Analisi e Gestione di un FMS

Progetto per il corso PMCSN 2024/2025

Anastasia Brinati
Università degli studi di Roma Tor Vergata
Roma, Italia
anastasia.brinati@alumni.uniroma2.eu
0321654

Giulio Appetito
Università degli studi di Roma Tor Vergata
Roma, Italia
giulio.appetito@alumni.uniroma2.eu
0321669

I. Introduzione

A marzo 2015, un'interruzione di 12 ore dell'Apple Store è costata all'azienda 25 milioni di dollari [2]; ad agosto 2016, un'interruzione di corrente di cinque ore in un centro operativo ha causato 2.000 voli cancellati e una perdita stimata di 150 milioni di dollari per Delta Airlines [3].

Questi eventi evidenziano quanto sia cruciale la continua operatività dei servizi web per le aziende che vendono prodotti o servizi ai consumatori attraverso piattaforme online, sottolineando l'importanza di mantenere un'elevata disponibilità: non solo per proteggere i ricavi e la reputazione aziendale, ma anche per garantire la fiducia dei clienti e la soddisfazione degli utenti finali.

Si è soliti misurare la disponibilità di un servizio cloud sulla base della percentuale di volte in cui, in un periodo temporale prefissato, il servizio stesso è pubblicamente accessibile. Tale valore viene chiamato *uptime* e fa riferimento all'ammontare di tempo, su base percentuale, in cui un server web resta attivo e accessibile dall'esterno. Molti hosting provider pubblicizzano la percentuale di uptime per indicare la percentuale di disponibilità dei propri servizi su base annuale, mensile e/o settimanale. L'unità di misura complementare è il downtime (o tempo di disservizio).

La tabella seguente mostra alcuni esempi di uptime e downtime, per comprendere, ad esempio, come un uptime del 90% equivale a circa 36 giorni di disservizio all'anno (non necessariamente consecutivi), periodo nel quale sarà impossibile accedere al servizio. Percentuali ottimali di uptime vanno dal 99,1% al 99,5% e oltre.

I downtime di un sito web possono essere causati da manutenzione delle macchine server, problemi di rete o configurazioni errate delle stesse. [4]

Per questo, uno degli aspetti piú critici di tali sistemi é la gestione delle failure: la natura distribuita implica che i componenti del sistema possano fallire indipendentemente, portando a potenziali interruzioni del servizio e perdita di dati.

La gestione delle failure é un'attivitá complessa che richiede una combinazione di strategie di prevenzione, monitoraggio, tolleranza e recupero. Investire in tali tecniche non solo migliora l'affidabilitá e la disponibilitá del sistema, ma contribuisce anche a garantire un'esperienza utente continua e di alta qualitá.

Uptime %	Downtime su base annuale	Downtime su base mensile*	Downtime su base settimanale
90%	36.5 giorni	72 ore	16.8 ore
95%	18.25 giorni	36 ore	8.4 ore
97%	10.96 giorni	21.6 ore	5.04 ore
98%	7.30 giorni	14.4 ore	3.36 ore
99%	3.65 giorni	7.20 ore	1.68 ore
99.5%	1.83 giorni	3.60 ore	50.4 minuti
99.8%	17.52 ore	86.23 minuti	20.16 minuti
99.9%	8.76 ore	43.2 minuti	10.1 minuti
99.95%	4.38 ore	21.56 minuti	5.04 minuti
99.99%	52.56 minuti	4.32 minuti	1.01 minuti

Fig. 1: Tabella tempistiche di downtime

A. Il ciclo MAPE

Un approccio efficace per affrontare questa sfida é l'implementazione del ciclo MAPE (Monitor, Analyze, Plan, Execute). È importante sottolineare che il ciclo MAPE è un processo iterativo, ovvero, si tratta di un ciclo continuo di osservazione, analisi, pianificazione ed esecuzione, che permette un miglioramento costante della resilienza del sistema.

Il **Monitoraggio** rappresenta la fase iniziale in cui vengono raccolti e registrati dati in tempo reale sull'operatività del sistema, incluse metriche di performance, log di errori e dati sull'utilizzo delle risorse. Questo processo è essenziale per mantenere una visione costante dello stato di salute del sistema e per rilevare tempestivamente eventuali anomalie.

Segue la fase di **Analisi**, i cui obiettivi sono identificare pattern di errore, correlare problemi di performance con eventi

specifici (es. aumento di traffico, aggiornamenti software, ...), fino a prevedere potenziali guasti o degrado delle prestazioni, tramite analisi predittiva e tecniche di machine learning.
Una volta terminata l'Analisi si procede con la **Pianificazione**, al fine di sviluppare strategie di soluzione e/o mitigazione per i problemi identificati, creare playbook per la risoluzione automatica e manuale dei problemi (es. Ansible) e definire piani di scalabilitá per gestire i picchi di traffico (es. Kubernetes).
L'ultima fase é l'**Esecuzione**, in cui si eseguono le azioni pianificate e ci si assicura che le azioni correttive siano eseguite tempestivamente e con successo. Esempi pratici di Esecuzione sono la scalabilità automatizzata (computazione playbook Ansible per lanciare istanze AWS EC2 aggiuntive durante i picchi di traffico), il riavvio dei servizi (processamento script di riavvio automatico) oppure la manutenzione programmata (processamento script di pulizia e ottimizzazione del database durante le finestre di manutenzione pianificate).

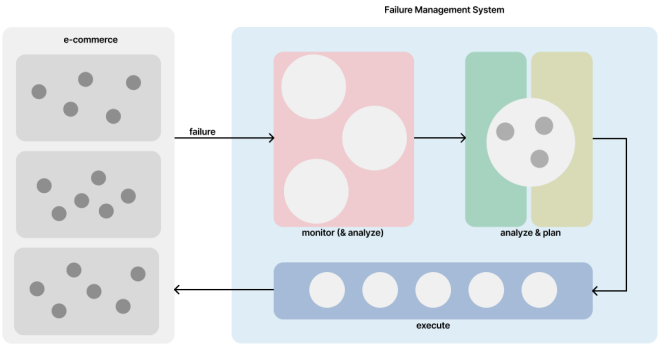


Fig. 2: Modello FMS

II. Sistema

Con il presente studio approfondiamo l'esempio del Failure Management System (FMS) per una rete di server di un'azienda di e-commerce, progettato con l'architettura che troviamo in figura 3. L'FMS in questione prevede per la fase di monitoraggio tre nodi/containers che eseguono *Prometheus*, raccogliendo, elaborando ed archiviando log contenenti metriche di performance, log di errori e dati derivanti dai nodi sull'utilizzo delle risorse. Ogni container Prometheus è configurato per monitorare specifici aspetti del sistema, descritti come segue:

- **log del server web** (Apache, Nginx): questi log raccolgono informazioni sulle richieste HTTP gestite dai server web, comprese le risposte di errore, includendo dettagli come l'ora dell'errore, l'URL richiesto, il codice di stato HTTP, l'indirizzo IP del client.
  Questi dati sono cruciali per identificare problemi di configurazione, sovraccarichi del server, e altre anomalie che possono compromettere l'accessibilità del sito web.
  Ad esempio, un errore HTTP 500 indica un problema lato server che impedisce l'elaborazione corretta delle richieste;
- **log del database** (MySQL, PostgreSQL): i log del database registrano eventi legati all'esecuzione delle query SQL e alle operazioni del database.
  Possono includere informazioni sul consumo di risorse come l'utilizzo della CPU e della memoria durante l'esecuzione di query intensive, entrambe fondamentali per ottimizzare le prestazioni del database, rilevare query inefficienti, e prevenire problemi di scalabilità.
  Ad esempio, un log potrebbe indicare che una query SQL specifica sta causando un carico eccessivo sulla CPU, rallentando le prestazioni del database;
- **log delle applicazioni** (errori runtime, stack trace): questi log catturano gli errori che si verificano durante l'esecuzione delle applicazioni, come errori di runtime e stack trace.
  Possono anche registrare metriche sulle prestazioni, come la latenza di rete e il tempo di risposta delle applicazioni.
  Ad esempio, un log potrebbe mostrare che la latenza di rete aumenta durante certi intervalli di tempo, suggerendo un possibile problema di rete o di scalabilità dell'applicazione.

