OPERATION BIG DATA

-Relazione secondo progetto Big Data-

Davide Cassetta

Matricola: 472436

Sommario

1 – INTRODUZIONE

L'analisi di stream viene sempre più utilizzata per eseguire azioni in base ai dati raccolti real time, infatti la conoscenza in tempo reale di cosa sta accadendo permette di reagire in tempi brevi nel correggere problemi o nel cogliere opportunità e fare prevenzione. Questo tipo di analisi viene utilizzata per diversi scopi come monitoraggio e alerting in campo industriale, prevenendo anomalie e problemi, o in quello bancario per l'individuazione di frodi, per decisioni di mercato automatizzate, nei recommender system, per consigliare prodotti in tempo reale e ancora analisi di dati proveniente da ogni tipo di sensori.

Questo secondo progetto di Big Data consiste nella realizzazione di un'architettura lambda, in grado di eseguire sia analisi streaming su dati in arrivo real time, che analisi batch su dati memorizzati in un database NoSql, in certi intervalli di tempo. Un'architettura lambda in generale permette di avere una visione d'insieme, grazie alle analisi batch, e una visione in tempo reale grazie all'analisi streaming ed eventualmente fondere i due tipi di analisi per ricavarne informazioni combinate.

I dati sono stati presi dal sito di Kaggle e sono contenuti in 22 file csv: l'insieme dei file contiene i risultati delle partite classificate della stagione 5, chiamata Operation Velvet Shell (Febbraio 2017 – Aprile 2017), di Raimbow Six Siege, un videogioco fps (First-Person Shooter) multipiattaforma, multigiocatore. Lo scopo dell'analisi è quella di fornire statistiche sia ai giocatori, che possono fare certe scelte guidate da quest'ultime, sia agli sviluppatori, che possono regolare i parametri del gioco per renderlo più equilibrato possibile, evitando situazioni frustranti e poco divertenti per chi usufruisce del gioco. Il flusso streaming viene creato ad hoc da un sottoinsieme dei file tramite Kafka, simulando l'arrivo di dati real time delle partite. Si è scelto di non utilizzare l'insieme si dati completo perché troppo grande (20 GB), cosa che avrebbe richiesto tempi troppo lunghi per un'analisi streaming dell'intero file.

Figura 1 - Raimbow Six Siege - Operation Velvet Shell

2 - CASO DI STUDIO E OBIETTIVI

2.1 – Descrizione dominio

Il caso di studio analizzato nel progetto ha come argomento Raimbow Six Siege, un videogioco sparatutto multipiattaforma (pc, ps4 e xbox), multigiocatore che ha avuto grande successo dalla sua uscita arrivando a contare circa 100 milioni di giocatori e 1.12 miliardi di incassi.

Ogni partita è divisa in più round, la prima squadra a vincerne 4 vince la partita. I giocatori vengono divisi in 2 team da 5 giocatori l'uno e ogni team, in ogni round, ricopre il ruolo di attaccante o difensore, alternando tra i 2 ruoli tra i vari round; un round viene vinto quando vengono uccisi tutti i membri della squadra avversaria o quando viene portato a termine un obiettivo che dipende dalla modalità di gioco (Presidio, Ostaggio o Bomba); se gli attaccanti non riescono a vincere entro il tempo limite la vittoria va ai difensori. Ogni giocatore all'inizio del round sceglie un operatore con cui giocare: un operatore è un personaggio che dispone di certe armi e attrezzatura unica che può aiutare molto nello svolgersi del round e gli operatori sono diversi per attacco e difesa; i difensori ad esempio, dispongono di attrezzatura per rallentare gli attaccanti e impedirgli di accedere all'obiettivo, mentre gli attaccanti possono neutralizzare le difese create dall'altro ruolo e dispongono di armi più letali.

