RaftShop Versione Specifiche: 1.2

Progetto di Laboratorio di Sistemi Operativi A.A. 2016-2017
Saverio Giallorenzo | Università di Bologna

1 Descrizione

RaftShop è un sistema distribuito *fault-tolerant* (tollerante ai guasti) per il commercio elettronico. L'architettura di RaftShop è composta da 3 entità principali: i Client, i Server e l'Administrator. A questi si aggiunge il Network Visualiser, un tool amministrativo per monitorare lo stato dei Server.

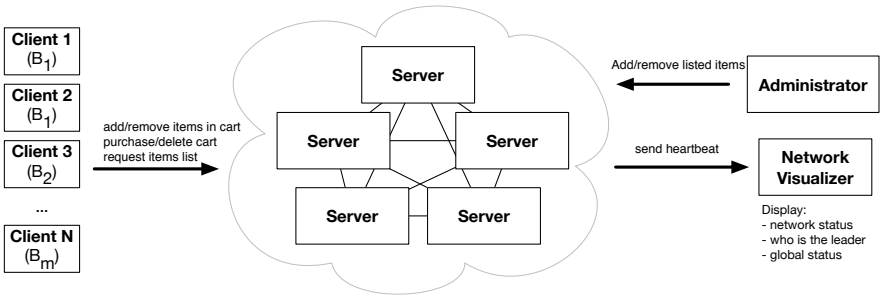

Figure 1: Rappresentazione del sistema RaftShop

1.1 Client

I Client rappresentano i clienti di RaftShop. I clienti possono:

Richiedere la lista delle merci vendute in RaftShop. La risposta contiene i nomi dei beni e le relative quantità disponibili.

Richiedere gli attuali elementi in un carrello.

Registrare un nuovo carrello presso i Server. Il nome del carrello è unico all'interno di tutto il sistema. Più clienti possono intervenire sullo stesso carrello, indicandone il nome.

Aggiungere/Rimuovere elementi in un carrello, inviando una richiesta contenete nome del carrello, nome del bene e quantità relativa. Se possibile, la richiesta viene esaudita, i.e., se i prodotti richiesti sono in vendita, per l'aggiunta, e se gli elementi sono nel carrello, per la rimozione. In caso di aggiunta, gli oggetti inseriti nel carrello vengono rimossi dalla disponibilità generale dei Server. In caso di rimozione, gli oggetti rimossi ritornano ad essere disponibili presso i Server.

Acquistare o Cancellare il Carrello Un cliente può decidere di acquistare o cancellare il carrello. Se acquistato, il carrello va in archivio e il relativo nome rimane riservato, gli oggetti acquistati vengono rimossi dal sistema. Se cancellato, il carrello viene rimosso e il relativo nome ritorna disponibile; gli oggetti ivi presenti ritornano disponibili.

Il Client offre una semplice interfaccia utente (via terminale) per l'inserimento e la valutazione dell'esito dei comandi.

1.2 Administrator

L'Administrator rappresenta l'amministrazione del sistema di commercio elettronico che può **aggiungere** e **rimuovere oggetti** all'interno del sistema. Oltre ad una semplice interfaccia utente (via terminale), può prevedere, all'avvio, il caricamento automatico di una serie di prodotti prestabiliti (e.g., 5 tipi di prodotti con diverse quantità).

1.3 Server

I Server costituiscono il sistema di commercio elettronico distribuito. Oltre all'interazione coi Client e l'Administrator, i Server costituiscono un sistema distribuito tollerante ai guasti, implementando un archivio ridondato: *i*) degli oggetti venduti (nomi e quantità), *ii*) dei carrelli aperti coi relativi oggetti e *iii*) dei carrelli acquistati (i.e., archiviati).

Per implementare l'archivio ridondato, i Server di RaftShop implementano Raft[1], un semplice e sicuro algoritmo per il consenso distribuito. La spiegazione dettagliata del protocollo è alla Sezione 1.4.