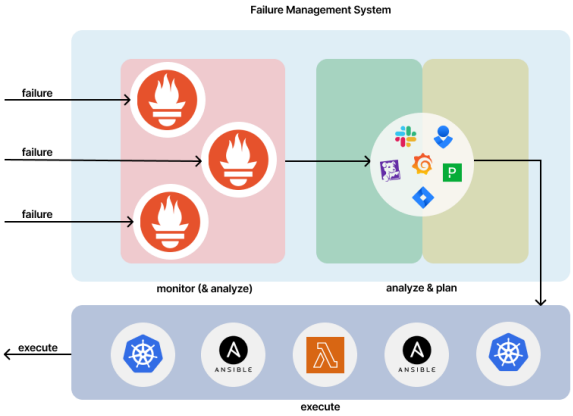


Fig. 3: Modello d'esempio di un FMS

Il monitoraggio in tempo reale di una vasta rete di server può generare un'enorme quantità di dati: gestire, archiviare e analizzare questi dati in modo efficiente richiede risorse significative e una robusta infrastruttura dati. I messaggi di log provenienti da una rete di nodi con dispositivi eterogenei, raccolti e generati da cause distinte, comportano dimensioni e velocità nell'arrivo differenti dei messaggi. I dati raccolti dai container Prometheus vengono inviati al cluster di containers che si occupa di analisi, visualizzazione e pianificazione con strumenti quali AlertManager (e. g. PagerDuty) e Grafana, per realizzare dashboard interattive in tempo reale, identificare anomalie e pattern di errore ricorrenti, generare avvisi su potenziali problemi di performance, generare piani sotto forma di script di automazione (e. g.) playbook Ansible.
Nella fase di Esecuzione del ciclo MAPE, il Failure Management System (FMS) esegue i piani d'azione elaborati nelle fasi precedenti. Tuttavia, è importante sottolineare che l'effettiva implementazione di questi piani avviene diretta-

mente tramite le componenti operative dell'infrastruttura di e-commerce monitorata, che implementano le azioni necessarie per ripristinare il corretto funzionamento del sistema.
In pratica, l'FMS termina la sua funzione al momento della generazione dei piani d'azione. Pertanto, il nostro studio si concentra sull'analisi di un FMS, ad esclusione dell'effettiva esecuzione delle azioni correttive, la quale è delegata alle componenti operative interne dell'infrastruttura di e-commerce. Questo approccio permette al sistema di mantenere un'elevata disponibilità e affidabilità, garantendo una risposta tempestiva e mirata ai problemi identificati.
Dal punto di vista dell'architettura, l'FMS in esame si articola di 2 cluster di nodi:

- il primo costituito da 3 containers Prometheus, dedicati al monitoraggio;
- il secondo, dedito all'analisi e pianificazione, composto di un container Alertmanager (PagerDuty + creazione playbook) ed uno Grafana.

Il tutto è deployato tramite Docker Compose su una singola macchina, la quale ha a disposizione 4 core.

III. Aspetti Critici

Lo scaling orizzontale è una scelta non sempre congeniale quando, come in questo caso, si hanno risorse limitate appena sufficienti per le mansioni considerate.

Per quanto riguarda i tempi d'attività di un container Prometheus ci si riferisce principalmente ai tempi di risposta delle query, ovvero l'operazione da parte di Prometheus per il salvataggio e l'aggregazione dei messaggi di log ricevuti. La maggior parte richiede da meno di 100 millisecondi a meno di un 1 secondo, in quanto opera con poche metriche e/o dati su un breve periodo di tempo (e. g. ottenere l'ultimo valore di una metrica per un singolo target); raramente, alcune possono richiedere da 1 a 5 secondi: si tratta di query che aggregano dati da molteplici target, o che richiedono operazioni complesse come rate, histogram quantiles o join tra molte serie temporali su periodi di tempo lunghi (settimane o mesi). Equivalentemente, per i container AlertManager e Grafana, sono previste tempistiche differenti in base al tipo di **query**: la maggior parte di queste richiede da meno di 100 millisecondi a meno di un 1 secondo (e. g. elaborazione di dashboard semplici e routing moderato); compaiono raramente query che aggregando dati da molteplici target, prevedendo operazioni complesse come rate, histogram quantiles o join tra molte serie temporali su periodi di tempo lunghi (settimane o mesi), possono richiedere da 1 a 3 secondi.
Dato che entrambi i cluster operano in tempo reale, il tempo necessario per analizzare i dati e decidere sulle azioni da intraprendere può variare fino a circa mezzo secondo. [7] [11]
Per quanto riguarda l'esecuzione, se si utilizza un sistema di orchestrazione come Kubernetes, il provisioning di nuove risorse (come nuovi pod) può richiedere anche da 1 a 5 minuti a seconda del carico del cluster e della disponibilità delle risorse. D'ora in avanti nella trattazione con il termine 'sistema' faremo riferimento solo ed unicamente al FMS con l'architettura d'esempio.

IV. Obiettivi

Il Failure Management System (FMS) deve avere un tempo di risposta massimo per prevenire rallentamenti percepibili dagli utenti. Ad esempio, nel caso in cui fosse necessario scalare il sistema aggiungendo un nodo, la decisione deve essere presa dall'FMS nel minor tempo possibile. In totale, ci si aspetta che il tempo di reazione dell'FMS sia non superiore a mezzo secondo.

- **Obiettivo 1**: *Mantenere il tempo medio di risposta del sistema al di sotto di* 0.5 *secondi.*

Per aumentare l'affidabilità complessiva del sistema di e-commerce, l'obiettivo è mantenere la disponibilità del sistema pari ad almeno il 90% di uptime su base annua, ovvero downtime massimo di 16.8 ore settimanali (6 min/h). [4]

- **Obiettivo 2**: *Mantenere il tempo medio di risposta per log del server web, al di sotto di* $\frac{6}{5208.34025 * 11.925\%} = 0.0096603645$ *secondi.* [8]

Qui, 5208.34025 è il tasso medio di arrivo totale dei messaggi di log al FMS, e 0.11925 è la percentuale di arrivi di messaggi di log del server web, come mostreremo inseguito.

V. Perdita e Profitto

Secondo uno studio condotto da Gartner nel 2014 il costo **medio** del tempo di inattività è di 5.600 dollari al minuto. Da un report Avaya dello stesso anno è emerso che le medie variavano da 2.300 a 9.000 dollari al minuto, e dal 2014, quella cifra è in aumento. Un report realizzato dal Ponemon Institute nel 2016 aumenta la media di Gartner da 5.600 dollari al minuto a quasi 9.000 dollari al minuto. Per le piccole imprese, il numero scende al livello più basso ma comunque significativo compreso tra 137 e 427 dollari al minuto. Il punto in cui un'azienda si posiziona in questo spettro molto ampio dipende da una serie di fattori, tra cui il verticale di settore, le dimensioni dell'organizzazione e il modello di business (un sito di e-commerce ha ovviamente più da perdere se si verifica un'interruzione del Web rispetto a un'azienda con punti vendita fisici). Quanto più il modello di business si basa sul tempo di attività, tanto maggiori sono, ovviamente, le perdite causate dal tempo di inattività. Per il colosso dell'e-commerce Amazon, il cui intero modello di business si basa sul tempo di attività, i costi stimati sono di circa 13,22 milioni di dollari all'ora [5]. Implementare e mantenere un FMS sofisticato può essere costoso. I costi includono software, hardware, risorse umane e formazione, oltre alle spese operative continue per mantenere il sistema aggiornato e funzionante, per cui è essenziale ottimizzare l'utilizzo delle risorse e ponderare le scelte di scaling orizzontale e verticale. Nel caso di studio in questione, consideriamo l'azienda di e-commerce come una piccola impresa, per cui un minuto di inattività equivale a perdere $137\$ = 127,80 eur$.

