SIMULAZIONE DI FLUTTUAZIONI NEL CARICO IN SISTEMI CON AUTOSCALING

Alessandro Finocchi, Emanuele Gentili, Giuseppe Marseglia

Università degli studi di Roma Tor Vergata
Ingegneria Informatica

17 Settembre 2025

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Agenda

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

Oggetto di Studio

- Sistema multi-server dotato di un meccanismo di routing dinamico basato sullo stato del sistema e della sua congestione.
- Le risorse a disposizione nel sistema sono:
  - Web Server: risorse primarie per l'evasione delle richieste.
  - Spike Server: risorsa secondaria per la gestione delle congestioni.
- Routing dinamico basato sullo stato: quando il sistema determina che i Web Server si trovano in una condizione di congestione dovuta ad un improvviso picco di carico, indirizza le richieste in entrata verso lo Spike Server.

Obiettivo dello Studio

- L'obiettivo è il "right-sizing problem": identificare il minor numero di risorse che devono essere allocate per rispettare degli indicatori di QoS.
- Gli indicatori di QoS da rispettare sono calcolati in funzione del 95-esimo percentile del tempo di risposta delle richieste.
- Si vogliono individuare:
  - Il numero di Web Server da allocare.
  - Se lo Spike Server porta un impatto positivo.
  - La politica di routing dinamico per la gestione delle congestioni.

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

Modello concettuale

Il Load Controller:

- Monitora il numero di job nei server e determina il livello di congestione.
- Fa il routing dei job a seconda del livello di congestione.
- Non introduce ritardi o code.

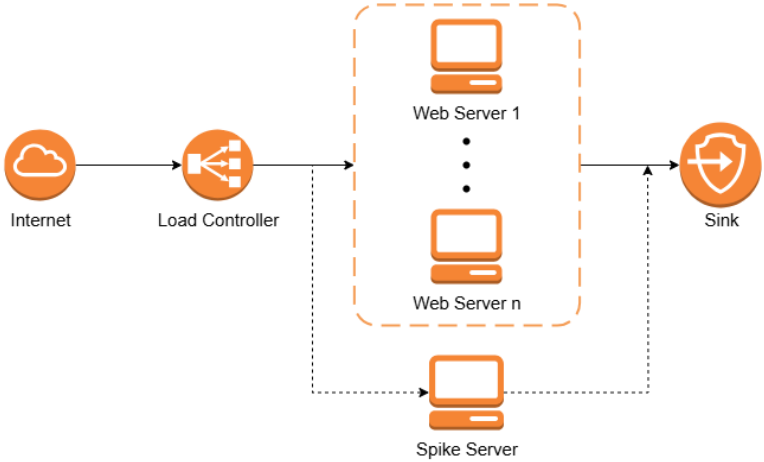


Figure 1: Il modello concettuale del sistema

- **Lo stato del sistema** è composto dallo stato dei server.
  Lo stato di ogni server è caratterizzato da:
  - Una capacità di calcolo, espressa in $\frac{\text{operazioni}}{\text{s}}$.
  - L'insieme delle richieste che il server sta processando. Ciò è particolarmente importante poiché i server seguono una policy Processor Sharing.
- **Gli eventi** possibili nel sistema sono:
  - Arrivo di un job al sistema.
  - Completamento di un job.

- Per simulare **fluttuazioni short-term** i tempi di interarrivo e le richieste dei job sono modellati con v.a. iperesponenziali.
- Per simulare **fluttuazioni long-term** è stato definito uno scenario in cui il tasso di arrivo cambia periodicamente ogni $500\ s$, si susseguono:
  - $400s$ con tasso di arrivi "basso": $3.0\ \frac{\text{arrivi}}{s}$ e $C^2 = 4$.
  - $100s$ con tasso di arrivi "alto": $8.0\ \frac{\text{arrivi}}{s}$ e $C^2 = 4$.