1.3.1 Network Visualizer

Il Network Visualiser è un tool amministrativo da terminale per il monitoraggio dei Server. Quando un Server diventa Leader (vedere § 1.4) invia un messaggio a cadenza costante al Network Visualizer riportando lo stato del network, tra cui: la propria identità come Leader, quali Server sono attivi e quali vengono ritenuti inattivi e lo stato globale del sistema (e.g., i carrelli attivi col relativo contenuto, quelli archiviati, etc.). Sulla base dei dati ricevuti, il Network Visualiser visualizza lo stato del sistema. Non si indicano particolari formattazioni per tale visualizzazione, l'importante è che sia possibile utilizzare detta visualizzazione per constatare lo stato del sistema.

1.4 Raft

RaftShop implementa il consenso distribuito tra i propri Server tramite Raft. Seguendo tale algoritmo, in RaftShop un Server può diventare il Leader che gestisce la ricezione, l'esecuzione e la replicazione sicura (sugli altri Server) dei comandi inviati dai Client e dall'Administrator. In RaftShop, i Client e l'Administrator possono interagire solo con il Leader. Se un Client cerca di interagire con un Server diverso dal Leader, il Server interrogato risponde con l'indirizzo del Leader. Il Leader decide quando gli altri Server, chiamati *Followers*, possono replicare in maniera sicura i comandi inviati. Tale processo avviene seguendo l'algoritmo distribuito di Raft, basato sulla duplicazione sicura di un *log* (registro) contenete i comandi da replicare. Dato che RaftShop contempla la caduta dei Server, è possibile che il Leader cada. In generale, in assenza di un Leader, i Server indicono nuove elezioni per individuarne uno nuovo. Raft decompone il problema del consenso distribuito in tre sotto-problemi: *i*) Elezione del Leader; *ii*) Replicazione del Log; *iii*) Controllo di Sicurezza.

Concetti Base di Raft Ogni Server può transitare in uno di tre stati: Leader, Follower e Candidato. Normalmente esiste un Leader e tutti gli altri Server sono suoi Followers. Raft suddivide il tempo in *termini* di lunghezza arbitraria. I termini vengono indicati da un intero monotono crescente (1,2,3,...). Ogni temine inizia con un'elezione nella quale i Candidati

[1] https://raft.github.io/raft.pdf

cercano di diventare il Leader. Se nessun candidato ha la maggioranza assoluta dei voti, il termine passa vacante, si inizia un nuovo termine e si procede ad una nuova elezione.

I termini permettono ai Server di capire se un Candidato possiede più o almeno la stessa quantità di informazione rispetto a loro. Ogni Server conserva il valore del *termine corrente* a cui ha partecipato. Il valore del termine corrente viene scambiato ad ogni comunicazione tra i Server. Un Server che riceve una comunicazione da un altro Server con un termine corrente maggiore aggiorna il proprio termine corrente al nuovo valore. Se un Server scopre di avere un termine corrente vecchio, diventa un Follower. Se un Server riceve una comunicazione con un termine corrente vecchio, rigetta la richiesta.

I Server comunicano usando due operazioni: `RequestVote` e `AppendEntries`. `RequestVote` viene usata dai Candidati per richiedere voti dagli altri Server durante le elezioni. `AppendEntries` viene usata dal Leader per replicare il log.

Elezione del Leader Raft usa un meccanismo di heartbeat[2] per iniziare l'elezione del Leader. Quando un Server parte, si trova nello stato di Follower. Un Server rimane nello stato di Follower finché riceve comunicazioni valide da un Leader o dai Candidati. Un Leader invia delle comunicazioni periodiche tramite l'operazione `AppendEntries` (con nessuna nuova voce del log da aggiungere) a tutti i Server per mantenere la propria leadership. Se un Follower non riceve comunicazioni entro un tempo casuale chiamato *Election Timeout*, il Server assume che il Leader sia caduto, diventa un Candidato e inizia una nuova elezione.