VI. Modello concettuale

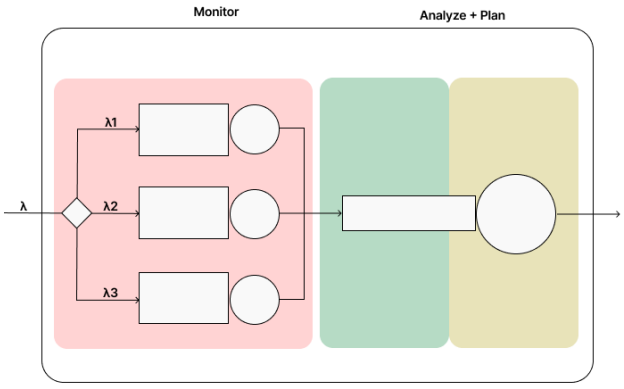


Fig. 4: Modello concettuale del FMS

A. Componenti del sistema

Il sistema è suddiviso nelle seguenti macroaree:

- **Monitoraggio**: permette la diagnosi di alert e guasti segnalati al sistema. Questa zona prevede 3 centri per la ricezione dei messaggi di log, distinti come descritto in precedenza (log del server web, log del database, log delle applicazioni). In quest'area si identifica il tipo di segnalazione al fine di indirizzare in messaggio verso la giusta priorità;
- **Analisi&Pianificazione**: raccoglie ogni log analizzandone le cause ed elaborando un piano d'azione che possa risolvere il guasto o arginare il problema.

B. Descrizione degli Eventi

Nel contesto del FMS, un *messaggio di log* corrisponde ad un *job* che arriva al sistema e, indipendentemente dalla tipologia del messaggio, passa attraverso tutte le fasi del ciclo, nello stesso ordine. Utilizzeremo i termini log e job in modo interscambiabile per indicare un'entità che entra nel sistema.
Per il centro di Monitoraggio:

- l'evento di arrivo di un job coincide con l'arrivo di un messaggio di log al sistema;
- l'evento di uscita per il centro è dato dal processamento del servizio del job da parte di uno dei 3 serventi, ovvero l'identificazione del problema e/o guasto.

Un guasto viene letto da un solo server: se lo trova occupato, il job viene messo in coda secondo una politica FIFO astratta, senza poter abbandonare il sistema, in quanto il guasto non è stato risolto.
A seguire, il messaggio di errore è indirizzato all'area di Analisi&Pianificazione, dove viene ripartito verso la coda più congeniale alla sua gestione in base alla sua priorità. Sono presenti **N** serventi dediti a tale operazione. Se tutti i serventi sono occupati, il job rimane in attesa secondo una politica FIFO astratta.

Per il centro di Analisi&Pianificazione:

- l'arrivo di un job coincide con l'uscita di un job dal centro di Monitoraggio;
- l'evento di uscita per il centro è dato dal processamento del servizio del job, che include l'analisi del log e la pianificazione del piano esecutivo per risolvere il problema identificato.

L'evento di uscita per il centro appena descritto coincide con l'evento di uscita dal sistema. Nello stato attuale dell'analisi, tutte le code sono definite con priorità FIFO astratte.

VII. Modello delle specifiche

A. Tasso medio di arrivo

Il parametro λ (tasso medio di arrivo) è stato derivato a partire dal dataset dell'*ACM DEBS 2024 Grand Challenge* [13], che riporta dati di monitoraggio S.M.A.R.T. raccolti nell'arco di 23 giorni. Tramite il numero medio di messaggi di log ricevuti giornalmente, otteniamo:

$$\lambda = 125000.166\frac{job}{day} = 5208.34025\frac{job}{h}$$

$$= 86.805670833\frac{job}{min} = 1.446\frac{job}{s}$$

Successivamente, al fine di calcolare le probabilità di routing verso ciascun centro dell'area di monitoraggio, e conseguentemente i relativi tassi di arrivo $\lambda_1, \lambda_2, \lambda_3$., abbiamo utilizzato come riferimento le probabilità riportate nel dataset *Synthetic Log Data of Distributed System* [9]. Il dataset, infatti, riporta i LogLevel per ogni messaggio di log, che abbiamo riassunto nelle seguenti percentuali per livello di criticità:

- *FATAL* (11.925%)
- *ERROR* (15.851%)
- *WARNING* (26.181%)
- *INFO* (20.142%)
- *DEBUG* (25.901%)

Questi ultimi sono stati mappati sui tre tipi di messaggi di log da noi considerati:

- il livello *FATAL* è stato associato ai messaggi di log del server web;
- il livello *ERROR* è stato associato ai messaggi di log del database;
- infine, i livelli *WARNING, INFO e DEBUG* sono stati raccolti tutti sotto il termine WARNING e mappati sui messaggi di log delle applicazioni.

B. Area di Monitoraggio

Tale area viene modellata con 3 centri $M/L_{\mathrm{n}}/1$. La distribuzione dei tempi di interarrivo è Esponenziale di parametro λ, e ciascun centro riceve rispettivamente un flusso di tasso:

- $\lambda_1 = \lambda \cdot p_{\mathrm{fatal}}$;
- $\lambda_2 = \lambda \cdot p_{\mathrm{error}}$;
- $\lambda_3 = \lambda \cdot p_{\mathrm{warning}}$;

La distribuzione dei servizi per ciascun servente è data da una distribuzione Log-Normale(μ, σ^2), (possibilmente troncata a 5 sec), avente come media: $E[S] = 0.5$ secondi.

La distribuzione log-normale è utile per modellare variabili aleatorie sempre positive, asimmetriche e con una lunga coda verso destra, che rappresenta valori che possono essere molto più grandi della media. Questa distribuzione riesce a catturare adeguatamente la variabilità e gli outlier tipici dei tempi di risposta dei container. Secondo il principio della massima entropia, se non si conosce nulla di una distribuzione, eccetto che appartiene a una certa classe (solitamente definita in termini di proprietà o misure specificate), allora si dovrebbe scegliere la distribuzione con la maggiore entropia come default meno informativo. In teoria dell'informazione, l'entropia di una variabile casuale rappresenta il livello medio di "informazione", "sorpresa" o "incertezza" intrinseco ai possibili risultati della variabile stessa. Data una variabile casuale discreta X, che assume valori nell'insieme $\mathcal{X}$ ed è distribuita secondo p:

$$p : \mathcal{X} \to [0, 1]$$

l'entropia $H(X)$ è definita come:

$$H(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

dove $\sum$ denota la somma sui possibili valori della variabile.

La motivazione è duplice: primo, massimizzare l'entropia minimizza la quantità di informazioni a priori integrate nella distribuzione; secondo, molti sistemi fisici tendono a muoversi verso configurazioni di entropia massima nel tempo.

Come suggerito dal libro di testo, inoltre, la distribuzione log-normale è una valida alternativa alla distribuzione Erlang per modellare i tempi di servizio. Le variabili casuali Erlang sono spesso utilizzate per modellare tempi di servizio casuali, in particolare quando il servizio è definito da una serie di sottoprocessi indipendenti. Questo è coerente con il fatto che essa rappresenta la somma di n variabili casuali esponenziali. Nessun job può lasciare nessuna delle code di monitoraggio.

C. Area di Analisi&Pianificazione

Tale area viene modellata con un centro $X/L_{\mathrm{n}}/1$.
La distribuzione dei servizi è rappresentata da una Log-Normale troncata a 3 secondi, avente come media: $E[S] = 0.3$ secondi.
Nessun job può lasciare la coda di analisi&pianificazione.

VIII. Modello computazionale

Per la realizzazione del progetto, si è optato per l'utilizzo di Python. Per la simulazione del sistema, è stato utilizzato il paradigma Next-Event Simulation.

