

UNIVERSITÀ DEGLI STUDI DEL SANNIO

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

**Data science**

---

Homework 3

---

*Prof:*

Pecchia Antonio

*Studenti:*

Cinelli Jessica, 399000529

Mazzitelli Francesco C., 399000532

ANNO ACCADEMICO 2022-2023

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Hadoop e paradigma Map-Reduce</b>	<b>4</b>
2.1	Inizializzazione del file system distribuito . . . . .	4
2.2	Somma delle visite per mese . . . . .	6
2.3	Conteggio del numero di siti unici . . . . .	7
2.4	Conteggio del numero minimo di visite . . . . .	8
<b>3</b>	<b>Storm e analisi di flussi di dati</b>	<b>9</b>
3.1	Inizializzazione Storm . . . . .	10
3.2	Applicazione Storm: Temperature Topology . . . . .	11
3.3	Risultati delle elaborazioni con Storm . . . . .	12
	<b>Appendice A Somma delle visite per mese</b>	<b>15</b>
A.1	Mapper . . . . .	15
A.2	Reducer . . . . .	15
A.3	Driver . . . . .	15
	<b>Appendice B Conteggio del numero di siti unici</b>	<b>16</b>
B.1	Mapper . . . . .	16
B.2	Reducer . . . . .	16
B.3	Driver . . . . .	16
	<b>Appendice C Conteggio del numero minimo di visite</b>	<b>17</b>
C.1	Mapper . . . . .	17
C.2	Reducer . . . . .	17
C.3	Driver . . . . .	17
	<b>Appendice D Applicazione Storm: TemperatureTopology</b>	<b>18</b>
D.1	SocketSpout.java . . . . .	18
D.2	SamplingBolt.java . . . . .	19
D.3	AlertBolt.java . . . . .	20
D.4	AlertLogBolt.java . . . . .	21
D.5	TemperatureTopology . . . . .	22

## Elenco delle figure

1	Tools . . . . .	3
2	Inizializzazione File System . . . . .	5
3	Creazione cartelle e push dei dati . . . . .	5
4	Esecuzione del JAR . . . . .	6

5	Risultato somma delle visite . . . . .	6
6	Risultato conteggio del numero di siti unici . . . . .	7
7	Risultato conteggio del numero minimo di visite . . . . .	8
8	Esempio topologia di un'applicazione Storm . . . . .	9
9	Risultato conteggio del numero minimo di visite . . . . .	10
10	Topologia dell'applicazione Storm realizzata . . . . .	11
11	Risultati . . . . .	12
12	GUI . . . . .	13
13	Grafo della topologia dispiegata . . . . .	14

# 1 Introduzione

Il seguente studio<sup>1</sup> ha avuto come obiettivo lo studio e l'approfondimento di alcuni dei tool principali per l'analisi dei dati:

- **Hadoop**: piattaforma che agevola lo sviluppo di applicazioni Map-Reduce su file system distribuiti. Mette a disposizione un framework di sviluppo e l'Hadoop Distributed File System (HDFS), un file system distribuito scalabile e portabile.
- **Storm**: framework di calcolo per l'elaborazione di flussi distribuiti. Questo consente quindi di effettuare elaborazioni "real-time" tramite la costruzione di una topologia i cui nodi prendono di "spout" (Sorgente) e "bolt" (Componenti di elaborazione).

Entrambe le applicazioni sono state realizzate in Java utilizzando come riferimento quanto svolto durante le esercitazioni.



((a)) Hadoop



((b)) Storm

Figura 1: Tools

---

<sup>1</sup>È possibile visionare l'intero progetto al link <https://github.com/jessicacinelli/Homework3.git>.

## 2 Hadoop e paradigma Map-Reduce

Il framework MapReduce è composto da diverse funzioni per ogni step:

- **Input Reader:** componente adibita alla lettura dei dati dal file system distribuito e divide l'input in diversi split
- **Map Function:** componente adibita alla generazione di zero o più coppie (chiave, valore) in uscita.
- **Partition Function:** componente adibita alla memorizzazione sul disco e al partizionamento delle coppie bufferizzate in R sezioni .
- **Reduce Function:** componente adibita alla riduzione dei valori mappati in coppie chiave valore, generando di fatto un risultato globale.
- **Output Writer:** componente adibita alla scrittura dei risultati in memoria di massa e una volta finiti tutti i task di Map e Reduce.

Hadoop nasconde la maggior parte delle funzioni adibite alla gestione dei dati esponendo delle interfacce che consentono la sola implementazione delle funzioni di Map e Reduce, consentendo allo sviluppatore di concentrarsi solo sull'implementazione di tali funzioni.

### 2.1 Inizializzazione del file system distribuito

L'intero file system ha un'architettura gerarchica basata sull'interconnessione di due diverse tipologie di nodi:

- **Namenode:** Chiamato anche nodo master, può tenere traccia dei file, gestire il file system e accedere ai metadati di tutti i dati memorizzati al suo interno. In particolare, il namenode contiene dettagli sul numero di blocchi, le posizioni dei datanode in cui sono archiviati i dati, dove sono archiviate le repliche e altri dettagli.
- **Datanode:** Memorizza i dati in esso sotto forma di blocchi. Questo è anche noto come nodo slave e memorizza i dati effettivamente presenti nell'HDFS ed è responsabile di operazioni in lettura e scrittura da parte del client.

