Analisi in tempo reale di difetti nella produzione L-PBF

Utilizzando Flink e Spark

Gentili Emanuele & Donnini Francesco

Università degli studi di Roma Tor Vergata
Ingegneria Informatica

Luglio 10, 2025

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Roadmap

Roadmap

Il contesto d'applicazione

- Lo scopo di questo progetto è rispondere ad alcune query su dati di monitoraggio del processo produttivo Laser Powder Bed Fusion L-PBF, utilizzando l'approccio di processamento orientato ai data-stream con Apache Flink.
- L-PBF é soggetta a diversi tipi di difetti, dovuti a impuritá del materiale, instabilitá termiche o errori di calibrazione.
- In particolare, si richiede di effettuare un'analisi in tempo reale delle immagini di OT, cioè di istantanee termiche del letto di polvere durante il processo di stampa, una foto per ogni strato, con il fine di stimare la probabilità che il pezzo in produzione abbia dei difetti.

Il dataset

- Ogni foto viene divisa in settori della stessa grande denominati tile
- Ogni tile è identificato da:
  - seq_id: numero di sequenza unico per l'elemento di input
  - print_id: identificatore dell'oggetto in corso di stampa
  - tile_id: identificatore del tile all'interno del layer
  - layer: coordinata z del layer corrente
  - tiff: dati binari che rappresentano il tile in formato TIFF.

Struttura della soluzione proposta

- La soluzione proposta utilizza Flink per rispondere alle tre query e Spark per rispondere alle prime due.
- In particolare sono stati utilizzati per implementare la pipeline:
  - analisi basata su soglia (Q1)
  - windowing (Q2)
  - analisi degli outliers (Q2)
  - clustering degli outliers (Q3)

Roadmap

La sorgente dei dati in Flink

- Utilizzo delle `DataSource` API
- Il componenete fondamentale è `SourceReader` che può essere eseguito in parallelo in un task manager e si occupa di consumare i dati, in questo caso dal server micro-challenger, producendo uno o più stream di tuple, in questo caso batch.
- Il cuore di questo componente è la funzione [1] questa funzione viene chiamata ripetutamente fino a quando il server non ha più dati da inviare.
- Dato che non è necessario un componente per la lettura dei dati l'architettura risultante è quella in figura [1]

La funzione del source reader

```java
@Override
public InputStatus pollNext(ReaderOutput<Batch> readerOutput) throws Exception {
    var o = client.nextBatch(benchmark);
    if (o.isPresent()) {
        var entryTime = Instant.now();
        var batch = deserializer.deserialize(o.get());
        batch.setEntryTime(entryTime);
        readerOutput.collect(batch);
        return InputStatus.MORE_AVAILABLE;
    }
    return InputStatus.END_OF_INPUT;
}
```

Listing 1: Codice per la lettura dei dati dal challenger a Flink

Architettura per il processamento con Flink

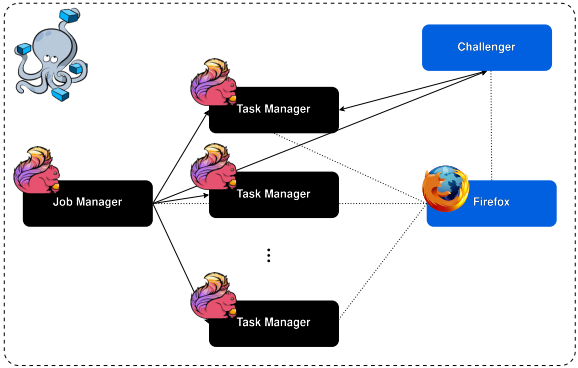

Figure 1: Ogni componente nella figura è un container Docker, il sistema viene eseguito tramite Docker Compose.

La presenza di Firefox

- Il container che esegue Firefox è stato utilizzato per accedere alle UI senza dover esporre le porte verso l'host.
- Accesso alla UI del challenger per vedere le performance (presenti nelle tabelle nella sezione esperimenti)
- Accesso alla UI di Flink per vedere il grafo dell'applicazione [2]

Il grafo dell'applicazione Flink

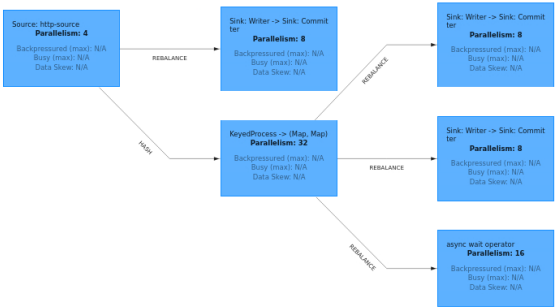

Figure 2: DAG dell'applicazione Flink che riporta il grado di parallelismo scelto per ogni operatore