  La media del tasso degli interarrivi risultante su tutto il periodo di $500\ s$ è di $4.0\ \frac{\text{arrivi}}{s}$.
- Le richieste dei job sono modellate con una v.a. iperesponenziale di media 1000 operazioni e $C^2 = 4$.

- I singoli server sono modellati come dei centri $G/H_2/1/PS$: la presenza del routing dinamico cambia la distribuzione degli arrivi ai singoli server, che non si può più dire sia iperesponenziale.

- Sia Web Server che Spike Server sono stati implementati con una capacità pari a $C = 1000 \ \frac{\text{operazioni}}{s}$.
  Quindi la distribuzione del tempo di servizio è iperesponenziale con media pari a $E[S] = \frac{E[Z]}{C} = 1 \ s$ e coefficiente di variazione pari a 4.

- Lo stato di un server che implementa PS può essere descritto tramite:
  - Capacità del server C, in $\frac{\text{operazioni}}{s}$.
  - Lista con i job in corso di servizio $jobs = \{jobs_1, jobs_2, ...\}$, dove $jobs_i$ indica la richiesta rimanente del job i-esimo, espressa in operazioni.
  - $|jobs|$ indica il numero di job attualmente in corso di servizio.
- In PS è possibile approssimare l'esecuzione come se la capacità complessiva del server venisse equamente spartita tra tutti i job in servizio: $\frac{C}{|jobs|}$

È possibile determinare come lo stato evolve fino al successivo arrivo o completamento, assumendo di conoscere gli istanti di arrivo al server, con il seguente metodo:

1. Determinare il prossimo completamento.
   Il primo job a completare sarà quello con la richiesta rimanente minore e completerà dopo un intervallo di lunghezza:

$$c_{\text{next}} = \min_i \frac{job_i}{\frac{C}{|jobs|}}$$

2. Determinare il prossimo arrivo.
   Indicato con a_{next}.

3. Determinare l'avanzamento.
   È necessario procedere per step, poiché al cambiamento di $|jobs|$ cambia la capacità nominale assegnata ad ogni job.

$$\text{advance} = \min(c_{\text{next}}, a_{\text{next}}).$$

4. Aggiornare le richieste rimanenti dei job che erano attivi.
   Queste diminuiscono della capacità nominale moltiplicata per il tempo in cui questa viene applicata:

$$\forall i : job_i = job_i - \frac{C}{|jobs|} \cdot \text{advance}$$

   Al più un job potrà avere richiesta rimanente pari a 0 operazioni, e questo può accadere se e solo se advance $= c_{\text{next}}$.

- SI rappresenta il numero totale di job nel sistema, ed è usato come indicatore di congestione.
- $SI^{\max}$ rappresenta una soglia impostabile sull'indicatore SI.
  m è il numero di Web Server.
- Ci sono tre fasce di intensità:
  - **Bassa intensità di carico**: $SI < m \cdot SI^{\max}$
    I job vengono assegnati ai Web Server, secondo Least Used.
  - **Media intensità di carico**: $m \cdot SI^{\max} \leq SI < (m+1) \cdot SI^{\max}$
    I job vengono assegnati allo Spike Server.
  - **Alta intensità di carico**: $(m+1) \cdot SI^{\max} \leq SI$.
    I job possono essere assegnato sia ai Web Server sia allo Spike Server, secondo Least Used.

Lo SLO è stato definito *(da noi)* sul tempo di risposta dei job con la seguente **euristica**:

1. Si calcola il 95-esimo percentile del tempo di servizio. Questo è un lower bound per il 95-esimo percentile del tempo di risposta.

$$C^2 = \frac{1}{2p(1-p)} - 1 \implies p = \frac{1}{2}\left(1 + \sqrt{\frac{C^2+1}{C^2-1}}\right)$$

$$\lambda_1 = \lambda \cdot 2p, \quad \lambda_2 = \lambda \cdot 2(1-p)$$

$$\Downarrow$$

$$F(x) = \int_0^x p\lambda_1 e^{-\lambda_1 t} + (1-p)\lambda_2 e^{-\lambda_2 t} dt = 0.95$$

Per un iperesponenziale con $E[S] = 1, C^2 = 4$, il valore calcolato è $S_{95} = 3.714886\ s$.