Per iniziare una nuova elezione, il Candidato incrementa il proprio termine corrente, vota per sé stesso e invia in parallelo a tutti i Server la richiesta di voto tramite l'operazione `RequestVote`. Un Candidato rimane tale finché: *a*) il Candidato vince l'elezione; *b*) un altro Candidato vince l'elezione; *c*) l'elezione si conclude senza vincitore.

Nel caso *a*), un Candidato vince un elezione se riceve la maggioranza assoluta dei voti dei Server nell'ambito dello stesso termine. Ogni Server può votare per un solo Candidato nell'ambito di un termine. Il voto avviene con politica first-come-first-served (la prima candidatura ricevuta eleggibile viene votata). Quando un Candidato vince l'elezione, manda a tutti i Server il proprio heartbeat, prevenendo l'avvio di una nuova tornata elettorale.

Nel caso *b*), un Candidato ha richiesto voti, riceve una comunicazione di `AppendEntries` con un termine maggiore o uguale al suo termine corrente, riconosce il nuovo Leader e diventa Follower. Se al contrario il termine di `AppendEntries` è minore del termine corrente del Candidato, questo scarta la richiesta e continua ad attendere l'esito del voto.

Nel caso *c*), a causa di una dispersione di voti non è stato possibile eleggere un candidato. In questo caso il termine passa vacante, i Candidati incrementano il proprio termine corrente e ripetono la procedura di richiesta voti. Per evitare il possibile ripetersi infinito di termini vacanti, Raft usa degli Election Timeout casuali (ad ogni timeout, per ogni Server) entro un intervallo fisso, e.g., 150–300ms. Questo incrementa le probabilità che i timeout avvengano scaglionati, permettendo ad un solo Server di diventare Candidato, richiedere le elezioni e diventare Leader prima che un altro Server vada in timeout e richieda le elezioni a sua volta.

Lo stesso meccanismo viene inoltre usato per gestire le elezioni vacanti. Alla richiesta di voti, i Candidati fanno partire un Election Timeout. Se, prima del timeout, il Candidato riceve o la maggioranza assoluta dei voti o l'heartbeat valido di un Leader, l'elezione è considerata riuscita. Al contrario, allo scattare del timeout il termine passa vacante e viene iniziata un'elezione.

Replicazione del Log Quando un Leader viene eletto, questo serve le chiamate dei Client e dell'Administrator. Ognuna di queste chiamate contiene un comando che modifica la lista

[2] https://en.wikipedia.org/wiki/Heartbeat_(computing)

di prodotti, quella dei carrelli attivi/archiviati o il contenuto di un carrello. Una volta ricevuto il comando, il Leader lo aggiunge al proprio log come una nuova voce.

Il log è visto come un vettore dove ogni voce aggiunta è associata a *i*) un indice intero incrementale corrispondente alla posizione della voce nel log, *ii*) il termine in cui il comando è stato ricevuto e *iii*) il comando ricevuto.

Aggiunta la voce nel proprio log, il Leader invia a tutti i Followers in parallelo il comando di `AppendEntries` per replicare la nuova voce. Quando la voce è stata *correttamente replicata* (chiarito di seguito), il comando ad essa relativo viene eseguito dal Leader. Al termine dell'esecuzione del comando, il Leader riporta il successo dell'esecuzione al Client che ha inviato il comando. Il Leader giudica un log correttamente replicato se la maggioranza dei Server risponde positivamente alla richiesta di replicazione della voce del log.

Inoltre, il Leader tiene traccia di due informazioni: 1) l'*indice dell'ultima voce eseguita* (i.e., giudicata correttamente replicata) e 2) il *termine* e l'*indice* della voce nel log *immediatamente precedente* alla nuova voce da aggiungere. Entrambe le informazioni vengono incluse in ogni comando di `AppendEntries` (heartbeat inclusi).

1) viene usato dai Server per conoscere quali comandi contenuti nel proprio log possono eseguire. L'esecuzione avviene in maniera sequenziale, partendo dall'indice della propria ultima voce eseguita fino ad arrivare alla voce indicata nel comando di *AppendEntries*.

