Progetto di Sistemi Distribuiti e Pervasivi

Laboratorio di Sistemi Distribuiti e Pervasivi

Aprile 2019

1 Descrizione del progetto

Lo scopo del progetto è quello di realizzare un sistema per la gestione intelligente dell'energia elettrica prodotta da un complesso di case. Il consumo elettrico di ogni casa viene misurato costantemente da uno *smart meter* che monitora l'utilizzo degli elettrodomestici. L'architettura del sistema da sviluppare è mostrata in Figura 1. Il complesso di case costituisce una

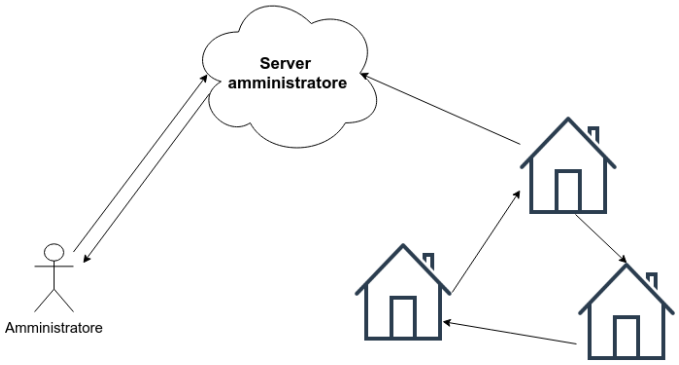

Figura 1: Architettura

rete peer-to-peer dinamica volta alla gestione decentralizzata del consumo elettrico. All'interno di questa rete, ogni *Casa* deve informare il vicinato del suo consumo elettrico in tempo reale, in modo tale che ogni casa possa calcolare il consumo energetico complessivo. Inoltre, quando una casa necessita di consumare una quantità di energia superiore alla norma, deve prima chiedere il consenso alle altre case in modo tale da non superare la quota complessiva prevista per il complesso. Le informazioni relative al consumo di corrente condominiale vengono periodicamente fornite dalle case ad un server remoto, chiamato *Server amministratore*, tramite il quale gli amministratori possono consultare i consumi energetici per calcolare le spese. Il *Server amministratore* permette anche alle case di aggiungersi o rimuoversi alla rete peer-to-peer del condominio in modo dinamico.

2 Applicativi da implementare

Per il progetto è necessario implementare separatamente i seguenti applicativi:

- *Casa*: una specifica casa del complesso condominiale
- *Server amministratore*: server REST che riceve i dati dalle case e che permette la gestione dinamica della rete peer-to-peer delle case
- *Amministratore*: un client che interroga il *Server amministratore* per interrogare i dati sul consumo energetico

È quindi importante notare che ogni *Casa* è un **processo** a sè stante, e non deve quindi essere implementata come un thread. In seguito vengono forniti dettagli sugli applicativi da realizzare.

3 Casa

Ogni *casa* è un **processo** che si occupa di analizzare le misurazioni dello smart meter e di coordinarsi con le altre case per a) calcolare in modo distribuito il consumo energetico complessivo, b) inviare statistiche al *server amministratore* e c) richiedere al condominio di poter consumare per un limitato lasso di tempo una quantità di corrente elettrica superiore alla media.

3.1 Inizializzazione

Una *casa* deve essere inizializzata specificando il suo identificatore, il numero di porta di ascolto per la comunicazione tra *case* e l'indirizzo del *server amministratore*. Una volta avviato, il processo *casa* deve avviare il suo smart meter (vedi Sezione 3.2) e registrarsi al condominio tramite il *server amministratore*. Se l'inserimento va a buon fine (ovvero, non esistono altre case con lo stesso identificatore), la *casa* riceve dal server l'elenco di *case* presenti nella rete peer-to-peer, in modo tale da poter presentarsi per entrare in maniera decentralizzata nella rete. Quindi, la rete peer-to-peer di case deve costruirsi senza alcun ausilio da parte del *Server Amministratore*, il cui unico compito è quello di fornire l'elenco di case già presenti nel condominio.

3.2 Smart meter

Ogni casa ha un suo *smart meter* che produce periodicamente delle misurazioni riguardanti il consumo di corrente in tempo reale. Ogni singola misurazione è caratterizzata principalmente da:

- Valore letto (in kiloWatt)

- Timestamp in millisecondi

La generazione di queste misurazioni viene svolta da un opportuno simulatore, il cui codice viene fornito per semplificare lo sviluppo del progetto. Il simulatore assegna come timestamp alle misurazioni il numero di secondi passati dalla mezzanotte.
Al link *http://ewserver.di.unimi.it/sdp/simulation_src_2019.zip* è quindi possibile trovare il codice necessario. Questo codice andrà aggiunto come package al progetto e NON deve essere modificato. In fase di inizializzazione, ogni casa dovrà quindi occuparsi di far partire il suo simulatore per generare misurazioni di consumo di corrente.