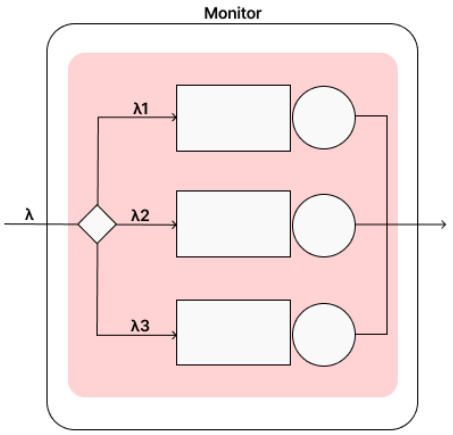


Fig. 5: Centro di Monitoraggio

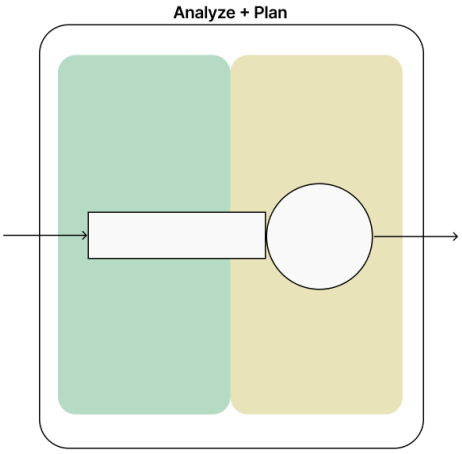


Fig. 6: Centro di Analisi&Pianificazione

A. Struttura

Il progetto è organizzato con la seguente struttura:

- la directory *subsystems*, contiene i singoli centri modellati come classi;
- la directory *libs* contiene tutti i moduli predefiniti in Python da Philip Steele, tradotti dal codice orginale di Steve Park & Dave Geyer in ANSI C;
- la directory *utils* contiene varie componenti utili per la modularizzazione ed il riuso del codice, come ad esempio event.py;
- infine, esternamente abbiamo i programmi original-main.py e system.py, bettermain.py e bettersystem.py, che inizializzano il sistema ed esegueno le simulazioni ad orizzonte finito o infinito.

```python
class MonitoringCentre:
    number = []                 # time integrated numbers in the node  */
    queue = []                  # in the queues  */
    service = []                # in service     */
    departed = []               # number served                        */
    servers = 0                 # number of servers                    */

    AnastasiaBrinati
    def __init__(self, servers) -> None:...

    2 usages  AnastasiaBrinati
    def get_events(self) -> list:...

    1 usage  AnastasiaBrinati
    @staticmethod
    def route_arrival() -> int:
        r = rngs.random()
        if r <= FATAL:
            return 0
        elif FATAL < r <= FATAL+ERROR:
            return 1
        else:
            return 2
```

(a) Codice implementato per il centro di monitoraggio, parte 1

```python
def get_arrival(self, stream) -> tuple[Any, int]:
    # ---------------------------------------------
    # * generate the next arrival time
    # * same stream, different routing
    # * ---------------------------------------------
    # */
    global ARRIVAL_TEMP

    rngs.selectStream(stream)
    ARRIVAL_TEMP += rvgs.Exponential(1/LAMBDA)

    centre = self.route_arrival()
    return ARRIVAL_TEMP, centre
```

```python
4 usages  AnastasiaBrinati *
def get_service(self, stream) -> int:
    # ---------------------------------------------
    # * generate the next service time for each
    # * server independently (each their own stream)
    # * ---------------------------------------------
    # */
    rngs.selectStream(stream)
    return rvms.idfLognormal(a, b, u)
    # TO VERIFY:
    #return rvgs.Exponential(MU)
```

(b) Codice implementato per il centro di monitoraggio, parte 2

```python
class PlanningCentre:
    number = 0                  # time integrated number in the node    */
    queue = []                  # in the queue    */
    service = []                # in service      */
    departed = 0                # number served                         */

    AnastasiaBrinati
    def __init__(self) -> None:...

    1 usage  AnastasiaBrinati
    def get_events(self) -> list:...

    ...

    2 usages  AnastasiaBrinati *
    def get_service(self, stream) -> int:
        # ---------------------------------------------
        # * generate the next service time
        # * ---------------------------------------------
        # */
        rngs.selectStream(stream)
        return rvms.idfLognormal(a, b, u)
        # TO VERIFY:
        #return rvgs.Exponential(mu)
```

(c) Codice implementato per il centro di analisi&pianificazione

```python
# service times distribution:                                          */
#               X =  Lognormal(a, b)   x > 0                           */
# The mean and variance are                                            */
#              μ = mean = exp(a + 0.5*b*b)  = 0.55                     */
#              σ^2 = variance = (exp(b*b) - 1)*exp(2*a + b*b) = 0.45   */
# Follows that                                                         */
#              a = log(μ) - b^2/2                                      */
#              b = sqrt ( log( [σ^2 / log(μ)^2] + 1 ) )                */
a = -0.843414
b = 0.714951

# TRUNCATION                                                           */
# right tail cut at x = 5 seconds                                      */
#                 β = 1-Pr(X ≤ x) = 1-F(x)                             */
# determine left-tail and right-tail truncation probabilities α and β  */
# in our case only the right truncation is needed                      */
alpha = rvms.cdfLognormal(a, b, x=0.0)                          # α */
beta = 1.0 - rvms.cdfLognormal(a, b, x=5.0)                     # β */
# constrained inversion                                              */
u = rvgs.Uniform(alpha, 1.0 - beta)
# now we can use: rvms.idfLognormal(a, b, u)                         */
```

(d) Codice implementato per troncare la LogNormal a b=5secondi, seguendo le linee guida suggerite dal libro di testo. [1]

Fig. 7: Modello Computazionale

IX. VERIFICA

Con la fase di verifica, verifichiamo che il modello computazionale implementato risulti *corretto*. Per la realizzazione delle verifiche, si farà uso di tempi di servizio esponenziali, in quanto, altrimenti, non si avrebbe un confronto valutabile dal punto di vista analitico con le distribuzioni lognormali troncate. Il confronto avverrà considerando il caso stazionario, con simulazioni ad orizzonte infinito adottando la tecnica delle *batch means*, che tratteremo con più dettaglio in seguito.

A. Centro di monitoraggio

$\lambda = 1.446 job/sec$
$\mathbb{E}[S] = 0.55 sec$
$p_1 = 0.12, \quad p_2 = 0.16, \quad p_3 = 1 - p_1 - p_2$

$\lambda_i = p_i \cdot \lambda, \quad \lambda_1 = 0.17352 job/sec, \quad \lambda_2 = 0.23136 job/sec, \quad \lambda_3 = 1.04112 job/sec$
$\rho_i = \lambda_i \cdot \mathbb{E}[S], \quad \rho_1 = 0.095436, \quad \rho_2 = 0.127248, \quad \rho_3 = 0.572616$

$\mathbb{E}[T_{q_i}] = \dfrac{\rho_i \cdot \mathbb{E}[S]}{1 - \rho_i} \quad \mathbb{E}[T_{q_1}] = 0.0580277 sec, \quad \mathbb{E}[T_{q_2}] = 0.080190 sec, \quad \mathbb{E}[T_{q_3}] = 0.736898 sec$
$\mathbb{E}[T_q] = \sum_{i=1}^{3} (p_i \cdot \mathbb{E}[T_{q_i}]) = 0.562379 sec$

$\mathbb{E}[T_{S_i}] = \mathbb{E}[S] + \mathbb{E}[T_{q_i}]$
$\mathbb{E}[T_S] = \sum_{i=1}^{3} (p_i \cdot \mathbb{E}[T_{S_i}]) = 1.112379 sec$

Formula	Analytical Result	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.5503609	0.5480248044018853	$+/-$ 0.0041935296275504436
$\mathbb{E}[T_{S_1}]$	0.6080277	0.6090930284411377	$+/-$ 0.0024814535490544193
$\mathbb{E}[T_S]$	1.1003609	1.0978848095708016	$+/-$ 0.004515655381576862
ρ_1	0.095436	0.09596328087193738	$+/-$ 0.0005658879170999494
ρ_2	0.127248	0.12671301217861775	$+/-$ 0.000688077149394328
ρ_3	0.572616	0.5892615152411455	$+/-$ 0.003082013468347346

TABLE I: Tabella di verifica per il confronto dei risultati analitici ed empirici del centro di monitoraggio.

