UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

Titolo

Sottotitolo (alcune volte lungo - opzionale)

Supervisore
Montresor Alberto

Laureando
Giust Alberto

Anno accademico 2018/2019

Ringraziamenti

...thanks to...

Indice

Sommario

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Sommario è un breve riassunto del lavoro svolto dove si descrive l'obiettivo, l'oggetto della tesi, le metodologie e le tecniche usate, i dati elaborati e la spiegazione delle conclusioni alle quali siete arrivati.

Il sommario dell'elaborato consiste al massimo di 3 pagine e deve contenere le seguenti informazioni:

- contesto e motivazioni
- breve riassunto del problema affrontato
- tecniche utilizzate e/o sviluppate
- risultati raggiunti, sottolineando il contributo personale del laureando/a

1 STEM, STEAM e computational STEAM

Il termine STEM indica un percorso di studi incentrato su quattro discipline: Scienze, Matematica, Tecnologia e Ingegneria [https://www.britannica.com/topic/STEM-education]. L'acronimo nasce nel 2001 negli Stati Uniti per raggruppare le principali aree di studio necessarie per lo sviluppo e l'innovazione nell'era dell'informazione. La sua evoluzione in STEAM vuole unire le quattro discipline sopra citate all'Arte, in modo da proporre un piano volto alla formazione di ragazzi in grado sia di applicare il metodo scientifico per la risoluzione di problemi più o meno complessi, sia di sviluppare la propria fantasia e il pensiero critico. Ai giorni nostri, infatti, non basta essere fortemente competenti in ambito scientifico e tecnologico, ma è necessario anche essere creativi, originali e innovativi. STEAM si pone quindi l'obiettivo di completare quello che STEM proponeva anni prima, integrandolo ed adattandolo per essere più efficace e aggiornato.

L'informatica, però, non è espressa direttamente in STEAM, ma è inglobata nel concetto di Technology. Questo può rivelarsi riduttivo se si associa questa disciplina all'aspetto meramente tecnico, in quanto essa si occupa dello studio di metodi generali per risolvere problemi mediante sistemi di calcolo, ed utilizza il calcolatore solamente come strumento [http://www.treccani.it/enciclopedia/informatica/] per accelerare il processo computazionale.

L'aspetto pratico e tecnico è sicuramente più visibile ai "non addetti ai lavori", ma non è l'unico: l'informatica può essere vista ad un livello più alto, come una scienza che si occupa di problem solving, risoluzione di problemi. L'informatico quindi non deve solo analizzare i problemi, ma deve essere capace anche di proporre una soluzione corretta ed efficiente, raggiungibile grazie a precisione, creatività e ragionamento [fonte].

L'informatica, inoltre, assume ulteriore importanza come supporto per lo sviluppo delle altre materie scientifiche. Ha rivoluzionato lo studio e l'analisi dei dati raccolti da altre discipline semplificando

e abbreviando enormemente i tempi di calcolo, tanto da diventare fondamentale nei laboratori di ricerca. La scienza computazionale (è così chiamata qualsiasi ramo delle scienze che utilizza la potenza di calcolo offerta dai moderni calcolatori per risolvere problemi [fonte]) è una disciplina che si sta sviluppando enormemente, tanto da diventare il terzo "pilastro" dell'indagine scientifica, affiancando teoria e sperimentazione [fonte].

Da queste considerazioni nasce l'idea di un percorso di approfondimento chiamato Computational STEAM, mirato a rafforzare le competenze di informatica all'interno del Liceo Scientifico, opzione Scienze Applicate. Si propone di integrare questa disciplina con le altre materie scientifiche insegnate, offrendo agli studenti gli strumenti necessari per poter approfondire e toccare con mano come queste discipline si stanno sviluppando nel mondo del lavoro e della ricerca, senza però tralasciare le materie umanistiche e artistiche, indispensabili per sviluppare un pensiero critico.

1.1 La situazione in Italia

L'idea di adattare gli studi superiori scientifici ai principi del modello STEAM si appoggia alla riforma Gelmini del 2010, più precisamente la sezione dedicata al Liceo Scientifico con l'aggiunta del piano chiamato Scienze Applicate. Questo percorso di studi guida lo studente ad approfondire e a sviluppare le conoscenze e le abilità necessarie per seguire lo sviluppo della ricerca scientifica e tecnologica [https://miur.gov.it/liceo-scientifico-opzione-scienze-applicate]. Vengono rafforzate e approfondite discipline quali tecnologia, matematica, fisica, chimica, biologia, scienze della terra e informatica. Con particolare riferimento a quest'ultima, nel Decreto Ministeriale 211 del 7 ottobre 2010 "Indicazioni Nazionali", allegato F si fa riferimento alla necessità di "utilizzare gli strumenti per la risoluzione di problemi appresi per la soluzione di problemi significativi connessi principalmente allo studio delle altre discipline, acquisendo la consapevolezza dei vantaggi e dei limiti dell'uso degli strumenti e dei metodi informatici e delle conseguenze sociali e culturali di tale uso". Alla fine di tale percorso lo studente è in grado di utilizzare i principali software, scegliendo di volta in volta il più adatto.