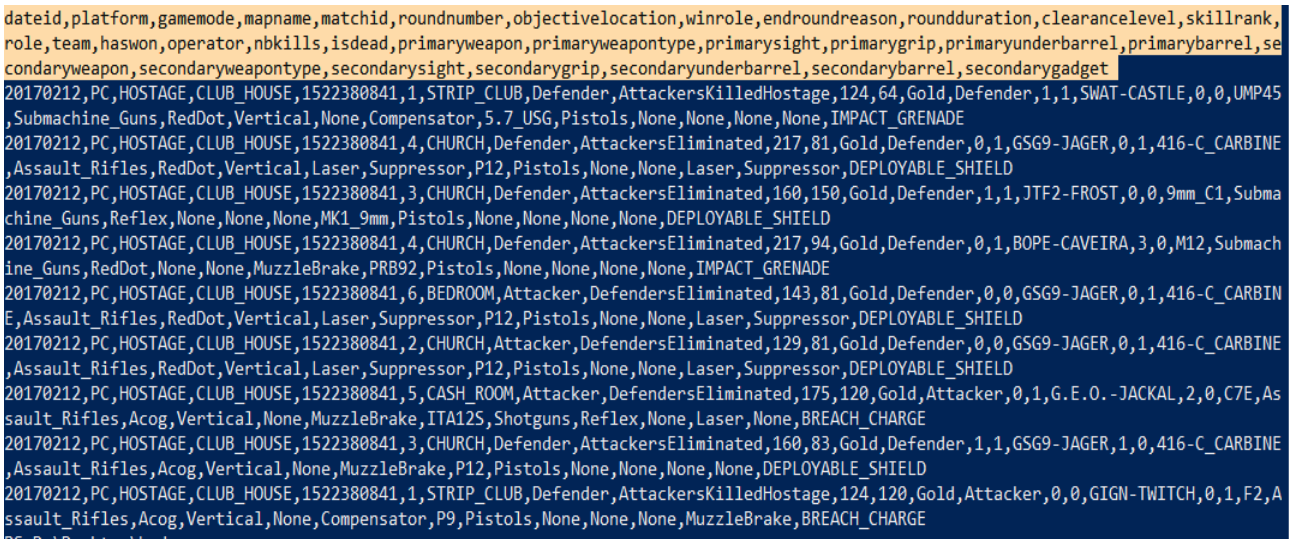
2.2 – Obiettivi

L'obiettivo delle analisi è fornire informazioni utili sia agli sviluppatori, per rendere più stabile ed equilibrato il gioco, modificando i parametri delle mappe di gioco o degli operatori, che ai giocatori che vogliano visualizzare classifiche e statistiche sul loro profilo o sul mondo di gioco in generale. Ogni 3-4 mesi infatti inizia una nuova stagione e vengono rilasciati nuovi contenuti che possono portare a sbilanciamenti nel gioco; analisi real time e batch possono fornire informazioni che evidenzino questi problemi e si possono programmare aggiornamenti appositi per correggerli o prevenirli.

2.3 – Dataset

Il dataset contiene i risultati di tutte le partite per le piattaforme pc, ps4 e xbox della stagione 5 di Raimbow Six Siege che va da Febbraio 2017 ad Aprile 2017 ed è stato preso dal sito di Kaggle. L'intero dataset è grande 20 GB, troppo per un'analisi streaming e quindi sono state estratte un numero limitato di righe relative ai giorni che vanno dal 10 al 16 del mese di febbraio e tutte le analisi sono state effettuate su questo sottoinsieme. La granularità è molto fine, ogni riga contiene il risultato di un

giocatore, in un certo round di una certa partita, con informazioni molto dettagliate come numero uccisioni, operatore, armi, se è morto, ruolo, livello e così via.



```
dateid,platform,gamemode,mapname,matchid,roundnumber,objectivelocation,winrole,endroundreason,roundduration,clearancelevel,skillrank,
role,team,haswon,operator,nbkills,isdead,primaryweapon,primaryweapontype,primarysight,primarygrip,primaryunderbarrel,primarybarrel,se
condaryweapon,secondaryweapontype,secondarysight,secondarygrip,secondaryunderbarrel,secondarybarrel,secondarygadget
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,1,STRIP_CLUB,Defender,AttackersKilledHostage,124,64,Gold,Defender,1,1,SWAT-CASTLE,0,0,UMP45
,Submachine_Guns,RedDot,Vertical,None,Compensator,5.7_USG,Pistols,None,None,None,None,IMPACT_GRENADE
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,4,CHURCH,Defender,AttackersEliminated,217,81,Gold,Defender,0,1,GSG9-JAGER,0,1,416-C_CARBINE
,Assault_Rifles,RedDot,Vertical,Laser,Suppressor,P12,Pistols,None,None,Laser,Suppressor,DEPLOYABLE_SHIELD
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,3,CHURCH,Defender,AttackersEliminated,160,150,Gold,Defender,1,1,JTF2-FROST,0,0,9mm_C1,Subma
chine_Guns,Reflex,None,None,None,MK1_9mm,Pistols,None,None,None,None,DEPLOYABLE_SHIELD
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,4,CHURCH,Defender,AttackersEliminated,217,94,Gold,Defender,0,1,BOPE-CAVEIRA,3,0,M12,Submach
ine_Guns,RedDot,None,None,MuzzleBrake,PRB92,Pistols,None,None,None,None,IMPACT_GRENADE
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,6,BEDROOM,Attacker,DefendersEliminated,143,81,Gold,Defender,0,0,GSG9-JAGER,0,1,416-C_CARBIN
E,Assault_Rifles,RedDot,Vertical,Laser,Suppressor,P12,Pistols,None,None,Laser,Suppressor,DEPLOYABLE_SHIELD
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,2,CHURCH,Attacker,DefendersEliminated,129,81,Gold,Defender,0,0,GSG9-JAGER,0,1,416-C_CARBINE
,Assault_Rifles,RedDot,Vertical,Laser,Suppressor,P12,Pistols,None,None,Laser,Suppressor,DEPLOYABLE_SHIELD
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,5,CASH_ROOM,Attacker,DefendersEliminated,175,120,Gold,Attacker,0,1,G.E.O.-JACKAL,2,0,C7E,As
sault_Rifles,Acog,Vertical,None,MuzzleBrake,ITA12S,Shotguns,Reflex,None,Laser,None,BREACH_CHARGE
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,3,CHURCH,Defender,AttackersEliminated,160,83,Gold,Defender,1,1,GSG9-JAGER,1,0,416-C_CARBINE
,Assault_Rifles,Acog,Vertical,None,MuzzleBrake,P12,Pistols,None,None,None,None,DEPLOYABLE_SHIELD
20170212,PC,HOSTAGE,CLUB_HOUSE,1522380841,1,STRIP_CLUB,Defender,AttackersKilledHostage,124,120,Gold,Attacker,0,0,GIGN-TWITCH,0,1,F2,A
ssault_Rifles,Acog,Vertical,None,Compensator,P9,Pistols,None,None,None,MuzzleBrake,BREACH_CHARGE
```

Figura 2 - Prime righe del dataset

Ogni riga contiene 31 campi:

- dateId
- platform
- gameMode
- mapName
- matchId
- roundNumber
- objectiveLocation
- winRole
- endRoundReason
- roundDuration (durata del round in secondi)
- clearanceLevel (livello giocatore)
- skillRank (classificazione grado giocatore)
- role
- team
- hasWon(se ha vinto il round)
- operator
- nbKills (numero uccisioni)
- isDead (se è morto durante il round)
- primaryWeapon
- primaryWeaponType
- primarySight
- primaryGrip
- primaryUnderBarrel
- primaryBarrel
- secondaryWeapon
- secondaryWeaponType
- secondarySight
- secondaryGrip
- secondaryUnderBarrel
- secondaryBarrel
- secondaryGadget

