Progetto SDP 2021

Laboratorio di Sistemi Distribuiti e Pervasivi

Aprile 2021

1 Descrizione del progetto

Lo scopo del progetto è quello di realizzare un sistema che gestisca le consegne degli ordini ricevuti dal sito di e-commerce **Dronazon**, tramite l'utilizzo di droni. In Figura 1 è mostrata l'architettura del sistema da sviluppare. Dronazon ha a disposizione uno stormo di droni distribuiti all'interno di una smart-city per gestire le proprie consegne. I droni della smart-city si devono organizzare per eleggere un drone master. Ogni volta che un cliente di Dronazon effettua un nuovo ordine, questa informazione verrà passata al drone master della smart-city. Il drone master stabilirà poi quale drone si dovrà occupare di tale consegna.

Ogni drone ha installato un sensore che rileva il livello di inquinamento dell'aria. Inoltre, ogni consegna implica un consumo di batteria. Quando il livello di batteria residua di un drone è al di sotto del 15%, il drone è costretto ad abbandonare il sistema. Periodicamente, i droni devono comunicare ad un server remoto, chiamato Server Amministratore, le informazioni relative al numero di consegne effettuate, al numero di chilometri percorsi, al livello di inquinamento dell'aria rilevato ed al livello di batteria rimanente. Gli amministratori di Dronazon potranno monitorare il proprio sistema di consegne attraverso il Server Amministratore. Inoltre, tramite il Server Amministratore è anche possibile registrare e rimuovere droni dal sistema dinamicamente.

2 Applicativi da implementare

Per il progetto, è richiesta l'implementazione dei seguenti applicativi:

- *Dronazon*: applicativo che simula un sito di e-commerce, generando periodicamente nuove consegne da dover effettuare e comunicando le informazioni relative ad esse allo stormo di droni
- *Drone*: uno specifico drone del sistema
- *Server Amministratore*: server REST che riceve statistiche dai droni e che permette la gestione dinamica della rete di droni

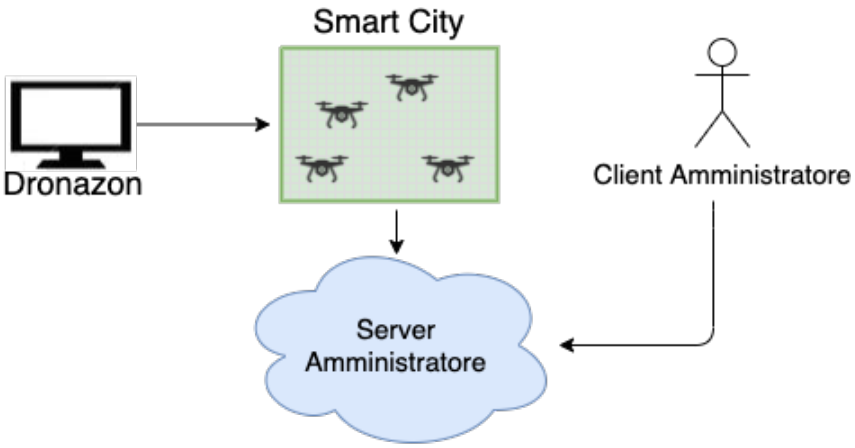

Figura 1: Architettura del sistema.

- *Amministratore*: client che interroga il *Server Amministratore* per ottenere informazioni sulle statistiche relative ai droni e alle consegne

Notare che ogni *Drone* è un **processo** a sè stante, e quindi non deve essere implementato come un thread. In seguito vengono forniti dettagli sugli applicativi da realizzare.

3 Rappresentazione interna della smart-city

La smart-city viene rappresentata come una griglia 10x10 (vedi Figura 2). Ogni cella della griglia rappresenta un chilometro quadrato della smart-city. Al momento della registrazione all'interno del sistema, un drone verrà posizionato in una cella scelta casualmente. Per semplicità, una singola cella può contenere più droni.