2. Utilizzando la formula di Kingman si approssima il tempo in coda che un job sperimenterebbe se il sistema fosse $G/G/1$.

$$\begin{aligned} E[W_q] &\approx \frac{\rho}{1-\rho} \cdot \frac{c_a^2 + c_s^2}{2} \cdot E[S] \\ &\Downarrow \rho = 0.6, c_a^2 = c_s^2 = 4, E[S] = 1 \\ &\approx \frac{0.6}{1-0.4} \cdot \frac{4+4}{2} \cdot 1 = 6 \ s \end{aligned}$$

3. Si fissa SLO_{95} come la somma del percentile del tempo di servizio e del tempo medio in coda:

$$\text{SLO}_{95} = S_{95} + E[W_q] = 9.714886 \ s$$

Codice dell'avanzamento degli eventi e della gestione di un completamento.

`Calendar` è implementata tramite una `Map`. Ciò sottolinea come solo il prossimo evento venga mantenuto.

```
while (continueSimulating(s)) {
    /* Compute the next event time */
    Event nextEvent = calendar.nextEvent();
    nextEvent.process(s, visitor);
}
...
public void visit(SystemState s, CompletionEvent event)  {
    /* Get the current clock and the one of this arrival */
    double startTs = s.getCurrent();
    double endTs = event.getTimestamp();
    /* Advance job execution */
    servers.computeJobsAdvancement(startTs, endTs, true);
    /* Generate next completion */
    double nextCompletionTs = servers.activeJobExists() ?
        servers.computeNextCompletionTs(endTs) : INFINITY;
    Event nextCompletion = new CompletionEvent(nextCompletionTs);
    s.addEvent(nextCompletion);
    /* Update the current system clock */
    s.setCurrent(endTs);
}
```

Le statistiche dei server sono raccolte come indicato dall'analisi operazionale.

```
public void updateServerStats(
    double startTs, double endTs,
    double jobNum, Double completedJobResponseTime) {
    /* Check if there has been a completion */
    boolean isCompletion = completedJobResponseTime != null;
    /* Update statistics */
    if(jobNum > 0) {
        this.nodeSum    += (endTs - startTs) * jobNum;
        this.serviceSum += (endTs - startTs);
    }
    if (isCompletion){
        this.completedJobs++;
        /* Update SLO indicator */
        if (completedJobResponseTime <= RESPONSE_TIME_95PERC_SLO)
            this.jobRespecting95percSLO++;
        /* Update mean response time */
        meanRT += (completedJobResponseTime - meanRT) / completedJobs;
    }
}
```

La stessa configurazione può essere ripetuta più volte.

Viene fatta la `plantSeed` solo alla prima ripetizione.

Per ripetibilità vengono salvati:

1. I parametri della configurazione.
2. I seed di tutte le ripetizioni.

```
RunConfiguration c;
for (int i = 0; i < REPEAT_CONFIGURATION; i++) {
    run(c, i);
    log(c, i);
}
...
public void run(RunConfiguration c, int repetiton){
    /* Plant the seeds only if the first
     * running the new configuration */
    if (repetition == 0) {
        R.plantSeeds(SEED);
    }
    ...
    /* Log to CSV the initial seed for
     * each stream for replayability */
    for (int stream = 0; stream < TOTAL_STREAMS; stream++) {
        R.selectStream(stream);
        // actual code is more complex
        saveStreamSeed(stream, R.getSeed());
    }
    ...
}
```

