UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Consumo Energetico di Strutture Dati in Java: Uno Studio Empirico

RELATORE

Prof. **Fabio Palomba**

Prof. **Dario Di Nucci**

Università degli Studi di Salerno

CANDIDATO

Nicolò Delogu

Matricola: 0512106214

Anno Accademico 2021-2022

INSERIRE QUI UNA DEDICA O UNA CITAZIONE

Sommario

Questa tesi presenta uno studio dettagliato del consumo energetico delle diverse implementazioni di Java Collection Framework (JFC). Per ogni metodo messo a disposizione dalle API, presentiamo il suo consumo di energia a seconda del caso d'uso e al variare del volume di dati gestito. Una volta ottenuti i profili energetici di ogni struttura dati e rilevati i metodi piu vantaggiosi per ogni implementazione, presentiamo un approccio di ottimizzazione energetica per i programmi Java basato su chiamate a metodi JFC nel codice sorgente di varie librerie e applicazioni, optando opportunamente per la scelta più ecologica o per quella meno ecologica rispetto ai profili prodotti precedentemente. Infine, presentiamo i risultati ottenuti per ogni caso in cui è stata rilevato un miglioramento o degradamento rispetto alle modifiche effettuate.

Indice

Elenco delle figure

CAPITOLO 1

Introduzione

1.1 Motivazioni e Obiettivi

I crescenti costi energetici legati alle TIC nelle organizzazioni e le preoccupazioni ambientali della società stanno cambiando il modo in cui gli ingegneri del software sviluppano i loro prodotti. Durante il secolo precedente, il miglioramento dei tempi di esecuzione era l'obiettivo principale nello sviluppo di hardware/software e nella programmazione di linguaggi e i loro compilatori sono stati progettati per produrre sistemi veloci. Oggigiorno il consumo energetico sta diventando il collo di bottiglia di tali sistemi. Di conseguenza, le librerie offerte attraverso i linguaggi di programmazione devono tener conto di questa nuova realtà. In questa tesi abbiamo condotto uno studio dettagliato in termini di consumo energetico della libreria Java Collections Framework (JCF). Abbiamo considerato tre diversi gruppi di strutture di dati, vale a dire insiemi, elenchi e mappe, e per ognuno di essi, abbiamo studiato il consumo di energia di ciascuna delle sue diverse implementazioni e metodologie. Abbiamo monitorato l'energia consumata da ciascuna API trattando set di dati di piccole, medie e grandi dimensioni e prendendo in cosiderazione le operazioni di inserimento, di iterazione e di ricerca randomica. Il primo risultato del nostro studio è stata una quantificazione dell'energia spesa per ciascun metodo di ogni struttura considerata, successivamente impiegata per definire dei profili energetici specifici per ogni metodologia e finalizzati a guidare i programmatori nelle loro scelte implementative. La consapevolezza delle problematiche odierne sul consumo energetico può essere utile non solo per guidare

gli sviluppatori nella scrittura di prodotti software a basso impatto ambientale, ma anche nell'ottimizzazione del codice legacy. Come secondo risultato abbiamo calcolato staticamente quali implementazioni e metodi sono usati nel codice sorgente di varie librerie e applicazioni, per poi confrontare i dati ottenuti con i profili prodotti precedentemente, individuando, se la scelta implementitativa equivalente presenti un consumo energetico minore o maggiore rispetto a quella originale. Infine, abbiamo trasformato manualmente il codice sorgente inserendo delle modifice «ecologicamente vantaggiose o svantaggiose». I nostri risultati preliminari mostrano che l'energia consumata aumenta o decrementa in tutti i prodotti software che abbiamo testato a seconda della modifica apportata.
I dettagli implementativi dell'applicazione sono disponibili al seguente link:
https://github.com/XJustUnluckyX/DataStrucutreTestApp/tree/master

1.2 Struttura della tesi

Il documento è strutturato come segue: il secondo capitolo presenta lo stato dell'arte dell'argomento trattato; il terzo capitolo descrive uno studio simile al nostro preso in considerazione come punto di partenza e di confronto, il quarto capitolo descrivono dettagliatamente le tecnologie hardware e software utilizzate per quantificare l'energia consumata da ogni metodo testato, presenta l'analisi dei consumi energetici delle diverse implementazioni delle strutture dati di Java Collection Framework e descrive la metodologia scelta per ottimizzare i programmi che fanno uso di JCF API; nel quinto capitolo valutiamo le minacce alla validità della nostra analisi e nel sesto i possibili sviluppi per lavori futuri.

CAPITOLO 2

Stato dell'arte

L'efficienza energetica del software è un processo iterativo che consiste in numerosi passaggi per ottenere un codice sorgente e/o un ambiente di esecuzione migliori senza influire sulle funzionalità, l'evoluzione e la manutenzione del software stesso. Per produrre un software efficiente dal punto di vista energetico, è necessario innanzitutto stimarne fedelmente il consumo energetico e monitorarne l'evoluzione nel tempo. Nella letteratura scientifica sono stati presentati numerosi strumenti per misurare il consumo energetico del software, ma si possono distinguere principalmente due categorie: strumenti "hardware" e strumenti "software".

2.1 Fattori Discriminanti

Studi precedenti [1] hanno dimostrato che esistono numerose metriche che possono condizionare il risultato finale della misurazione del consumo energetico e ne definiscono aspetti fondamentali come applicabilità e affidabilità. Essi possono riguardare sia le componenti hardware, ovvero la strumentazione, sia quelle software, ovvero le tecnologie. Tra le principali sono state individuate:

Efficienza: indica il numero complessivo di casi di test da effettuare per identificare sia le misurazioni con un setup "peggiore" sia quelle che possono risultare anomale;

Granuralità: la misurazione del consumo energetico può essere a grana fine (valutazione del contributo individuale di ciascuna linea di codice per l'energia consumata), mid-grained

(riguarda l'energia consumata da un blocco di codice o da un metodo/procedura) o a grana grossa (semplicemente dichiarare l'energia consumata dal esecuzione del programma in un dato periodo di tempo).

Effetto Hawthorne: è necessario tenere conto del fatto che le proprietà che cerchiamo di misurare possano essere influenzate dal processo di misurazione. In particolare, il consumo di una qualsiasi linea di codice sarà probabilmente influenzata dalla strumentazione stessa, riducendo così l'affidabilità delle misure. Dunque se l'influenza delle misurazioni su di esse è minima o costante, allora potremmo scegliere di ignorarla o fattorizzarla ed escluderla. Tuttavia, poiché il funzionamento di un frammento di codice è sensibile ad almeno una parte del contesto, non possiamo presumere che l'effetto della strumentazione sia costante o minimale;

Specificità: esiste un divario piu o meno esteso tra l'applicabilità di un' approccio e il grado di informazione che esso può restituire. Per essere specifici, non possiamo semplicemente testare l'energia consumata, ma sarebbe buona prassi fornire valutazioni dettagliate sul dove l'energia viene consumata. Una valutazione così specifica potrebbe mettere meglio in evidenza i modi per ridurre il consumo energetico. Per esempio, l'approccio RAPL (Running Average Power Limit) [2], è stato sviluppato da Intel per distinguere tra l'energia consumata nella CPU, l'accesso casuale dinamico alla memoria, e il cosiddetto «CPU uncore» (come le cache e le unità di elaborazione grafica on-chip). Questa specificità, così strettamente accoppiata con l'hardware valuta, facendo luce per quanto riguarda le cause del consumo, ma le informazioni che produce sono naturalmente pertinenti solo al singolo dispositivo;

