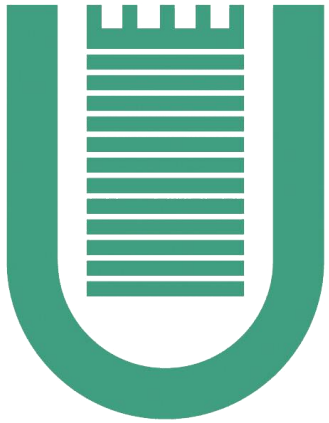


UNIVERSITA' degli STUDI di ROMA

TOR VERGATA

Presentazione SDCC

Milia, Perrone, Pusceddu

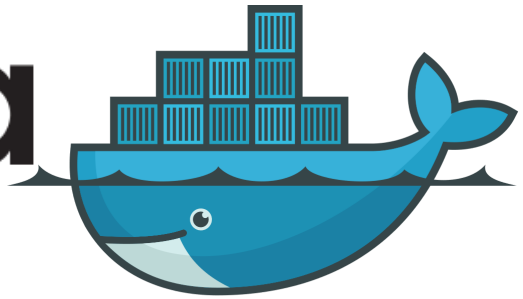
AGENDA

- Architettura del sistema
- Docker
- Kubernetes
- EKS
- MongoDB
- Spring, React
- ApacheKafka
- ApacheStorm
  - Query1
  - Query2
  - Query Controllo
  - Query Stato dei semafori
- Calcolo latenza e throughput