2) vengono usati per un controllo di consistenza da parte dei Server: alla ricezione, il Server controlla che l'indice dell'ultima voce nel proprio log ed il suo corrispondente termine corrispondano con quelli in 2). Se questo è falso, la richiesta viene scartata.

Per correggere eventuali disallineamenti, Raft considera come valida la versione del log del Leader. In caso di disallineamenti, il Leader deve trovare l'ultima voce del log sulla quale il proprio log e quello del Server disallineato concordano, rimuovendo eventuali voci presenti dopo tale voce e mandando un'apposita operazione di `AppendEntries` che rispetti il controllo di consistenza. Per questo il Leader mantiene lo stato del *prossimo indice* del log di ogni Follower. Se il Leader riceve una risposta negativa ad un comando di `AppendEntries` da un Follower, decrementa il valore del prossimo indice del Follower e invia un nuovo comando di `AppendEntries` considerando la voce all'indice precedente come nuova rispetto al log del Follower. Il ciclo viene ripetuto fino a riportare il log del Follower in pari rispetto a quello del Leader.

Controllo di Sicurezza Raft aggiunge alcuni controlli di sicurezza per prevenire inconsistenze dovute, e.g., a un Server rimasto disallineato che diventa Leader, portando alla possibile inconsistenza nello stato dei Followers (che hanno già eseguito dei comandi che il nuovo Leader non contiene nel proprio log). Per questo, Raft limita l'eleggibilità dei Candidati a solo coloro il cui stato corrisponde all'esecuzione di tutte le voci considerate eseguite dai votanti. Tale controllo viene fatto sui valori di 2) (i.e., l'indice e il termine dell'ultima voce nel log): se tali valori sono almeno uguali a quelli del Server votante, questo da il proprio voto al Candidato (sempre seguendo la regola definita al paragrafo sull'elezione del Leader).

Tabelle Riassuntive Si riportano di seguito le tabelle riassuntive dei concetti su Raft spiegati ai paragrafi precedenti.

Stato del Server

currentTerm l'ultimo termine visto dal Server. Viene inizializzato a 0 e cresce in modo monotono.
votedFor l'indirizzo del candidato a cui è stato dato il voto nel termine corrente (non definito in caso contrario).
log le voci nel log; ogni voce è associata a un comando, un termine ed un indice monotono crescente (partendo a 1).
commitIndex l'indice dell'ultima voce del log da eseguire (partendo da 0, crescendo in maniera monotona).
lastApplied l'indice dell'ultima voce del log eseguita (partendo da 0, crescendo in maniera monotona).
nextIndices l'associazione tra ogni Follower e l'indice immediatamente successivo della voce nel log da mandare a quest'ultimo (presente solo nel Leader; inizializzato all'indice dell'ultima voce del log del Leader + 1; inizializzato dopo ogni elezione);
matchIndices l'associazione tra ogni Follower e l'indice dell'ultima voce replicata del log del Leader (presente solo nel Leader; inizializzato a 0 ad ogni elezione; incrementato in maniera monotona)

Operazione `AppendEntries`

Argomenti
term il termine del Leader
leaderID l'indirizzo del Leader
prevLogIndex l'indice della voce del log immediatamente precedente le nuove voci da aggiungere
prevLogTerm il termine della voce del log indicata da `prevLogIndex`
entries le voci del log da includere (vuoto per l'heartbeat)
leaderCommit `commitIndex` del Leader

La risposta dei Followers può contenere:
- **Term**: `currentTerm` del Follower per permettere al Leader di aggiornarne lo stato
- **Success**: `true` se il Follower includeva una voce corrispondente a `prevLogIndex` e `prevLogTerm`

La logica dei Follower per la risposta a `AppendEntries` è:
- Rispondi `false` se `term < currentTerm`;
- Rispondi `false` se il log non contiene una voce all'indice `prevLogIndex` con termine uguale a `prevLogTerm`;
- Se esiste una voce allo stesso indice di una voce da aggiungere e le due hanno termini diversi, cancella la voce esistente e tutte quelle successive ad essa;
- includi tutti le voci non ancora presenti nel log;
- se `leaderCommit > commitIndex`, assegnare a `commitIndex` il minimo tra `commitIndex` e l'indice dell'ultima voce inclusa.

