# Esercitazione di Sistemi Distribuiti e Pervasivi

## Suggerimenti di Sviluppo e Progettazione II

Gabriele Civitarese
gabriele.civitarese@unimi.it

EveryWare Lab
Università degli Studi di Milano
-
Docente: Claudio Bettini

Slide tratte da versioni precedenti di Letizia Bertolaja e Dario Freni

# Disclaimer

- Il contenuto di questi lucidi è da considerarsi puramente indicativo
  - Verranno forniti dei suggerimenti di sviluppo e di progettazione
  - Ogni studente può effettuare scelte differenti
- Le direttive su come svolgere il progetto sono riportate nel testo del progetto presente sul sito del corso.

# La Rete di Nodi Edge

- A tutti gli effetti è una rete P2P
  - Ogni singola applicazione *Nodo edge* sarà sia client che server
- Ogni nodo edge deve essere simulato da un singolo **processo** (miraccomando, non un thread!)
- Ogni nodo edge riceve misurazioni da un certo numero di sensori
- I nodi si sincronizzano tra di loro per inviare le misurazioni al *server cloud*
  - La comunicazione tra nodi avviene direttamente tramite socket o grpc (a vostra scelta)
  - La comunicazione con il *server cloud* avviene tramite librerie per chiamate REST
  - Adottate SEMPRE formati di dati standard (XML, JSON, Protocol Buffer,...)

## Come si crea la rete?

- All'inizio del sistema la griglia è vuota e il sever cloud è pronto a ricevere richieste di inserimento
- Ogni volta che facciamo partire un nodo edge, automaticamente tenta di entrare nella griglia
- Se la registrazione va a buon fine, il nodo riceve la lista degli altri nodi partecipanti
  - Se è il primo nodo della rete, si auto proclama coordinatore
  - Altrimenti, deve comunicare con gli altri nodi per entrare nella rete
- Il protocollo di entrata nella rete dipende dalla gestione della topologia
  - Come strutturiamo la rete?
  - Tutti i nodi devono conoscersi tra di loro?

# La rete (cont.)

## Uscita dalla rete ed elezione

- Un *nodo edge* può esplicitamente uscire dalla rete dei nodi edge
- In questo caso deve solo comunicare l'uscita al server cloud e chiudere tutte le comunicazioni con l'esterno
- Gli altri *nodi edge* della rete devono accorgersi della sua uscita
  - Come?
- Se esce il coordinatore, gli altri *nodi edge* della rete devono accorgersene e indire un'elezione
- Implementare Bully o elezione ad anello
  - Vietato usare soluzioni centralizzate

# Problemi da considerare

## Casi limite

- Cosa succede quando due nodi edge entrano contemporaneamente nella griglia?
- Cosa succede quando un nodo edge entra nella rete mentre qualche altro nodo esce?
- Cosa succede quando un nodo entra durante un'elezione?
- Cosa succede quando un nodo esce durante un'elezione?
- Come gestire le statistiche prodotte durante un'elezione?
- ...

Risulta fondamentale considerare tutti i possibili casi limite! In fase di discussione del progetto vi verranno **sicuramente** richiesti. Testateli con le *sleep()*.

# Simulatore di sensori

## Implementare il simulatore di sensori

- Per semplificare lo sviluppo, ogni sensore è gestito come un thread
  - Teoricamente dovrebbero essere processi
- È quindi semplicemente necessario lanciare tanti thread di simulazione quanti sono i sensori specificati da argomento
- La parte più sostanziosa è l'implementazione dell'interfaccia *SensorStream*

## Come implementiamo l'interfaccia?

- Ogni stream deve avere un puntatore al nodo edge di riferimento
- Questo riferimento è dinamico e soggetto a variazioni
  - Potrebbe addirittura non esserci un nodo edge assegnato
  - Cosa facciamo in questo caso con le misurazioni?
- È necessario gestire il caso in cui un *nodo edge* non sia più raggiungibile
- Problemi di sincronizzazione?
- La richiesta continua del *nodo edge* più vicino è l'unico caso di busy waiting ammesso
  - È stato deliberatamente inserito nel testo del progetto solo per semplificarvi lo sviluppo
  - Non è realistico e non è una best practice
  - Va gestito per forza su un thread separato?

# Misurazioni

- Ogni misurazione è caratterizzata da
  - Id sensore
  - Tipologia di sensore
  - Valore letto
  - Timestamp
- Ogni sensore manda in stream le proprie misurazioni ad un *nodo edge*
- Il *nodo edge* deve versare le misurazioni provenienti dai vari sensori su un unico buffer
  - Usato per calcolare periodicamente le statistiche che vengono inviate al coordinatore
  - Problemi di sincronizzazione?

# Statistiche locali e globali

## Calcolo statistiche

- Ogni *nodo edge* calcola periodicamente (ogni 40 misurazioni) statistiche locali
- Ogni volta che viene calcolata una statistica locale, viene mandata al coordinatore
  - Cosa fare della statistica locale se il coordinatore non risponde?
- Il coordinatore risponde con la più recente statistica globale
- Il coordinatore riceve statistiche locali e ogni 5 secondi computa la statistica globale
  - Solo le più recenti
  - Evitare di comunicare una stessa statistica locale più volte
- Il coordinatore trasmette statistiche locali (compresa la propria) e globali al server cloud
- Risulta fondamentale capire come assegnare timestamp (o intervalli temporali?) alle statistiche

# Gestione messaggi

- All'arrivo di un qualsiasi messaggio, il nodo edge deve esaminarlo
- Sviluppando solo server multithread, la ricezione dei messaggi è concorrente
- Un buffer condiviso di messaggi e un pool di thread per gestirli possono essere una buona soluzione
- Una buona stesura del protocollo di comunicazione può semplificare il codice che esamina i messaggi in ingresso
  - esempio di formato messaggi:
    `header content [parameters] timestamp`
- Può essere conveniente considerare i diversi *stati* dell'applicazione durante l'interpretazione dei messaggi

# Pattern Singleton

- In diversi problemi di sincronizzazione è necessario un oggetto di coordinazione
- In un sistema con un dispatcher e molti thread, il problema è riferirsi alla stessa istanza
- Possibilità:
  - Il dispatcher crea l'oggetto.
  - I working thread ricevono i riferimenti degli oggetti condivisi tramite il costruttore
  - Problema: all'aumentare del numero di riferimenti la gestione degli oggetti condivisi si complica.
- Soluzione:
  - Ogni thread si "prende" da solo l'oggetto.
  - Uso del pattern Singleton per accedere all'unica istanza di una determinata classe

# Pattern Singleton (2)

```
public class Singleton {
        private static Singleton instance = null;

        public synchronized static Singleton getInstance() {

                if (instance==null)
                        instance = new Singleton();

                return instance;
        }


        //Il costruttore private impedisce la creazione di istanze da parte di classi esterne
        private Singleton() {/*...some code...*/}


        ...
}
```

Buon lavoro!