3 – DESCRIZIONE LAVORO E RISULTATI

3.1 – Descrizione architettura

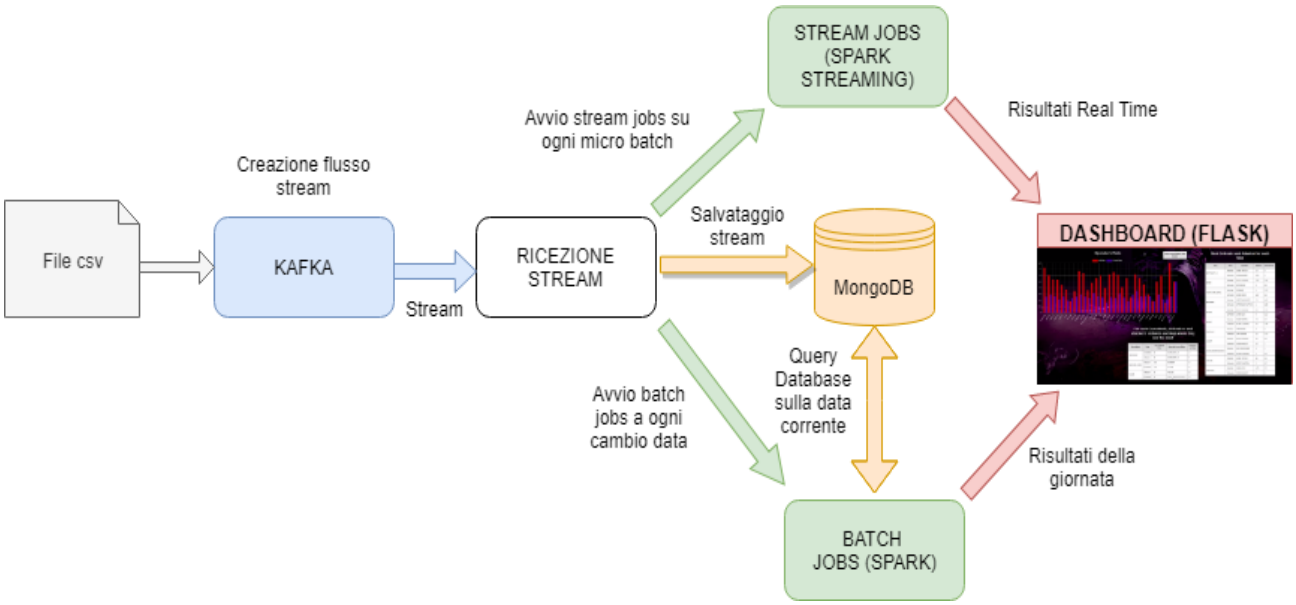


Figura 3 – Architettura Lambda

Il lavoro svolto ha portato alla realizzazione di un'architettura lambda in cui abbiamo l'esecuzione di streaming jobs e di batch jobs. I moduli sono stati realizzati in linguaggio Python e per la parte web della dahsboard sono stati utilizzati HTML, CSS e Javasctipt. Lo stream viene realizzato tramite Kafka, una tecnologia di messaggi distribuiti di tipo publish-subscribe, utilizzata per lo stream processing. I dati vengono presi dal file csv e inviati sul topic creato in precedenza (un topic è una categoria di messaggi alla quale il consumer può registrarsi per ricevere lo stream); il Producer, creato nel modulo KafkaProducer.py, è responsabile dell'invio delle righe del file sul canale di comunicazione. Le righe vengono preelaborate, scartando i campi non di interesse, e raggruppate in base alla data e al matchId, simulando l'invio di dati real time delle partite dei giocatori appena concluse, nei vari giorni del mese; il raggruppamento per data permette di simulare un'architettura che lavora a lungo termine su più giorni, e di eseguire i batch jobs su ogni giornata.

Il flusso viene ricevuto dal modulo SparkStreaming.py in cui risiede il Consumer, che viene fornito dalla libreria pyspark.streaming.kafka tramite la funzione createDirectStream(), e viene elaborato da SparkStreaming, una tecnologia molto simile a Spark, che si concentra sull'elaborazione di flussi streaming: questo in realtà

è un sistema near real time in quanto non processa le singole righe streaming in arrivo (come fa invece Storm) ma le divide in micro-batch ed esegue i jobs su un singolo micro batch alla volta. Il modulo si occupa inoltre di salvare le righe ricevute in un database NoSql (MongoDB), tramite una struttura dati di tipo dict contenente i campi delle righe, in modo da poter essere recuperate per i batch jobs. Una volta elaborati i dati con gli streaming jobs, i risultati sono inviati alla dashboard tramite un sistema di push request che fa uso della teconologia Flask che verrà illustrata successivamente.

Per quanto riguarda i batch jobs vengono avviati come sottoprocesso nel modulo batchJobs.py ogni volta che avviene un cambiamento nella data: infatti avendo raggruppato le righe per data, questo ci permette di simulare l'avvio di un batch jobs ogni 24 ore (anche se ovviamente ogni elaborazione batch viene avviata al cambio data e quindi in un tempo molto inferiore). I dati vengono recuperati da MongoDB, tramite una query basata sul campo data; i batch jobs vengono quindi eseguiti su tutte le righe relative ad una certa giornata e i risultati vengono inviati alla dashboard e visualizzati.