2.2 Strumenti Hardware

Questa categoria di strumenti richiede delle componenti fisiche dedicate per misurare il consumo energetico generalmente noto per essere molto preciso ma non a grana fine, e spesso si incorrere in una strumentazione con un costo finanziario aggiuntivo. Nella maggior parte dei casi, si tratta di dispositivi/schede che valutano l'energia consumata dall'alimentatore in joule o Wh. Il modo in cui essi vengono utilizzati consiste nel misurare la corrente consumata da un computer o da un altro apparecchio prima e durante l'esecuzione del software. Il corrispondente consumo energetico viene dunque dedotto sottraendo la misurazione di partenza a quella finale. In primo luogo, WattsUp Pro [3] è un dispositivo che viene solitamente installato tra computer/server e la loro fonte di alimentazione per registrare il consumo

energetico totale ad una velocità di campionamento massima di 1 campione al secondo e ad una corrente massima di 15 ampere. Il dispositivo ha una memoria interna che può essere utilizzata per registrare le misure.
Quest'ultime possono riguardare un'ampia gamma di dati, tra cui i costi massimi potenziali, attuali, energetici e cumulativi. I dati memorizzati in memoria possono essere scaricati su un computer tramite un cavo USB. Il prodotto funziona anche con un software separato Logger Pro o LabQuest App per creare grafici, calcoli e profili dei dispositivi. Se è necessario misurare contemporaneamente il consumo energetico di un cluster, potrebbero essere necessari molti dispositivi WattsUp Pro; tuttavia il prezzo di tale dispositivo oscilla tra 50 e 120 dollari e ciò potrebbe risultare molto costoso per chiunque voglia avviare un progetto simile.
Un'alternativa alle soluzioni precedentemete esposte potrebbero essere PowerMon [4] e PowersMon2 [5]. Questi dispositivi di monitoraggio dell'alimentazione funzionano tra l'alimentatore di un sistema e una scheda madre, per analizzare i compromessi sul consumo di energia nelle applicazioni software e informatiche. PowerMon monitora tensione e corrente su sei binari DC e riporta le misure fino a 50 campioni al secondo tramite un'interfaccia USB, consentendo il monitoraggio dell'host di destinazione o di un host separato. Rispetto al suo predecessore, PowerMon2 è un po' più piccolo e permette di essere utilizzato in un server chassis da 3,5 pollici. Dispone inoltre di 2 ulteriori guide DC (per un totale di 8) per periferiche aggiuntive, come dischi e GPU. PowerMon2 utilizza anche un cavo USB per registrare le misure di potenza ad una velocità massima di 3 KHz. Il prezzo di un singolo dispositivo PowerMon è superiore a 150 dollari.
Come i dispositivi PowerMon, PowerInsight [6] è un altro strumento costruito su una scheda esterna basata su un processore ARM Cortex, progettato da Penguin Computing per realizzare strutture particolarmente complesse. PowerInsight può essere collegato a un massimo di 15 componenti (dischi, GPU, ecc.). È inoltre progettato per funzionare all'interno di un cluster. Ogni scheda inserita tra la scheda madre di un computer e il suo alimentatore è dotata di una porta Ethernet, che viene utilizzata per inviare e acquisire dati da/verso il nodo principale il quale aggrega e salva i dati per il consumo energetico del cluster. Le misurazioni sono state testate con lo strumento PowerInsight con una frequenza di campionamento di 1 campione al secondo.
GreenMiner [7] è uno strumento di misura ben noto utilizzato nella letteratura, principalmente per misurare il consumo energetico delle applicazioni mobile [8]. Si tratta di una testsuite hardware/software che esegue test su numerosi dispositivi e misura sia il consumo energetico di ogni test e sia quello dell'intero dispositivo. Il lato client GreenMiner è un Raspberry

Pi che funge da banco di prova. Esso utilizza un dispositivo Android per eseguire i test e raccoglie i risultati da una scheda Arduino che monitora il consumo energetico. L'energia viene misurata tramite un chip di misura dell'energia (INA219) che campiona e aggrega le misure con una frequenza di 5 MHz. Il banco prova registra e carica le misure INA219 sul servizio web GreenMiner che rappresenta il lato server, responsabile del trattamento e dell'analisi dei dati di misura dell'energia.

2.2.1 Variazioni Energetiche

Per variazioni energetiche, ci riferiamo al problema della stabilità nella misurazione del consumo energetico del software. Ciò significa che, compromettendo lo stato della singola misurazione all'interno dell'ambiente, si possono verificare diversi errori e incongruenze nell'insieme delle misurazioni effettuate.
Questa variazione è stata spesso correlata al processo di fabbricazione delle CPU e dei componenti, ma è stata anche oggetto di molti studi, nei quali sono stati presi in considerazione diversi aspetti che potrebbero influire e variare il consumo energetico tra le esecuzioni. La correlazione tra la temperatura del processore e il consumo energetico è stato uno dei percorsi più esplorati. Joakim/Kisroski [9] hanno affermato che processori identici possono presentare variazioni significative di consumo energetico senza uno stretto legame causa-effetto tra la temperatura e le prestazioni del processore stesso.
Le differenze piu ampie registrate nel consumo energetico sono state quelle per gli stati di carico minimo e di carico massimo. Tuttavia, è necessario precisare che l'effetto termico del processore è uno dei fattori che maggiormente contribuiscono alla flattuazione dell'energia. E' bene tenere a mente che la correlazione tra la temperatura della CPU e la variazione del consumo energetico è molto stretta. Questo rende il rumore termico un fattore chiave da considerare quando si confrontano i risultati ottenuti. Anche la temperatura dell'ambiente è stata anche discussa in altri documenti come fattore candidato per la variazione di energia impiegata da un processore. Infatti, il consumo energetico può variare a causa delle condizioni dovute all'ambiente esterno; tuttavia, la temperatura all'interno di un data center non comporta grandi variazioni da un nodo all'altro.
La commutazione dello spot di due server non influisce sostanzialmente sul loro consumo energetico. Inoltre, la modifica dei componenti hardware, come il disco rigido, la memoria o anche l'alimentatore, non influisce sulla variazione di energia di un nodo, rendendolo principalmente legato al processore. I ricercatori informatici hanno anche notato durante i loro esperimenti che le versioni obsolete di una CPU potrebbero costituire una delle cause prindipali di inconsistenze energetiche.

2.3 Strumenti software

A differenza delle strumentazioni hardware, le tecnologie software non richiedono requisiti fisici specifici, dunque è sempre possibile farne uso a patto che siano compatibili con il tipo di ambiente e sistema operativo su cui si intende utilizzarli. Tuttavia, la problematica principale di questo tipo di strumenti è legata alla loro reperibilità e disponibilità. Non esiste di fatto un framework comune che permetta a un eventuale gruppo di ingegneri del software di poter condividere facilmente i risultati ottenuti. Per tale motivo, chiunque voglia interessarsi alla problematica del consumo energetico del software, si vede costretto ad affrontare una scelta tra l'utilizzo di uno dei tanti tool messi a disposizione nelle repository sul web oppure lo sviluppo di uno specifico per i propri obbiettivi. Nella letteratura scientifica spesso sono stati utilizzati i seguenti servizi:

- **GreenMiner Web Service**: il servizio web Green Miner [7] abilita la distribuzione e raccolta di test di green mining e relativi risultati. Esso è responsabile dell'aggregazione, della persistenza, dell'archiviazione, dell'analisi dei dati, dei test e della pianificazione delle esecuzioni dei test. Inoltre, ne traccia i risultati e fornisce l'accesso a dati grezzi e aggregati.

- **GreenTracker**: Al fine di stimare il consumo di energia di qualsiasi applicazione software distribuita, Green Tracker [10] raccoglie informazioni sulla CPU del computer. Esegue alcuni compiti iniziali di "benchmarking". Agli utenti viene richiesto di specificare quali solo le classi di software che desiderano testare e richiede di specificare in quali sistemi software ogni classe è attualmente installata sul proprio computer. Green Tracker cerca quindi di determinare il consumo medio della CPU per ciascuna delle classi specificate. Apre prima un programma specifico (ad esempio il browser) e poi, in un intervallo specificato, salva un file di testo contenente i seguenti elementi: timestamp, ID del processo, utilizzo della CPU e nome del comando[].