Il collegamento con le altre materie scientifiche e umanistiche è già presente su carta nel decreto, ma ha avuto difficoltà a svilupparsi nella pratica per varie ragioni: in primo luogo, il numero di ore offerte non sono sufficienti per poter sviluppare in maniera esaustiva gli argomenti, dando la possibilità agli studenti di poter applicare gli strumenti appresi in un contesto interdisciplinare, mantenendo comunque un adeguato equilibrio tra teoria e pratica. Inoltre la mancanza di docenti tecnici da affiancare durante le esercitazioni [https://www.orizzontescuola.it/itp-danno-tecnico-pratico-della-riforma-gelmini/] e l'adeguamento dei testi all'utilizzo di nuove tecnologie, più comprensibili e intuitive per i ragazzi, hanno contribuito a questo rallentamento. Le basi di questo percorso di studi partono da questa riforma, che però non dà molta libertà, per una mancanza oggettiva di tempo in primis.

L'Italia, durante il XIX e il XX secolo, è stata una tra le prime nazioni al mondo per innovazione e progresso. Il XXI secolo, che ci ha catapultati nell'era dell'informazione, necessita di un ulteriore sforzo per rimanere al passo. Con questa proposta si vuole dare ulteriore spazio all'informatica ed alla scienza computazionale all'interno degli istituti superiori italiani, in modo da poter offrire un percorso di studi adatto a coloro che vogliono sviluppare competenze trasversali in ambito scientifico, sfruttando la potenza degli odierni calcolatori, mantenendo un giusto equilibrio con le materie umanistiche indispensabile per lo sviluppo del pensiero critico.

1.2 Temi ed argomenti affrontati

Come detto già in precedenza, questo percorso formativo avrà una forte componente trasversale e interdisciplinare: l'informatica deve essere insegnata insieme e come supporto con altre discipline scientifiche, trasmettendo l'importanza del suo ruolo nella società moderna. Le tematiche affrontate non sono inserite in percorsi "verticali" di specializzazione, ma di più ampio respiro in modo da offrire le basi, seguendo lo spirito generalista del liceo rimandando gli approfondimenti agli studi universitari.

Computational science L'utilizzo dell'informatica come strumento per lo studio di modelli matematici/informatici è alla base della ricerca moderna, come anche l'utilizzo di questi modelli applicati a fenomeni reali, raccogliendo i dati e iterando il processo più volte, seguendo i principi di design ingegneristico.

Data Science La crescente digitalizzazione ha portato ad un esponenziale accumulo di dati raccolti. Questi dati, per poter essere utili, devono essere elaborati attraverso adeguati modelli statistici e matematici. Il ruolo del data scientist sta diventando sempre più importante nella società odierna ed è essenziale capire come i nostri dati vengono elaborati in modo da migliorare la nostra esperienza in rete. Esistono numerosi dati disponibili liberamente che possono essere utilizzati per esperienze laboratoriali, cosicché gli studenti possano capire quali sono le basi su cui si appoggiano i numerosi algoritmi utilizzati dalle grandi aziende.

Intelligenza artificiale Oggigiorno siamo circondati da sistemi "intelligenti" che cercano di semplificare la nostra vita. Numerose scelte vengono prese da algoritmi di cui non è chiesto conoscerne l'implementazione. È indispensabile per uno studente del liceo scientifico comprendere i principali, in modo da diventare un cittadino e non un semplice consumatore passivo. In questo senso, il ruolo delle discipline umanistiche, storiche e filosofiche è fondamentale.

2 Protocolli epidemici nel dettaglio

Un protocollo epidemico è un modello di comunicazione che trae ispirazione dallo studio della diffusione di epidemie. In modo intercambiabile si può utilizzare il termine protocollo di gossip, derivante dall'omonimo fenomeno studiato nell'ambito delle scienze sociali come metodo efficace per il passaggio di informazioni in una rete sociale.

Il gossip e le epidemie sono stati analizzati e le loro caratteristiche sono state implementate in reti informatiche, nello specifico in sistemi distribuiti. Le regole su cui si basa il loro funzionamento sono semplici, ma allo stesso tempo sono robusti veloci ed affidabili. Inoltre, non è necessario alcun controllo centralizzato, condizione necessaria affinché possano essere utilizzati nei sistemi distribuiti