La dashboard è stata realizzata facendo uso soprattutto di Javascript e della sua libreria per la creazione di grafici, Chart.js. Per inviare i dati in tempo reale alla dashboard è stato necessario l'utilizzo di Flask, un web application framework scritto in Python: questo permette il push del contesto grazie a una request di tipo POST, in modo da aggiornare la pagina web in base ai dati ricevuti. Le request vengono inviate alla pagina web dagli stream jobs e dai batch jobs quando finiscono l'elaborazione, permettendo così l'aggiornamento dei grafici visualizzati con gli ultimi dati elaborati. L'aggiornamento della pagina web è implementato nel modulo flask.py: in particolare la pagina iniziale '/ ' prende come valori iniziali da mostrare degli array vuoti; quando i moduli dei vari job inviano dati, li inviano alla pagina '/updateData' che prende i dati nella request e aggiorna i valori precedenti, rimpiazzando gli array vuoti con i dati ricevuti; ogni 2 secondi viene quindi chiamata '/refreshData' che aggiorna i dati visualizzati tramite Javascript senza dover ricaricare la pagina. È possibile alternare la visualizzazione degli streaming jobs e batch jobs tramite un button posto in alto centralmente. La pagina web è implementata nel file index.html.

3.2 – Streaming Jobs e Risultati

I micro-batch sui quali vengono eseguiti i job hanno una lunghezza di 1 secondo e permettono così di elaborare contemporaneamente un numero di righe cha varia tra le 4000 e le 6000.

Figura 4 - Dashboard streaming jobs

3.2.1 – Stream job 1

Descrizione: per ogni operatore calcolare il kill ratio (rapporto tra le uccisioni e i morti) e round ratio (rapporto tra round vinti e numero round giocati) aggiornandoli in tempo reale.

Per memorizzare i dati relativi ad ogni operatore è stata utilizzata la funzione di checkpoint SparkStreamingContext.checkpoint(): questa permette di salvare informazioni sugli RDD generati in modo da poter combinare i dati tra batch diversi utilizzando updateStateByKey(updateFunction), un'altra funzione che permette di combinare i dati del batch corrente con i dati salvati dei batch precedenti. In questo modo è possibile aggiornare in tempo reale uccisioni e morti degli operatori e calcolare il nuovo ratio da inviare alla dashboard. Per questo job vengono utilizzati un mapper ed un reducer:

- mapStreamOperators: viene splittata la riga e presi i campi operator, nbKills, isDead, haswon e vengono inseriti killRatio (inizialmente uguale al numero delle uccisioni) e roundRatio (uguale a 1, in quanto ogni riga è relativa ad 1 round); l'operatore viene utilizzato come chiave e il resto come value.
- reduceStreamOperators: vengono sommate le uccisioni, le morti, il numero di round vinti e i round giocati e vengono calcolati il kill ratio (uccisioni / morti) e il round ratio (round vinti / round giocati).

Output: (operator, (sumKills, sumDead, killRatio, nbRoundWon, totRoundsPlayed, roundRatio))

Dopo aver applicato il mapper e il reducer viene creato un checkpoint e viene chiamata updateStateByKey() che ha come argomento la fuzione che salva e aggiorna i valori di uccisioni, morti, round vinti e round giocati in modo da poter calcolare il ratio aggiornato in tempo reale.
Viene poi chiamata la funzione sendToDashboardOperators() che invia una request di tipo POST alla dashboard contenente i valori aggiornati, che vengono visualizzati in un bar chart.
Tutto ciò viene ripetuto su ogni micro-batch.

Risultati real time

Essendo job real time, che cambia continuamente nel tempo, verranno mostrati i risultati della dashboard ad un certo istante di tempo. Dai risultati si può dedurre che certi operatori vengono presi dai giocatori che preferiscono concentrarsi sulle uccisioni, cercando di eliminare la squadra avversaria, piuttosto che sul portare a termine l'obiettivo del round; in particolare l'operatore Glaz, ruolo attaccante, è dotato di un fucile di precisione e la sua specialità è proprio quella di stanare e occuparsi dei difensori, come si evince dal grafico, visto che ha un rapporto uccisioni morti più alto degli altri operatori.