Amazon EKS

APACHE STORM

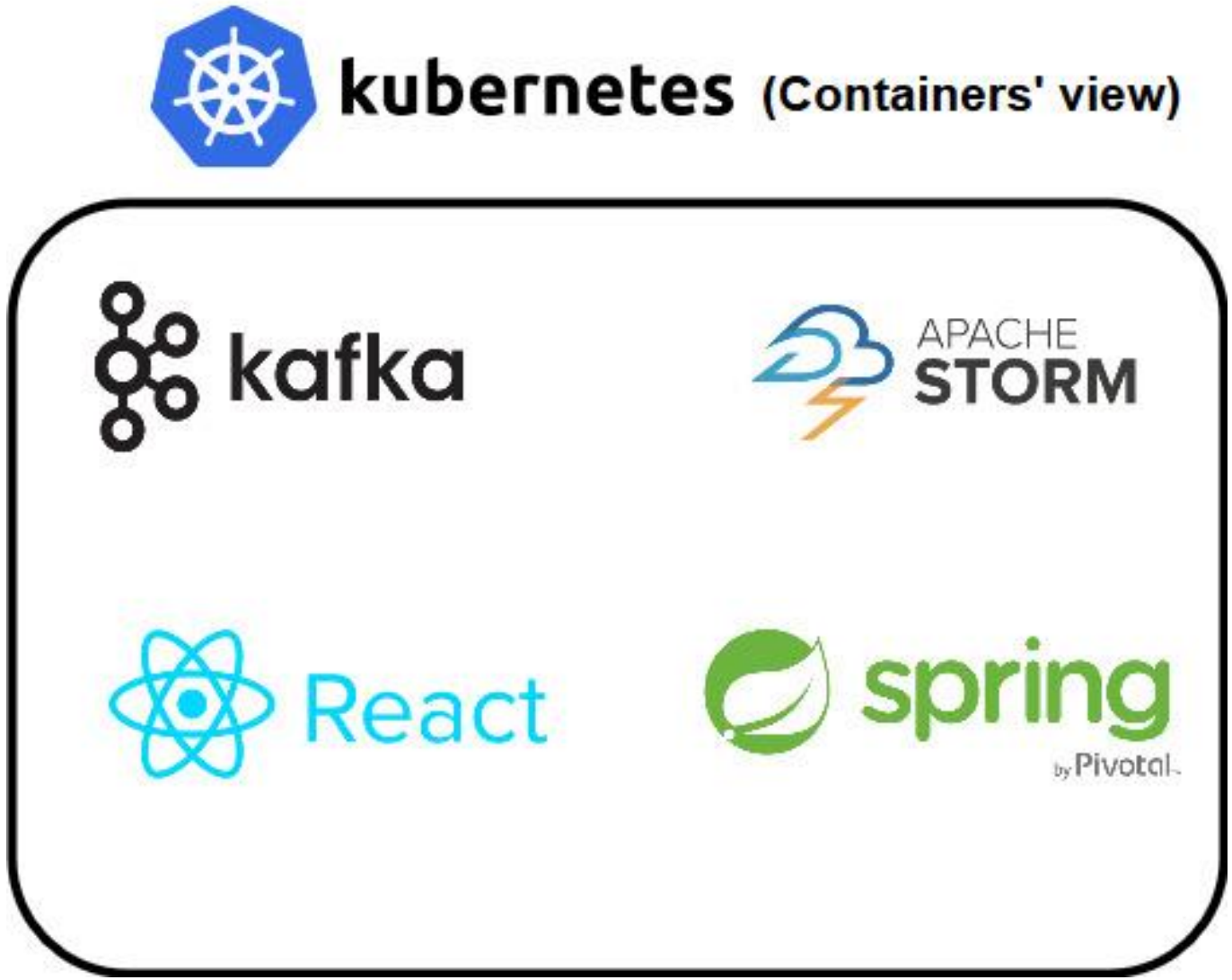
kafka



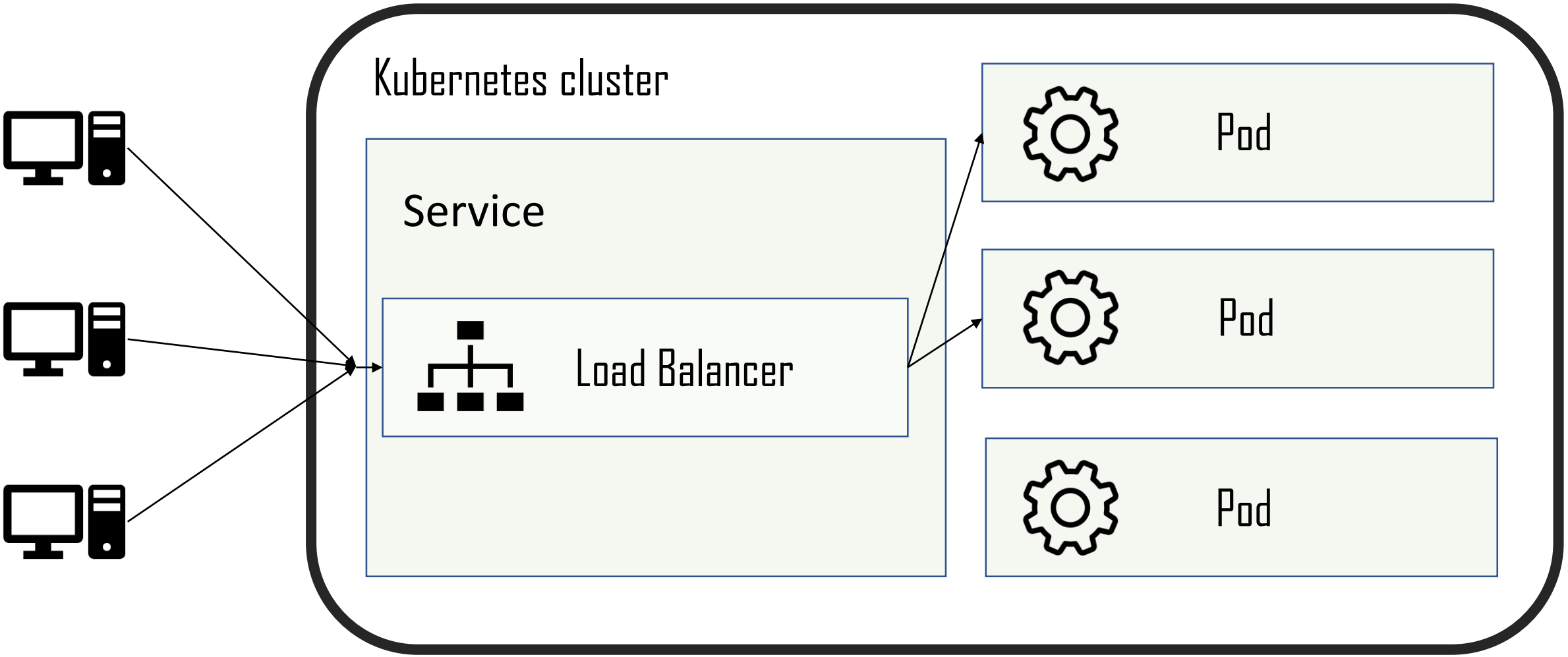
docker

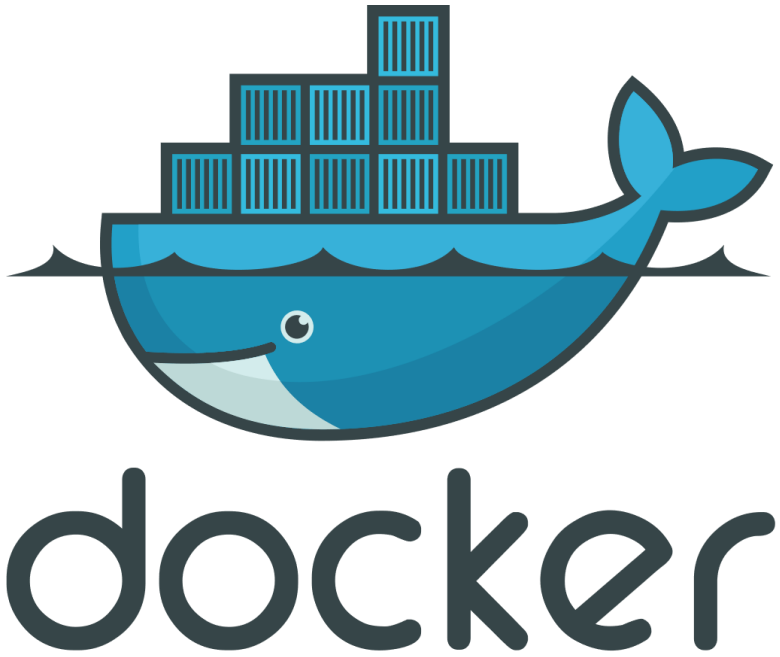
ARCHITETTURA DEL SISTEMA

- Software open source che permette di distribuire e gestire applicazioni in container in modo scalabile.
- Self-healing: riavvia i container che hanno subito un fallimento in modo trasparente all'utente.
- Scaling orizzontale: permette lo scale up o down con semplici comandi.


kubernetes (Containers' view)
kafka
APACHE
STORM
React
spring
by Pivotal

Kubernetes





- Docker è un progetto open-source che automatizza il deployment di applicazioni all'interno di container software.
- Vantaggi:
  - Gestione rapida del deploy.
  - Portabilità.

- Servizio gestito per Kubernetes.
- Gestisce automaticamente la disponibilità e la scalabilità dei nodi.
- EKS si integra con diversi servizi AWS per fornire scalabilità e sicurezza per le applicazioni. Tra questi servizi, Elastic Load Balancing per la distribuzione del carico, IAM per l'autenticazione, Amazon VPC per l'isolamento.

Vantaggi:

- Bassa complessità nell'utilizzo
- Garantisce alta disponibilità
- È scalabile orizzontalmente
- Ha query ottimizzate per la gestione di big data
- Ha un'ottima documentazione
- Offre partizionamento del db
- Utilizzato in applicazioni real-time

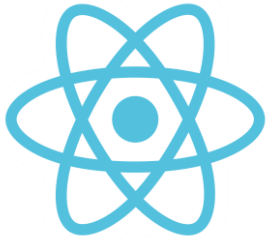
Collection progettate:

- SdccRank
- SdccMedian
- StateTrafficLight
- SdccIntersection

Spring, React

- Framework utilizzato per il backend
- CRUD incrocio
- get stato dei semafori
- Riparazione luce semaforo
- get informazioni query1
- get informazioni su query2

- Libreria utilizzata per il frontend
- Utilizza Redux per effettuare chiamate temporizzate al backend
- Visualizzazione stato dei semafori
- Visualizzazione semafori rotti
- Visualizza,inserisci,modifica,elimina incrocio
- Visualizzazione informazioni query1
- Visualizzazione informazioni query2