2.1 Storia

I protocolli epidemici vengono analizzati per la prima volta in un documento scritto da Alan Demers [fonte] nel 1987. Il problema riscontrato da Demers e dai suoi colleghi nella rete interna del centro di ricerca di Xerox a Palo Alto era il seguente: mantenere la consistenza tra più copie di un database presente nelle diverse macchine. Il loro obiettivo consisteva nel progettare algoritmi robusti, efficienti e con elevata scalabilità, tenendo conto del tempo necessario per la distribuzione di un aggiornamento in una rete ed il traffico generato. Dopo aver esaminato un protocollo best-effort chiamato direct mail, che prevede l'invio in broadcast a tutti i nodi di una rete dell'aggiornamento ricevuto ed aver notato che non era efficiente né affidabile (è possibile che un messaggio venga perso e non è sempre scontato che un nodo conosca tutti i componenti in una rete), le loro attenzioni si sono spostate verso due tipologie di protocolli epidemici: anti-entropy e rumor-mongering. Entrambi prevedono lo scambio periodico di informazioni tra nodi scelti in modo casuale, cercando di risolvere le differenze che intercorrono tra i due database dei rispettivi end-point. Attraverso questi approcci si può ottenere consistenza eventuale (eventual consistency), una forma di consistenza debole, tale per cui un sistema di storagee garantisce che se non ci sono ulteriori aggiornamenti, tutti gli accessi ritorneranno il valore più aggiornato [eventual_consistency].

I protocolli di gossip sono utilizzati oggigiorno in numerosi contesti: in sistemi peer-to-peer che necessitano di scambio di informazioni, come il sistema di condivisione di file BitTorrent [bittorrent] oppure nella rete BitCoin [serve fonte se possibile, ho trovato solo forum che ne parlano].

2.2 Information dissemination

La prima applicazione che verrà analizzata riguarda la distribuzione di informazioni, come avviene naturalmente nel gossip. Gossip nei sistemi distribuiti significa scambiarsi informazioni in modo periodico e probabilistico tra due membri [gossiping_in_distributed_systems]. È importante notare che questo processo viene eseguito ripetutamente e di per se non ha una condizione di terminazione, ma verrà introdotta parlando del modello SIR e degli algoritmi di rumor-mongering.

2.2.1 Componenti e notazione

Si consideri una rete composta da un numero fissato P di nodi. Il grafo generato sarà completo, ovvero ogni nodo potrà comunicare direttamente con tutti gli altri nodi. Ogni nodo contiene una variabile chiamata *value* inizialmente uguale per tutti. I nodi potranno scambiarsi messaggi contenenti *value*, la quale avrà un attributo *value.timestamp* che indicherà la data e l'ora dell'ultimo aggiornamento. L'obiettivo di questi algoritmi sarà: in assenza di ulteriori aggiornamenti, *value* sarà uguale in tutti i nodi.

Ogni nodo possiede anche un attributo status che potrà assumere tre valori, ispirati alla terminologia utilizzata in epidemiologia:

- **Susceptible**(S): un nodo che non è venuto a conoscenza di un aggiornamento
- **Infected**(I): un nodo che è venuto a conoscenza dell'aggiornamento e lo sta distribuendo attivamente
- **Removed**(R): un nodo che è venuto a conoscenza dell'aggiornamento ma non lo distribuisce più

Gli stati *susceptible* e *infected* verranno utilizzati nel modello SI, mentre aggiungendo lo stato *removed* si parlerà di modello SIR.

2.2.2 Epidemie semplici

Il modello SI, chiamato anche anti-entropy o delle epidemie semplici, è il primo protocollo studiato nel centro di ricerca di Xerox. Come già detto, un nodo potrà trovarsi nello stato *susceptible* o *infected*. L'invio periodico di informazioni è scandito da un timer proprio del nodo, impostato inizialmente al valore Δ. Quando il timer scende a zero, il nodo invia il messaggio secondo lo stile scelto ed imposta nuovamente il timer. È interessante notare ogni nodo esegue queste operazioni una sola volta per round.

Nel documento originale vengono esposti e studiati tre stili per il modello SI, che cambiano il modo in cui i componenti della rete comunicano e risolvono le differenze.

Si utilizzerà la seguente notazione: n indica il numero totale dei nodi presenti sulla rete ($|P|$), mentre i valori s e i indicano rispettivamente il rapporto $|S|/n$ e $|I|/n$. Chiaramente $s + i = 1$.

Push

Il primo stile è detto stile push: un nodo infetto sceglierà in modo casuale un vicino a cui inviare informazioni. il nodo destinazione verificherà se le informazioni ricevute sono più aggiornate e, in caso affermativo, cambierà il proprio valore. Un nodo quindi, se infetto, invierà sempre un messaggio ad un altro nodo, indipendentemente dal fatto che il nodo destinazione conosca o meno l'aggiornamento. Si può facilmente intuire che lo stile push è più efficace quando il numero di nodi infetti è basso. In uqesta situazione, infatti, il numero di nodi infetti tenderà a raddoppiare ad ogni round e dopo $O(\log_2 n)$ round il valore i si avvicinerà a $1/2$. Quando invece i supera $1/2$ la situazione cambia: se definiamo s_t il rapporto di nodi suscettibili al roud t, possiamo calcolare il numero atteso di nodi suscettibili al round $t + 1$ come:

$$E(s_{t+1}) = s_t\Big(1 - \frac{1}{n}\Big)^{n(1-s_t)} \tag{2.1}$$

Nel dettaglio: un nodo rimane suscettibile se al round t era suscettibile (s_t) e non è stato contattato da nessuno dei nodi infetti ($1 - 1/n$ indica la probabilità che un nodo non contatti il nodo suscettibile, ripetuto per il numero di nodi infetti $n(1 - s_t)$).