Il 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 il seguente metodo:

- $void\ addMeasurement(Measurement\ m)$

È dunque necessario creare una classe che implementi l'interfaccia.

Il thread del simulatore usa il metodo $addMeasurement$ per riempire il buffer. Ogni casa deve produrre periodicamente una statistica relativa al suo consumo energetico. Per questo motivo, deve effettuare periodicamente una media dei dati nel buffer. In particolare, è necessario implementare la tecnica della *sliding window* presentata durante le lezioni di teoria, considerando una dimensione del buffer di 24 misurazioni e un overlap del 50%.

3.3 Sincronizzazione distribuita

3.3.1 Calcolo del consumo complessivo

Ogni volta che una casa produce una media (come spiegato precedentemente), deve essere immediatamente comunicata alle altre case del condominio insieme all'istante di tempo in cui è stata calcolata (timestamp). È necessario progettare un'architettura che permetta la condivisione peer-to-peer dei dati riguardanti il consumo energetico. Ogni *casa* deve periodicamente stampare a schermo il consumo complessivo del condominio. Il consumo complessivo deve essere calcolato ogni volta che si ottengono nuove informazioni per ogni casa nella rete. Una specifica statistica relativa al consumo di una casa non deve contribuire per più di una volta al calcolo del consumo complessivo condominiale.

3.3.2 Invio delle informazioni al server

Le case devono anche coordinarsi per inviare periodicamente al *Server Amministratore* il consumo di ogni casa (statistica locale) e il consumo complessivo del condominio (statistica globale). Implementare una soluzione che

permetta al gruppo di *case* di non trasmettere al server queste informazioni in modo ridondante.

3.3.3 Richiesta di consumo superiore alla norma

Il condominio permette di sfruttare, per periodi limitati, $6kW$ da condividere tra tutte le case per gestire consumi di corrente extra. Questi kilowatt possono essere usufruiti previo accordo con tutti i condomini, in modo da non eccedere il tetto masssimo. Da linea di comando, una *Casa* deve fornire un'interfaccia per richiedere al condominio il consenso per il consumo extra di corrente elettrica. Questa richiesta implica di richiedere al condominio di poter sfruttare $3kW$ per 5 secondi. Si implementi uno degli algoritmi di mutua esclusione distribuita visti a lezione (a scelta tra *Token Ring* e *Ricart-Agrawala*) per fare in modo che il condominio garantisca che il consumo extra di corrente non superi il tetto massimo. Quindi, non più di due case contemporaneamente possono richiedere un consumo superiore alla media. È necessario prevedere un sistema di coda per gestire le richieste che non possono essere accontentate immediatamente. Una volta che una *Casa* ha ottenuto il permesso dal condominio, deve invocare il metodo *boost*() del simulatore. Questo metodo incrementa automaticamente il consumo energetico di $3kW$ per 5 secondi. Una volta terminata l'esecuzione del metodo *boost*(), la casa deve avvisare il condominio che ha terminato l'utilizzo della corrente extra, in modo da permettere ad eventuali altre case in coda di sfruttare la corrente extra a disposizione.

3.4 Uscita dalla rete

Ogni casa può decidere esplicitamente di uscire dalla rete peer-to-peer in qualsiasi momento. L'applicazione *Casa* deve quindi predisporre un'interfaccia a linea di comando per permettere di inserire un comando relativo all'uscita dalla rete. Quando questo comando viene inserito, prima di terminare la casa deve comunicare al *Server Amministratore* e alle altre case della sua uscita. Si assume che una *casa* non possa uscire dalla rete in modo incontrollato.

4 Server amministratore

Il *server amministratore* si occupa di ricevere dal condominio le statistiche che verranno poi interrogate dagli amministratori. Inoltre, si occupa di gestire l'ingresso e l'uscita di case dalla rete peer-to-peer. Deve quindi offrire diverse interfacce REST per: a) gestire la rete di case, b) ricevere le statistiche sul consumo elettrico e c) permettere agli amministratori di effettuare interrogazioni. Il *server amministratore* non deve in alcun modo coordinare

la topologia di comunicazione della rete peer-to-peer delle case e non deve occuparsi di risolvere problemi di mutua esclusione distribuita.

4.1 Interfaccia per le case

4.1.1 Inserimento

Quando vuole inserirsi nel sistema, una *casa* deve comunicare al server:

- Identificatore
- Indirizzo IP
- Numero di porta sul quale è disponibile per comunicare con le altre *case*

Il server aggiunge una *casa* al suo elenco di case presenti nel conodminio solo se non esiste un'altra casa con lo stesso identificatore. Se l'inserimento va a buon fine, il *server amministratore* restituisce alla *casa* la lista di *case* già presenti nella rete, specificando per ognuno indirizzo IP e numero di porta per la comunicazione.