- **Seo's Framework**: Seo e al.[11] hanno definito un framework per la stima del consumo energetico di sistemi software pervasivi basati su Java. L'obiettivo principale del framework è quello di consentire ad un ingegnere di prendere decisioni quando si adatta l'architettura di un sistema, in modo tale che venga ridotto il consumo di energia dei dispositivi hardware con una durata della batteria limitata. Il framework assume esplicitamente una prospettiva basata sui componenti, il che lo rende adatto a un'ampia classe di odierne applicazioni distribuite, integrate e pervasive e fornisce un approccio

che facilita la stima del consumo energetico di un sistema sia durante la costruzione sia a runtime.

E' importante ricordare che questi sono solo alcune delle alternative possibili e che la varietà di questi prodotti comporta inevitabilmente una difficoltà maggiore nel confrontare i profili energetici prodotti a causa dei diversi parametri presi in considerazione.

CAPITOLO 3

Background Study

3.1 Introduzione

Nelle fasi inziali di progettazione del nostro lavoro, è stato necessaria un'attenta ricerca di lavori simili a quello proposto in questa tesi al fine di trovarne uno che potesse fungere sia come punto di riferimento sia di confronto. Al termine di questo ricerca è stato selezionato un articolo di Samir Hasan e al. [8], in cui si discute il ruolo delle strutture dati nell'almbito del consumo energetico del software. Gli autori dell'articolo hanno creato profili dettagliati dell'energia consumata dalle operazioni comuni effettuate su List, Map e Set utilizzando l'architettura *GreenMiner* [7]. I risultati prodotti mostrano che i tipi di dati alternativi per queste astrazioni differiscono notevolmente in termini di consumo energetico a seconda delle operazioni prese in considerazione.

3.2 Decostruzione dello studio

3.2.1 Classi analizzate

Le classi Java Collections memorizzano gruppi di oggetti e forniscono API per accedere, modificare o iterare sugli elementi. Java fornisce implementazioni riutilizzabili e convenienti di strutture dati e algoritmi popolari mediante Java Collection Framework (JFC). E' bene ricordare che esitono anche molte altre implementazioni da parte di terzi di strutture analoghe.

Nell'articolo oltre alle strutture offerte da JFC, il team ha anche analizzato quelle messe a disposizione da Apache Commons Collections e da Trove.

3.2.2 Processo di misurazione

Per misurare l'effetto dell'utilizzo di diverse collezioni su diversi carichi di lavoro, è stata creata un'applicazione Android di base. Quest'applicazione di test visualizza uno schermo vuoto inattivo, dove ogni test unitario per la test-app è un esperimento separato. In ognuno di essi è stata creata una classe Collections, inizializzata con alcuni dati e successivamente è stata svolta un particolare operazione (inserimento, iterazione etc.) su di essa. L'energia consumata dal test è stato misurata e registrata con il framework di riferimento Green Miner. In ogni test, sono stati aggiunti N elementi all'interno (all'inizio, nel mezzo o alla fine in base al caso d'uso) della struttura dati (il valore di N varia da 1 a 5000, in totale sono state utilizzate 13 taglie di input diverse). Ogni run di GreenMiner esegue test unitari per un uso specifico caso(ad esempio l'inserimento all'inizio di una lista). Ogni test-case, dati tutti i parametri, è stato eseguito 20 volte, ed i risultati sono stati raccolti e raggruppati, per determinare una media delle 20 misurazioni effettuate. Ciò ha fornito ulteriore coerenza ai dati che gli autori hanno raccolto da GreenMiner. Alla fine i dati prodotti sono stati plottati per generare dei profili energetici per ogni caso d'uso. Questo approccio presentava tuttavia alcuni problemi. In primo luogo, dovevano assicurarsi che ogni unità di prova riscontrasse lo stesso overhead. In secondo luogo, dato che i frammenti di codice erano di piccole dimensioni, l'energia consumata sarebbe potuta risultare troppo ridotta per essere osservabile. In terzo luogo, la Java Virtual Machine (JVM) può invocare il Garbage Collector in fasi indefinite del test, e questo potrebbe aver inquinato i risultati. Infine, l'attuale energia consumata da una suite di test varia da dispositivo a dispositivo. Il sistema GreenMiner è collegato a 4 dispositivi diversi; è lecito dunque pensare che l'energia effettiva consumata varia leggermente quando lo stesso test viene eseguito su dispositivi diversi. Per fare fronte a queste problematiche il team ha preso diverse misure tra cui:

Settaggio dell'overhead: è stato creata una nuova istanza di tutte le strutture con un metodo setUp(), indipendentemente da quella effettivamente usata per la prova particolare. Per esempio, quando si inseriscono elementi in una LinkedList, tutte e cinque le istanze della lista sono state create prima con il metodo setUp () per poi essere popolate nello step successivo.

Produzione di cambiamenti osservabili: all'interno di un metodo di test, gli autori hanno ripetuto l'invocazione delle API più volte. Per esempio, quando si inserivano 50 elementi, c'erano 20 tentativi di: (1) invocare setUp (); (2) effettuare gli inserimenti effettivi; (3) invocare tearDown()(tutti i test sono stati progettati in modo simile). Tenendo conto di ciò, i dati prodotti risultano essere un aggregato anziché riferirsi alla prestazione di una singola esecuzione. Questa scelta determina effetti osservabili sul consumo di energia rispetto alle dimensioni delle collezioni testate.

3.3 Risultati

Il team ha realizzato dei profili sul consumo energetico dei comuni metodi API forniti dalle implementazioni List, Map e Set, e registrato come esso varia con le dimensioni degli input. Nello specifico, si sono posti le seguenti sei domande di ricerca e hanno organizzato i loro risultati sulla base di esse:

E' possibile identificare un'implementazione List che offra un consumo ottimale per inserimenti, iterazione e accesso casuale?

E' possibile identificare un'implementazione Map che offra un consumo ottimale per inserimenti, iterazione e accesso casuale?

E' possibile identificare un'implementazione Set che offra un consumo ottimale per inserimenti, iterazione e accesso casuale?

Da quale dimensione di input le differenze energetiche diventano significative?

C'è una differenza notevole nel consumo di energia quando le collezioni contengono elementi diversi?

E' possibile utilizzare i profili per scegliere l'implementazione più efficiente dal punto di vista energetico di List, Map e Set?

I dettagli dei risultati contenenti maggiori informazioni sul lavoro svolto sono disponibili al seguente link: https://sites.google.com/site/collectionsenergy/

CAPITOLO 4

Data Structure Profiling

4.1 Premesse

La scelta della strumentazione è stata effettuata tentedo conto della reperibilità delle componenti e prendendo in cosiderazione l'architettura GreenMiner. A causa dell'indisponibilità del servizio web di Green Miner e alla sua complessità, abbiamo deciso di progettare un circuito più semplice che non fa uso di RawsberryPi. Inoltre ammiamo sostuito il test device dell'architettura (Galaxy Nexus) con uno smartphone piu recente (nel nostro caso uno Xiaomi modello 11T) a causa dell'obsolescenza del primo.