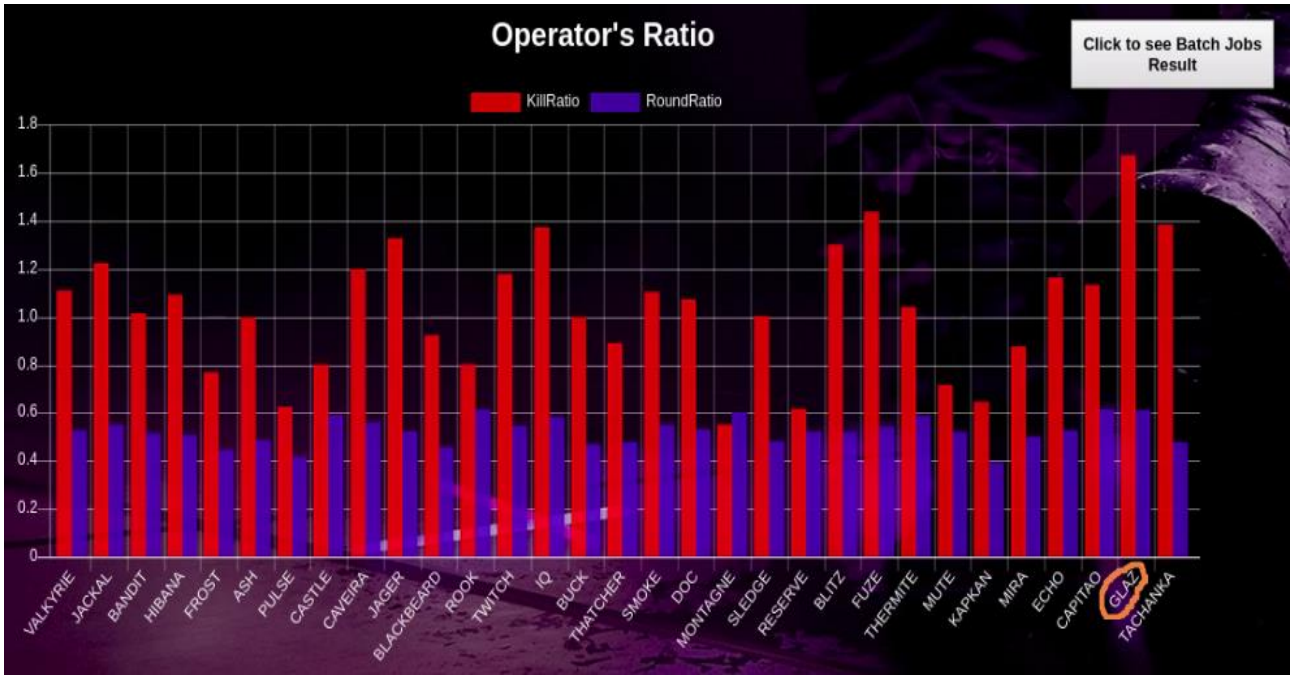


Figura 5 - Screen Job 1 Streaming

3.2.2 – Stream job 2

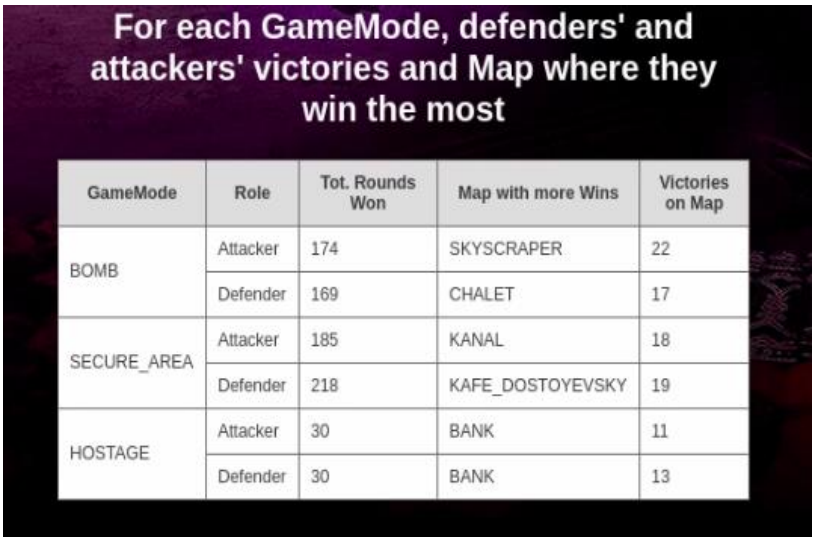
Descrizione: per ogni modalità di gioco calcolare vittorie di attaccanti e difensori, la mappa in cui vincono di più e il numero di vittorie su quella mappa.

In questo job vengono utilizzati 2 mapper e 2 reducer:

- map1StreamGameMode: la riga viene splittata e vengono presi i valori relativi a gameMode, role, map e haswon e vengono usati i primi 3 come chiave e haswon come valore.
- reduce1StreamGameMode: vengono sommate le vittorie (haswon) di ciascun ruolo in ciascuna modalità di gioco, su ciascuna mappa.
- map2StreamGameMode: in questo secondo mapper vengono usati come chiave gameMode e role e come value map, round vinti su quella mappa e round vinti in totale che verranno calcolati in fase di reduce.
- Reduce2StreamGameMode: tra due righe con la stessa chiave viene presa la mappa con un numero di round vinti maggiore e vengono sommati i round totali vinti da quel ruolo in quella modalità di gioco.

Output: ((gameMode, role), (map, totRoundWon, roundWonOnMap)).
Successivamente l'output viene inviato alla dashboard con la funzione sendToDashboardGameMode(); in questo caso non volendo dare troppe responsabilità al modulo SparkStreaming.py l'aggiornamento dei dati viene fatto nel modulo dedicato a Flask, nel quale viene mantenuto il risultato dei job sui dati ricevuti in precedenza e vengono aggiornati i valori con quelli appena ricevuti tramite una request POST. I risultati vengono quindi visualizzati in una table.

Risultati real time



GameMode	Role	Tot. Rounds Won	Map with more Wins	Victories on Map
BOMB	Attacker	174	SKYSCRAPER	22
	Defender	169	CHALET	17
SECURE_AREA	Attacker	185	KANAL	18
	Defender	218	KAFE_DOSTOYEVSKY	19
HOSTAGE	Attacker	30	BANK	11
	Defender	30	BANK	13

In questo job si nota come le modalità di gioco siano piuttosto equilibrate tra attaccanti e difensori con una leggera prevalenza dei difensori per la modalità Secure_Area. Tuttavia, con l'aumentare dei dati stream in arrivo, i valori cambiano significativamente, rendendo difficile trovare dei pattern

Figura 6 – Screen job 2 Streaming

comuni, segno che gli sviluppatori sono riusciti a rendere equilibrate le 3 modalità di gioco.

3.2.3 – Stream job 3

Descizione: per ogni mappa trovare l'operatore attaccante e difensore migliore (ratio uccisioni e ratio round più alti).

In questo job abbiamo 1 mapper e 1 reducer:

- mapper1Map: la riga viene splittata e vengono presi map, operator, role, nbKills, isDead, haswon; vengono utilizzati map e operator come chiave e il resto come value con l'aggiunta di un contatore per calcolare il numero dei round.
- reducer1Map: vengono sommate uccisioni, morti, round vinti e numero di round totali e viene aggiunto il ruolo di una delle righe, visto che ogni operatore ha un ruolo fisso.