Regole dei Server

Tutti i Server

Se `commitIndex > lastApplied`	incrementa `lastApplied` ed esegui l'istruzione della voce del log corrispondente (a `lastApplied`)
Se una richiesta o una risposta da un altro Server riporta un termine `term > currentTerm`	diventa Follower e assegna `currentTerm = term`

Followers

se `Election Timeout` scatta senza ricevere `AppendEntries` dal Leader o senza richieste di voto	diventa Candidato

Condidati

Diventando candidato	incrementa `currentTerm`; vota per sé stesso; resetta l'`Election Timeout`; invia in parallelo `RequestVote` agli altri Server
Se riceve la maggioranza dei voti dai Server	diventa Leader
Se riceve `AppendEntries` valido	diventa Follower
Se `Election Timeout` scatta	inizia una nuova elezione

Leader

Vincendo l'elezione	manda in parallelo a ogni Server un'`AppendEntries` con nessuna voce (heartbeat), fa partire un `Heartbeat Timeout` (il timer è resettato ogni volta che avviene una `AppendEntries`)
Se riceve un comando da un Client	Aggiunge il comando al log locale, notifica i Followers e risponde dopo aver eseguito il comando
Se `Heartbeat Timeout` scatta	manda in parallelo a tutti i Server un'`AppendEntries` con nessuna voce (heartbeat)
Se l'indice dell'ultima voce nel log del Leader è $\geqslant$ del `nextIndex` (all'interno di `nextIndices`) di uno dei Followers	manda `AppendEntries` con le voci del log del Leader che partono da `nextIndex`: se l'operazione ha successo, aggiorna `nextIndex` e `matchIndex` per il Follower; se l'operazione fallisce, decrementa `nextIndex` e ripete `AppendEntries`
Se per la maggioranza di Followers `matchIndex` $\geqslant$ `i`, con `i > commitIndex` e il termine della i-esima voce nel log è uguale a `currentTerm`	`commitIndex = i`

Operazione `RequestVote`

Argomenti
term il termine del Candidato
candidateID l'indirizzo del Leader richiedente il voto
lastLogIndex l'indice della voce del log immediatamente precedente le nuove voci da aggiungere
lastLogTerm il termine della voce del log indicata da `lastLogIndex`
La risposta dei Server può contenere:
- **Term**: `currentTerm` del Server per permettere al Candidato di aggiornarne lo stato
- **VoteGranted**: `true` se il Server vota per il Candidato

La logica dei Server per la risposta a `RequestVote` è:
- Rispondi `false` se `term < currentTerm`;
- se `votedFor` è non definito (o è uguale a `candidateID`), `lastLogIndex` è maggiore o uguale all'indice dell'ultima voce del log e `lastLogTerm` è maggiore o uguale al termine dell'ultima voce del log, vota per il Candidato.

2 Consegna del Progetto

2.1 Implementazione

Il progetto deve essere sviluppato utilizzando il linguaggio Jolie. Non ci sono requisiti riguardo ai protocolli (*Protocol*) e i media (*Location*) utilizzati per realizzare la comunicazione tra i componenti di RaftShop. L'obiettivo dell'implementazione riguarda la gestione concorrente di risorse condivise. L'importante è dimostrare che il comportamento implementato gestisca i tipici problemi di concorrenza su risorse condivise (e.g., il tipico problema Readers-Writers).

2.1.1 Reading-Writing

Uno degli argomenti di discussione per lo sviluppo del progetto è la gestione delle risorse come i comandi dei Clients e dell'Administrator. Il problema è riconducibile al Reader-Writer. È importante pensare a come sviluppare queste funzionalità tra i componenti che tentano di aggiungere/rimuovere prodotti sui Server o di aggiungere/archiviare/rimuovere carrelli e il loro contenuto in prodotti. Ai fini della valutazione è bene riportare le scelte fatte per implementare questi meccanismi.