4 Dronazon

Dronazon è un processo che simula un sito di e-commerce. In particolare, tale processo deve generare un nuovo ordine per cui è richiesta una consegna ogni 5 secondi. Ogni ordine è caratterizzato da:

- ID
- Punto di ritiro
- Punto di consegna

Punto di ritiro e di consegna sono espressi come coordinate cartesiane di una cella della griglia rappresentante la smart-city. Gli ordini generati da Dronazon, devono essere comunicati tramite il protocollo MQTT visto a lezione.

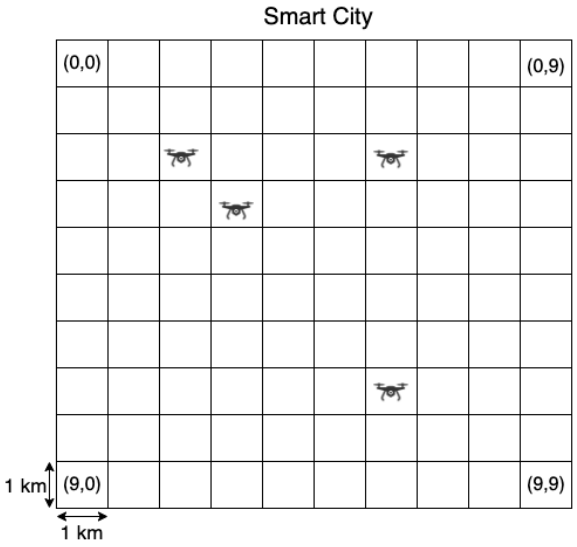

Figura 2: Rappresentazione della smart-city

Per semplicità, si assume che il broker MQTT della smart-city sia attivo al seguente indirizzo `tcp://localhost:1883`. Dronazon deve connettersi a tale broker come client MQTT e assumerà il ruolo di publisher. Ogniqualvolta viene generato un nuovo ordine, Dronazon pubblica tale aggiornamento sul seguente topic MQTT: *dronazon/smartcity/orders/*.

5 Drone

Ogni drone è simulato da un **processo** che si occupa di

- coordinarsi con gli altri droni tramite gRPC per:
  - eleggere il drone master della smart-city tramite l'algoritmo di elezione ring-based visto durante il corso di teoria
  - inviare statistiche al server amministratore
- effettuare le consegne assegnate dal drone master
- uscire dal sistema quando il proprio livello di batteria si trova al di sotto del 15%

5.1 Struttura della rete

La rete di droni ha una struttura ad anello. La creazione di tale rete deve essere gestita in modo decentralizzato, senza essere guidata dal server amministratore. E' quindi necessario tenere in considerazione i possibili problemi

di sincronizzazione distribuita che possono avvenire in ingresso ed uscita dall'anello.

5.2 Inizializzazione

Un drone deve essere inizializzato specificando

- ID
- Porta di ascolto per la comunicazione tra droni
- Indirizzo del server amministratore

Una volta avviato, il processo drone deve registrarsi al sistema tramite il server amministratore. Se l'inserimento andrà a buon fine (ovvero, non esistono altri droni con lo stesso ID), il drone riceve dal server:

- la propria posizione di partenza nella smart-city (es: $(8, 6)$)
- l'elenco di droni già presenti nella smart-city

Ricevute queste informazioni, il drone deve quindi inserirsi nella rete ad anello in maniera decentralizzata, senza l'aiuto del server amministratore. Quando l'inserimento del drone nella smart-city va a buon fine, esso dovrà avviare il proprio sensore per il rilevamento dell'inquinamento dell'aria. Nel caso in cui non ci siano altri droni all'interno della smart-city, il drone si auto-proclama master. Altrimenti, il drone deve presentarsi agli altri droni della smart-city inviando la propria posizione nella griglia e capire chi sia il drone master. Durante questa fase di presentazione, il drone non necessita di comunicare il proprio livello di batteria in quanto si assume che sia pari al 100%.

5.3 Consumo di batteria

Al termine di ogni consegna un drone perderà un percentuale di batteria pari al 10%. Si noti che per semplicità si assume che la percentuale di batteria consumata per una consegna venga calcolata solo al temine di quest'ultima.