B. Centro di analisi&pianificazione

$\lambda = 1.446$ job/sec, $\quad \mathbb{E}[S] = 0.3$ sec
$\rho = \lambda \cdot \mathbb{E}[S] = 0.4338$

$\mathbb{E}[T_q] = \frac{\rho \cdot \mathbb{E}[S]}{1-\rho} = 0.229848$ sec

$\mathbb{E}[T_S] = \mathbb{E}[S] + \mathbb{E}[T_q] = 0.529848$ sec

Formula	Analytical Result	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.229848	0.23054151490056773	$+/-$ 0.0013103771327328043
$\mathbb{E}[T_S]$	0.529848	0.5308612826681952	$+/-$ 0.001527552929089431
ρ	0.4338	0.440033428717642	$+/-$ 0.0007184446942382347

TABLE II: Tabella di verifica per il confronto dei risultati analitici ed empirici del centro di analisi&pianificazione.

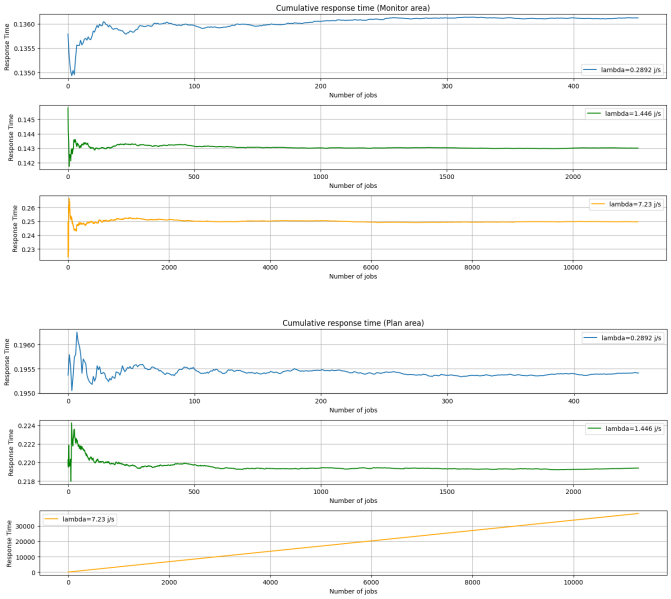
C. Controlli di consistenza

Oltre alla corrispondenza fra i valori prodotti dalla simulazione ed i risultati analitici, osserviamo anche il rispetto dei seguenti criteri di consistenza:

- $\mathbb{E}[T_S] = \mathbb{E}[S] + \mathbb{E}[T_q]$
- $0 < \rho < 1$

X. Validazione

Giunti alla fase di validazione, ci si domanda se il modello computazionale realizzato è consistente con il sistema in analisi. Non essendo in possesso di dati relativi al sistema in esame con cui effettuare un confronto, per presentare una validazione comunque significativa, abbiamo deciso di controllare che il valore delle metriche vari al variare del tasso di arrivo. A parità di configurazione, un'utilizzazione più elevata implica un peggioramento delle prestazioni del sistema, e viceversa. In seguito, sono riportati gli intervalli di confidenza e i grafici a dimostrazione di un peggioramento nelle prestazioni dei centri.



XI. Design degli esperimenti

Segue la sezione dedicata agli esperimenti sul sistema in esame coinvolgendo simulazioni ad orizzonte finito ed infinito.

A. Simulazione ad orizzonte finito

La simulazione è stata effettuata sul sistema con la tecnica della Replicazione, di modo da ottenere stime indipendenti per le statistiche transitorie.
Il seed iniziale è quello di default della libreria pari a 123456789, inizializzato esternamente al ciclo che ripete le simulazioni, utilizzando lo stato finale di ciascuna sequenza di numeri casuali come stato iniziale della replicazione successiva. Il tempo di simulazione è stato posto pari a 2000000. L'intervallo di confidenza è stato individuato seguendo i passi descritti dall'algoritmo 8.1.1. del libro di testo [1]. Scegliendo come livello di confidenza 95% = (1 - α)%, considerando il tipico valore $\alpha = 0.05$, e computando le statistiche campionarie quali media e deviazione standard, possiamo calcolare il valore t^* = idfStudent$(n-1, 1-\frac{\alpha}{2})$, con il quale possiamo calcolare gli estremi degli intervalli. In questo modo, dato un n sufficientemente grande, possiamo affermare di essere $(1-\alpha)\cdot 100\%$ *confident* che la media ricada nell'intervallo definito, ovvero, se andassimo a considerare molteplici intervalli di confidenza in questo modo, allora approssimativamente il $(1-\alpha)\cdot 100\%$ di questi coprirebbe il vero valore della media. [1]

Formula	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.00848381170632739	$+/-$ 0.0001668062862291411
$\mathbb{E}[T_{S_1}]$	0.13621629451514936	$+/-$ 0.0001413429145800615
$\mathbb{E}[T_S]$	0.14300242067438107	$+/-$ 0.0001668062862289719
ρ_1	0.02350863413345103	$+/-$ 0.00029793455082677
ρ_2	0.0312804359207796	$+/-$ 0.00031439179360743895
ρ_3	0.13983502640712936	$+/-$ 0.0006961722520697222

TABLE III: Risultati esperimenti centro di monitoraggio, simulazione ad orizzonte finito con n=20 replicazioni.

Formula	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.02827598108285837	$+/-$ 0.0004572327957054056 6
$\mathbb{E}[T_S]$	0.2193861084938665	$+/-$ 0.0004572327957051367
ρ	0.27641984405618525	$+/-$ 0.0013581087132465335

TABLE IV: Risultati esperimenti centro di analisi&pianificazione, simulazione ad orizzonte finito con n=20 replicazioni.

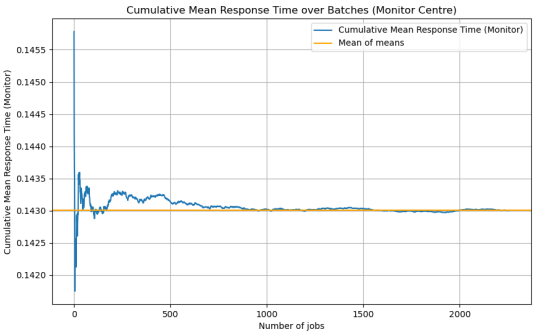
Al fine di rimanere fedeli alla realtà del sistema in esame, poichè un FMS implementato sulle linee guida del paradigma MAPE è costantemente in funzione, operando ciclicamente, ci concentreremo principalmente sull'esecuzione delle simulazioni a orizzonte infinito, per cui si assume che l'ambiente del sistema sia statico (non c'è scaling orizzontale).

B. Simulazione ad orizzonte infinito

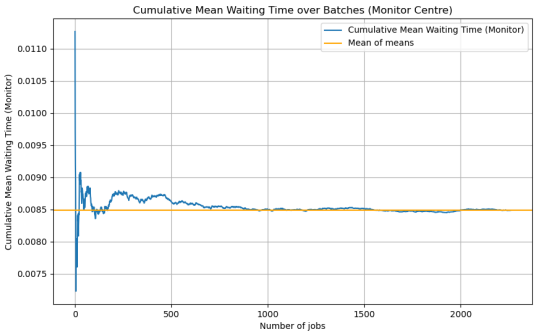
La simulazione ad orizzonte infinito prevede che il sistema venga simulato per un tempo virtualmente infinito; al fine di realizzare questo scenario, si è scelto di utilizzare un tempo di simulazione molto grande. Per la simulazione del sistema è stato adottato il metodo *Batch Means* al fine di valutare il comportamento del sistema nello stato stazionario, in quanto elimina il bias dello stato iniziale: le statistiche di ogni batch (ad eccezione del primo) sono inizializzate allo stato del sistema nel momento in cui i contatori statistici del batch sono resettati. [1]

1) Come scegliere (b, k)?: Poichè la scelta della coppia (b, k) ha impatto sulla dimensione dell'intervallo di stima è stato necessario pesarne la decisione tenendo conto delle seguenti osservazioni:

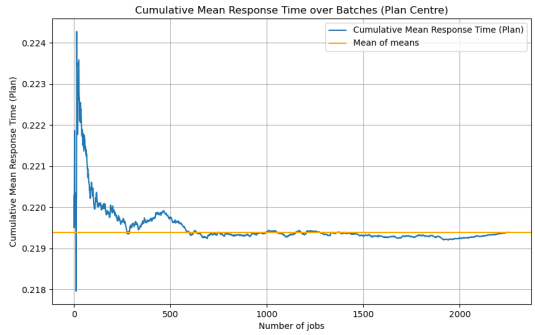
- la media di b variabili, indipendenti o meno, tende a diventare Normal man mano che si incrementa b e



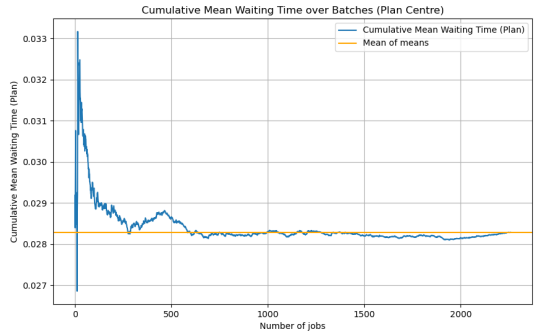
(a) Cumulative response time for base system (Monitor Centre)



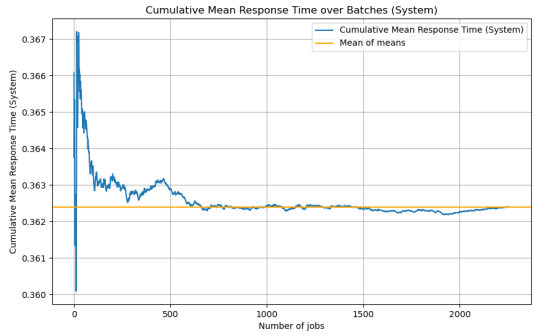
(b) Cumulative waiting time for base system (Monitor Centre)



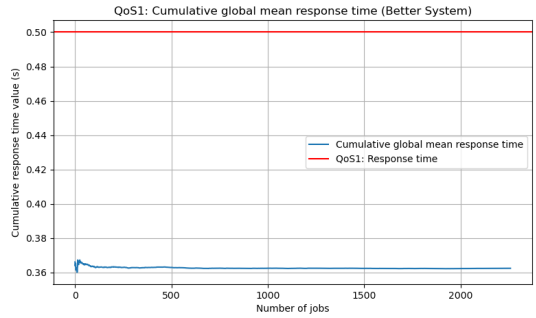
(c) Cumulative response time for base system (Plan Centre)



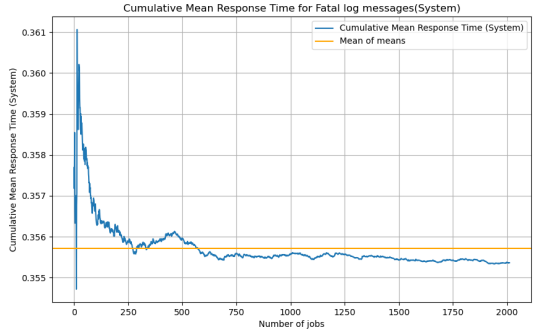
(d) Cumulative waiting time for base system (Plan Centre)



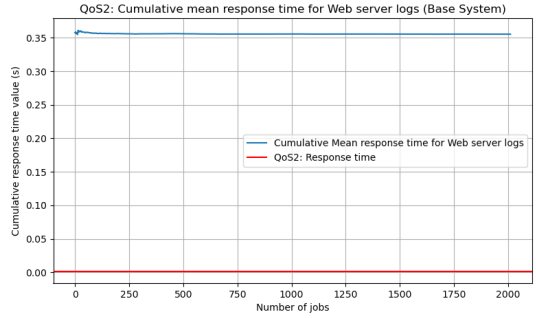
(e) Cumulative response time for base system (System)



(f) QoS1: Cumulative response time for base system (System)



(g) Cumulative response time for Fatal jobs base system (System)



(h) QoS2: Cumulative response time for Fatal jobs base system (System)

Fig. 8: Experiments for base system

l'autocorrelazione è ridotta, diventando prossima allo zero;
- fissato il numero di jobs, b non può essere troppo grande, altrimenti k risulterà troppo piccolo, producendo un largo intervallo di stima. [1]

Con riferimento alla linea guida di Banks, Carson, Nelson, and Nicol (2001, page 438), proposta dal libro di testo [1], è stato adoperato il programma acs.py per identificare la coppia (b, k) = (128, 16), affinchè la *lag-one-autocorrelation* fra le batch means fosse inferiore a 0.2. Per queto motivo i grafici degli esperimenti della stazionarietà mostrano fino ad un numero di jobs pari a $n = k \cdot b = 2048$.

XII. Conclusioni

Grazie agli esperimenti condotti è evidente che il primo obiettivo risulti abbondantemente soddisfatto dal sistema in esame, con un margine di quasi 0.2 secondi dal QoS imposto; per quanto riguarda invece il secondo requisito di qualità, è altrettanto evidente che il sistema non riesca a garantire con le risorse a disposizione al momento una percentuale di uptime ottimale.

Rispetto alla realtà, stiamo sottoponendo l'FMS ad un carico di richieste eccessivo rispetto all'e-commerce di una piccola azienda, per cui sin dall'origine è pensato e progettato. Nonostante ciò, la violazione del secondo vincolo è la motivazione che ci fa considerare il seguente modello migliorativo, affinché, senza l'adozione di scaling orizzontale e/o verticale si possa ottenere una più elevata disponibilità del sistema in termini di uptime.

XIII. Modello migliorativo

Il miglioramento è stato pensato per valutare se e quanto è possibile migliorare la disponibilità del sistema, nel continuo rispetto dei QoS imposti, nel caso in cui le richieste al sistema non vengano più gestite per ordine di arrivo, senza tuttavia apportare modifiche all'architettura attuale, ovvero senza ricorrere a soluzioni quali scaling verticale e/o orizzontale.

1) Obiettivi miglioramento: Nel modello migliorativo abbiamo mantenuto gli stessi obiettivi del modello base, con particolare attenzione volta al secondo requisito al fine di soddisfarlo più abbondantemente.

2) Modello Concettuale: Il modello migliorativo prevede gli stessi utenti del modello base, condividendone gli stessi stati ed eventi, variando però nella gestione delle code. L'introduzione di code di priorità senza prelazione per lo scheduling al centro di Analisi&Pianificazione consente di favorire un particolare tipo di richiesta (FATAL: log del server web), in modo tale da cercare di soddisfare entrambi gli obiettivi imposti. L'assenza di prelazione persiste in quanto le query di Grafana e AlertManager devono essere eseguite per intero e senza interruzioni per motivi di consistenza.

3) Modello delle Specifiche: Per quanto riguarda il modello delle specifiche, vengono utilizzati gli stessi dati ottenuti in precedenza, con gli stessi tassi di arrivo, di servizio; le probabilità di routing verso il centro di monitoraggio restano invariate, mentre nel caso del centro analisi&pianificazione

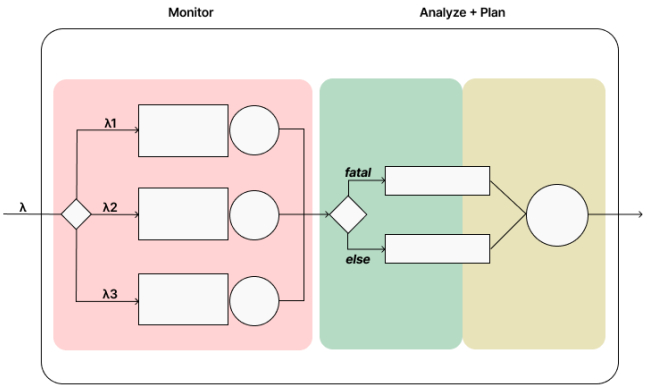


Fig. 9: Modello concettuale miglioramento del FMS

troviamo uno scheduling astratto senza prelazione con due classi di priorità, per cui alle richieste di tipo *fatal* (circa il 12% del totale) viene assegnata la massima priorità.