Output: ((map, operator), (role, sumKills, sumDead, nbRoundWon, totRoundsPlayed))

Essendo più complicato il job in questo caso, visto che per ogni mappa dobbiamo tener conto delle statistiche di tutti gli operatori, in modo da aggiornarle e fornire il miglior operatore attaccante e difensore in tempo reale, i dati vengono memorizzati in una mappa che viene aggiornata nella funzione sendToDashboardMaps(): in particolare la mappa aggiornata viene riversata in una lista temporanea, quest'ultima ordinata sulla base del killRatio / 2 + roundRatio (si è scelto di dare maggiore importanza alle vittorie dei round piuttosto che alle uccisioni, perché più determinanti per i fini del gioco), e viene preso il primo attaccante e il primo difensore della lista ordinata, inviati alla dashboard e visualizzati in una table.

Risultati real time

Best Defender and Attacker for each Map

Map	Role	Operator	killRatio	RoundRatio
BARTLETT_U.	Attacker	GIGN-TWITCH	1.4	0
	Defender	GSG9-JAGER	2.75	0
BANK	Attacker	G.E.O.-JACKAL	2.33	0.14
	Defender	SAT-ECHO	2	0.5
HEREFORD_BASE	Attacker	BOPE-CAPITAO	1	0.4
	Defender	SWAT-CASTLE	2	0.17
BORDER	Attacker	SAT-HIBANA	3	0.12
	Defender	BOPE-CAVEIRA	2	1
PLANE	Attacker	SPETSNAZ-GLAZ	2.12	0
	Defender	GIGN-ROOK	2.5	0.17
KANAL	Attacker	GIGN-TWITCH	1.5	0.2
	Defender	BOPE-CAVEIRA	2.5	0.25
FAVELAS	Attacker	JTF2-BUCK	2	0
	Defender	SAS-SMOKE	2.5	0.17
YACHT	Attacker	SPETSNAZ-FUZE	1	0.11
	Defender	SAS-SMOKE	1.33	0.2
KAFE_DOSTOYEVSKY	Attacker	GSG9-IQ	2.5	0.25
	Defender	NAVYSEAL-VALKYRIE	1.64	0.11
HOUSE	Attacker	GIGN-TWITCH	2.5	0.8
	Defender	BOPE-CAVEIRA	2	0.5
OREGON	Attacker	SWAT-ASH	1.25	0.5
	Defender	GSG9-BANDIT	7	0
CLUB_HOUSE	Attacker	SPETSNAZ-FUZE	2.67	0
	Defender	GSG9-JAGER	2.17	0.29

Figura 7 – Screen job 3 Streaming

Da questi risultati si evidenzia il fatto che l'operatore Caveira, ricorre come miglior difensore in diverse mappe; infatti Caveira è un operatore che non si concentra sul piazzare trappole o difendere l'obiettivo, ma punta a indebolire e cogliere di sorpresa gli attaccanti per ucciderne il più possibile e quindi risulta spesso uno dei difensori più utilizzati e più utile su molte mappe.

3.3 – Batch Jobs e Risultati

I batch jobs vengono eseguiti ogni volta che c'è un cambiamento di data: infatti i dati arrivano dallo stream ordinati per data e quindi, quando questa cambia, si possono fare operazioni su tutte le righe della data precedente, salvate in MongoDB. Il modulo batchJobs.py recupera dal database tutte le righe relative alla data passata ed esegue i jobs su quest'ultime, inviando i risultati alla dashboard.

Figura 8 - Dashboard batch jobs

3.3.1 – Batch job 1

Descrizione: le prime 5 divisioni della giornata con il punteggio più alto (assegnando 5 punti per ogni uccisione e 15 per ogni round vinto), con livello medio dei giocatori e tempo giocato con quella divisione.

La divisione è la squadra a cui è assegnato l'operatore (ogni divisione ha un numero di operatori che va da 2 a 4), e si può estrarre dal campo operatore, essendo nella forma 'divisione-nomeOperatore'.
Abbiamo 2 mapper ed 1 reducer:

- mapper1Divisions: ogni riga viene splittata e vengono presi divisione (ottenuta da un ulteriore split del campo operator su '-'), kills, haswon, clearanceLevel, roundDuration (espresso in secondi); kills viene moltiplicato per 5, haswon per 15 e vengono sommati tra loro (kills * 5 + haswon * 15). Inoltre, viene messo un contatore che verrà usato per calcolare il livello medio dei giocatori.

- reducer1Divisions: vengono sommati tra loro punti, livelli dei giocatori, contatore e tempo giocato con quella divisione (sommando la durata dei round in secondi).
- mapper2Divisions: calcola il livello medio (somma livello giocatori / contatore) e trasforma il tempo da intero a stringa nella forma ore:minuti:secondi.

Output: (division, (points, avgLevel, totTime))

Dopo aver eseguito i mapper e il reducer viene applicata la funzione takeOrdered() che ordina le divisioni in base ai punti e prende le prime 5. Dopodiché viene chiamata la funzione sendToDashboardDivisions() che invia alla dashboard il risultato del job e la data sulla quale è stato calcolato.
Le 5 divisioni verranno quindi rappresentate in una tabella.

Risultati

Figura 9 - Screen batch job 1 (giorno 14-02-2017)

Il gioco dà la possibilità di sbloccare gli operatori con una moneta di gioco, che si può guadagnare giocando le partite; le prime 4 divisioni sono quelle che hanno un costo degli operatori più economico rispetto alle altre, come si può notare dal livello medio dei giocatori: infatti la 5° divisione ha un livello medio più alto rispetto alle altre, proprio perché ha degli operatori più costosi che vengono quindi sbloccati a livelli più alti. Ovviamente ciò influisce anche sui punti totali e sul tempo di gioco, dato che il numero di giocatori che potrà accedere a quella divisione sarà inferiore rispetto alle altre.