La prima operazione svolta è stata l'inizializzazione del Namenode e l'avvio del file system distribuito.

```

File Edit View Terminal Go Help
student@xubuntu-VirtualBox: ~/Desktop
2023-05-20 04:14:19,082 INFO namenode.NameNode: STARTUP_MSG:
*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = xubuntu-VirtualBox/127.0.0.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.2.1
STARTUP_MSG: classpath = /home/studente/hadoop-3.2.1/etc/hadoop:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/kerb-admin-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-text-1.4.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jai-imageio-1.7.25.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/httpclient-4.5.6.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/kerby-asn1-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-net-3.6.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jcip-annotations-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jsp-api-2.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-collections-3.2.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/woodstox-core-5.0.3.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jackson-xml-1.9.13.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/kerby-util-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jetty-servlet-9.3.24.v20180605.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/asm-5.0.4.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jackson-databind-2.9.8.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/kerb-server-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jetty-server-9.3.24.v20180605.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/error_prone_annotations-2.2.0.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/accessors-smart-1.2.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/animal-sniffer-annotations-1.17.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/javax-servlet-api-3.0.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jobjc-annotations-1.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jetty-http-9.3.24.v20180605.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-configuration-2.1.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jcxf-0.1.54.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/curator-recipes-2.13.0.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/kerby-pkix-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-compress-1.18.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/commons-io-2.5.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jackson-mapper-asl-1.9.13.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/nimbus-jose-jwt-4.4.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/jetty-webapp-9.3.24.v20180605.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/token-provider-1.0.1.jar:/home/studente/hadoop-3.2.1/share/hadoop/common/lib/rx2-1.1.jar:/home/studente/hadoop-3.2.

```

((a)) Inizializzazione Namenode

```

File Edit View Terminal Go Help
2023-05-20 04:14:22,211 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2023-05-20 04:14:22,222 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2023-05-20 04:14:22,222 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
2023-05-20 04:14:22,232 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2023-05-20 04:14:22,232 INFO util.GSet: VM type = 32-bit
2023-05-20 04:14:22,233 INFO util.GSet: 0.029999999329447746% max memory 786 MB = 241.5 KB
2023-05-20 04:14:22,233 INFO util.GSet: capacity = 2^16 = 65536 entries
2023-05-20 04:14:22,417 INFO namenode.FSImage: Allocated new BlockPoolId: BP-934547740-127.0.1.1-1684570462379
2023-05-20 04:14:22,503 INFO common.Storage: Storage directory /tmp/hadoop-studente/dfs/name has been successfully formatted.
2023-05-20 04:14:22,974 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-studente/dfs/name/current/fsimage.cpt.00000000000000000000 using no compression
2023-05-20 04:14:23,341 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-studente/dfs/name/current/fsimage.cpt.00000000000000000000 of size 403 bytes saved in 0 seconds
2023-05-20 04:14:23,446 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-20 04:14:23,486 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when metadata shutdown.
2023-05-20 04:14:23,487 INFO namenode.NameNode: SHUTDOWN_MSG:
*****
SHUTDOWN_MSG: Shutting down NameNode at xubuntu-VirtualBox/127.0.0.1
student@xubuntu-VirtualBox: ~/Desktop$ cd ..
student@xubuntu-VirtualBox: ~$ ls
apache-activemq-5.8.0 Downloads jms1.1 TESTfile1 zookeeper-3.4.12
apache-storm-1.2.3 hadoop-3.2.1 ossmc-hdfs-2.9.3 weka
Desktop JAR23 spark-2.4.5 workspaceJava
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ ls
bin include lib LICENSE.txt NOTICE.txt sbin studente
etc input libexec logs README.txt share test.txt
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [xubuntu-VirtualBox]
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$

```

((b)) Start DFS

Figura 2: Inizializzazione File System

Dopo aver inizializzato il file system distribuito sono state create le cartelle di output e di input e quest'ultima è stata popolata con i dati su cui effettuare le elaborazioni.

Browsing HDFS

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwx-r-x	studente	supergroup	0 B	May 20 04:19	0	0 B	input

Showing 1 to 1 of 1 entries

Previous Next

Hadoop, 2019.

File Edit View Terminal Go Help

```

$ = 1,5,25
2023-05-20 04:14:22,222 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2023-05-20 04:14:22,222 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
2023-05-20 04:14:22,232 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2023-05-20 04:14:22,232 INFO util.GSet: VM type = 32-bit
2023-05-20 04:14:22,233 INFO util.GSet: 0.029999999329447746% max memory 786 MB = 241.5 KB
2023-05-20 04:14:22,233 INFO util.GSet: capacity = 2^16 = 65536 entries
2023-05-20 04:14:22,417 INFO namenode.FSImage: Allocated new BlockPoolId: BP-934547740-127.0.1.1-1684570462379
2023-05-20 04:14:22,503 INFO common.Storage: Storage directory /tmp/hadoop-studente/dfs/name has been successfully formatted.
2023-05-20 04:14:22,974 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-studente/dfs/name/current/fsimage.cpt.00000000000000000000 using no compression
2023-05-20 04:14:23,341 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-studente/dfs/name/current/fsimage.cpt.00000000000000000000 of size 403 bytes saved in 0 seconds
2023-05-20 04:14:23,446 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-20 04:14:23,486 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when metadata shutdown.
2023-05-20 04:14:23,487 INFO namenode.NameNode: SHUTDOWN_MSG:
*****
SHUTDOWN_MSG: Shutting down NameNode at xubuntu-VirtualBox/127.0.0.1
student@xubuntu-VirtualBox: ~/Desktop$ cd ..
student@xubuntu-VirtualBox: ~$ ls
apache-activemq-5.8.0 Downloads jms1.1 TESTfile1 zookeeper-3.4.12
apache-storm-1.2.3 hadoop-3.2.1 ossmc-hdfs-2.9.3 weka
Desktop JAR23 spark-2.4.5 workspaceJava
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ ls
bin include lib LICENSE.txt NOTICE.txt sbin studente
etc input libexec logs README.txt share test.txt
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [xubuntu-VirtualBox]
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ hdfs dfs -mkdir /user
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ hdfs dfs -mkdir /user/studente
student@xubuntu-VirtualBox: ~/hadoop-3.2.1$ hdfs dfs -mkdir /user/studente/input

```

((a)) Creazione cartelle

Browsing HDFS

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--	studente	supergroup	375 B	May 20 04:25	1	128 MB	websites.txt

Showing 1 to 1 of 1 entries

Previous Next

Hadoop, 2019.

