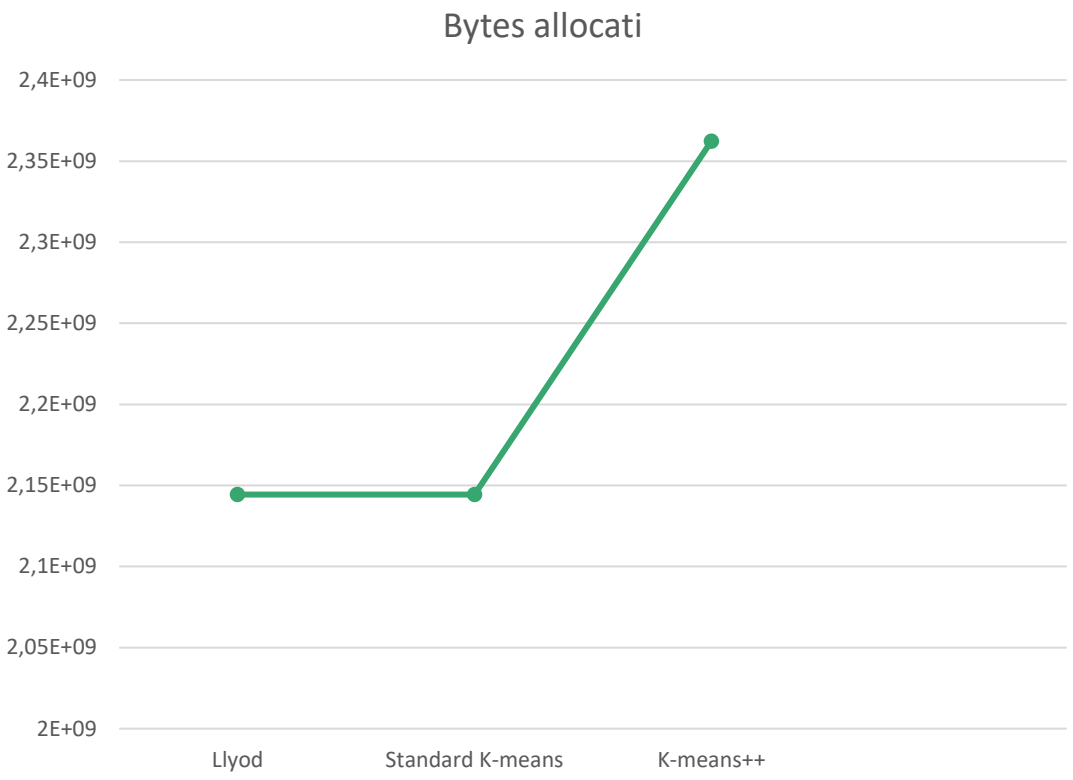
K-means++							
Latenza		Memoria allocata		Bytes allocati		Numero iterazioni	
Cluster	5	Punti	200	Mapper	2	Reducer	2
1m38.5738274s		1072560 allocs/op		181671784 bytes/op		15	
Cluster	5	Punti	200	Mapper	5	Reducer	2
3m32.7970837s		1061626 allocs/op		175356432 bytes/op		13	
Cluster	5	Punti	500	Mapper	5	Reducer	2
5m36.2310378s		6198016 alloc/op		923242496 bytes/op		21	
Cluster	5	Punti	500	Mapper	10	Reducer	5
9m6.8351506s		6110487 alloc/op		903597352 bytes/op		17	
Cluster	5	Punti	800	Mapper	10	Reducer	10
5m11.8577959s		15352244 alloc/op		2362312448 bytes/op		9	

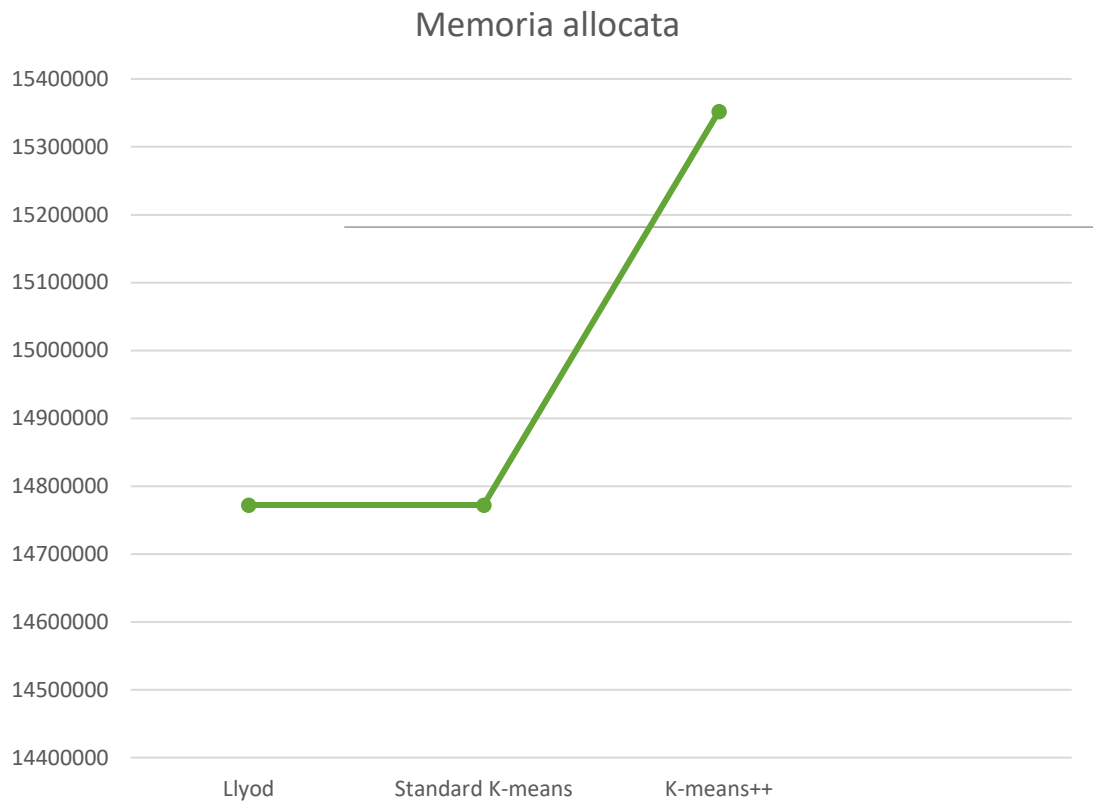
Punti:800 – Mapper:10, Reducer:10





Punti:800 – Mapper:10, Reducer:10





Conclusioni

Prendendo in considerazioni i cluster con un numero di punti pari a 800, un numero di mapper pari a 10 e un numero di reducer pari a 10, è possibile notare che l'algoritmo Llyod e l'algoritmo Standard K-means non presentano grandi differenze dovuto al fatto che i centroidi di ogni cluster vengono scelti inizialmente in modo casuale.

Salvo alcune eccezioni, nella maggior parte dei casi si nota una differenza con l'algoritmo K-means++ con una latenza minore rispetto agli altri due ma con un'allocazione di memoria e bytes maggiore, come mostrato nei grafici seguenti.

Grazie per l’attenzione!