Questo valore può essere approssimato per n molto grandi a $s_t e^{-(1-s_t)}$. Come dimostra Pittel [on_spreading_a_rumor] il numero di round atteso $T(n)$ per informare tutti i nodi in una rete è

$$T(n) = \log_2 n + \ln n + O(1) = O(\log n) \tag{2.2}$$

dove $\log_2 n$ deriva dalla prima fase, $\ln n$ da quella finale, mentre la fase intermedia, molto veloce, dura un numero costante di cicli.

Pull

Il secondo stile analizzato è lo stile pull: i nodi chiedono informazioni ad altri nodi, inviando il proprio timestamp. Se questi ultimi possiedono un aggiornamento più recente, invieranno un messaggio in risposta che verrà utilizzato dal nodo iniziale per cambiare il proprio valore.

A differenza dello stile push, quest'ultimo risulta poco efficace quando i nodi infetti sono pochi. Il numero atteso di nodi non ancora infeti dopo $t+1$ round può essere espresso come:

$$E(s_{t+1}) = s_t \cdot s_t = s_t^2 \tag{2.3}$$

in quanto un nodo rimane non informato se nel round precedente era suscettibile ed ha contattato un nodo a sua volta suscettibile. Può accadere che un nodo infetto dovrà aspettare alcuni round prima di venir contattato, rendendo questi round inutili per lo scopo dell'algoritmo. Nonostante ciò, con alta probabilità dopo $O(logn)$ round metà dei nodi sarà infetta. La fase finale invece è molto più rapida in quanto, aumentando il numero di nodi infetti, aumenta la probabilità per un nodo suscettibile di contattare un nodo infetto.

Push-Pull

La soluzione migliore proposta si basa su una combinazione dei due stili precedenti. Lo stile push-pull lavora nel modo seguente: all'azzeramento del timer, un nodo invia un messaggio ad un altro nodo scelto tra i vicini in modo casuale, il quale controllerà il timestamp ed a seconda del risultato invierà una risposta oppure aggiornerà il proprio valore. E' più rapido in quanto sfrutta i punti di forza dei protocolli push e pull (nella fase iniziale sfrutterà il push, nella parte finale il pull). Karp [randomized_rumor_spreading] ha dimostrato che il numero atteso di round per infettare tutti i nodi è $O(\log \log n)$.

Riassumendo, il modello SI è efficace in quanto permette di distribuire su tutta la rete un aggiornamento, in quanto un nodo infetto continuerà (idealmente per sempre) ad inviare o ricevere messaggi. Nonostante ciò questo può rivelarsi un peso non indifferente per la rete in quanto questo modello prevede l'invio del database completo all'interno del messaggio e non il singolo aggiornamento. Se gli aggiornamenti in una rete sono rari, la maggior parte dei messaggi diventa inutile, perché i nodi continueranno a contattarne altri che già sono a conoscenza dell'aggiornamento.

2.2.3 Epidemie complesse

Il modello SIR viene introdotto per risolvere il problema della non terminazione del modello precedente e per aumentare l'efficienza. I nodi potranno assumere lo stato rimosso, ovvero nodi che conoscono l'aggiornamento ma non lo distribuiscono più. Come per le epidemie semplici, inizialmente i nodi sono tutti suscettibili: quando uno di questi viene a conoscenza di un aggiornamento, diventa infetto ed incomincia ad inviare messaggi agli altri nodi. Eventualmente questi nodi potranno "perdere" interesse nel distribuire il proprio aggiornamento, cambiando così il suo stato in rimosso. Quando nella rete non ci sarà più nessun nodo infetto, l'algoritmo termina. Il passaggio dallo stato infetto allo stato rimosso può essere influenzato dai seguenti fattori:

- **Come** (How):
  - **counter**: un nodo passerà allo stato rimosso dopo k contatti
  - **coin**: un nodo passerà allo stato rimosso con probabilità $1/k$
- **Quando** (When)
  - **feedback**: la valutazione avverrà quando un nodo contatta un altro nodo che era già a conoscenza dell'aggiornamento
  - **blind**: la valutazione avverrà ad ogni round

Si ottengono così quattro possibili combinazioni: *feedback/counter, blind/coin, feedback/coin, blind/counter*. Verranno analizzati *feedback/counter* e *blind/coin* utilizzando lo stile push, ma le considerazioni potranno essere applicate allo stesso modo anche per gli altri algoritmi.

Per confrontare i diversi protocolli che si basano sul modello SIR si utilizzano i seguenti criteri:

- **Residuo**: indica il numero di nodi ancora suscettibili al termine dell'algoritmo (viene indicato con s^*). Non è garantito infatti, come invece accade nel modello SI, che tutti i nodi verranno a conoscenza dell'aggiornamento. Può verificarsi una situazione in cui tutti i nodi si trovano nello stato suscettibile oppure rimosso, senza quindi aver ottenuto consistenza in tutta la rete.