File Edit View Terminal Go Help

```

student@xubuntu-VirtualBox: ~/Desktop$ ls
student@xubuntu-VirtualBox: ~/Desktop$ hdfs dfs -put websites.txt /user/studente/input/websit
2023-05-20 04:25:21,786 INFO sasl.DataTransferClient: SASL encryption trust check: localH
ostTrusted = false, remoteHostTrusted = false
student@xubuntu-VirtualBox: ~/Desktop$

```

((b)) Caricamento dataset

Figura 3: Creazione cartelle e push dei dati

## 2.2 Somma delle visite per mese

L'obiettivo della prima applicazione è calcolare il numero di visite mensili per ogni mese. Sono state implementate le seguenti classi Java:

- Map: riceve in input il contenuto del file e produce in output una tupla (*chiave, valore*): la chiave è il mese, il valore è il numero di visite.
- Reduce: riceve in input la tupla prodotta dal Mapper e calcola il numero di visite mensili. Produce in output la coppia (*mese, numero di visite mensili*).
- Driver: ha il compito di abilitare l'esecuzione dell'applicazione e di gestirne le configurazioni.

L'implementazione dell'applicazione è consultabile nell'appendice A.

```

studente@ubuntu-VirtualBox:~/JAR23$ hadoop jar SumOfMonthlyVisit-1.jar visits.VisitsDriver input
2023-05-20 05:14:56,964 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2023-05-20 05:14:57,102 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s)
2023-05-20 05:14:57,102 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2023-05-20 05:14:57,418 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2023-05-20 05:14:57,680 INFO input.FileInputFormat: Total input files to process : 1
2023-05-20 05:14:57,736 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-20 05:14:58,110 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1173327027_0001
2023-05-20 05:14:58,110 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-05-20 05:14:58,277 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2023-05-20 05:14:58,279 INFO mapreduce.Job: Running job: job_local1173327027_0001
2023-05-20 05:14:58,280 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2023-05-20 05:14:58,292 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-05-20 05:14:58,293 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup_temporary folders under output directory:false, ignore cleanup failures: false
2023-05-20 05:14:58,294 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2023-05-20 05:14:58,366 INFO mapred.LocalJobRunner: Waiting for map tasks
2023-05-20 05:14:58,367 INFO mapred.LocalJobRunner: Starting task: attempt_local1173327027_0001_m_00000_0
2023-05-20 05:14:58,404 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-05-20 05:14:58,404 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup_temporary folders under output directory:false, ignore cleanup failures: false
2023-05-20 05:14:58,439 INFO mapred.Task: Using ResourceCalculatorProcessFree : [ ]
2023-05-20 05:14:58,444 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/user/studente/input/webcities.txt-0-375
2023-05-20 05:14:58,860 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2023-05-20 05:14:58,860 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2023-05-20 05:14:58,861 INFO mapred.MapTask: soft limit at 83886080
2023-05-20 05:14:58,861 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2023-05-20 05:14:58,861 INFO mapred.MapTask: kvstart = 26214396; length = 653600
2023-05-20 05:14:58,866 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTaskMapOutputBuffer
2023-05-20 05:14:58,912 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false

```

((a)) Esecuzione del JAR

Figura 4: Esecuzione del JAR

((a)) Risultato somma delle visite con un reducer

((b)) Risultato somma delle visite con più reducer

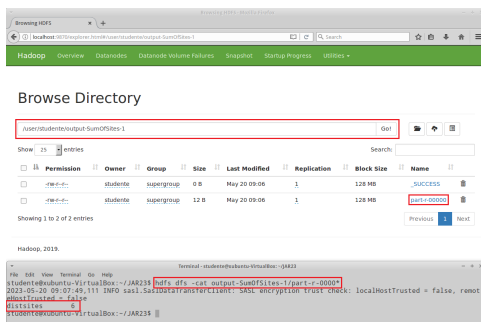
Figura 5: Risultato somma delle visite

## 2.3 Conteggio del numero di siti unici

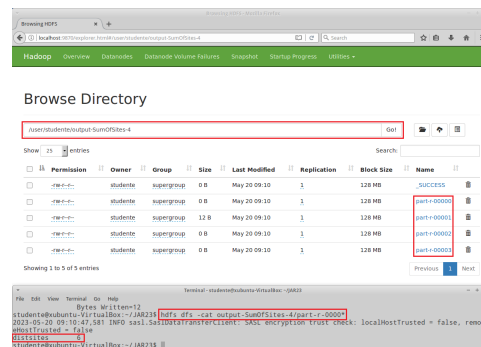
L'obiettivo della seconda applicazione è calcolare il numero di siti che compaiono all'interno del dataset. Sono state implementate le seguenti classi Java:

- Map: riceve in input il contenuto del file e produce in output una tupla (*chiave, valore*): la chiave è una stringa "risultato", il valore è una stringa rappresentante il nome del sito.
- Reduce: riceve in input la tupla prodotta dal Mapper e calcola il numero di siti unici. Si è utilizzato un HashSet che garantisce, tramite il salvataggio all'interno dello stesso, l'unicità della stringa *sito*. Il valore relativo al conteggio è ottenuto tramite il metodo `.size()` associato alla struttura dati. Produce in output la coppia (*Stringa, numero di siti unici*).
- Driver: ha il compito di abilitare l'esecuzione dell'applicazione e di gestirne le configurazioni.

L'implementazione dell'applicazione è consultabile nell'appendice B.