Figura 10 - Screen batch job 1 (giorno 16-02-2017)

3.3.2 – Batch job 2

Descrizione: primi 3 operatori della giornata con killRatio (uccisioni / morti) e roundRatio (round vinti / round giocati) più alto.

Similmente al job 1 streaming vengono usati un mapper ed un reducer:

- mapper1Operators: viene splittata la riga e presi i campi operator, kills, isdead, haswon e viene usato un contatore per i round giocati. La chiave è operator e gli altri campi il value.
- reducer1Operators: vengono sommati tra loro uccisioni, morti, round vinti, round giocati e vengono calcolati killRatio e roundRatio.

Successivamente viene applicato un ordinamento sulla base di (killRatio / 2 + roundRatio), dando più importanza al roundRatio, e vengono presi i primi 3 operatori. Questi vengono poi inviati alla dashboard tramite sendToDashboardOperators() e visualizzati tramite un bar chart.

Risultati

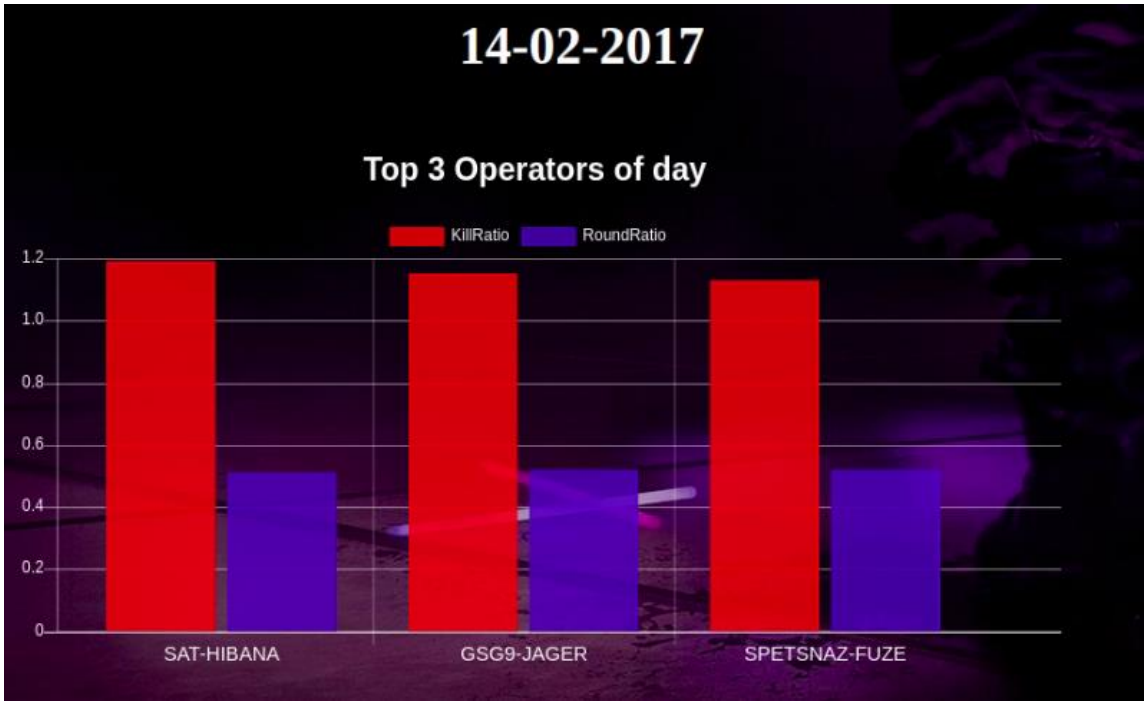


Figura 11 - Screen batch job 2 (giorno 14-02-2017)

Figura 12 - Screen batch job 2 (giorno 16-02-2017)

3.3.3 – Batch job 3

Descrizione: per ogni piattaforma calcolare round totali giocati, mappa più giocata (basato sul tempo di gioco) e tempo totale di gioco.

In questo caso vengono utilizzati 3 mapper e 3 reducer:

- mapper1Platform: la riga viene splittata e presi i campi platform, map matchId, roundNB, roundDuration e viene inserito un contatore per il numero di round. Vengono usati come chiave platform, map, matchId e roundNB e il resto come value.
- reducer1Platform: questo reducer riscrive la durata del round, andando ad escludere i tempi relativi allo stesso round dello stesso match.
- mapper2Platform: utilizza come chiave platform e map e come valore numero round giocati e tempo di gioco.
- reducer2Platform: somma i round totali e il tempo relativi ad una certa piattaforma su una certa mappa.
- finalMapperPlatform: usa come chiave platform e come value map, round totali e tempo.
- finalReducerPlatform: trova per ogni piattaforma la mappa sulla quale il tempo è maggiore e somma il tempo totale, i round totali e converte il tempo nel formato ora:minuti:secondi.

Output: (platform, (mappa, numero round totali, tempo totale, tempo in H:M:S)).

Vengono poi inviati i dati alla dashboard con la funzione sendToDashboardPlatform() e vengono rappresentati con 2 grafici: una tabella che elenca i risultati calcolati per le 3 piattaforme, in base alla giornata e un line chart che mostra l'andamento del numero di round giocati per ogni piattaforma in tutti i giorni calcolati finora.