- **Traffico**: indica il numero di messaggi inviati. Spesso si utilizza il traffico medio, definito come

$$m = \frac{\text{traffico totale}}{\text{numero di nodi}} \tag{2.4}$$

- **Ritardo**: può essere espresso come ritardo medio t_{avg}, ovvero la differenza tra il momento dell'infezione iniziale e l'arrivo di un aggiornamento, mediato sul numero di nodi, oppure come ritardo totale t_{max}, cioè il tempo necessario affinché l'ultimo nodo riceva l'aggiornamento.

Definiamo, come per il modello SI, s, i e r come la il rapporto di nodi suscettibili, infetti e rimossi rispetto al numero di nodi, in modo tale che $s + i + r = 1$.

L'andamento dei seguenti algoritmi può essere modellato attraverso l'utilizzo delle seguenti equazioni differenziali [The mathematics of infectious diseases]:

$$\begin{aligned} \frac{ds}{dt} &= -\beta is \\ \frac{di}{dt} &= \beta is - \gamma i \\ \frac{dr}{dt} &= \gamma i \end{aligned} \tag{2.5}$$

dove β rappresenta il tasso di contagio mentre γ, chiamato in epidemiologia tasso di recupero, è un valore che dipende da k e dal numero di nodi ancora suscettibili, più precisamente

$$\gamma = \frac{1}{k}(1 - s) \tag{2.6}$$

Possiamo utilizzare le prime due per risolvere il sistema di equazioni differenziali partendo dal loro rapporto e supponendo beta uguale a 1:

$$\begin{aligned} \frac{di}{ds} &= \frac{si - \frac{1}{k}(1-s)i}{-si} \\ &= \frac{\frac{\cancel{i}(ks-1+s)}{k}}{-\cancel{i}s} \\ &= \frac{1 - ks - s}{ks} \\ &= \frac{1}{ks} - 1 - \frac{1}{k} \\ &= \frac{1}{ks} - \frac{1+k}{k} \end{aligned} \tag{2.7}$$

integrando si ottiene

$$s(i) = \frac{1}{k}\ln s - \frac{1+k}{k} + c \tag{2.8}$$

dove c è costante di integrazione che si può calcolare considerando che inizialmente la funzione è espressa come $i(1 - 1/n) = 1/n$ che tende a 0 per n molto grandi

$$c = \frac{k+1}{k} \tag{2.9}$$

che porta alla seguente soluzione

$$i(s) = \frac{k+1}{k}(1-s) + \frac{1}{k}\ln s \tag{2.10}$$

Questa equazione può essere utilizzata per calcolare s^* quando $i(s^*) = 0$

$$s^* = e^{(-k-1)(1-s^*)} \tag{2.11}$$

Il risultato è una funzione implicita su s^*, la quale mostra che il residuo diminuisce esponenzialmente all'aumentare di k.

È possibile notare inoltre che tutte le varianti dell'algoritmo condividono la stessa relazione tra traffico e residuo. Considerando ogni messaggio inviato ha probabilità pari a 1/n di contattare un nodo specifico, la probabilità di rimanere suscettibile dopo l'invio di m*n messaggi è pari a:

$$s(m*n) = \left(1 - \frac{1}{n}\right)^{nm} \tag{2.12}$$

che con n grandi può essere approssimato come $s = e^{-m}$. Come si può notare dalla tabella 1, il ritardo è l'unico parametro che distingue le varianti: osservando i dati si può notare che feedback/counter offre ritardo inferiore a parità di k.

2.2.4 Dettagli implementativi

Nelle pagine precedenti sono stati descritti i protocolli basati sui modelli SI e SIR utilizzati per la distribuzione di informazioni, senza parlare di come questi possano essere utilizzati in casi reali. Innanzitutto anti-entropy e rumor-mongering possono essere utilizzati in contemporanea sulla medesima rete: si suppone infatti che un protocollo basato sul modello SIR può eventualmente terminare senza che si sia ottenuta consistenza su tutta la rete. Per evitare che ciò accada, è possibile eseguire un algoritmo anti-entropy meno frequentemente.

Valori multipli

È molto probabile che i protocolli devono lavorare con più che singoli valori, e quindi la comparazione può diventare molto onerosa. Molto spesso infatti, lavorando con database per esempio, il confronto diventa quasi completamente inutile perché gran parte dei dati sono simili. È possibile risolvere in parte questo problema utilizzando dei checksum calcolati a partire dalla copia del database contenuta nel nodo, ricalcolando ogni volta che quest'ultimo viene aggiornato. Il primo scambio tra nodi avviene attraverso il confronto dei relativi checksum, eseguendo il confronto completo dei database solo se si trovano in disaccordo. Purtroppo la computazione dei checksum tende ad essere molto spesso diversa se l'aggiornamento ai nodi arriva in momenti tanto diversi, quindi la rete tenderà comunque a dover confrontare i dati completi. Una soluzione più complessa prevede di definire un intervallo di tempo che sia sufficiente per far ricevere l'aggiornamento a tutti i nodi. Ogni nodo contiene una lista con gli aggiornamenti più recenti, all'interno di questo spazio di tempo, che viene scambiata ed utilizzata per aggiornare checksum e database. Solo nel caso di un ulteriore disaccordo, viene eseguito il confronto con l'intero database. È importante notare che la scelta dell'intervallo di tempo è cruciale per il funzionamento: se viene fatta in maniera sbagliata, i checksum tenderanno a differenziarsi, aumentando il traffico rendendolo superiore al traffico generato dai protocolli anti-entropy senza checksum.