((a)) Risultato conteggio del numero di siti unici con un reducer



((b)) Risultato conteggio del numero di siti unici con più reducer

Figura 6: Risultato conteggio del numero di siti unici

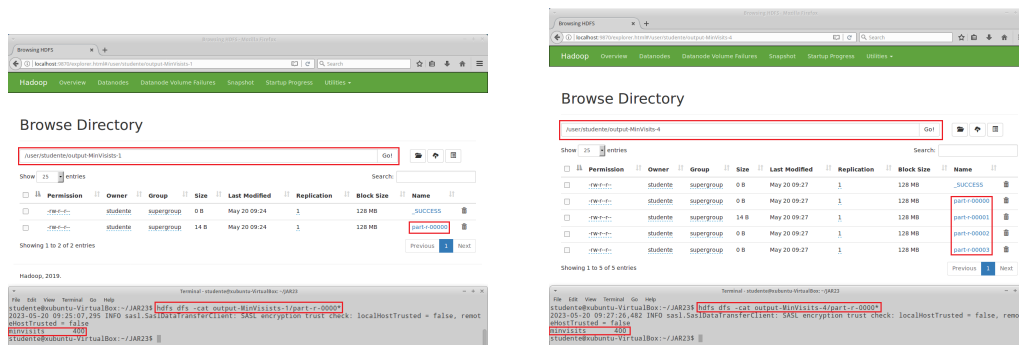


## 2.4 Conteggio del numero minimo di visite

L'obiettivo della terza applicazione è calcolare il numero minimo di visite dell'intero dataset. Sono state implementate le seguenti classi Java:

- Map: riceve in input il contenuto del file e produce in output una tupla (*chiave, valore*): la chiave è una stringa "risultato", il valore è una l'intero rappresentante il numero di visite per sito.
- Reduce: riceve in input la tupla prodotta dal Mapper e calcola il numero di siti unici. Si è utilizzata una lista che, tramite la funzione Collections.min(lista) ha agevolato le operazioni di calcolo del *minimo*. Produce in output la coppia (*Stringa, valore minimo di visite*).
- Driver: ha il compito di abilitare l'esecuzione dell'applicazione e di gestirne le configurazioni.

L'implementazione dell'applicazione è consultabile nell'appendice C.



((a)) Risultato conteggio del numero minimo di visite con un reducer

((b)) Risultato conteggio del numero minimo di visite con più reducer

Figura 7: Risultato conteggio del numero minimo di visite

### 3 Storm e analisi di flussi di dati

Storm è un framework che agevola la realizzazione di applicazioni orientate all'elaborazione dei flussi di dati. Il tutto si basa sulla realizzazione di una pipeline con una topologia ben definita caratterizzata dalle seguenti componenti:

- **Spout:** Elemento della pipeline che è in grado di agganciarsi alla sorgente dati, realizzare un'operazione preliminare di filtraggio dei dati ed emettere uno stream di dati filtrati verso i nodi con competenze elaborative.
- **Bolt:** Elemento della pipeline che consuma lo stream di dati per svolgervi operazioni di:
  - Filtraggio
  - Summarization
  - Aggregazioni
  - Join
  - Salvataggio su un DB
  - Creazione di un nuovo stream

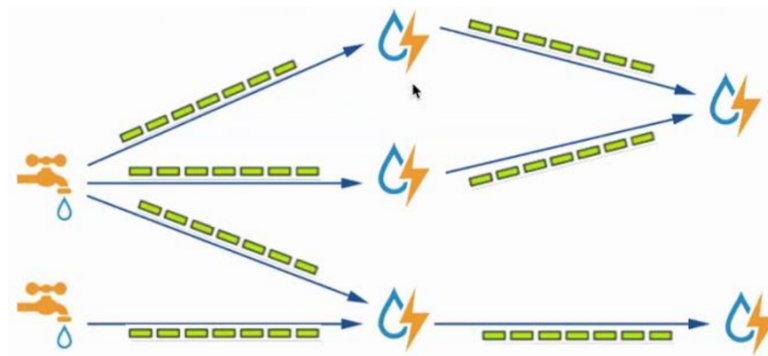


Figura 8: Esempio topologia di un'applicazione Storm

### 3.1 Inizializzazione Storm

Nel caso si utilizzasse un `TopologySubmitter` è necessario avviare esplicitamente i processi come segue.

La prima operazione effettuata è stata l'avvio onfigurazione, naming, fornitura del Server **ZooKeeper**, un servizio centralizzato per il mantenimento delle informazioni di cdi sincronizzazione distribuita e fornitura di servizi di gruppo. Sostanzialmente questo si comporta come il core principale del cluster che consente la comunicazione tra Nimbus e Supervisor.

- **Nimbus** fa le veci di un Master Node in quanto detiene informazioni sulla topologia della rete, è il responsabile della distribuzione del codice nel cluster, si occupa dell'assegnazione di task alle macchine ed effettua monitoraggio per fallimenti.
- Il **Supervisor** ascolta il lavoro assegnato alla sua macchina; avvia/arresta i processi del worker secondo le necessità
- Il **Worker** è la componente terminale dell'infrastruttura e può essere uno Spout o un Bolt.
- Infine viene avviato il processo **UI** che consente la visualizzazione di informazioni sul cluster direttamente da browser, in quanto web applicaton.

```

File Edit View Terminal Go Help
student@ubuntu-VirtualBox:~/zkServer$ ./zkServer.sh start
ZooKeeper_JMX enabled by default
Using config: /home/student/zookeeper-3.4.12/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
student@ubuntu-VirtualBox:~/zkServer$

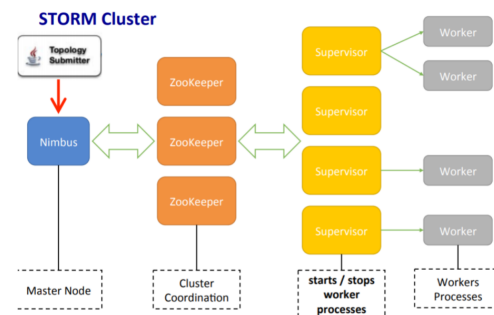
File Edit View Terminal Go Help
student@ubuntu-VirtualBox:~/zkServer$ storm nimbus
Running: /usr/lib/jvm/java-8-openjdk-1386/bin/java -server -Ddaemon.name=nimbus -Dstorm.options= -Dstorm.home=/home/student/apache-storm-1.2.3 -Dstorm.log.dir=/home/student/apache-storm-1.2.3/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file=/home/student/apache-storm-1.2.3/conf -Dlog4j.daemon= -Dlog4j.name=nimbus -Dlog4j.selector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector -Dlog4j.configurationFile=/home/student/apache-storm-1.2.3/log4j2/cluster.xml org.apache.storm.daemon.nimbus

File Edit View Terminal Go Help
student@ubuntu-VirtualBox:~/zkServer$ storm supervisor
Running: /usr/lib/jvm/java-8-openjdk-1386/bin/java -server -Ddaemon.name=supervisor -Dstorm.options= -Dstorm.home=/home/student/apache-storm-1.2.3 -Dstorm.log.dir=/home/student/apache-storm-1.2.3/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file=/home/student/apache-storm-1.2.3/conf -Dlog4j.daemon= -Dlog4j.name=supervisor -Dlog4j.selector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector -Dlog4j.configurationFile=/home/student/apache-storm-1.2.3/log4j2/cluster.xml org.apache.storm.daemon.supervisor

File Edit View Terminal Go Help
student@ubuntu-VirtualBox:~/zkServer$ storm ui
Running: /usr/lib/jvm/java-8-openjdk-1386/bin/java -server -Ddaemon.name=ui -Dstorm.options= -Dstorm.home=/home/student/apache-storm-1.2.3 -Dstorm.log.dir=/home/student/apache-storm-1.2.3/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file=/home/student/apache-storm-1.2.3/conf -Dlog4j.daemon= -Dlog4j.name=ui -Dlog4j.selector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector -Dlog4j.configurationFile=/home/student/apache-storm-1.2.3/log4j2/cluster.xml org.apache.storm.ui.core