N.B. Uno dei punti di valutazione riguarda l'uso del parallelismo nel progetto. E.g., usare `execution { sequential }` per prevenire scritture (e letture) parallele è una soluzione al problema considerato, ma rappresenta anche una notevole perdita in termini di concorrenza, incidendo negativamente sulla valutazione del progetto. Analogamente, il costrutto `synchronized( t ){ ... }` deve essere usato con raziocinio per massimizzare il grado di parallelismo del proprio progetto.

2.2 Report

Circa 4–5 pagine (8–10 facciate) – singola colonna, font-size 12 – in formato PDF. Nel Report vengono discusse:

- la struttura del progetto (la divisione delle funzionalità tra i servizi e la loro gerarchia);
- le scelte più importanti fatte sul progetto (e.g., qual'è la logica di gestione dei comandi ricevuti dai Server) e come sono state sviluppate le principali funzionalità (anche con snippets di codice);
- i problemi principali riscontrati, le alternative considerate e le soluzioni scelte;
- le istruzioni per eseguire il progetto (in generale) e quelle per eseguire una demo di esecuzione. La demo deve contenere almeno 2 Client, 1 Administrator, 5 Server e 1 Network Visualiser, per osservare l'esecuzione dei comandi. La sequenza di esecuzione suggerita è:
  - avvio dei Server;
  - avvio dell'Administrator (con eventuale caricamento batch di prodotti);
  - avvio dei Client e del Network Visualiser.

A questo indirizzo è disponibile un canovaccio, in markdown, con la struttura del report. (Pandoc e altri tools permettono di creare PDF da markdown). È possibile scrivere il report nel formato preferito (.doc, .tex), l'importante è che il PDF generato rispetti la struttura del succitato template.

Il report di progetto è molto importante e buoni progetti con report scadenti possono risultare insufficienti. Il report non deve essere la ripetizione delle specifiche, deve contenere le idee, le discussioni e le scelte progettuali, soprattutto per quelli considerati punti chiave.

Mantenere una documentazione completa e in linea con lo sviluppo è di estrema importanza per costituire uno storico delle scelte fatte, utile anche come knowledge-base per nuovi progetti (e.g., riuso di soluzioni, oltre che di codice).

È buona norma scrivere una bozza del report *durante* lo sviluppo del progetto, documentando le discussioni e le soluzioni adottate.

2.3 Gruppi

I gruppi possono essere costituiti da un minimo di 3 a un massimo di 5 persone. I gruppi che intendono svolgere questo progetto, devono comunicare via email a

saverio.giallorenzo@gmail.com

entro l'**11 Maggio 2016** la composizione del gruppo. L'email deve avere come oggetto "GRUPPO LSO" e contenere il **nome del gruppo** e una riga per ogni componente del gruppo, con cognome, nome e matricola. Inoltre deve essere presente un **indirizzo email di riferimento** a cui mandare le notifiche al gruppo. Per esempio:

```
Oggetto: Gruppo LSO
Corpo email:
  NomeGruppo
  - Cognome_1, Nome_1, Matricola_1
  - ...
  - Cognome_n, Nome_n, Matricola_n

Referente: nome.cognome@studio.unibo.it
```

Per completare l'iscrizione, ogni componente deve iscriversi su AlmaEsami alla prova "progetto AA 16/17". L'appello è unico e indipendente dalla date di consegna del progetto e della discussione. Chi non comunicherà la composizione del gruppo entro l'11 Maggio non potrà consegnare il progetto. Nel caso, **chi non riuscisse a trovare un gruppo**, lo comunichi il prima possibile, entro e non oltre il **9 Maggio 2016**, a saverio.giallorenzo@gmail.com, con una mail con oggetto "CERCO GRUPPO LSO", specificando i propri dati (nome-cognome-matricola-email) ed eventuali preferenze legate a luogo e tempi di lavoro. Si cercherà di costituire gruppi di persone con luoghi e tempi di lavoro compatibili. Le persone senza un gruppo verranno raggruppate il prima possibile (e non sarà possibile modificare i gruppi formati).