La lettura dei dati in Spark

- In Spark la lettura dei dati avviene mediante Kafka
- Un componente (producer) interagisce con il challenger per recuperare i dati e pubblicarli su un topic
- Spark sarà il consumer che legge i dati connettendosi al broker di Kafka
- L'architettura risultante sarà quella in figura [3]

Architettura per il processamento con Spark

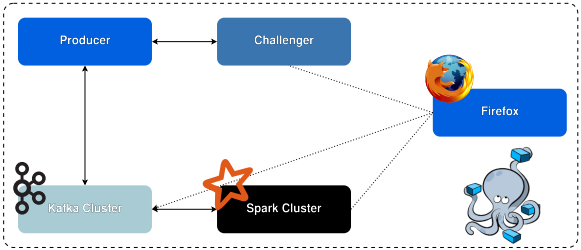

Figure 3: Il sistema viene eseguito tramite Docker Compose.

Roadmap

Analisi basata su soglia

La prima query

- Per ogni tile, identificare i punti dove la temperatura è sotto la soglia di 5000 (aree di vuoto) oppure sopra la soglia di 65000 (punti saturati).
- Durante la fase di deserializzazione è necessario effettuare il parsing dell'immagine dal formato tiff a una matrice di interi
- I punti saturati vengono quindi contati durante la fase di deserializzazione.

Windowing in Flink

- La query Q2 si concentra sugli stadi (2) e (3) della pipeline. Per ogni tile, mantenere una finestra scorrevole degli ultimi 3 layer. Per ogni tensore è necessario calcolarne gli outliers ossia i pixel la cui deviazione supera una certa soglia.
- Due soluzioni:
  - Una finestra scorrevole basata su event time: come timestamp viene utilizzato il layer dell'immagine convertito in secondi
  - Una finestra implementata ad-hoc come operatore stateful che utilizza un meccanismo di doppio buffering per gestire l'arrivo di tuple fuori ordine

- Per indicare che il layer deve essere usato come misura di tempo dalla finestra scorrevole è necessario definire una strategia di watermarking [2]
- L'ordinamento delle tuple nella finestra non è garantito quindi è necessario effettuarne l'ordinamento prima di passare il tensore all'algoritmo che ne calcola gli outlier.

```
WatermarkStrategy
  .<Batch>forMonotonousTimestamps()
  .withTimestampAssigner((TimestampAssignerSupplier<Batch>) context ->
    (TimestampAssigner<Batch>) (element, recordTimestamp) ->
      element.getLayer() * 1000L);
```

Listing 2: Definizione del campo layer del batch come event time.

```
public class KeyedBatchSlidingWindow implements Serializable {
    ...
    private final int size;
    private final int slide;
    private int startingOffset = 0;
    private final Deque<Batch> window = new ArrayDeque<>();
    private final Queue<Batch> buffer = new PriorityQueue<>(new BatchComparator());
    ...
}
```

Listing 3: Procedura di aggiunta di un nuovo batch alla finestra.

La finestra ad-hoc

- Ogni volta che un nuovo batch arriva all'operatore, viene aggiunto alla finestra se **in ordine**, nell'altro buffer altrimenti.
- Aggiungendo un nuovo elemento alla finestra, elementi precedentemente arrivati fuori ordine potrebbero, ora, entrare a farne parte. La procedura `fillWindow()` si occupa di effettuare questa operazione.

Procedura di aggiunta di un nuovo batch alla finestra

```
public boolean update(Batch b) {
  if (inorder(b)) {
    window.add(b);
    fillWindow();
    return true;
  }
  buffer.add(b);
  return false;
}
```

Listing 4: Procedura di aggiunta di un nuovo batch alla finestra.

Analisi e clustering degli outliers

- Per ogni punto p nel layer più recente di una finestra di tile viene calcolata la deviazione di temperatura locale
- Un punto viene classificato come outlier se la sua deviazione di temperatura locale supera la soglia di 6000.
- Una volta calcolati gli outliers, il flusso viene diramato a due operatori: uno è responsabile del ranking dei primi cinque outliers ordinati rispetto alla deviazione, mentre l'altro calcola i centri del cluster e quanti punti fanno parte di esso utilizzando l'algoritmo DBScan fornito dalla libreria `commons-math3` di Apache.
- Il ranking dei punti e i centri del cluster vengono salvati su file, inoltre quest'ultimi vengono inviati al challenger come richiesto dalla specifica.