```

((a)) Inizializzazione Storm



((b)) Architettura Storm

Figura 9: Risultato conteggio del numero minimo di visite

### 3.2 Applicazione Storm: Temperature Topology

L'applicazione Temperature Topology implementa una topologia Storm formata da uno Spout e tre Bolt, come in figura 10.

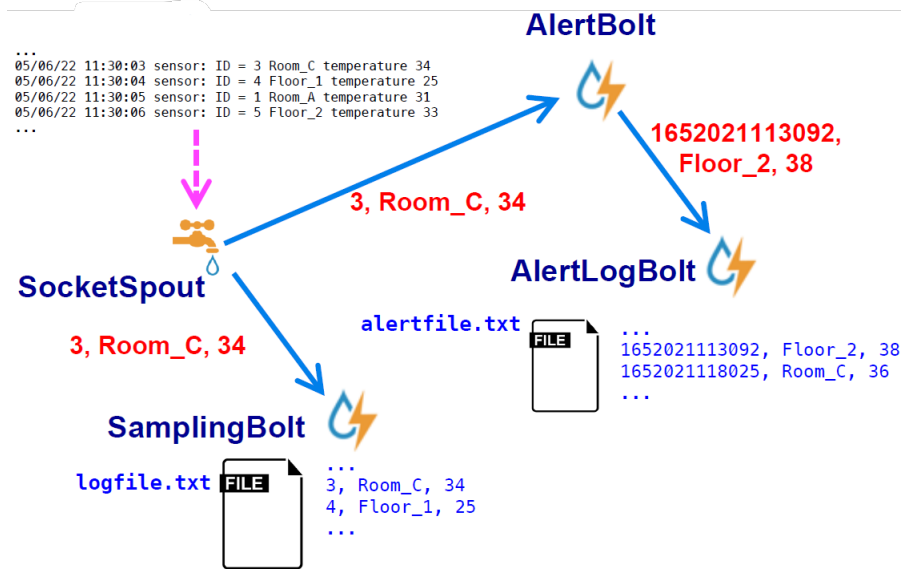


Figura 10: Topologia dell'applicazione Storm realizzata

**SocketSpout** rappresenta la sorgente dello stream di dati: legge da un server locale (temperatureStream.sh) le misure di temperatura ed emette delle tuple nel formato *ID, location, temperature* su un unico stream di uscita.

**SamplingBolt** è il bolt sottoscritto a SocketSpout che deve campionare il 20% dello stream letto. Quindi, applica una funzione hash sull'id, che può restituire un numero tra 0 e 5 (il server locale produce 5 id diversi) e campiona solo l'id la cui funzione di hash restituisce 0. I campioni vengono salvati su un file locale *logfile.txt*.

**AlertBolt** è il bolt sottoscritto a SocketSpout che per ogni tupla ricevuta deve effettuare un check sulla temperatura: se la temperatura è maggiore o uguale di 35, allora deve emettere sullo stream di output una tupla nel formato *timestamp, location, temperature*.

**AlertLogBolt** è il bolt sottoscritto a AlertBolt che memorizza le tuple ricevute su un file locale denominato *alertlog.txt*.

**TemperatureTopology** ha il compito di definire la configurazione della topologia di nostro interesse. In particolare, la classe *TopologyBuilder* permette di definire e istanziare la topologia. I metodi *setSpout()* e *setBolt()* consentono la registrazione degli Spout e dei Bolt definiti. La politica

di grouping è *shuffle grouping*, che prevede che le tuple dello stream vengano suddivise in maniera casuale tra gli executor associati ad un particolare Bolt.

La topologia è stata sottomessa utilizzando due alternative:

- *LocalCluster* che si occupa della simulazione del cluster Storm e, tramite una sua istanza è possibile sottomettere e avviare la topologia creata dal metodo `createTopology()` dell'oggetto builder;
- *TopologySubmitter* che emula un cluster reale in cui sono presenti Nimbus e Supervisor, i quali comunicano mediante il server Zookeeper.

L'implementazione dell'applicazione è consultabile nell'Appendice D.

### 3.3 Risultati delle elaborazioni con Storm

A seguito delle operazioni di dispiegamento ed esecuzione dei Jar risultanti sono stati prodotti in output due file:

- **logfile.txt** contenente i valori campionati al 20% dello stream totale basandosi sull'ID.
- **alertfile.txt** contenente tutti i valori generati dal server con valore superiore a 35.