Modello Computazionale

Generazione v.a. e multistream

```
public Exponential(Rngs r,
    double mean,
    int stream) {
    ....
}
```

```
public double gen() {
    // actual code is more complex
    r.selectStream(stream);
    return -this.mean
        * Math.log(1.0 - this.r.random());
}
```

```
public CHyperExponential(Rngs r,
    double variationCoefficient,
    double mean,
    int stream1, int stream2, int stream3) {
    ...
    this.exp1 = new Exponential(r, stream1, mean1);
    this.exp2 = new Exponential(r, stream2, mean2);
    this.bernoulli = new Bernoulli(r, stream3, p);
}
```

```
public double gen() {
    double x = this.bernoulli.gen();
    if (x == 1.0) {
        return this.exp1.gen();
    } else {
        return this.exp2.gen();
    }
}
```

Media e varianza sono aggiornate online. In particolare per la varianza è stato usato l'**algoritmo di Welford**.

Gli intervalli di confidenza sono stati calcolati con la `Student`, generata dalla classe `Rvms()` della stessa libreria del PRNG.

```
public void printIntervalEstimation(...) {
    double u, t, w;
    /* compute variance */
    double responseTimeS =
        Math.sqrt(this.responseTimeV / this.currBatch);
    double confidence = 1 - STATS_CONFIDENCE_ALPHA;
    /* interval parameter  */
    u = 1.0 - 0.5 * STATS_CONFIDENCE_ALPHA;
    /* critical value of t */
    Rvms rvms = new Rvms();
    t = rvms.idfStudent(this.currBatch - 1, u);
    /* interval half width */
    w = t * responseTimeS / Math.sqrt(this.currBatch - 1);

    System.out.print(
        "The expected value of Response Time is in the interval "
         + responseTimeX + " +/- " + w
         + " with confidence " + confidence
    );
}
```

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

Verifica

- Domanda: i risultati ottenuti dal nostro sistema sono coerenti con i risultati analitici?

$$\rho = \frac{\lambda}{\mu}, \quad E(T_S) = \frac{1}{\mu - \lambda}, \quad E(N_S) = \lambda \cdot E(T_S) \quad \Rightarrow$$

Table 1: Risultati teorici

λ	μ	ρ	$\mathbf{E(T_S)}$ (s)	$\mathbf{E(N_S)}$
0.6	1	0.6	2.5	1.5
0.8	1	0.8	5	4

Table 2: Risultati con intervalli di confidenza al 95%

Esp.	Distr(λ)	Distr(μ)	λ	μ	ρ	$\mathbf{E(T_S)}$ (s)	$\mathbf{E(N_S)}$
ver-b-01	exp	exp	0.6	1	0.6021 $\pm$ 0.00805	2.5108 $\pm$ 0.0981	1.512 $\pm$ 0.0660
ver-b-02	exp	exp	0.8	1	0.8028 $\pm$ 0.01071	5.0348 $\pm$ 0.4006	4.0449 $\pm$ 0.3454
ver-b-03	exp	h2	0.6	1	0.5948 $\pm$ 0.01279	2.4619 $\pm$ 0.1517	1.4834 $\pm$ 0.0986
ver-b-04	exp	h2	0.8	1	0.7930 $\pm$ 0.01630	4.5818 $\pm$ 0.4957	3.6809 $\pm$ 0.4154

Verifica

- Domanda: le variabili aleatorie sono generate correttamente?
  - È stato creato un pacchetto di test che confronta media e varianza realmente generate con i risultati teorici.
- Domanda: il clock della simulazione è monotono crescente?
  - Il controllo è fatto nel codice tramite tramite `assert`.

Validazione

- Domanda: il sistema creato è coerente con i risultati ottenuti dallo studio originale?