Un' ultima soluzione, che non prevede la scelta a priori di un intervallo di tempo, prevede la memorizzazione di un indice inverso del database basato sul timestamp. I nodi si scambiano gli aggiornamenti in ordine inverso di timestamp, ricalcolando i checksum fino ad ottenere un consistenza di questi ultimi. Non è comunque ottimale a causa del costo di mantenimento dell'indice inverso in ogni nodo.

Traffico generato

Un problema non trascurabile in casi reali è sicuramente il traffico che una rete può supportare. Questi protocolli prevedono che ad ogni round vengano inviati più messaggi contemporaneamente, rischiando quindi di sovraccaricare la rete. Demer aveva definito questo come connection limit, spiegando che

definire un limite è necessario sia in caso di puh che di pull. Questa condizione porta però ad un significativo peggioramento dei protocolli che utilizzano lo stile pull, mentre push diventa migliore. Il protocollo Scuttlebutt [flowgossip] definisce alcuni rimedi a questa limitazione, specificando un metodo per determinare in modo dinamico il tasso massimo di traffico che un nodo può generare senza dover creare un sistema di backlog per gli aggiornamenti.

Gestione dei fallimenti

Implementando i protocolli di gossip in contesti reali comporta il rischio di possibili fallimenti per fattori esterni. Per fortuna, la perdita di messaggi non aggrava sull'esecuzione se non rallentando il raggiungimento della consistenza (alcuni round diventano inutili). Se alcuni nodi smettono di "funzionare", gli scambi con questi diventano inutili e i protocolli rallentano la loro esecuzione. È comunque auspicabile mantenere lo stato attivo degli endpoint in modo da evitare questi inconvenienti.

2.3 Altri utilizzi dei protocolli epidemici

Negli scritti di Demers[1987], si supponeva che la rete fosse statica, ovvero il numero di nodi rimaneva fisso nel tempo. Inoltre ogni nodo aveva una visione totale, e quindi il grafo rappresentativo era completo. Infine il numero di macchine collegate era relativamente basso. Oggigiorno le reti sono molto più grandi, non sono completamente connesse e la lista di nodi direttamente connessi tra loro può variare per continue connessioni-disconnessioni. Questo sicuramente porta grandi vantaggi, come una maggior scalabilità. I protocolli epidemici possono essere adattati a questi grandi cambiamenti ed utilizzati per altri scopi, oltre alla distribuzione di informazioni. Esistono implementazioni dei protocolli epidemici per mantenere informazioni sullo stato di una rete peer-to-peer [membership], per rilevare errori e guasti[SWIMM], per implementare garbage collection[garbage_collection], per calcolare informazioni di aggregazione[aggregation], ...

2.3.1 Notazione

Si può definire un algoritmo di gossip generico, che si basa sulle implementazioni precedenti e descrive i vari momenti che devono essere tenuti in considerazione:

- **inizializzazione**: un nodo viene definito con il suo stato iniziale. Viene impostato il timer
- **allo scadere del timer**: il nodo sceglie un vicino, prepara il messaggio, lo invia al nodo scelto e reimposta il timer
- **al ricevimento di un messaggio di richiesta**: il nodo prepara la risposta e la invia. Il nodo legge la richiesta e la elabora
- **al ricevimento di un messaggio di risposta**: il nodo elabora la risposta.

Da queste situazioni si può notare come i metodi di elaborazione ed invio non sono stati implementati, ma questa operazione verrà fatta in seguito.

2.3.2 Peer sampling

Per la gestione di reti dinamiche e di grandi dimensioni, il mantenimento di una lista contenente tutti i nodi è oneroso e per nulla efficiente. A causa della possibilità di fallimenti o di nuovi nodi, questa lista deve essere aggiornata frequentemente, per poi essere diffusa a tutti.

Per risolvere questo problema si utilizza un sottoinsieme della lista completa, scelta per ogni nodo in maniera casuale. Il mantenimento di questo sottoinsieme è sicuramente più gestibile. Il metodo che si occupa di questo compito è detto getPeer(). Di seguito la spiegazione di Newscast[Newscast], un protocollo di membership management (un altro modo per chiamare il peer sampling).