4) Modello computazionale: Anche in questo caso le uniche differenze significative sulle entità modellate si riscontrano sul centro analisi&pianificazione, oltre che sul programma principale per l'esecuzione delle simulazione.

```
class PlanningCentre:
    number = 0                  # time integrated number in the node */
    queue = [[], []]            # in the queues                      */
    service = []                # in service                         */
    departed = 0                # number served                      */

    AnastasiaBrinati
    def __init__(self) -> None:...

    1 usage   AnastasiaBrinati
    def get_events(self) -> list:

        planning_events = []
        # one arrival to first queue              */
        arrival1 = event.Event()  # arrival
        arrival1.t = START
        arrival1.x = OFF
        planning_events.append(arrival1)

        # one arrival to second queue              */
        arrival2 = event.Event()  # arrival
        arrival2.t = START
        arrival2.x = OFF
        planning_events.append(arrival2)

        # departure from node
        departure = event.Event()  # departure
        departure.t = START
        departure.x = OFF
        planning_events.append(departure)

        return planning_events
```

Fig. 10: Modello computazionale centro di analisi&pianificazione migliorato

5) Verifica: La fase di verifica è stata eseguita come in precedenza andando a controllare che l'implementazione risultasse corretta per il nuovo centro di analisi&pianificazione.

$\lambda = 1.446$ job/sec, $\quad \mathbb{E}[S] = 0.3$ sec

$\rho = \lambda \cdot \mathbb{E}[S] = 0.4338$

$p_1 = 0.12$

$\lambda_1 = p_1 \cdot \lambda, \quad \lambda_1 = 0.17352$ job/sec

$\rho_1 = \lambda_1 \cdot \mathbb{E}[S], \quad \rho_1 = 0.095436$

$$\mathbb{E}[T_{q_1}] = \frac{\lambda \cdot \mathbb{E}^2[S]}{1-\rho_1} = 0.1338704171 \text{ sec}$$

$$\mathbb{E}[T_{q_2}] = \frac{\lambda \cdot \mathbb{E}^2[S]}{(1-\rho_1)\cdot(1-\rho)} = 0.2540982287 \text{ sec}$$

$\mathbb{E}[T_q] = p_1 \cdot \mathbb{E}[T_{q_1}] + (1-p_1) \cdot \mathbb{E}[T_{q_2}] = 0.229848$ sec

$\mathbb{E}[T_{S_1}] = \mathbb{E}[S] + \mathbb{E}[T_{q_1}] = 0.4438704171$ sec

$\mathbb{E}[T_S] = \mathbb{E}[S] + \mathbb{E}[T_q] = 0.529848$ sec.

Formula	Analytical Result	Empirical Result	Confidence Interval
$\mathbb{E}[T_{q_1}]$	0.1338704171	0.13723297021995143	$+/-$ 0.0010000973250875321
$\mathbb{E}[T_{q_2}]$	0.2540982287	0.24150748660368507	$+/-$ 0.0015364345493729275
$\mathbb{E}[T_{S_1}]$	0.4438704171	0.4376085216695721	$+/-$ 0.001532360713959895
$\mathbb{E}[T_S]$	0.529848	0.5295286125955603	$+/-$ 0.001511642964258574
ρ	0.4338	0.44049757468220496	$+/-$ 0.0007166431145017131

TABLE V: Tabella di verifica per il confronto dei risultati analitici ed empirici del centro di analisi & pianificazione.

In questo caso, oltre ai precedenti controlli di consistenza, è stato anche verificato che il centro di monitoraggio rimanesse invariato.

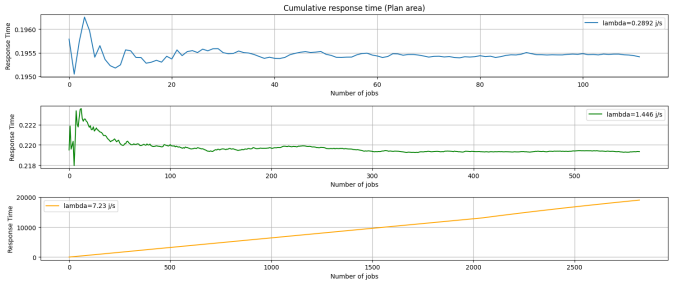
```
Monitor Centre
E[Tq] = 0.5480248044018853 +/- 0.0041935296275504436
E[Ts] = 1.0978848095708016 +/- 0.0045155655381576862
E[Ts1] = 0.6090930284411377 +/- 0.0024814535490544193
rho1 = 0.09596328087193738 +/- 0.0005658879170999494
rho2 = 0.12671301217861775 +/- 0.000688077149394328
rho3 = 0.5892615152411455 +/- 0.00308201346834734 6
```

Fig. 11: Risultati verifica sistema migliorativo

6) Validazione: La fase di validazione è stata svolta seguendo lo stesso schema adottato per il modello precedente: non disponendo di dati relativi al sistema in esame con cui effettuare un confronto, abbiamo di nuovo valutato il comportamento delle metriche al variare del tasso di arrivo. Di nuovo, a parità di configurazione, un'utilizzazione più elevata implica un peggioramento nelle performance, e viceversa. I grafici di seguito riportano la media cumulativa del tempo di risposta per la Plan area al variare di λ.

7) Design degli esperimenti:

- **Simulazione ad orizzonte finito.** Come spiegato in precedenza, dato il nostro caso di studio la simulazione ad orizzonte finito non è rappresentativa della realtà, dal momento che un FMS è costantemente operativo: questo risulta ovviamente vero anche per il modello migliorativo. Analogamente al modello base, la simulazione ad orizzonte finito è stata effettuata con la tecnica della *replicazione*, scegliendo un numero pari ad $n = 20$ replicazioni, il seed iniziale scelto è quello di default (123456789) e il tempo di simulazione è stato posto a 2; è bene sottolineare che questa finestra temporale non ha alcun riscontro con la realtà del caso di studio.



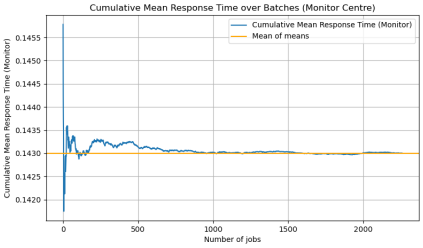
Formula	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.00848381170632739	$+/-$ 0.0001668062862291411
$\mathbb{E}[T_{S_1}]$	0.13621629451514936	$+/-$ 0.0001413429145800615
$\mathbb{E}[T_S]$	0.14300242067438107	$+/-$ 0.0001668062862289719
ρ_1	0.02350863413345103	$+/-$ 0.00029793455082677
ρ_2	0.0312804359207796	$+/-$ 0.00031439179360743895
ρ_3	0.13983502640712936	$+/-$ 0.0006961722520697222

TABLE VI: Tabella risultati centro di monitoraggio con simulazione ad orizzonte finito con n=20 replicazioni (sistema migliorativo)

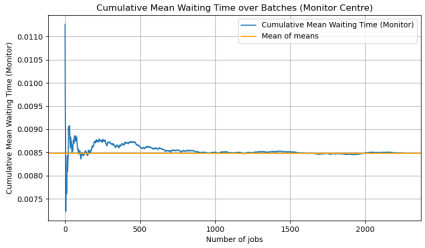
Formula	Empirical Result	Confidence Interval
$\mathbb{E}[T_q]$	0.02827598108285837	$+/-$ 0.0004572327957054056 6
$\mathbb{E}[T_{q_1}]$	0.025808794670697417	$+/-$ 0.0007191160526872091
$\mathbb{E}[T_{q_2}]$	0.028614765320999342	$+/-$ 0.0004905825268712809
$\mathbb{E}[T_S]$	0.2193861084938665	$+/-$ 0.0004572327957051367
ρ	0.27641984405618525	$+/-$ 0.0013581087132465335

TABLE VII: Tabella risultati centro di analisi&pianificazione con simulazione ad orizzonte finito con n=20 replicazioni (sistema migliorativo)

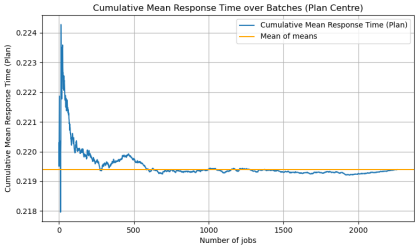
- **Simulazione ad orizzonte infinito.** *Come scegliere (b, k)?* Con riferimento alla linea guida adottata in precedenza, è stato adoperato il programma acs.py per identificare la coppia (b, k) = (128, 16), affinchè la *lag-one-autocorrelation* fra le batch means fosse inferiore a 0.2. Per queto motivo i grafici degli esperimenti della stazionarietà mostrano fino ad un numero di jobs pari a $n = k \cdot b = 2048$.