La pipeline in Spark

Le differenze rispetto al flusso in Flink sono

- Acquisizione dei dati tramite Kafka
- Windowing gestito attraverso l'operatore con stato `FlatMapGroupsWithStateFunction<>`
- I risultati non vengono inivati al challenger perché non è stata implementata la terza query.

Roadmap

Le configurazioni provate in Flink

- Su Flink sono state provate due configurazioni:
  - La versione che utilizza la finestra da noi proposta.
  - La versione che utilizza la finestra basata su event time.
- L'applicazione è stata eseguita su un cluster Flink costituito da un job manager e un numero di task manager pari a $n = 1, 2, 5$.
- È stata utilizzata la libreria NETEM per introdurre ritardi a livello di rete, tra i nodi del cluster, che seguono una distribuzione normale con parametri $\mu = 10ms, \sigma = 2ms$
- Inoltre il client applica un ritardo fisso di $100ms$ durante il trasferimento dei risultati al challenger.

Le metriche utilizzate

- tempo di risposta medio R: è stato raccolto il tempo d'ingresso di ogni batch nel sistema e il relativo tempo d'uscita
- throughput per operatore: ha richiesto la raccolta dei tempi d'ingresso di ogni batch nell'operatore. Viene calcolato come:

$$X = \frac{N}{t_{\text{out},M} - t_{\text{in},m}}$$

dove N è il numero di batch, $t_{\text{out},M}$ è il tempo d'uscita dell'ultimo batch e $t_{\text{in},m}$ è il tempo d'ingresso del primo batch nell'operatore.

Performance con 1 task manager

	R	σ	p_{99}	R_M	X
Q1	0.056	0.03	0.111	0.127	188.69
Q2	0.117	0.032	0.188	0.216	189.21
Q3	0.12	0.032	0.191	0.261	189.13

Table 1: Finestra ad-hoc con 1 task manager

	R	σ	p_{99}	R_M	X
Q1	0.062	0.03	0.117	0.204	150
Q2	0.32	0.079	0.516	0.583	152.484
Q3	0.325	0.083	0.522	0.582	152.464

Table 2: Finestra basata su event time con 1 task manager.

Performance con 2 task manager

	R	σ	p_{99}	R_M	X
Q1	0.058	0.029	0.112	0.183	182.77
Q2	0.127	0.032	0.188	0.286	184.44
Q3	0.128	0.032	0.19	0.292	184.59

Table 3: Finestra ad-hoc con 2 task manager

	R	σ	p_{99}	R_M	X
Q1	0.06	0.03	0.114	0.156	158.75
Q2	0.311	0.079	0.507	0.593	159.96
Q3	0.315	0.08	0.523	0.605	159.91

Table 4: Finestra basata su event time con 2 task manager.

Performance con 5 task manager

	R	σ	p_{99}	R_M	X
Q1	0.057	0.029	0.113	0.125	190.17
Q2	0.12	0.0346	0.192	0.349	191.28
Q3	0.124	0.0356	0.201	0.356	191.19

Table 5: Finestra ad-hoc con 5 task manager

	R	σ	p_{99}	R_M	X
Q1	0.061	0.03	0.115	0.168	158.16
Q2	0.302	0.082	0.503	0.572	159.43
Q3	0.307	0.083	0.509	0.6	159.43

Table 6: Finestra basata su event time con 5 task manager.

Performance raccolte dal micro-challenger

n	X	R	p_{99}	R_M
1	138.46	3s230ms172μs	8s744ms988μs	9s343ms1μs
2	136.57	2s965ms37μs	8s331ms604μs	8s820ms281μs
5	137.15	3s329ms929μs	9s43ms943μs	9s764ms892μs

Table 7: Finestra ad-hoc, metriche raccolte dal micro-challenger.

n	X	R	p_{99}	R_M
1	133.29	1s340ms864μs	3s693ms998μs	4s142ms919μs
2	135.73	1s650ms993μs	4s677ms523μs	5s234ms734μs
5	136.11	1s614ms705μs	4s692ms951μs	5s423ms926μs

Table 8: Finestra basata su event time, metriche raccolte dal micro-challenger.

Le performance di Spark

	R	σ	p_{99}	R_M	X
Q1	0.006	0.0027	0.014	0.105	9.62
Q2	1.525	0.445	3.019	4.052	9.65

Table 9: 1 Worker.

	R	σ	p_{99}	R_M	X
Q1	0.011	0.023	0.066	0.551	9.15
Q2	0.976	0.979	2.945	22.741	22.22

Table 10: 3 Worker.