- Piattaforma middleware di scambio di messaggi più popolare al mondo
- Piattaforma distribuita su un cluster
- Utilizzato per il data injection verso ApacheStorm
- Integrato in ApacheStorm
- Produttore scrive i dati su un topic
- Utilizza Zookeeper
- Offre salvataggio dello stream di dati
- Servizio affidabile con diversi tipi di semantiche

kafka

- ApacheStorm è un sistema di calcolo distribuito, a tolleranza di errore e open source
- Nel progetto è stato utilizzato tale framework per elaborare i flussi di dati provenienti dai sensori in real-time
- La scelta di ApacheStorm è dovuta al fatto che tale framework è maggiormente utilizzato e documentato e risulta didatticamente migliore perché offre delle api più a basso livello rispetto a Flink e Hearon
- Flessibilita in termini di linguaggio
- Scalabilita dinamica
- Interfaccia utente
- Affidabilità e tolleranza ai guasti

Topologia query 1

QUERY 1

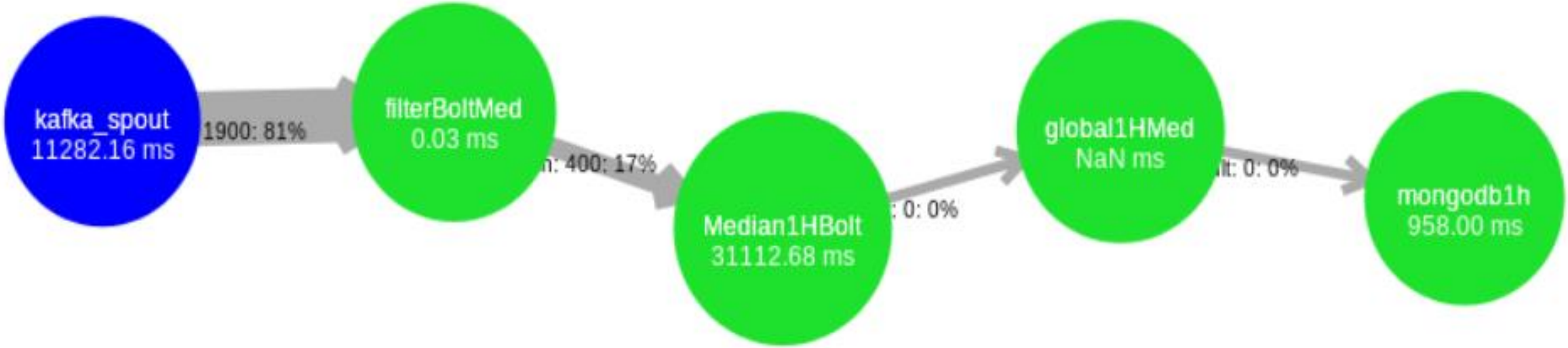
Scopo: Calcolare la classifica dei TOP K Incroci (con K=10) con la velocità media maggiore, in finestre temporali da 15 minuti,1 ora e 24 ore.

Spout: Lo Spout è unico ed è di tipo KafkaSpout.Prende le tuple dal topic di Kafka e le propaga nella topologia.

Bolt usati:

- **FilterBolt**: Unisce le tuple provenienti dai sensori per formare incroci.
- **AvgBolt**: WindowBolt che calcola la velocità media per ogni incrocio nelle finestre temporali da 15M 1H o 24H.
- **IntermediateRankBolt**: Calcolano le classifiche parziali
- **GlobalRankBolt**: Unisce ordinatamente le classifiche parziali
- **MongoDBBolt**: Aggiorna le classifiche nel DB.

Topologia query 2



QUERY 2

Scopo: Calcolare le intersezioni che hanno la mediana del numero di veicoli superiore al valore della mediana globale dei veicoli che hanno attraversato tutte le intersezioni in finestre temporali da 15M 1H e 24H.