Osservando la figura 11 è possibile notare come l'esecuzione dei due Bolt sia avvenuta in parallelo in quanto i valori riportati in entrambi i file coincidono.

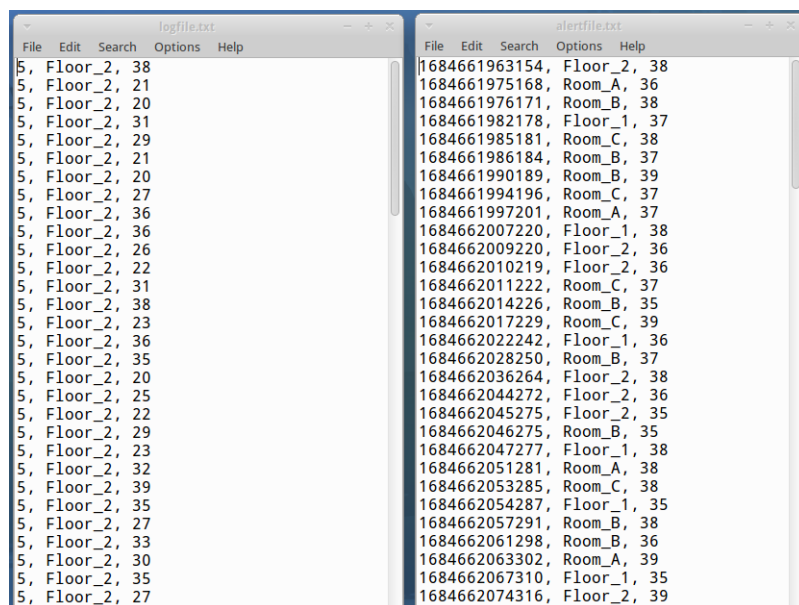


Figura 11: Risultati

Tramite la GUI, raggiungibile all'indirizzo localhost:8080, è stato possibile raccogliere informazioni sulla topologia dispiegata e accedere ai valori relativi ai singoli Bolt e Spout.

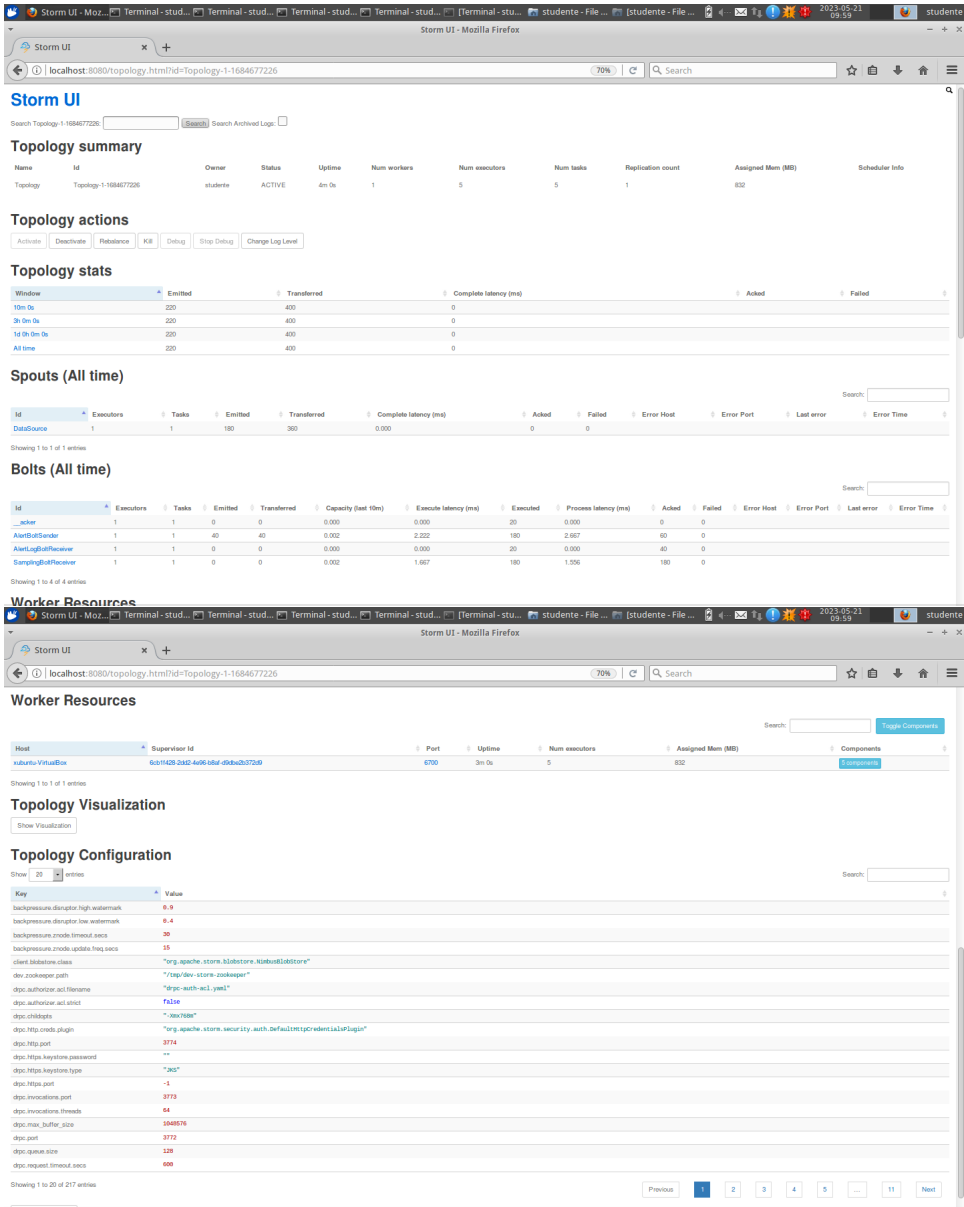


Figura 12: GUI

Nella sezione Topology Actions è possibile arrestare la topologia e rimuoverla da Storm. La topologia non viene più visualizzata e per poterla riattivare è necessario che venga nuovamente dispiegata.