5.4 Chiusura esplicita

Si assume che ogni drone possa terminare solamente in maniera controllata. In particolare, ogni drone chiede al server amministratore di uscire dal sistema non appena il suo livello di batteria è inferiore al 15%. Inoltre, un drone può richiedere autonomamente di uscire dalla rete tramite una richiesta esplicita (e.g., *quit*) da linea di comando. In entrambi i casi, per uscire dalla rete un drone che non è master deve eseguire i seguenti passi:

1. terminare l'eventuale consegna di cui si sta occupando, comunicando poi al drone master le informazioni descritte nella Sezione 5.6.2
2. chiudere forzatamente le comunicazioni con gli altri droni, senza curarsi di comunicare loro la propria uscita dalla rete
3. chiedere al server amministratore di uscire dalla rete

Se invece il drone in questione è il master, è necessario che compia i seguenti passi:

1. terminare l'eventuale consegna di cui si sta occupando, memorizzando le informazioni descritte nella Sezione 5.6.2
2. disconnettersi dal broker MQTT della smart-city
3. assicurarsi di assegnare le consegne pendenti ai droni della smart-city
4. chiudere forzatamente le comunicazioni con gli altri droni, senza curarsi di comunicare loro la propria uscita dalla rete
5. inviare al server amministratore le statistiche globali della smart-city
6. chiedere al server amministratore di uscire dal sistema

Al fine di semplificare lo sviluppo del progetto, non e' necessario gestire le consegne generate da Dronazon nell'intervallo di tempo che intercorre tra l'uscita dalla rete del drone master corrente e l'elezione del successivo. Questo implica che tali consegne andranno perse. I droni della rete devono essere in grado di rilevare l'assenza di un altro drone. In tal caso, dovranno ricostruire l'anello. Quando un drone si accorge dell'assenza del master, esso deve avviare l'elezione di un nuovo drone master tramite l'algoritmo ring-based visto a lezione (Chang and Roberts). Come drone master, dovrà essere eletto il drone con il livello di batteria residua maggiore. In caso di parità, viene scelto il drone con ID maggiore. Si noti che se un drone sta effettuando una consegna durante una elezione, bisogna considerare come livello di batteria quello che avra' al termine della consegna. Non appena viene eletto un nuovo drone master, tutti gli altri droni dovranno comunicargli la propria posizione all'interno della smart-city. La rilevazione dell'assenza di un drone, la ricostruzione dell'anello e l'elezione ring-based non devono essere guidate in alcun modo dal server amministratore.

5.5 Sensori di inquinamento

Ogni sensore di inquinamento produce periodicamente delle misurazioni relative al livello di polveri sottili nell'aria (PM10). Ogni singola misurazione di sensore è caratterizzata da:

- Valore di PM10 letto
- Timestamp in millisecondi

La generazione di queste misurazioni viene svolta da opportuni simulatori, il cui codice viene fornito per semplificare lo sviluppo del progetto. Ogni simulatore assegna come timestamp alle misurazioni il numero di secondi passati dalla mezzanotte.
E' possibile scaricare il codice necessario direttamente dalla pagina Moodle del corso, nella sezione *progetti*. Questo codice andrà aggiunto come package al progetto e NON deve essere modificato. In fase di inizializzazione, ogni drone dovrà quindi occuparsi di far partire il simulatore necessario per generare misurazioni del sensore.

Ogni simulatore è un thread che consiste in un loop infinito che simula periodicamente (con frequenza predefinita) le misurazioni, aggiungendole ad una struttura dati. Di questa struttura dati viene fornita solo l'interfaccia (Buffer), la quale espone due metodi:

- $void\ add(Measurement\ m)$
- $List<Measurement>\ readAllAndClean()$

è dunque necessario creare una classe che implementi l'interfaccia. Ogni drone avrà un singolo sensore e quindi un solo buffer.

I thread di simulazione usano il metodo $addMeasurement$ per riempire la struttura dati. Il metodo $readAllAndClean$ deve invece essere usato per ottenere tutte le misurazioni contenute nella struttura dati. Al termine di una lettura, $readAllAndClean$ deve occuparsi di svuotare il buffer in modo da fare spazio a nuove misurazioni. In particolare, i dati dei sensori devono essere processati usando la tecnica della *sliding window* presentata durante le lezioni di teoria, considerando una dimensione del buffer di 8 misurazioni ed un overlap del 50%. Quando la dimensione del buffer raggiunge le 8 misurazioni, deve essere effettuata la media dei dati nel buffer. Le medie calcolate in questo modo verranno poi trasmesse insieme alle informazioni sulle consegne.