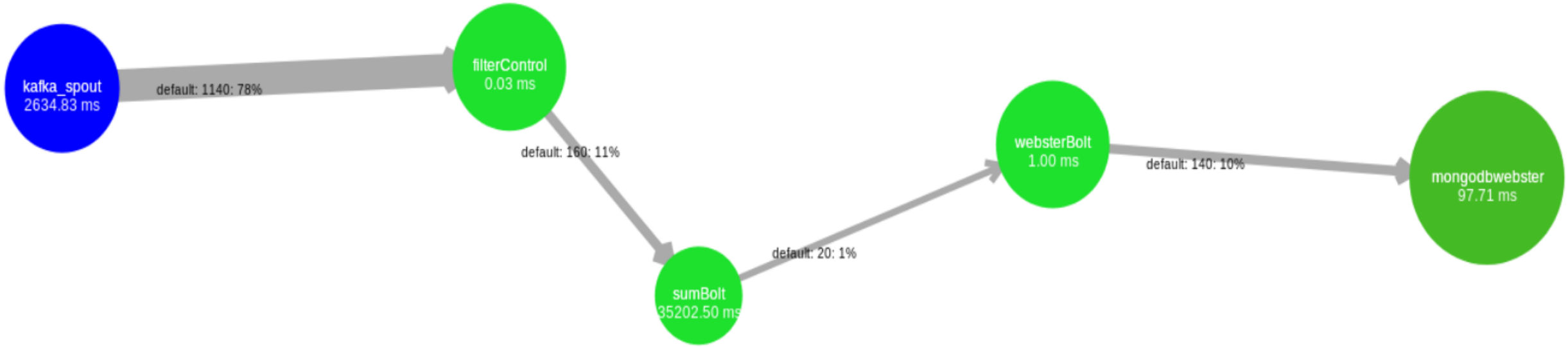
Spout: Lo Spout è unico ed è di tipo KafkaSpout.Prende le tuple dal topic di Kafka e le propaga nella topologia.

Bolt usati:

- **FilterMedianBolt:** Unisce le tuple provenienti dai sensori per formare incroci.
- **MedianBolt**: Calcola la mediana del numero di veicoli per singolo incrocio utilizzando Tdigest in finestre temporali da 15M 1H e 24H.
- **GlobalMedianBolt:** Calcola mediana globale e restituisce incroci con la mediana maggiore della mediana globale.
- **MongoDBBolt**: Aggiorna il DB con la lista degli incroci restituita dalla topologia.

Trade-off sul calcolo della mediana: Tdigest elimina collo di bottiglia dovuto all'ordinamento,ma restituisce mediana approssimata con un piccolo errore.

Topologia sistema di controllo



TOPOLOGIA SISTEMA CONTROLLO

Scopo: Implementazione sistema di controllo per calcolare le durate delle lampade verdi e rosse per ridurre il traffico.

Spout: Lo Spout è unico ed è di tipo KafkaSpout.Prende le tuple dal topic di Kafka e le propaga nella topologia.

Bolt usati:

- **FilterControlBolt:** Unisce le tuple provenienti dai sensori per formare incroci.
- **SumBolt**: Calcola in una finestra temporale il numero di veicoli che attraversano ogni sensore per poi calcolare il flusso ad ogni semaforo.
- **WebsterBolt:** Applica l'algoritmo di Webster per calcolare durate luci verdi e rosse.
- **MongoDBBolt**: Aggiorna il DB con la durata delle lampade per ogni semaforo.

Webster è un algoritmo locale(valido per singolo incrocio).
Supponendo incroci sufficientemente distanti diventa un buon algoritmo anche su un insieme di incroci.

Topologia stato dei semafori

TOPOLOGIA STATO SEMAFORI

Scopo: Capire quali Incroci contengono semafori con lampade guaste.

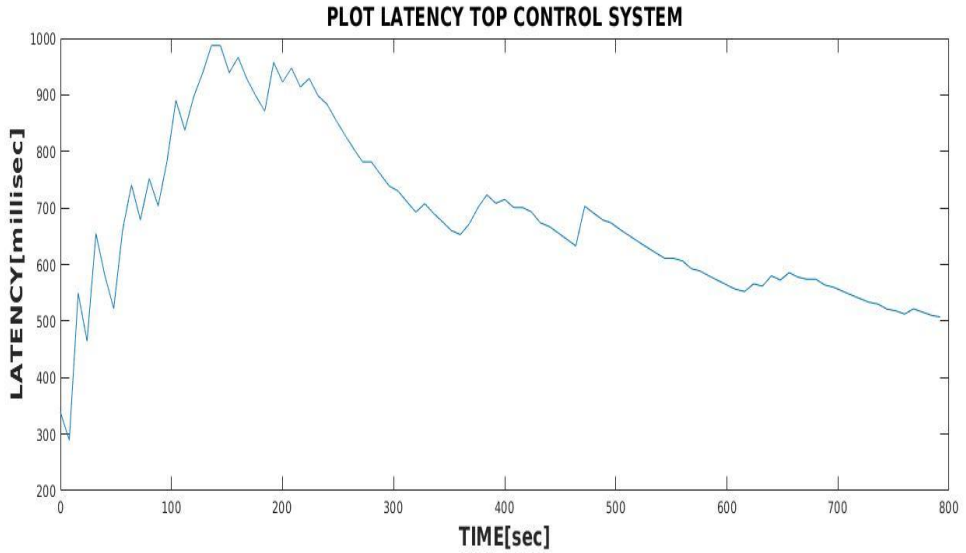
Spout: Lo Spout è unico ed è di tipo KafkaSpout.Prende le tuple dal topic di Kafka e le propaga nella topologia.