4.2 Prototipo

Il modello nella figura 3.1 presenta tutte le componenti hardware impiegate nello studio. Nel dettaglio, abbiamo utilizzato una scheda ArduinoUno (R3) collegata con un cavo usb al calcolatore che si occupa di gestire le misurazioni (vengono reperite tramite un opportuno script eseguito sull'ambiente ArduinoIDE) e connessa, tramite jumper cables, al chipset INA219, responsabile per l'effetivo tracciamento della corrente consumata(l'unità è stata posizionata su una breadboard per evitare la saldatura diretta dei cavi). Il sensore di corrente è a sua volta interconnesso con un jumper cable ad un modulo usb breakout, ovvero la porta d'ingresso del dispositivo da testare

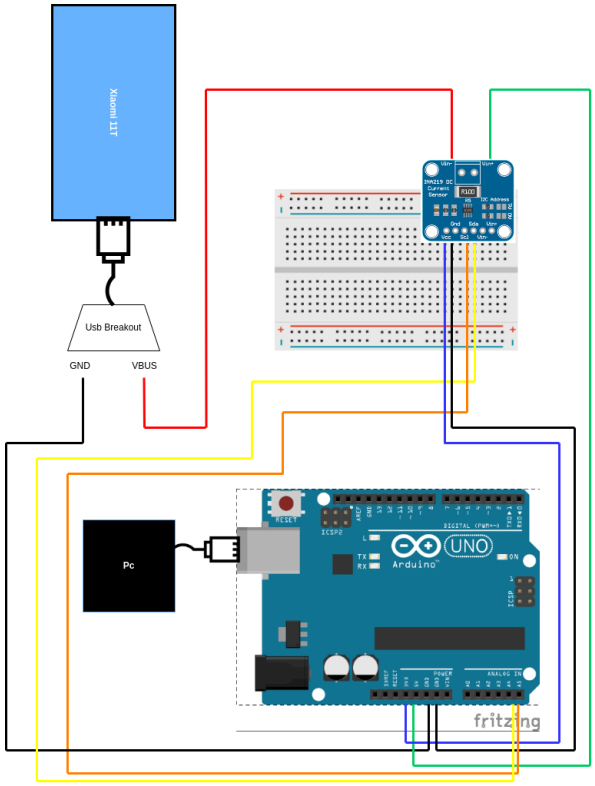

Figura 4.1: Componenti fisiche del framework, connesse tramise Usb Breakout, jumper calbes, Breadboard e interfaccia I2C

Arduino Uno R3 Board: Arduino Uno R3 [12] è una scheda microcontrollore basata su un microcontrollore ATmega328 AVR rimovibile, dual-inline-package (DIP). Dispone di 20 pin di ingresso/uscita digitali (di cui 6 possono essere utilizzati come uscite PWM e 6 possono essere utilizzati come ingressi analogici).

INA219: L'INA219 [13] è un monitor di potenza e di corrente di derivazione con un'interfaccia compatibile con i moduli I2C e SMBUS. Il dispositivo monitora sia la variazione dell'intensità della corrente di derivazione sia la tensione di alimentazione del bus, con tempi di conversione e filtraggio programmabili. Un valore di calibrazione programmabile, combinato con un moltiplicatore interno, consente la lettura diretta della corrente in ampere. Un ulteriore registro moltiplicatore calcola la potenza in watt. L'interfaccia dispone di 16 indirizzi programmabili. L'INA219 rileva i cambiamenit sui bus che possono variare da 0 a 26 V. Il dispositivo utilizza un'unica alimentazione da 3 a 5,5 V, assorbendo un massimo di 1 mA di corrente. L'INA219 funziona correttamente se la temperatura varia tra –40°C e 125°C.

4.3 Scripting

Ai fini del nostro studio, abbiamo scritto e utilizzato il seguente script:

```
#include <Wire.h>
#include <Adafruit_INA219.h>

  Adafruit_INA219 ina219;

 float voltage_V = 0,shuntVoltage_mV,busVoltage_V;
 float current_mA = 0;
 float power_mW = 0;
 float energy_Wh=0;
 long time_s=0;

  void setup(void)
  {
     Serial.begin(115200);//Settaggio baud per la lettura del
     segnale analogico
     uint32_t currentFrequency;
     ina219.begin();inizializza i registri di INA219
     Serial.println("Measuring voltage and current with INA219");
  }

  void loop(void) //richiama getData
  {
    getData();
    delay(2000);
  }

  void getData()//funzione di misurazione
  {
    // converte time to sec
    time_s=millis()/(1000);
    //get V+
```

```
    busVoltage_V = ina219.getBusVoltage_V();
     // get (V+ - V-), ritorna un valore in millivolt
    shuntVoltage_mV = ina219.getShuntVoltage_mV();
    //millivolts effettivamente consumati
    voltage_V = busVoltage_V + (shuntVoltage_mV / 1000);
    //restituisce la corrente secondo la legge di Ohm
    current_mA = ina219.getCurrent_mA();
    //calcolo dell'energia consumata in watt
    power_mW=current_mA*voltage_V;
    //energia in watt/hour
    energy_Wh=(power_mW*time_s)/3600;


     Serial.print("Bus␣Voltage:␣␣␣");
     Serial.print(busVoltage_V); Serial.println("␣V");
     Serial.print("Shunt␣Voltage:␣");
     Serial.print(shuntVoltage_mV); Serial.println("␣mV");
     Serial.print("Load␣Voltage:␣␣");
     Serial.print(voltage_V); Serial.println("␣V");
     Serial.print("Current:␣␣␣␣␣␣␣");
     Serial.print(current_mA); Serial.println("␣mA");
     Serial.print("Power:␣␣␣␣␣␣␣␣␣");
     Serial.print(power_mW); Serial.println("␣mW");
     Serial.print("Energy:␣␣␣␣␣␣␣␣");
     Serial.print(energy_Wh); Serial.println("␣mWh");
     Serial.println("-----------------------------------------");
}
```

Bus Voltage	V+ mV
Shunt Voltage	[(V+) - (V-)] mV
Load Voltage	Bus - (Shunt/1000) mV
Current	mA
Power	Current * Load mW
Energy	Power * Time(s) mWh

Figura 4.2: La figura mostra le unità di misura utilizzate nella formalizzazione dei segnali digitali tramite ArduinoIDE

4.3.1 Parametri e unità di misura

Abbiamo formalizzato le letture dei segnali digitali ricevuti da ArduinoR3 e nello step successivo i dati estratti sono stati organizzati secondo la figura 4.1.

4.4 SetUp

Per misurare l'effetto dell'utilizzo di diverse collezioni su diversi carichi di lavoro, abbiamo creato una semplice applicazione Android. Essa mostra a schermo un'interfaccia (ListView) responsive che permette di rimuovere o inserire nella struttura dati testata un numero arbitrario di oggetti String. Per massimizzare la precisione, l'applicazione è stata testata utilizzando una struttura dati per volta, dunque sono state eseguite una serie di test in maniera indipendente. L'energia consumata da ogni test è stata misurata tramite l'architettura descritta nella sezione 4.2 e successivamente utilizzata per definire dei profili energetici relativi ad ogni struttura analizzata. I dettagli implementativi dell'applicazione sono disponibili al seguente link: https://github.com/XJustUnluckyX/DataStrucutreTestApp/tree/master

4.5 Strutture Dati Analizzate

4.5.1 Map

Per quanto riguarda l'implementazione del tipo **Map**, Java Collection Framework offre le seguenti opzioni: **HashMap**, **LinkedHashMap** e **TreeMap**. I profili energetici mostrano chiaramente che TreeMap si è rivelata come la scelta peggiore in ogni tipo di operazione analizzata. Di conseguenza è opportuno focalizzarsi sul confronto tra HashMap e LinkedHshMap. Esse non presentano differenze significative e dunque è possibile considerarle come strutture dati intercambiabili, anche se HashMap tende ad avere un consumo medio inferiore rispetto a LinkedHashMap sebbene esso sia alquanto ridotto.