5.6 Sincronizzazione distribuita

5.6.1 Gestione delle consegne

Ogniqualvolta Dronazon genera una nuova consegna da effettuare, questa viene pubblicata dal sito di e-commerce tramite protocollo MQTT sul topic *dronazon/smartcity/orders/* (vedi Sezione 4). Quando viene eletto un nuovo master, questo, prima di effettuare qualsiasi operazione, deve attendere di ricevere la posizione degli altri droni della smart-city, senza il supporto del server amministratore. Successivamente, il drone master deve connettersi al broker MQTT della smart-city e registrarsi come subscriber al topic

dronazon/smartcity/orders/, per ricevere informazioni sulle nuove consegne. Quando un drone master riceve una richiesta per una nuova consegna, esso la assegnerà al drone della smart-city (incluso sè stesso) che rispetta i seguenti criteri:

- il drone non deve essere già impegnato in un'altra consegna
- tra i droni che rispettano i criteri precedenti, viene scelto quello più vicino al luogo di ritiro dell'ordine con maggior livello di batteria residuo
- nel caso in cui più di un drone rispetti tutti i criteri elencati in precedenza, verrà selezionato per la consegna il drone con ID maggiore

Per calcolare la distanza tra due punti $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ della smart-city, è necessario usare la seguente formula:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Se non ci sono droni disponibili per effettuare una consegna, questa viene inserita in un'apposita coda gestita dal drone master. Ogni volta che un drone porta a termine una consegna comunicando le relative statistiche al drone master, quest'ultimo utilizza i criteri precedentemente descritti per assegnare le consegne pendenti che si trovano nella coda.

Quando un drone riceve una consegna da effettuare, il tempo impiegato per consegnare l'ordine viene simulato da una `Thread.sleep()` di 5 secondi. Terminata la consegna, il drone comunica al master alcune informazioni, che verranno descritte nella Sezione 5.6.2. Per semplicità, si assume che un drone porta sempre a termine con successo la consegna a lui assegnata.

5.6.2 Calcolo delle informazioni del drone

Ogni volta che un drone effettua una consegna deve trasmettere al drone master le seguenti informazioni:

- timestamp di arrivo al luogo di consegna
- nuova posizione del drone che coincide con la posizione di consegna
- chilometri percorsi per raggiungere i punti di ritiro e consegna
- medie delle misurazioni relative al livello di inquinamento dell'aria rilevate a partire dall'ultima consegna effettuata (vedi Sezione 5.5)
- il proprio livello di batteria residuo

Queste informazioni verranno poi utilizzate dal drone master per calcolare le statistiche globali della smart-city. Ogni 10 secondi un drone deve stampare a schermo: il numero totale di consegne effettuate, i chilometri percorsi e la percentuale di batteria residua.

5.6.3 Invio delle informazioni al server

I droni devono coordinarsi per inviare periodicamente al *Server Amministratore* le *statistiche globali* della smart-city. Sarà il drone master a calcolare le *statistiche globali* ogni 10 secondi, considerando la media delle informazioni ricevute dai singoli droni al momento delle loro consegne (vedi Sezione 5.6.2). Nello specifico le *statistiche globali* sono le seguenti:

- media del numero di consegne effettuate dai droni
- media dei chilometri percorsi dai droni
- media del livello di inquinamento rilevato dai droni
- media del livello di batteria residuo dei droni

Quando il drone master termina di calcolare le **statistiche globali**, esso le invia al *server amministratore* insieme al timestamp in cui le statistiche sono state calcolate.