Newscast - non completo

Il protocollo Newscast funziona nel modo seguente: ad ogni round un nodo sceglie un vicino casuale tra la sua vista parziale di c elementi. Successivamente, stabilita la connessione, i nodi si inviano

reciprocamente una la loro lista più il proprio indirizzo con il timestamp aggiornato. Infine stilano una lista dei c collegamenti più recenti, eliminando i restanti. L'algoritmo generico viene completato nel seguente modo:

- **selectNeighbor()**: sceglie un noda dalla vista parziale
- **prepareRequest()** e **prepareReply()**: ritorna la vista più il proprio indirizzo con il timestamp aggiornato
- **merge()**: ritorna i più recenti collegamenti, scelti tra la propria vista e quella inviata dall'altro nodo

2.3.3 Failure detection

La rilevazione di errori in sistemi distribuiti è un problema complesso[failure_detection] principalmente perché molto spesso un processo può sembrare problematico anche se in realtà è semplicemente lento oppure la connessione è limitata. È necessario verificare con accuratezza che un nodo sia "guasto", in modo da limitare il più possibile i falsi positivi mantenendo una conoscenza parziale dei nodi il più aggiornata possibile.

Definizione del problema

Il protocollo SWIM [SWIM] offre un sistema di failure detection e, utilizzando il principio del gossip, un modo scalabile ed efficiente per distribuire le informazioni ricevute, mantenendo così aggiornata la lista dei membri che ogni nodo possiede. Si può dire quindi che combina sia il membership management e la failure detection.

Il problema della scalabilità delle precedenti soluzioni è dovuto principalmente all'utilizzo della tecnica di heartbeating: un membro Mi è dichiarato guasto da un altro membro Mj quando non riceve messaggi "heartbeat" (messaggi con counter incrementale) per un determinato numero di periodi consecutivi [SWIM].

Algoritmo

L'algoritmo è realizzato in modo da dividere le componenti di failure detection e di membership update, senza utilizzare il metodo heartbeat[SWIM]:

- **failure detection**: componente che rileva i guasti
- **dissemination**: componente che invia informazioni riguardo i membri che entrano o lasciano la rete

SWIM Failure Detector Utilizza due parametri: T indica il periodo del protocollo, mentre k indica il numero di nodi appartenenti al gruppo che si occupa di contattare un nodo che non ha inviato risposte. Ad ogni round (di lunghezza T), un nodo M_i sceglie un altro nodo M_j dalla sua lista di vicini e invia un messaggio *ping* e aspetta la risposta *ack*. Se questa non arriva dopo un determinato periodo, inferiore a T, M_i contatta i k nodi scelti all'avvio del protocollo e chiede loro, attraverso un messaggio *ping-req*, di contattare il nodo M_j. Se nessuno di questi nodi riceve risposta ed invia la conferma della salute del nodo M_j, questo viene dichiarato guasto nella lista di M_i, che si occuperà di distribuire la scoperta a tutta la rete. **Dissemination component** Se un nodo imposta un nodo come guasto, inizia a distribuire questa informazione a tutta la rete. La versione base del protocollo SWIM utilizza un sistema multicast, mentre una versione più robusta sfrutta il principio della diffusione delle epidemie, quindi i protocolli epidemici.

Riduzione dei fasi positivi

La versione base del protocollo SWIM rischia di aumentare il numero di nodi considerati guasti nel caso di perdita di pacchetti oppure per una temporanea disattivazione del nodo in questione. Per evitare questo problema viene introdotto protocollo chiamato Suspicion protocol [SWIM]: quando un

nodo Mi non riceve informazioni da un nodo Mj, ne dai nodi che lo aiutano a contattarlo, imposta il nodo M_j come Sospetto. Se un altro nodo riceve questa informazione, marca nella sua lista il nodo come sospetto.

Se un nodo riesce a contattare un nodo sospetto, comincia ad inviare un messaggio $Alive(M_j)$, il quale sovrascrive il sospetto di guasto. Quando un nodo rimane sospetto per un certo periodo di tempo, senza avere ulteriori informazioni, viene marcato come guasto in modo definitivo. Oltre allo stato di *Suspect/Alive/Confirm*(guasto), viene associato anche un contatore, generato in modo da gestire la successione di messaggi e la loro eventuale sovrascrittura all'interno della lista.

3 Relazione tirocinio

In questo capitolo viene esposto un resoconto del tirocinio avvenuto tra i mesi di aprile e maggio 2019 presso il Liceo Galileo Galilei (Trento, TN), il liceo Leonardo Da Vinci (Trento, TN) e l'istituto tecnico tecnologico Buonarroti-Pozzo (Trento, TN). L'attività è stata strutturata in due parti: una parte teorica di lezione interattiva e una parte laboratoriale in cui i ragazzi svolgevano degli esercizi. Lo scopo del tirocinio è stato quello di proporre un modo diverso di visualizzare ed utilizzare l'informatica presso gli istituti superiori, seguendo i principi del modello Computational STEAM (vedi capitolo 1). L'argomento trattato, i protocolli epidemici (vedi capitolo 2), è stato scelto perché adatto a far comprendere come modelli e algoritmi studiati in altri campi (la biologia e la matematica) possano essere adattati ed utilizzati con efficacia in ambito informatico (nello specifico la comunicazione in sistemi distribuiti).