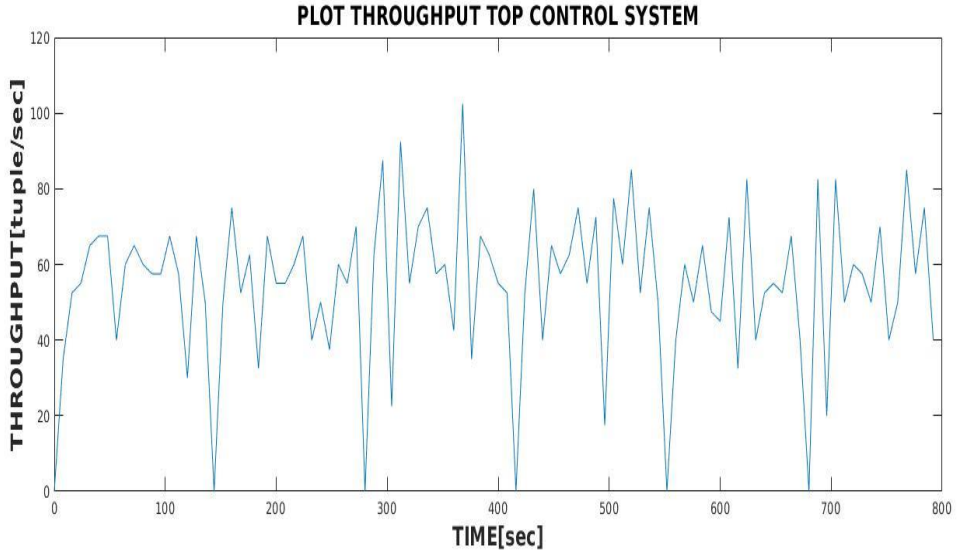
Bolt usati:

- **CheckStateBolt**: Per ogni tupla capisce se il relativo sensore ha lampada guasta.
- **MongoDBBolt**: Memorizza nel DB le info dei sensori con lampade guaste.

L'amministratore di sistema ha la possibilità di segnare i sensori come riparati e quindi eliminare dal DB la relativa entry che segna il sensore come guasto.

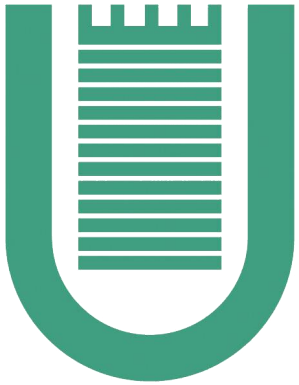
Calcolo latenza e throughput





Procedura calcolo Throughput e latenza:

- Dati per calcolo throughput e latenza ottenuti tramite **Storm UI REST API**: Chiamata **http://<ui-host>:<ui-port>/api/v1/ topology/:topology_id** ritorna le informazioni sulla topologia **topology_id** in formato JSON.
- Ogni L secondi chiamata **Storm UI REST :**
- Vengono estratti i valori della latenza e numero tuple emesse dall'avvio della topologia.
- I dati vengono memorizzati in un file CSV
- I dati vengono elaborati:
  - Throughput in [t,t+L]=TupleEmesse in [t,t+L]/L=(TupleEmesse[0,t+L]-TupleEmesse[0,t])/L.
- I dati elaborati vengono plottati con Matlab.



UNIVERSITA' degli STUDI di ROMA
TOR VERGATA

Grazie per l'attenzione