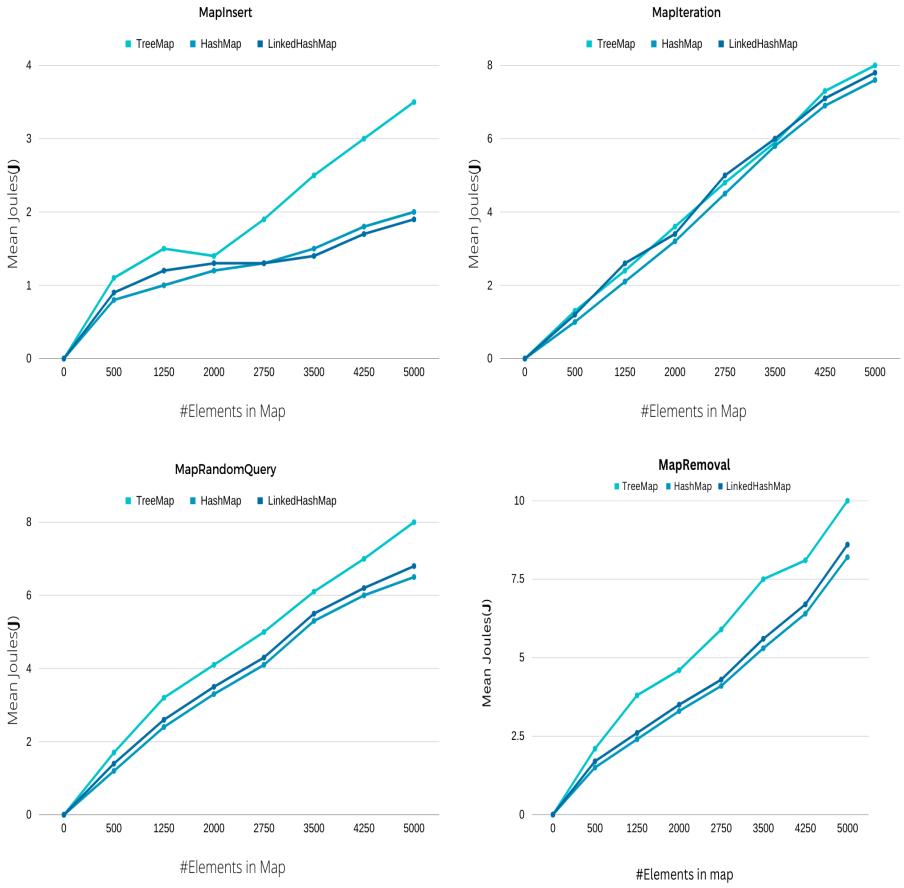

Figura 4.3: Profili energetici per Map. La figura mostra le prestazioni per le operazioni di inserimento, di rimozione, d'iterazione e di accesso casuale per TreeMap, HashMap, LinkedHashMap

4.5.2 Set

Per ciò che è relativo al tipo **Set**, le strutture dati messe a disposizione da JFC sono: **HashSet**, **LinkedHashSet** e **TreeSet**. E' possibile notare che il loro comportamento è pressochè identico a quello riscontrato durante il `profiling` della struttura dati Map.
Infatti, anche in questo caso la variante TreeSet rappresenta la scelta peggiore per l'implementazione di Set.
Viceversa HashSet e LinkedHashSet possiedono un grado di bontà notevolmente migliore della loro controparte e come visto per HashMap e LinkedHashMap, HashSet si è dimostrato migliore per tutte le operazioni sebbene di un margine alquanto ridotto.

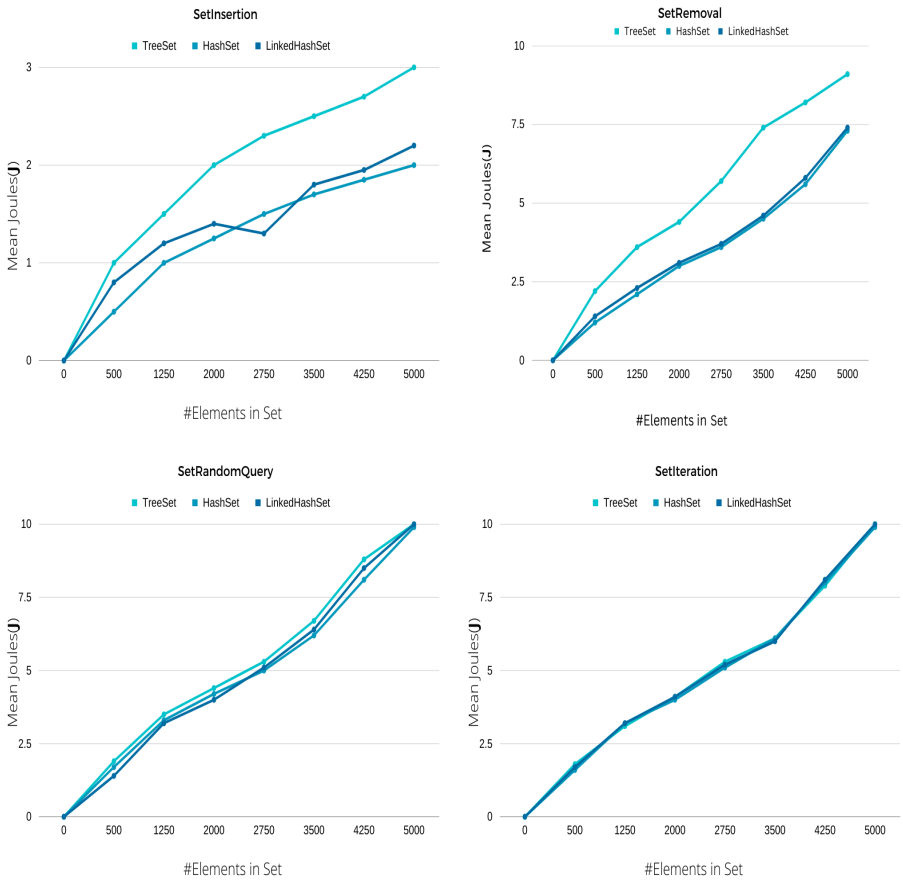

Figura 4.4: Profili energetici per Map. La figura mostra le prestazioni per le operazioni di inserimento, di rimozione, d'iterazione e di accesso casuale per TreeSet, HashSet, LinkedHashSet

4.5.3 List

Le strutture dati offerte da JFC [14] per la realizzazione del tipo **List** sono **ArrayList** e **LinkedList**, pertanto la nostra app utilizza un'implementazione per ciascuna di esse. Come è possibile notare dai profili energetici, le due strutture dati presentano un consumo medio abbstanza ridotto nei casi di operazioni di inserimento e di rimozione all'inizio e alla fine della lista. Tuttavia, è altrettanto evidente che la loro bontà diminuisca vertiginosamente per le stesse operazioni qualora esse vadano a modificarne la parte centrale.
In particolare, ArrayList si è dimostrata più vantaggiosa nei seguenti casi: inserimento nel mezzo, inserimento alla fine e ricerca casuale. L'operazione di iterazione semplice risulta essere mediamente efficiente in tutti i casi e la crescita della linea di consumo è regolare mentre la ricerca casuale è risultata essere la peggiore performante per LinkedList.