3.1 Contesto e organizzazione del corso

Abbiamo presentato il corso tra aprile e maggio presso i seguenti istituti superiori: liceo Scientifico Galileo Galilei di Trento, liceo Scientifico Leonardo Da Vinci di Trento e istituto tecnico e tecnologico Buonarroti-Pozzo. Abbiamo scelto questo periodo in quanto più adatto sia per i ragazzi che per gli insegnanti ordinari. Ci siamo concentrati sul liceo scientifico, in quanto il percorso di studi Computational STEAM è indirizzato a questa tipologia di scuola, ma abbiamo deciso di provare a portare l'esperienza anche in un istituto tecnico, per cercare di capire quali possono essere le differenze riguardo la preparazione dei ragazzi, in modo da adattare le lezioni al meglio. Abbiamo strutturato le lezioni nel seguente modo: un'ora per l'introduzione, e le restanti 6 ore per il contenuto del corso in sé. La lezione introduttiva e l'ultima sono state utilizzate anche per compilare un questionario introduttivo ed un conclusivo (entrambi anonimi), i cui risultati verranno utilizzati come supporto alla stesura di questo capitolo. È stata utilizzata l'applicazione Moduli Google. Abbiamo raccolto le risposte di 164 ragazzi per il questionario iniziale, 161 per quello finale (- risposte da rimuovere). Le classi analizzate sono terze, quarte e quinte, perché gli argomenti trattati risultano più adatti a studenti con conoscenze che vengono fornite a partire dal terzo anno in poi. La scelta delle classi è stata fatta dagli insegnanti di informatica. Le classi sono prevalente composte da maschi, il che non è affatto sorprendente osservando la composizione nei corsi di studio di informatica ed ingegneria. I ragazzi hanno notato questa differenza numerica, motivandola principalmente con questa affermazione: "l'informatica è preferita dai ragazzi". Nonostante ciò, la figura dell'informatico è stata descritta in modo abbastanza lontano dai frequenti stereotipi: secondo i dati raccolti molti pensano che un informatico debba essere creativo e abbia grande capacità nel lavorare in gruppo.

3.2 Contenuto e metodo esecutivo

Il contenuto del corso è stato esposto utilizzando delle slide per la parte di lezione teorica, mentre per la parte applicativa è stata fornita una cartella condivisa con gli esercizi da svolgere e le loro spiegazioni nel dettaglio [http://www.tinyurl.com/protocolli-epidemici]. La parte teorica è stata presentata come lezione interattiva, dando spazio agli studenti di porci delle domande e di rispondere a quesiti da noi proposti. Ad eccezione della prima ora di lezione, tutte le altre hanno avuto una componente pratica,

in modo tale che i ragazzi potessero applicare quanto imparato. La difficoltà degli esercizi ea via via crescente, sia per la complessità degli algoritmi proposti, sia per un minore aiuto fornito in determinate situazioni (in alcuni casi è stato fornito solamente lo pseudocodice). Abbiamo scelto di sviluppare il corso sui protocolli epidemici (vedi capitolo 2) per diversi motivi: in primo luogo, l'argomento si basa su semplici regole che vengono utilizzate più volte in diversi contesti. I protocolli di gossip possono però essere studiati maggiormente e lasciano spazio anche ad un approfondimento autonomo. Inoltre sono sono un argomento interdisciplinare: vengono utilizzati in reti distribuite per la comunicazione, ma i principi su cui si basano sono studiati in epidemiologia, nell'ambito quindi della diffusione delle epidemie, ma anche dalle scienze sociali, riguardo il fenomeno del gossip. Questa interdisciplinarietà si dimostra adatta all'interno di questa esperienza che vuole portare i principi di Computational STEAM nelle scuole superiori. Nello specifico abbiamo abbiamo sviluppato il programma del corso nel seguente modo: nella prima lezione, utile come introduzione, abbiamo presentato l'argomento e la sua natura interdisciplinare, il contesto storico, la definizione del problema, le applicazioni reali e i principi di base. Nelle successive lezioni ci siamo concentrati, dopo aver fatto una breve introduzione sui grafi (utile per alcune definizioni ed assunzioni), sui due modelli SI e SIR presentando gli algoritmi e stili che li utilizzano, lasciando poi spazio ai ragazzi di implementarli su schermo. L'ambiente di sviluppo utilizzato è Processing (https://processing.org), libreria grafica di Java e IDE utile per creare simulazioni e metodi di visualizzazione efficace. Gli studenti si sono occupati di scrivere gli algoritmi, completando il codice fornito oppure interpretando lo pseudo codice. Abbiamo fornito una libreria (https://github.com/ainter21/epidemic), utile sia per alleggerire il carico di lavoro, sia per nascondere alcune componenti troppo complesse, come la parte di visualizzazione (lo scopo degli esercizi è quello di implementare l'algoritmo, mentre la parte di visualizzazione è solamente un supporto per rendere più chiari i concetti). Abbiamo fornito la documentazione utile per completare gli esercizi, con la spiegazione dei metodi e degli attributi.

Bibliografia

Allegato A Titolo primo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

A.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Allegato B Titolo secondo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1 Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

B.1.1 Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.