6 Server Amministratore

Il Server Amministratore si occupa di mantenere una rappresentazione interna della smart-city e di ricevere dal drone master le *statistiche globali* (vedi Sezione 5.6.3). Queste informazioni verranno poi interrogate dagli Amministratori. Tale server deve quindi offrire due diverse interfacce REST per:

- gestire la rete di droni e ricevere le statistiche
- permettere agli amministratori di effettuare interrogazioni

6.1 Interfaccia per i droni

6.1.1 Inserimento

Quando vuole inserirsi nel sistema, un drone deve comunicare al server amministratore:

- ID
- Indirizzo IP
- Numero di porta sul quale è disponibile per comunicare con gli altri droni

Al momento della registrazione all'interno del sistema, un drone verrà posizionato in una cella della smart-city, scelta casualmente. E' possibile aggiungere un drone nella rete solo se non esiste un altro drone con lo stesso identificatore. Se l'inserimento va a buon fine, il server amministratore restituisce al drone la lista di droni già presenti nella smart-city, specificando per ognuno ID, indirizzo IP e numero di porta per la comunicazione. Notare che l'inserimento nella rete ad anello del nuovo drone dovrà essere gestita in maniera distribuita, senza il supporto del server amministratore.

6.1.2 Rimozione

Quando un drone si accorge che il proprio livello di batteria è sceso al di sotto del 15% deve chiedere al server amministratore di uscire dalla rete. Inoltre, un drone può richiedere autonomamente di uscire dalla rete tramite una richiesta esplicita (e.g., *quit*) da linea di comando. Quando il server amministratore riceve una richiesta di rimozione da parte di un drone deve rimuoverlo dalla smart-city. E' importante notare che il server amministratore si dovrà solo occupare di rimuovere il drone dalla propria struttura dati che rappresenta la smart-city. Gli altri droni dovranno accorgersi autonomamente dell'uscita di un drone e, di conseguenza, essere in grado di ricostruire l'anello.

6.1.3 Statistiche

Il server amministratore deve predisporre un'interfaccia per ricevere dal drone master le statistiche globali della smart-city. E' sufficiente memorizzare queste informazioni in opportune strutture dati che permettano di effettuare successive analisi.

6.2 Interfaccia per gli amministratori

Il server amministratore deve fornire dei metodi per ottenere le seguenti informazioni:

- L'elenco dei droni presenti nella rete
- Ultime n statistiche globali (con timestamp) relative alla smart-city (vedi Sezione 5.6.3)
- Media del numero di consegne effettuate dai droni della smart-city tra due timestamp t1 e t2
- Media dei chilometri percorsi dai droni della smart-city tra due timestamp t1 e t2

7 Amministratore

L'applicazione amministratore è un'interfaccia a linea di comando che si occupa di interagire con l'interfaccia *REST* fornita dal server amministratore. L'applicazione deve quindi mostrare all'amministratore un semplice menu per scegliere uno dei servizi per amministratori offerti dal server (vedi Sezione 6.2), con la possibilità di inserire eventuali parametri richiesti.

8 Semplificazioni e limitazioni

Si ricorda che lo scopo del progetto è dimostrare la capacità di progettare e realizzare un'applicazione distribuita e pervasiva. Pertanto gli aspetti non riguardanti il protocollo di comunicazione, la concorrenza e la gestione dei dati di sensori sono considerati secondari.

Inoltre è possibile assumere che :

- nessun nodo si comporti in maniera maliziosa,
- nessun processo termini in maniera incontrollata

Si gestiscano invece i possibili errori di inserimento dati da parte dell'utente. Inoltre, il codice deve essere robusto: tutte le possibili eccezioni devono essere gestite correttamente.

Sebbene le librerie di Java forniscano molteplici classi per la gestione di situazioni di concorrenza, per fini didattici gli studenti sono invitati a fare **esclusivamente uso di metodi e di classi spiegati durante il corso di laboratorio**. Pertanto, eventuali strutture dati di sincronizzazione necessarie (come ad esempio lock, semafori o buffer condivisi) dovranno essere implementate da zero e saranno discusse durante la presentazione del progetto.

Per la comunicazione tra processi è necessario utilizzare il framework *gRPC*. Qualora fossero previste comunicazioni in broadcast, queste devono essere effettuate in parallelo e non sequenzialmente.