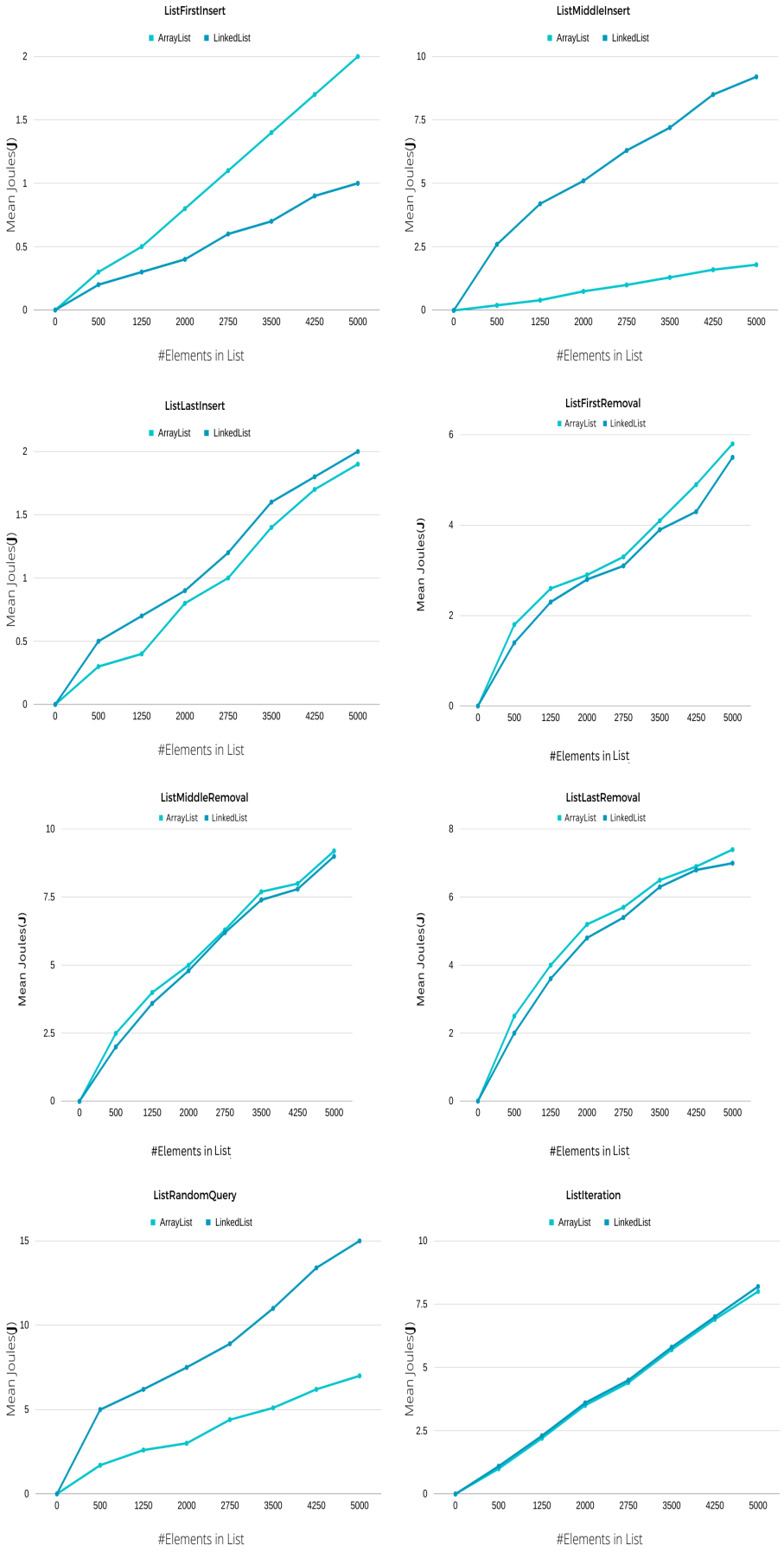

Figura 4.5: Profili energetici per List. La figura mostra le prestazioni per le operazioni di inserimento, di rimozione, d'iterazione e di accesso casuale per ArrayList e LinkedList

4.6 Quesiti e Metodo

I profili energetici, che abbiamo definito, confrontano le classi di *Collections* per ogni metodo API e suggeririscono migliori alternative. Tuttavia, quando le istanze di JCF [14] vengono utilizzate nelle applicazioni, più metodi API vengono invocati su ciascun oggetto, a seconda del ruolo dell'oggetto nel sistema e del carico del sistema. Ci aspettiamo quindi che l'impronta energetica di ogni oggetto in un'applicazione è determinata da una combinazione dell'impatto energetico di tutti i metodi API invocati. Per verificare questa supposizione ci siamo posti due domande fondamentali:

- Le diverse classi di *Collections* hanno un impatto energetico nelle applicazioni reali rispetto a quello che abbiamo trovato per strutture simili nei profili? In caso di risposta affermativa, quanto è grande l'impatto?
- Possiamo utilizzare i profili energetici per scegliere una classe alternativa e migliorare (o degradare) il consumo energetico di un'applicazione?

Per rispondere alla prima domanda, abbiamo modificato il codice sorgente delle test suites di librerie reali utilizzando classi alternative e abbiamo misurato l'energia consumata dal codice modificato. Inoltre per dare una risposta al secondo quesito abbiamo inserito all'interno del codice una modifica che va a sostituire la struttura dati utilizzata con una che, secondo i profili, vada a decrementare l'efficienza dell'applicazione.
Abbiamo rilevato tre tipi di istanze di *Collections*:

- istanze dichiarate come campi di una classe e utilizzate in più metodi;
- istanze create localmente in un metodo e utilizzate al suo intenro;
- istanze create localmente all'interno di metodi ma usate anche al loro esterno poiché vengono restituite come loro output;

Ogni insieme di istanze rilevato è stato inoltre sottoposto ad una verifica manuale per verificare la correttezza dei *call graphs* generati dalla libreria.
Per ogni istanza trovata, abbiamo identificato se essa è utilizzata in modo appropriato o se è disponibile un' alternativa migliore. Ad esempio, se una LinkedList viene utilizzata in un programma per inserimenti alla fine e iterazioni, consultando i nostri profili, ne sostituiremo l'istanza con una di ArrayList. Viceversa, abbiamo preparato delle versioni «peggiori» andando contro i risultati dei nostri profili. Dunque, dato che ArrayList è più efficiente dal punto di vista energetico di LinkedList per le operazioni comuni su una lista, invece di seguire la

scelta raccomandata, la versione «peggiore» del codice sostituisce ArrayList con LinkedList, prevedendo che questo cambiamento ne aumenti il consumo.
Le librerie testate includono: *Google JSON*, *Apache Commons Math*, *XStream* e *Apache Commons Configuration*.

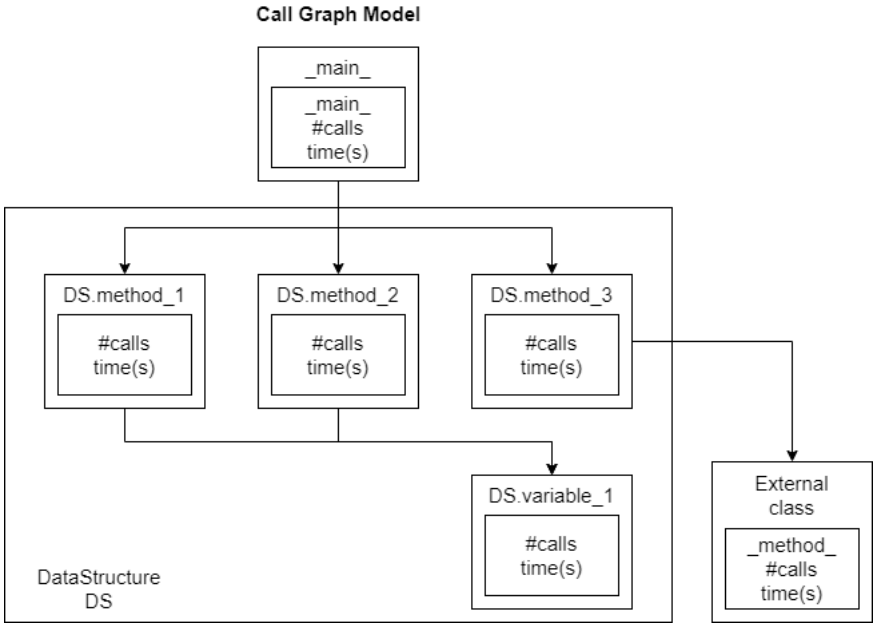

Figura 4.6: Schematizzazione callgraph

4.6.1 Google JSON

Gson [15] è una libreria che può essere usata per convertire gli oggetti Java nella loro rappresentazione JSON. Può anche essere usato per convertire una stringa JSON in un oggetto Java equivalente. Gson può lavorare con oggetti Java arbitrari, inclusi oggetti preesistenti di cui non si dispone di codice sorgente. Abbiamo utilizzato la versione 2.1 per il nostro studio, composta da circa 13 KLOC e una test suite di 16 KLOC.