2.4 Consegna e Date

Lo sviluppo del progetto avviene col supporto di Git (guida), così come la consegna del progetto, che viene fatta tramite un Tag di Git (spiegato di seguito). I gruppi devono creare un repository su GitLab.com seguendo la procedura descritta sotto. Creazione del gruppo:

- ogni membro del gruppo crea un account su GitLab;
- il referente del gruppo:
  - crea un progetto cliccando sul "+" in alto a destra nella schermata principale di GitLab, inserendo il nome `LabSO_NomeGruppo` (dove NomeGruppo è il nome registrato in precedenza del gruppo) e cliccando su "Create project";
  - una volta che il progetto è stato creato, aggiunge membri al progetto andando su "Settings" > "Members";
  - aggiunge ogni membro del gruppo come "Developer", cercandoli in base allo username registrato su GitLab;
  - aggiunge l'utente "thesave" come "Reporter".

Al momento della consegna, il repository dovrà contenere i sorgenti del progetto (e.g., sotto la directory "progetto") e la relazione, nominata `REPORT_LSO.pdf`. È possibile inserire un README in "progetto" che riporti come lanciare gli eseguibili del progetto.

Per effettuare la consegna, il referente del gruppo crea un Tag (chiamato anche release) del progetto. Per creare un Tag del progetto:

- nella pagina di progetto su GitLab, cliccare sulle voci "Repository" > "Tags"> "New tag";
- digitare come "Tag name" il nome "Consegna" e lasciare gli altri campi vuoti;
- cliccare su "Create tag" per eseguire la creazione del Tag di consegna di progetto.

Una volta creato il Tag, inviare una email di *notifica* di consegna (quindi senza alcun codice sorgente) con soggetto "CONSEGNA LSO - NOME GRUPPO" (dove NOME GRUPPO è il nome registrato del gruppo) a saverio.giallorenzo@gmail.com.

Il report va anche consegnato in forma cartacea nella casella del prof. Sangiorgi (piano terra del Dipartimento di Informatica, a fianco dell'ufficio).

Ci sono due date di consegna:

- 23:59 di Lunedì 3 Luglio 2017;
- 23:59 di Lunedì 18 Settembre 2016.

Come data di consegna, farà fede la data di creazione del Tag su GitLab. In seguito alle consegne, verranno fissate data e ora della discussione del progetto (per la seconda data di consegna è possibile che la discussione venga fissata al mese successivo) compatibilmente coi tempi di correzione (FCFS). La data di discussione verrà notificata ai gruppi tramite la mail di riferimento.

2.4.1 Valutazione del progetto

Il progetto deve rispettare la consegna descritta nelle Sezioni 1. Il progetto verrà valutato mediante una discussione di gruppo, con tutti i componenti del gruppo presenti. Non è possibile per i componenti di un gruppo effettuare la discussione in incontri separati. Al termine della discussione, ad ogni singolo componente verrà assegnato un voto in base all'effettivo contributo dimostrato nel lavoro di progetto. La valutazione del progetto è indipendente dal numero di persone che compongono il gruppo.

2.4.2 Note Importanti

- non si accettano email o richieste di eccezioni sui progetti con motivazioni legate a esigenze di laurearsi o di non voler pagare le tasse per un altro anno.
- chi copia o fa copiare, anche un sola parte del progetto, si vedrà invalidare completamente il progetto senza possibilità di appello. Dovrà quindi rifare un nuovo progetto l'anno successivo.

2.5 Domande sul progetto e ricevimento

Per le domande sul progetto si consiglia di usare il newsgroup del corso. Risposte corrette alle domande dei colleghi sul newsgroup verranno valutate positivamente in sede di esame. È possibile chiedere informazioni o un ricevimento via email, specificando la motivazione della richiesta, a saverio.giallorenzo@gmail.com.