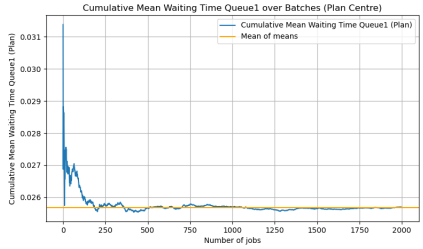
(a) Cumulative response time for better system (Monitor Centre)



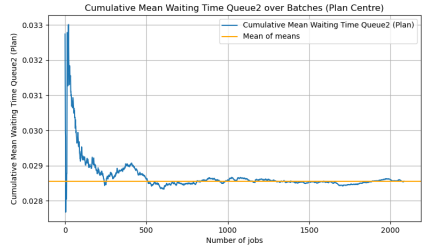
(b) Cumulative waiting time for better system (Monitor Centre)



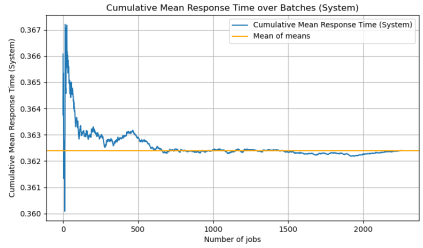
(c) Cumulative response time for better system (Plan Centre)



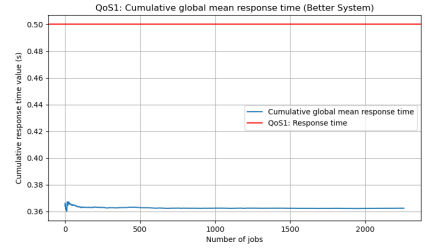
(d) Cumulative waiting time for Fatal log messages for better system (Plan Centre)



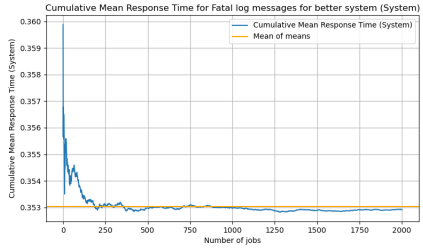
(e) Cumulative waiting time for non-Fatal log messages for better system (Plan Centre)



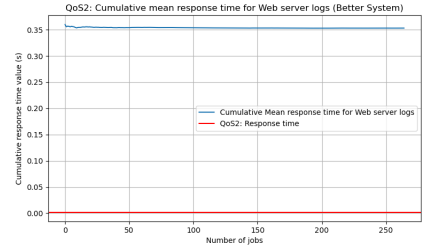
(f) Cumulative response time for better system (System)



(g) QoS1: Cumulative response time for better system (System)



(h) Cumulative response time for Fatal jobs better system (System)



(i) QoS2: Cumulative response time for Fatal jobs better system (System)

XIV. Conclusione

Grazie al miglioramento che impiega lo scheduling riusciamo, senza costi operazionali aggiuntivi, a migliorare di circa 2 millisecondi le prestazioni per i job di tipo *FATAL*. Nonostante siamo ancora lontani dall'obiettivo imposto, è necessario ribadire che i dati utilizzati per il calcolo delle stime sul tasso di arrivo sono relativi ad un sistema di dimensione assai maggiore rispetto all'azienda di e-commerce che stiamo considerando e ci poniamo di gestire con l'FMS descritto. Sebbene il miglioramento sul tempo di risposta per i job di tipo FATAL risulti minimo, in realtà ha un impatto notevole sui costi derivanti dal tempo di inattività: tramite alcuni calcoli, e grazie alle considerazioni riportate nella sezione relativa a perdita e profitto, otteniamo che ogni millisecondo di inattività ha un costo di:

$$\frac{137\$}{60000ms} = 0.002283\$/ms \tag{1}$$

Dunque, per ogni richiesta di tipo FATAL, il costo risparmiato sarebbe pari a:

$$0.002283\$/ms \times 2 = 0.004566\$/req \tag{2}$$

Per ogni giorno, dato il tasso medio di arrivo di richieste di tipo FATAL, il risparmio è dunque:

$$0.004566\$/req \times (125000.166 \times 0.11925) = 68.07\$/day \tag{3}$$

Questa quantità rapportata ad un anno di operatività diventa:

$$68.07 \times 365days = 24485.55\$/year \tag{4}$$

che indica un risparmio notevole nei costi di gestione per una attività di piccole dimensioni.

Una sperimentazione ulteriore può essere fatta con reverse engineering andando a simulare l'FMS in caso di bassa utilizzazione, schiacciando a zero i tempi d'attesa, per considerare quante richieste è in grado di gestire il sistema al massimo, al fine di rispettare il secondo requisito di qualità. Con il seguente tempo di risposta come lower bound, possiamo stabilire che il 90% di uptime è garantito potendo soddisfare, entro il tempo medio di risposta individuato, al massimo la seguente quantità di richieste:

$$6/(\mathbb{E}[T_{S_1}] \cdot 11.925\%) = 154,3862j/h \tag{5}$$

1) Possibili miglioramenti futuri: Al fine di ottenere una migliore rappresentazione dei tempi di servizio, sarebbe interessante sostituire l'adozione della distribuzione lognormale troncata con *the three-parameter Triangular (a, b, c) model*, che suggerito dal libro di testo, è più congeniale quando è noto il range di valori possibili ed il valore più probabile (la moda) [1]; tuttavia non essendo presente nelle librerie fornite richiede di studiarne l'implementazione.

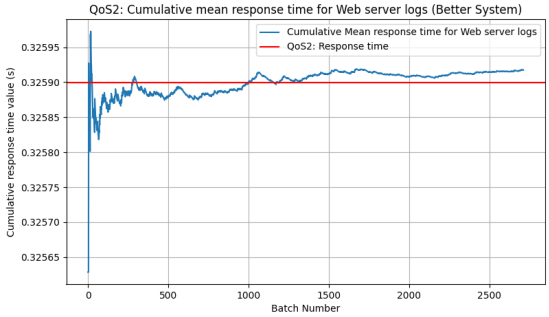


Fig. 12: Tempo di risposta medio per job *FATAL* nel caso migliorativo del FMS, con $\lambda = 0.01446$, (b,k)=(128,16);

References

[1] Leemis, Lawrence M., and Stephen Keith Park. Discrete-event simulation: A first course. Upper Saddle River: Pearson Prentice Hall, 2006.
[2] https://www.macobserver.com/tmo/article/app-store-downtime-cost-apple-25m-in-sales.
[3] https://money.cnn.com/2016/09/07/technology/delta-computer-outage-cost/.
[4] https://www.keliweb.it/billing/knowledgebase/162/Affidabilita-e-percentuali-di-uptimedowntime-di-un-hosting.html.
[5] https://www.atlassian.com/it/incident-management/kpis/cost-of-downtime.
[6] https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.
[7] https://prometheus.io/docs/introduction/overview/.
[8] riferimento sito n richieste 5xx.
[9] https://www.kaggle.com/datasets/shubhampatil1999/synthetic-log-data-of-distributed-system?resource=download
[10] https://www.linkedin.com/pulse/case-study-driving-e-commerce-success-prometheus-grafana-observability-wg3wc?trk=organization_guest_main-feed-card_feed-article-content
[11] https://community.grafana.com/
[12] https://en.wikipedia.org/wiki/Maximum_entropy_probability_distribution
[13] https://2024.debs.org/