Modifiche apportate: abbiamo trovato 53 istanze di ArrayList. Solo 4 di questi casi fanno parte del codice della libreria, mentre 49 si collocano nella test suite. Abbiamo studiato il codice delle istanze e cambiato 47 istanze in LinkedList. Tutte le istanze che sono state modificare vengono usate per inserimenti alla fine e operazioni di iterazione, dunque il cambiamento introdotto dovrebbe impattare negativamente sul rendimento energetico della libreria;

Impatto Energetico: con LinkedList il consumo energetico aumenta del 255% circa. Abbiamo identificato due fattori che contribuiscono a questo forte aumento. In primo luogo, la libreria

esegue i propri compiti dalle 3 alle 4 volte più lentamente quando vengono utilizzate le istanze LinkedList, che dunque portano ad un maggiore consumo di energia durante la fase ti test. In secondo luogo, la test suite di Gson ha numerosi test di prestazione che eseguono serializzazione e deserializzazione su input di grandi dimensioni che variano dai 2 ai 4 MB;

4.6.2 Apache Commons Math

La libreria Apache Commons Math [16] fornisce implementazioni di algoritmi matematici e statistici che non sono altrimenti disponibili nella distribuzione Java standard. Abbiamo utilizzato la versione 3.4.1 (209 KLOC app e 137 KLOC test). La libreria utilizza circa 167 istanze di ArrayList, mentre ne abbiamo contato circa 91 istanze nella test suite.

Modifiche apportate: su 258 istanze di ArrayList nel codebase e nella test suite, abbiamo individuato 83 istanze che erano usate principalmente per inserzioni finali (60 occorrenze), iterazioni (18 occorrenze) e accesso casuale (5 occorrenze). Dopo un'ispezione manuale, abbiamo trovato 169 istanze in piu che venivano usato allo stesso modo. I nostri profili suggeriscono che LinkedList è una scelta peggiorativa per questi casi e dunque abbiamo inserito questa modifica all'interno del codice;

Impatto Energetico: la versione modificata consumava il 12% in più di energia rispetto a quella precedente. Anche in questo caso, la libreria cambiata viene eseguita con circa 1.2 secondi di ritardo rispetto all' originale, e quindi consuma più energia durante l'esecuzione dei test case;

4.6.3 XStream

La libreria XStream [17] può essere usata per serializzare oggetti Java in XML e anche per deserializzarli. Abbiamo utilizzato la versione 1.5 con 34 KLOC di codebase e una suite di test di 30 KLOC. Sono state indetificate 33 istanze ArrayList nel codice della libreria e 128 nel codice di test.

Modifiche apportate: su un totale di 161 istanze di ArrayList, 156 istanze sono state convertite in LinkedList, prevedendo un maggiore consumo energetico;

Impatto Energetico: è stata rilevata una degradazione del 3.8% in caso di swapping delle istanze ArrayList con LinkedList. La versione modificata viene eseguita 1,2 volte più lentamente dell'originale determinando di conseguenza un maggiore consumo di energia;

4.6.4 Apache Commons Configuration

La libreria Apache Commons Configuration [18] facilita l'archiviazione e il recupero delle informazioni di configurazione per applicazioni Java. Abbiamo studiato la versione 1.10 che è costituita da 40 KLOC di codice di libreria, e una suite di test di 36 KLOC. Sono state individuate un totale di 13 istanze di LinkedList e 166 istanze di ArrayList.

Modifiche apportate: di queste 13 istanze LinkedList, 8 sono state utilizzate per inserimenti a fine lista e iterazione. Abbiamo trovato manualmente le altre 5, utilizzate con un *pattern* simile. Dal momento che i nostri profili indicano che ArrayList è una scelta migliore per queste operazioni, abbiamo cambiato queste 13 occorrenze in istanze di ArrayList, prevedendo una diminuzione dell'energia consumata dalla test suite.

Impatto Energetico: la modifica delle istanze LinkedList in ArrayList ha migliorato il consumo energetico del 2.5% circa. La versione modificata della libreria è stata eseguita 1.08 volte più velocemente rispetto all'originale, che è, probabilmente, il motivo per cui la quantità di energia consumata è risulta inferiore.

	Tipo	#Istanze	Tipo Modificato	Impatto	Consumo	Tempo
Gjson	ArrayList	53	LinkedList	NEGATIVO	^255%	^ 3-4 volte
Apache C.Math	ArrayList	252	LinkedList	NEGATIVO	^ 12%	^ 1.2 s
Xstream	ArrayList	156	LinkedList	NEGATIVO	^ 3,8%	^ 1,2 volte
Apache C.C.	LinkedList	13	ArrayList	POSITIVO	v 2.5%	v 1,08 volte

Figura 4.7: La figura sintetizza le modifiche apportate alla test suite di ogni libreria testata e le caratteristiche su cui esse hanno avuto effetto (tempo di esecuzione e consumo energetico).

CAPITOLO 5

Minacce alla validità

Esistono diversi fattori che incidono sulla validità del nostro lavoro e, per facilitarne la comprensione, abbiamo ritenuto opportuno individuare delle categorie in cui collocare ciascuna delle minacce rilevate (in base alle limitazioni che esse introducono all'interno dello studio).

Esse si dividono in:

- *Conclusion*: che influiscono sulla capacità di trarre conclusioni sulle relazioni tra le variabili indipendenti e le variabili dipendenti;
- *Internal*: che possono influenzare le variabili indipendenti rispetto alla causalità;
- *External*: condizioni che limitano la capacità di generalizzare i risultati dello studio;

5.1 *Conclusion*

Innanzitutto l'ostacolo piu grande, che riguarda l'affidabilità dei profili energetici e dei dati prodotti, è costituito dal rumore termico (interno) nei sistemi fisici di misurazione; in particolare gli smartphone si distinguono tra gli altri tipi di dispositivi poichè essi sono fortemente rumorosi e non deterministici a causa delle resistenze presenti nella batteria (questo aspetto ci ha portati a ripetere le nostre misurazioni almeno 15 volte per controllare le differenze di misura tra l'una e le altre). Inoltre un altro aspetto che potrebbe compromettere i risultati è quello della temperatura esterna, in quanto abbiamo verificato iterativamente che

il dispositivo di test presenta consumi energetici diversi a seconda delle condizioni climatiche esterne (maggiore la temperatura maggiore il consumo). E' dunque necessario tenere conto del fatto che le caratteristiche che cerchiamo di quantificare possono variare a causa dello stesso processo di misurazione. Ad esempio, il consumo di una qualsiasi linea di codice sarà probabilmente influenzata dall'interno framework di misurazione, riducendo così l'affidabilità dell'esperimento. Dunque se l'influenza su quest'ultime è minima o costante, allora potremmo scegliere di ignorarla oppure si potrebbe decidere di fattorizzarla ed escluderla in ipotetiche fasi successive di ottimizzazione dei dati. Tuttavia, poiché il funzionamento di un frammento di codice è altamente sensibile al contesto in cui essa viene eseguita, non possiamo prevedere con precisione se l'effetto della strumentazione risulti costante o minimale. Un'altra questione questione più sottile può sorgere a causa della gamma dei numeri che abbiamo raggiunto. Come abbiamo visto nei profili, la quantità di energia consumata è piuttosto piccola, specialmente per le piccole collezioni.