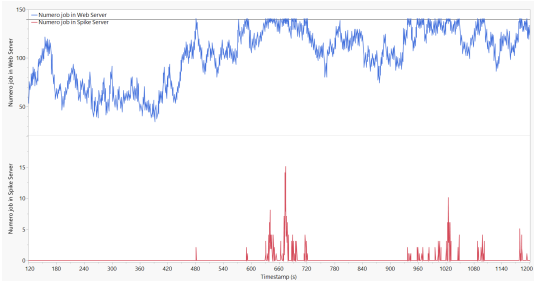


Figure 2: Risultati sistema creato

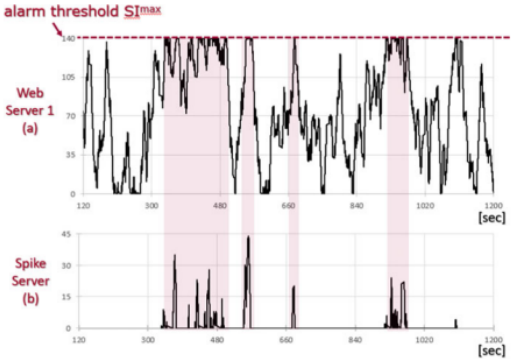


Figure 3: Risultati studio originale

Validazione

- Domanda: aumentare il numero di server migliora le performance?

Table 3: Esperimenti validazione

Esperimento	Distr(λ)	Distr(μ)	λ	μ	Num. WS
ver-b-01	exp	exp	0.6	1	1
val-b-02	exp	exp	0.6	1	2

Table 4: Intervalli di confidenza al 95%

Esperimento	ρ	$\mathbf{E}(\mathbf{T_S})$ (s)	$\mathbf{E}(\mathbf{N_S})$
ver-b-01	0.602121 $\pm$ 0.008058	2.510876 $\pm$ 0.098163	1.5129 $\pm$ 0.066007
val-b-02	0.604531 $\pm$ 0.005955	1.142166 $\pm$ 0.012234	0.689682 $\pm$ 0.009279

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

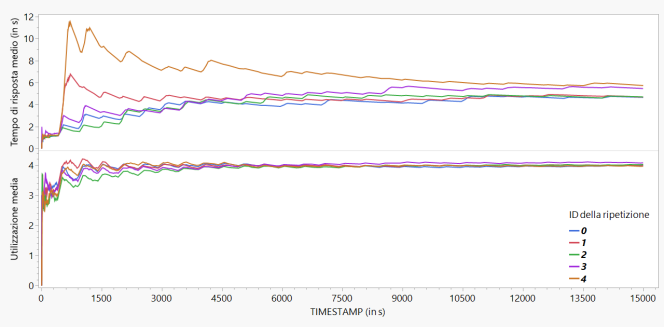
5 Studio dello Stazionario

Studio del Transitorio

- Lo studio del transitorio è stato eseguito con la tecnica delle **Replicazioni**, usando 64 replications, e intervalli di confidenza al 95%.
- Si è dimostrato che il sistema **converge** se la capacità massima allocabile per secondo è maggiore della richiesta per secondo dei job.

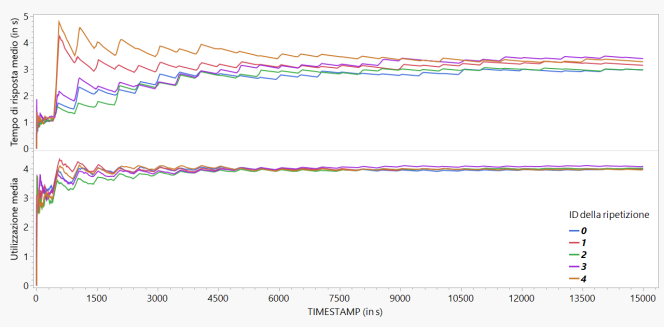
- Versione 1 del routing dinamico.
- 6 Web Server con