4.1.2 Rimozione

Una *casa* può richiedere esplicitamente di rimuoversi dalla rete. In questo caso, il server deve semplicemente rimuoverla dall'elenco di case del condominio.

4.1.3 Statistiche

Il *server amministratore* deve predisporre un'interfaccia per ricevere dalla rete le statistiche riguardanti il consumo elettrico condominiale. Il server riceverà sia statistiche globali (relative all'intero condominio) che locali (di ogni zona casa). È sufficiente memorizzare queste statistiche in una struttura dati che permetta successivamente l'analisi da parte degli amministratori.

4.2 Interfaccia per gli amministratori

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

- L'elenco delle case presenti nella rete
- Ultime n statistiche (con timestamp) relative ad una specifica *casa*
- Ultime n statistiche (con timestamp) condominiali
- Deviazione standard e media delle ultime n statistiche prodotte da una specifica *casa*

- Deviazione standard e media delle ultime n statistiche complessive condominiali

5 Amministratore

L'*Applicazione amministratore* è un'interfaccia a linea di comando che si occupa di interagire con l'interfaccia REST precedentemente presentata in Sezione 4. L'applicazione deve quindi semplicemente mostrare all'amministratore un menu per scegliere uno dei servizi di analisi offerto dal server, con la possibilità di inserire eventuali parametri che sono richiesti.

6 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 i 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**, escludendo quindi la parte di puntatori a strumenti avanzati di sincronizzazione. 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.

Nonostante alcune problematiche di sincronizzazione possano essere risolte tramite l'implementazione di server iterativi, per fini didattici si richiede di utilizzare server multithread. Inoltre, per le comunicazioni via socket, è richiesto di usare formati standard (ad esempio JSON, XML o Protocol Buffer) per lo scambio di dati. Qualora fossero previste comunicazioni in broadcast, queste devono essere effettuate in parallelo e non sequenzialmente. Alternativamente alle socket, è permesso utilizzare il framework *grpc* per la comunicazione tra processi. Lo studente può scegliere liberamente quale

delle due tecnologie utilizzare (socket, *grpc* o entrambe). In fase di presentazione del progetto, lo studente è comunque tenuto a conoscere almeno a livello teorico entrambe le tecnologie.

7 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). E' necessario presentare il progetto sul proprio computer.

Il codice sorgente dovrà essere consegnato al docente prima della discussione del progetto. Per la consegna, è sufficiente archiviare **esclusivamente il codice sorgente** 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.

Nel caso in cui il docente si accorga che una parte significativa del progetto consegnato non è stata sviluppata dallo studente titolare dello stesso, lo studente in causa: a) dovrà superare nuovamente tutte le prove che compongono l'esame del corso. In altre parole, se l'esame è composto di più parti (teoria e laboratorio), lo studente dovrà sostenere nuovamente tutte le prove in forma di esame orale (se la prova di teoria fosse già stata superata, questa verrà annullata e lo studente in merito dovrà risostenerla). b) non potrà sostenere nessuna delle prove per i 2 appelli successivi. Esempio: supponiamo che gli appelli siano a Febbraio, Giugno, Luglio e Settembre, e che lo studente venga riconosciuto a copiare all'appello di Febbraio. Lo studente non potrà presentarsi agli appelli di Giugno e Luglio, ma potrà sostenere nuovamente le prove dall'appello di Settembre in poi. Il docente si riserva la possibilità di assegnare allo studente in causa lo svolgimento di una parte integrativa o di un nuovo progetto.

8 Parti facoltative

8.1 Prima parte

In questa parte facoltativa, si vuole estendere il sistema con un sistema di notifiche push per gli amministratori. È necessario realizzare un'architettura che permetta agli amministratori di ricevere una notifica quando:

- Una nuova casa entra nella rete
- Una casa esce dalla rete
- Vengono richiesti consumi extra di corrente

Quando l'applicazione Amministratore riceve una notifica push, deve semplicemente stamparla su schermo. La soluzione proposta deve evitare il polling da parte degli amministratori.

8.2 Seconda parte

Una delle assunzioni nello sviluppo del progetto è che i processi non possono terminare in maniera controllata. In questa parte facoltativa, questa assunzione per le *case* non vale più. È quindi possibile che una *Casa* termini in modo incontrollato, senza riuscire a comunicare al *Server Amministratore* e alle altre case della sua uscita. Implementare quindi un protocollo distribuito decentralizzato che permetta alla rete di capire quando una *casa* ha concluso in maniera incontrollata la sua esecuzione. Quando questo accade, la rete peer-to-peer deve aggiornarsi e avvisare il *server amministratore*. È quindi necessario che tutti i meccanismi di sincronizzazione distribuita previsti dal progetto prevedano la terminazione incontrollata di altre case.

9 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.

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.