9 Presentazione del progetto

Il progetto è da svolgere individualmente. Durante la valutazione del progetto verrà richiesto di mostrare alcune parti del programma, verrà verificata la padronanza del codice presentato, verrà verificato il corretto funzionamento del programma e verranno inoltre poste alcune domande di carattere teorico inerenti gli argomenti trattati nel corso (parte di laboratorio). Verrà richiesto di eseguire una demo del progetto sul proprio computer, mentre il codice sorgente verrà analizzato sulla macchina del docente. Il codice sorgente dovrà essere consegnato al docente prima della discussione del progetto. Per

la consegna, è sufficiente archiviare il codice in un file zip, rinominato con il proprio numero di matricola (es. 760936.zip) ed effettuare l'upload dello stesso tramite il sito http://upload.di.unimi.it . Sarà possibile effettuare la consegna a partire da una settimana prima della data di ogni appello. La consegna deve essere tassativamente effettuata entro le 23:59 del secondo giorno precedente quello della discussione (es. esame il 13 mattina, consegna entro le 23.59 dell'11).

Si invitano inoltre gli studenti ad utilizzare, durante la presentazione, le istruzioni `Thread.sleep()` al fine di mostrare la correttezza della sincronizzazione del proprio programma. Si consiglia vivamente di analizzare con attenzione tutte le problematiche di sincronizzazione e di rappresentare lo schema di comunicazione fra le componenti. Questo schema deve rappresentare il formato dei messaggi e la sequenza delle comunicazioni che avvengono tra le componenti in occasione delle varie operazioni che possono essere svolte. Tale schema sarà di grande aiuto, in fase di presentazione del progetto, per verificare la correttezza della parte di sincronizzazione distribuita. Si ricorda che non è necessario mostrare la struttura delle classi del progetto, ma solo gli aspetti principali di sincronizzazione e di protocollo.

Durante la presentazione del progetto, sarà richiesto di testare il funzionamento del sistema avviando almeno 4 o 5 droni.

10 Casi di plagio

Non è tollerato il riutilizzo di codice sviluppato da altri studenti. In tale caso verranno applicate le sanzioni elencate nella sezione *student plagiarism* delle informazioni generali sul sito del corso.

11 Parti facoltative

Al fine di incentivare la presentazione del progetto nei primi appelli disponibili, lo svolgimento della parte facoltativa 1 diventa obbligatoria a partire dall'appello di Settembre (incluso), mentre entrambe le parti facoltative (1 e 2) saranno obbligatorie negli appelli successivi.

11.1 Prima parte

Nella prima parte facoltativa, il sistema deve essere esteso per dar la possibilità ai droni di ricaricare il proprio livello di batteria. Tale operazione potrà essere svolta da un solo drone alla volta. E' possibile richiedere di ricaricare il livello di batteria di un drone tramite richiesta esplicita da linea di comando (e.g., *recharge*). Quando si verifica tale evento, il drone deve richiedere agli altri droni di poter ricaricare la propria batteria, tramite l'algoritmo *Ricart and Agrawala* visto durante il corso di teoria. Quando ad un

drone viene concessa la possibilità di ricaricarsi, questo simulerà il processo di ricarica con una `Thread.sleep()` di 10 secondi. Durante questa fase, il drone non può ricevere consegne da effettuare. Terminata la fase di ricarica, il drone comunica al master della smart-city:

- la sua nuova posizione all'interno della smart-city, ovvero (0,0). Questa è la posizione in cui i droni possono ricaricare il proprio livello di batteria
- il suo nuovo livello di batteria residua, ovvero 100%

11.2 Seconda parte

Nello sviluppo del progetto abbiamo assunto che i processi terminino solo in maniera controllata. In questa parte facoltativa, questa assunzione decade per i droni. Quindi, sarà necessario implementare un protocollo distribuito decentralizzato che permetta alla rete di capire quando un drone ha terminato la propria esecuzione in maniera incontrollata. In questi casi, la rete di droni deve aggiornarsi ed avvisare il server amministratore dell'uscita incontrollata del drone dalla smart-city. Tutti i meccanismi di sincronizzazione distribuita previsti dal progetto devono tener conto della possibile terminazione incontrollata degli altri droni.

12 Aggiornamenti

Qualora fosse necessario, il testo dell'attuale progetto verrà aggiornato al fine di renderne più semplice l'interpretazione. Le nuove versioni del progetto verranno pubblicate sul sito del corso. Si consiglia agli studenti di controllare regolarmente il sito.