$$SI^{\max} = 5.0$$



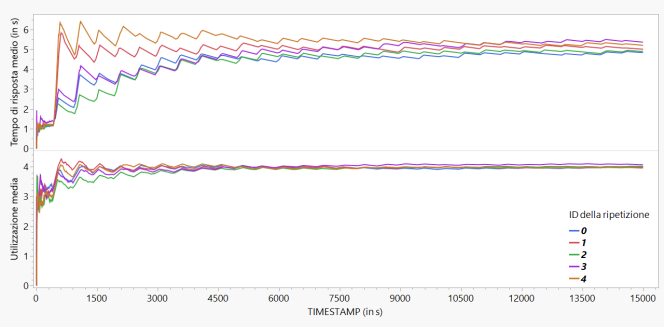
- Versione 2 del routing dinamico.
- 6 Web Server con

$$SI^{\max} = 0.4$$



- Versione 2 del routing dinamico.
- 6 Web Server con

$$SI^{\max} = 2$$



Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

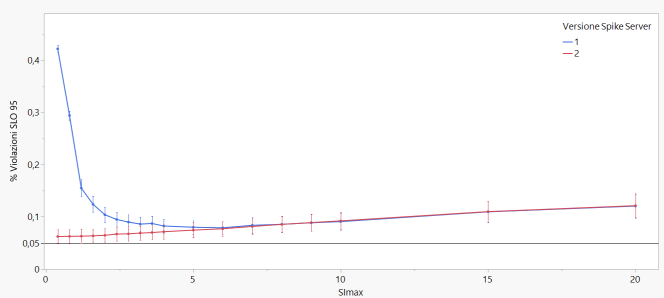
4 Studio del Transitorio

5 Studio dello Stazionario

Studio dello Stazionario

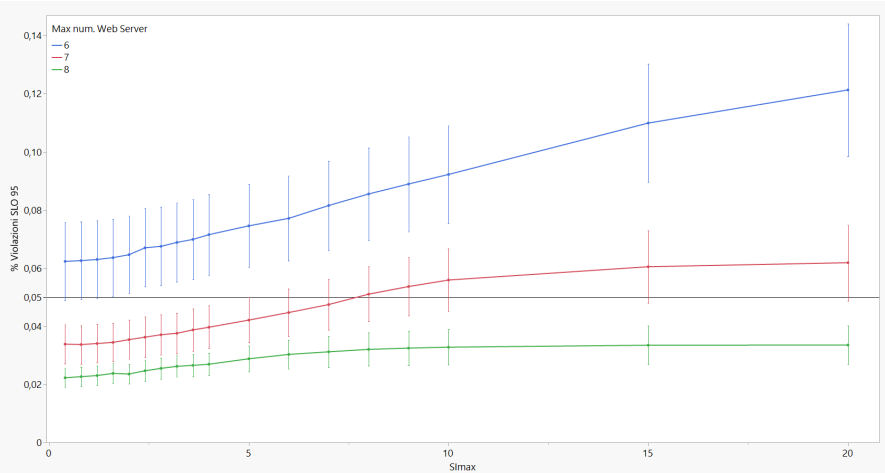
- Gli esperimenti ad orizzonte infinito sono stati eseguiti con la tecnica dei **Batch Means**, usando 128 batch di dimensione 512, e intervalli di confidenza al 95%.
- 3 gruppi di esperimenti.
  - Gruppo 1: ricerca miglior soluzione di routing dinamico.
  - Gruppo 2: ricerca della migliore configurazione.
  - Gruppo 3: necessità di sovra-dimensionare il sistema.

- La versione 1 dello Spike Server ha una grave instabilità per le soglie più piccole.
- Visto il comportamento in generale migliore della versione 2, si sceglie di usare questa d'ora in avanti.