5.2 *Internal*

Come gia illustrato nelle sezioni precedenti, abbiamo optato per uno specifico framework di lavoro con delle caratteristiche hardware e software ben definite. Naturalmente questa scelta differisce da quello dello studio presentanto nel terzo capitolo e differisce anche dagli altri lavori citati all'interno della nostra analisi. Di conseguenza è opportuno mettere in evidenza un'altra problematica fondamentale, ovvero stabilire quale delle combinazioni hardware/software massimizzi la precisione e l'attendibilità dei risultati e minimizzi, allo stesso tempo, i costi di assemblaggio e di *deployment*. Si tratta di una scelta cruciale in quanto essa determina in gran parte l'interno andamento delle varia iterazioni dell'esperimento; dunque è necessario optare per una soluzione che bilanci questi due aspetti nel miglior modo possibile (bisogna quindi determinare a priori quali tecnologie e strumenti potrebbero essere adoperati e quali no).

5.3 *External*

Nei capitoli precedenti è stato affermato che, al fine di produrre cambiamenti evidenti e osservabili nelle misure rilevate, le strutture dati testate sono state popolate in modo opportuno variando più volte la loro effetiva size, ovvero il numero di record inseriti al loro interno. Questa scelta è stata in qualche modo "forzata", poichè ci siamo accorti che il

consumo energetico effettivo per i metodi applicati a collezioni di dimensioni trascurabili (poche decine di record) risulta quasi indistinguibile da quello dell'applicativo testato senza fare uso di essi. Questa questione cosi delicata sulla taglia dell'input pone dei dubbi su un altro aspetto del nostro lavoro: il riscontro tra l'esperimento e la realtà (svolgendo una semplice analisi del contenuto è plausibile affermare che le linee guida fornite al termine dello studio siano applicabili principalmente a contesti che prevedono l'uso di strutture dati di dimensioni relativamente grandi).

CAPITOLO 6

Conclusioni e sviluppi futuri

6.1 Conclusioni

I nostri risultati forniscono una linea guida su alcuni scenari in cui i consumi energetici delle collezioni e delle classi ad esse associate divengono un punto focale nel contesto colletivo del settore dello sviluppo di software. Nel complesso, i nostri risultati saranno particolarmente utili per gli sviluppatori di software su larga scala che comunemente lavorano con grandi istanze di collezioni, e potranno guidare quest'ultimi nell'essere consapevoli dell'impatto energetico delle loro scelte implementative. Infine, il nostro lavoro potrebbe motivare lavori futuri, simili e non , per la definizione di nuovi approcci programmatici al fine di attribuire ai vari prodotti software una vera e propria classe energetica di appartenenza. Quest'ultima osservazione in particolare potrebbe rivelarsi utile non solo ai programmatori individuali ma anche ai grandi colossi del settore dello sviluppo informatico che, in questo modo, avrebbero a disposizione una metrica valutativa che potrebbe influenzare la scelta di un'eventuale prodotto di terzi parti da includere nella propria soluzione.

6.2 Sviluppi futuri

In futuro prevediamo di estendere il `profiling` svolto includendo nell'analisi anche le classe di collezioni non considerate precedentemente (cio implicherebbe una maggiore validità delle nostre stime rispetto a casi reali di programmi complessi che fanno uso di piu

framework simultaneamente). Inoltre, al fine di incrementare l'usabilità dei nostri profili, sarebbe opportuno reiterare l'esperimento svolto utilizzando una o piu architetture hardware /software diverse e metterne in evidenza i cambiamenti, se presenti(in questo modo si andrebbe a definire una forma di *working ground* comune con chiunque abbia svolto un lavoro simile al nostro).

Ringraziamenti

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non sarebbe stato possibile. Un ringraziamento particolare va al mio relatore Palomba che mi ha seguito, con la sua disponibilità, in ogni step della realizzazione dell'elaborato, fin dalla scelta dell'argomento. Grazie anche al mio correlatore Di Nucci e alla dottoressa Pontillo per i loro consigli e per avermi suggerito puntualmente le giuste modifiche da apportare alla mia tesi. Ringrazio di cuore la mia famiglia per avermi supportato nella mia scelta di intraprendere questo percorso universitario, per avermi permesso di portare a termine gli studi cosi da fondare le basi della mia futura carriera professionale e per il supporto che ricevo tutt'ora. Un ringraziamento particolare va alla mia ragazza Federica che mi è stata sempre a fianco nei momenti difficili, che ha contribuito con le sue idee a dare un tocco di originalità a questo elaborato e in cui ho sempre avuto modo di trovare un punto di riferimento. Infine, vorrei dedicare questo piccolo traguardo a me stesso, che possa essere l'incentivo giusto per continuare a impegnarmi e a realizzare le mie aspirazioni.

Bibliografia

[1] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 1–12, 2015. (Citato a pagina 3)

[2] "running-average-power-limit-energy." https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. (Citato a pagina 4)

[3] "Watts up pro." https://www.powermeterstore.com/p1206/watts_up_pro.php. (Citato a pagina 4)

[4] "Powermon." https://www.thornwave.com/products/bluetooth-battery-monitor-dc-power-meter. (Citato a pagina 5)

[5] D. Bedard, R. J. Fowler, M. Y. Lim, and A. Porterfield, "Powermon 2: Fine-grained, integrated power measurement," 2010. (Citato a pagina 5)

[6] J. Laros, P. Pokorny, and D. DeBonis, "Powerinsight - a commodity power measurement capability," pp. 1–6, 06 2013. (Citato a pagina 5)

[7] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. Campbell, , and S. Romansky, "Greenminer: a hardware based mining software repositories software energy consumption framework," in *International Working Conference on Mining Software Repositories (MSR 2014)*, pp. 12–21, 2014. (Citato alle pagine 5, 7 e 9)

[8] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *International Conference on Software Engineering (ICSE 2016)*, pp. 225–236, 2016. (Citato alle pagine 5 e 9)

[9] J. von Kistowski, H. Block, J. Beckett, C. Spradling, K.-D. Lange, and S. Kounev, "Variations in cpu power consumption," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, (New York, NY, USA), p. 147–158, Association for Computing Machinery, 2016. (Citato a pagina 6)

[10] N. Amsel and B. Tomlinson, "Green tracker: a tool for estimating the energy consumption of software," in *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Extended Abstracts Volume, Atlanta, Georgia, USA, April 10-15, 2010* (E. D. Mynatt, D. Schoner, G. Fitzpatrick, S. E. Hudson, W. K. Edwards, and T. Rodden, eds.), pp. 3337–3342, ACM, 2010. (Citato a pagina 7)

[11] C. Seo, S. Malek, and N. Medvidovic, "Estimating the energy consumption in pervasive java-based systems," in *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 243–247, 2008. (Citato a pagina 7)

[12] "Arduino uno r3." https://docs.arduino.cc/hardware/uno-rev3. (Citato a pagina 13)

[13] "Ina219." https://www.alldatasheet.com/view.jsp?Searchword=INA219AIDCN. (Citato a pagina 13)

[14] "Java collection framework." http://docs.oracle.com/javase/8/docs/technotes/guides/collections/. (Citato alle pagine 19 e 21)

[15] "Googlejson." https://github.com/google/gson. (Citato a pagina 22)

[16] "Apache common math." https://commons.apache.org/proper/commons-math/. (Citato a pagina 23)

[17] "Xstream." https://x-stream.github.io/. (Citato a pagina 23)

[18] "Apache common configuration." https://commons.apache.org/proper/commons-configuration/. (Citato a pagina 24)