Inoltre, è possibile consultare il grafo della topologia nella sezione Topology Visualization (figura 13).

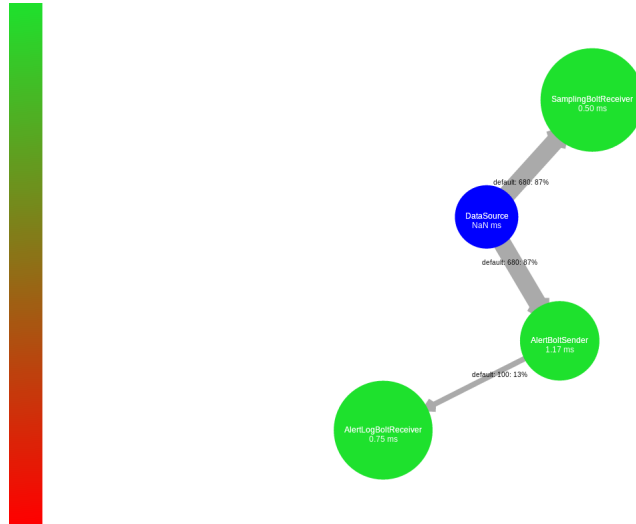


Figura 13: Grafo della topologia dispiegata

La componente blu rappresenta la prima componente della topologia. Il colore degli altri componenti della topologia indica se il componente supera la capacità del cluster: i componenti rossi rappresentano un collo di bottiglia e i componenti verdi sono componenti che operano all'interno capacità. Infine, lo spessore delle linee è un indice del flusso di dati: linee più spesse tra i componenti indicano flussi di dati più grandi.

## Appendice A Somma delle visite per mese

### A.1 Mapper

```
public class VisitsMap extends Mapper<LongWritable, Text, Text, IntWritable>{
    public void map ( LongWritable key ,Text value ,Context ctx) throws
        IOException,InterruptedException{
        String[] st = value.toString().split(" ");
        if( st != null){
            String month = new String(st[0]);
            int visits = new Integer(st[2]);
            ctx.write(new Text(month), new IntWritable(visits));
        }
    }
}
```

### A.2 Reducer

```
public class VisitsReduce extends Reducer<Text, IntWritable,Text, IntWritable>{
    public void reduce(Text month, Iterable <IntWritable> list , Context ctx) throws IOException,
        InterruptedException {
        int sum = 0;
        for(IntWritable value: list){
            sum += value.get();
        }
        ctx.write(month, new IntWritable(sum));
    }
}
```

### A.3 Driver

```
public class VisitsDriver {
    public static void main(String[] args) throws Exception{
        Job job = Job.getInstance();
        job.setJarByClass(visits.VisitsDriver.class);
        job.setJobName("SumVisits");
        job.setNumReduceTasks(4);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(visits.VisitsMap.class);
        job.setReducerClass(visits.VisitsReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        System.exit( job.waitForCompletion(true) ? 0:1);
    }
}
```



## Appendice B Conteggio del numero di siti unici

### B.1 Mapper

```
public class SitesMap extends Mapper<LongWritable, Text, Text, Text>{
    public void map ( LongWritable key ,Text value ,Context ctx) throws
        IOException,InterruptedException{
        String[] st = value.toString().split(" ");
        if( st != null){
            String sites = new String(st[1]);
            ctx.write(new Text("distsites"), new Text(sites));
        }
    }
}
```

### B.2 Reducer

```
public class SitesReduce extends Reducer<Text, Text,Text, IntWritable>{
    public HashSet<String> set = new HashSet<String>();
    public void reduce(Text site, Iterable <Text> list , Context ctx) throws IOException,
        InterruptedException {
        for(Text value: list){
            set.add(value.toString());
        }
        ctx.write(site, new IntWritable(set.size()));
    }
}
```

### B.3 Driver

```
public class SitesDriver {
    public static void main(String[] args) throws Exception{
        Job job = Job.getInstance();
        job.setJarByClass(sites.SitesDriver.class);
        job.setJobName("SumVisits");
        job.setNumReduceTasks(4);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(sites.SitesMap.class);
        job.setReducerClass(sites.SitesReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        System.exit( job.waitForCompletion(true) ? 0:1);
    }
}
```

## Appendice C Conteggio del numero minimo di visite

### C.1 Mapper

```
public class MinVisitsMap extends Mapper<LongWritable, Text, Text, IntWritable>{
    public void map ( LongWritable key ,Text value ,Context ctx) throws
        IOException,InterruptedException{
        String[] st = value.toString().split(" ");
        if( st != null){
            int visits = new Integer(st[2]);
            ctx.write(new Text("minvisits"), new IntWritable(visits));
        }
    }
}
```

### C.2 Reducer

```
public class MinVisitsReduce extends Reducer<Text, IntWritable,Text, IntWritable>{
    public List<Integer> listA = new ArrayList<Integer>();
    public void reduce(Text string, Iterable <IntWritable> list , Context ctx) throws IOException,
        InterruptedException {
        for(IntWritable value: list){
            listA.add(value.get());
        }
        int min = Collections.min(listA);
        ctx.write(string, new IntWritable(min));
    }
}
```

### C.3 Driver

```
public class MinVisitsDriver {
    public static void main(String[] args) throws Exception{
        Job job = Job.getInstance();
        job.setJarByClass(min.MinVisitsDriver.class);
        job.setJobName("SumVisits");
        job.setNumReduceTasks(4);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(min.MinVisitsMap.class);
        job.setReducerClass(min.MinVisitsReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        System.exit( job.waitForCompletion(true) ? 0:1);
    }
}
```