Risultati

Figura 13 - Screen batch job 3 (tabella giorno 14-02-2017)

Platform	Most Played Map	Tot. Rounds Played	Tot. Time Played H:M:S
PS4	BORDER	1599	83:16:46
XONE	KAFE_DOSTOYEVSKY	1117	57:11:26
PC	BORDER	494	26:17:29

Figura 14 - Screen batch job 3 (tabella giorno 16-02-2017)

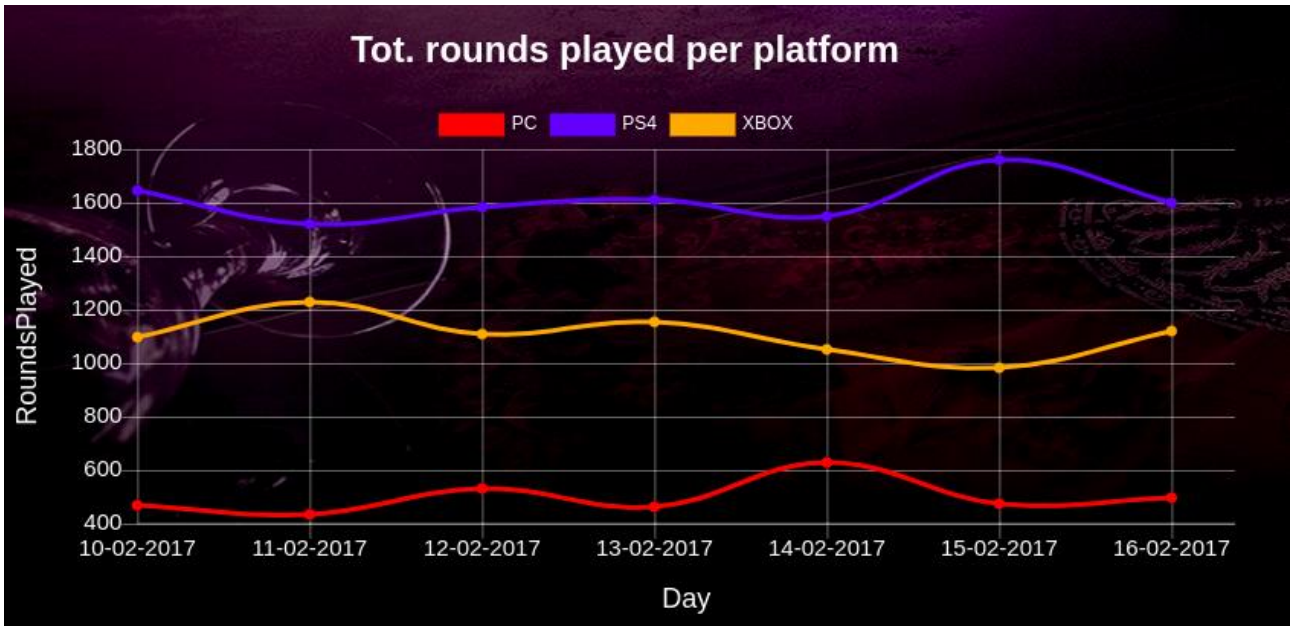


Figura 15 - Screen batch job 3 (line chart)

Si nota come la piattaforma con più tempo di gioco e round giocati sia la PS4, indipendentemente dal giorno, e al secondo posto l'XBOX.

4 - CONCLUSIONI

L'analisi di dati real time risulta utile anche in contesti nei quali potremmo non averne bisogno e può fornire risultati utili per prevenire problemi e agire di conseguenza. Nel caso analizzato i risultati possono fornire informazioni agli sviluppatori per stabilizzare il gioco ed evitare che certi operatori risultino troppo forti e decisivi su certe mappe e in certe modalità di gioco. Ogni mese vengono rilasciati numerosi aggiornamenti e ogni 3-4 mesi inizia una nuova stagione con il rilascio di nuove mappe e nuovi operatori che possono portare a sbilanciamenti; grazie alle analisi è possibile determinare se l'inserimento di questi nuovi operatori può portare problemi in modo da rilasciare aggiornamenti per cercare di equilibrare tutti gli operatori e invogliare i giocatori a provarli tutti. Anche l'analisi delle mappe può essere utile a questo scopo, per rilevare problemi su quelle aggiunte successivamente e capire se risultino bilanciate come le altre già presenti. Altre analisi possono invece fornire statistiche da visualizzare in classifiche, interessanti per i giocatori che cercano consigli sulla scelta degli operatori e delle divisioni.

L'analisi sulle piattaforme può invece essere sfruttata per promuovere il gioco laddove viene utilizzato di meno, con pubblicità mirate, riduzione dei prezzi o offerte a tempo sui contenuti, in modo da spingere all'acquisto un numero maggiore di giocatori che magari non dispone di una console di gioco, ma solo di un pc: è possibile inoltre vedere l'andamento durante queste campagne di promozione per capire se funzionano e se spingono un maggior numero di giocatori all'acquisto del gioco e quindi all'aumento del tempo di gioco su una certa piattaforma.

Su questi tipi di dato si possono effettuare molte altre analisi di vario genere, dipendenti dall'intento che si vuole raggiungere: ottimizzare la stabilità del gioco, invogliare una fetta sempre più grande di giocatori all'acquisto, visualizzare delle classifiche a punti in cui si contribuisce giocando le partite o calcolare le statistiche personali di un certo giocatore che potrà poi visualizzarle nel suo profilo utente privato. Il giocatore di conseguenza apprezza la partecipazione degli sviluppatori a fornire sempre maggiori contenuti mantenendo il gioco bilanciato e aggiornato. Un possibile sviluppo futuro, da integrare con quanto realizzato, potrebbe essere l'utilizzo di feedback e commenti utente, estratti sia da social network che dal forum ufficiale del gioco, per trovare e migliorare i punti deboli del gioco, cosa non facile da determinare da semplici statistiche o dati, ma solo giocando e interagendo con l'ambiente di gioco.

5 – BIBLIOGRAFIA

- Dataset: https://www.kaggle.com/maxcobra/rainbow-six-siege-s5-ranked-dataset