## Appendice D Applicazione Storm: TemperatureTopology

### D.1 SocketSpout.java

```
public class SocketSpout extends BaseRichSpout {

    private SpoutOutputCollector collector;
    private BufferedReader buf;

    @Override
    public void nextTuple() {
        String s= null;
        //impacchettiamo la stringa da inviare
        try {
            s=buf.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //Selezione dei campi dalla stringa letta: 05/06/22 11:30:04 sensor: ID = 4 Floor_1 temperature 25
        String [] st =s.split(" ");
        int id= new Integer(st[5]);
        String location= st[6];
        int temperature = new Integer(st[8]);

        //Emit della tupla
        collector.emit(new Values (id, location, temperature));
        System.out.println("spout: emesso: " + id+ ", " + location + ", " + temperature + " eseguito da "
            + Thread.currentThread().getName());
        Utils.sleep(1000);
    }

    @Override
    public void open(Map arg0, TopologyContext arg1, SpoutOutputCollector arg2) {
        collector = arg2;
        try {
            Socket s = new Socket ("127.0.0.1", 7777);
            buf= new BufferedReader(new InputStreamReader(s.getInputStream()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer arg0) {
        //definizione del formato della tupla da emettere
        arg0.declare(new Fields("ID", "location", "temperature"));
    }
}
```

## D.2 SamplingBolt.java

```
public class SamplingBolt extends BaseRichBolt {

    private OutputCollector collector;
    private PrintWriter pw;

    @Override
    public void execute(Tuple arg0) {
        // 1) lettura tupla e processing: "ID", "Location", "Temperature"
        int i = arg0.getIntegerByField("ID");
        String location = arg0.getString(1);
        int temperature = arg0.getIntegerByField("temperature");

        //stampiamo il valore della tupla
        System.out.println("    Bolt ricevuto "+ i + ", " + location + ", " + temperature + " da "
            +arg0.getSourceComponent() + " - " + arg0.getSourceStreamId());

        //Salvataggio delle tuple sul file logfile.txt
        //funzione di hash per campionare solo il 20% delle tuple ricevute
        int hash= Math.abs(i) % 5;
        if(hash == 0){
            pw.println(i +", " + location + ", " + temperature );
            pw.flush();
        }

        //ack della tupla
        collector.ack(arg0);
    }

    @Override
    public void prepare(Map arg0, TopologyContext arg1, OutputCollector arg2) {
        // TODO Auto-generated method stub
        collector=arg2;

        try {
            pw= new PrintWriter("/home/studente/logfile.txt");
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer arg0) {
    }

}
```

## D.3 AlertBolt.java

```
public class AlertBolt extends BaseRichBolt {

    private OutputCollector collector;
    private PrintWriter pw;

    @Override
    public void execute(Tuple arg0) {
        // TODO Auto-generated method stub
        // 1) lettura tupla
        int i = arg0.getIntegerByField("ID");
        String location = arg0.getString(1);
        int temperature = arg0.getIntegerByField("temperature");

        // stampiamo il valore della tupla
        System.out.println("    Bolt ricevuto "+ i + ", " + location + ", " + temperature + " da "
            +arg0.getSourceComponent() + " - " + arg0.getSourceStreamId());

        //2) emit della tupla con temperatura maggiore di 35
        if(temperature >= 35){
            long timestamp=System.currentTimeMillis();
            collector.emit(new Values (timestamp, location, temperature));
            System.out.println( "            ALERT: "+ timestamp + ", " + location + ", " + temperature );

            //ack della tupla
            collector.ack(arg0);
        }
    }

    @Override
    public void prepare(Map arg0, TopologyContext arg1, OutputCollector arg2) {
        // TODO Auto-generated method stub
        collector=arg2;
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer arg0) {
        // TODO Auto-generated method stub
        //definizione del formato della tupla emessa
        arg0.declare(new Fields("timestamp", "location", "temperature"));
    }
}
```

## D.4 AlertLogBolt.java

```
public class AlertLogBolt extends BaseRichBolt {

    private OutputCollector collector;
    private PrintWriter pw;

    @Override
    public void execute(Tuple arg0) {
        // TODO Auto-generated method stub
        // 1) lettura tupla e processing: "ID", "Location", "Temperature"
        long timestamp = arg0.getLongByField("timestamp");
        String location = arg0.getString(1);
        int temperature = arg0.getIntegerByField("temperature"); //se non ricordiamo il mapping tra la
            posizione e il count.

        //stampiamo il valore della tupla
        System.out.println("          ALERT LOG " + timestamp + ", " + location + ", " + temperature + "
            da " + arg0.getSourceComponent() + " - " + arg0.getSourceStreamId());

        //Salvataggio delle tuple sul file alertfile.txt
        pw.println(timestamp + ", " + location + ", " + temperature );
        pw.flush();

        //ack della tupla
        collector.ack(arg0);
    }

    @Override
    public void prepare(Map arg0, TopologyContext arg1, OutputCollector arg2) {
        // TODO Auto-generated method stub
        collector=arg2;

        try {
            pw= new PrintWriter("/home/studente/alertfile.txt");
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer arg0) {

    }
}
```

## D.5 TemperatureTopology

```
public class TemperatureTopology {

    public static void main(String[] args) throws InterruptedException, AlreadyAliveException,
        InvalidTopologyException, AuthorizationException {
        // TODO Auto-generated method stub
        Config cfg= new Config();
        cfg.setDebug(false);

        // Topologia DAG.
        TopologyBuilder builder =new TopologyBuilder();

        builder.setSpout("DataSource", new SocketSpout());

        builder.setBolt("SamplingBoltReceiver", new SamplingBolt()).shuffleGrouping("DataSource");
        builder.setBolt("AlertBoltSender", new AlertBolt()).shuffleGrouping("DataSource");
        builder.setBolt("AlertLogBoltReceiver", new AlertLogBolt()).shuffleGrouping("AlertBoltSender");

        //invio topologia:
        // 1) creazione local cluster

        LocalCluster cluster = new LocalCluster();
        cluster.submitTopology("TemperatureTopology",cfg, builder.createTopology());
        Thread.sleep(120000);
        cluster.shutdown();

        // 2) Storm Submitter
        //StormSubmitter.submitTopology("Topology",cfg, builder.createTopology());
    }
}
```