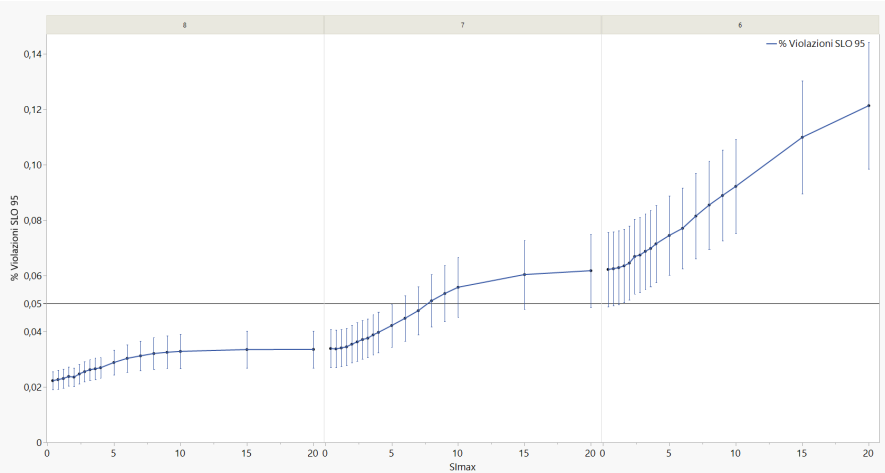
Studio dello Stazionario

Gruppo di esperimenti 2



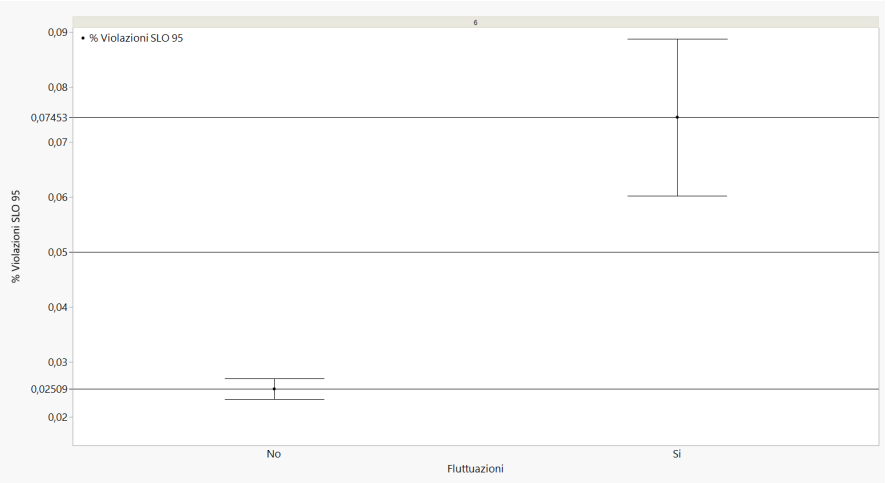
Studio dello Stazionario

Gruppo di esperimenti 2



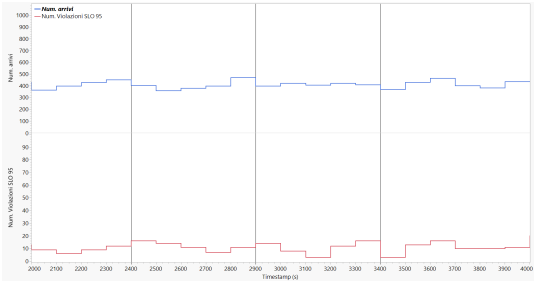
Studio dello Stazionario

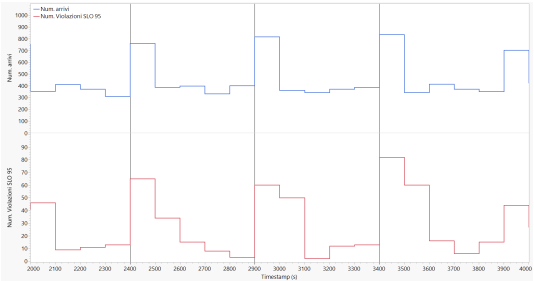
Gruppo di esperimenti 3



Studio dello Stazionario

Gruppo di esperimenti 3





Agenda

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

6 Conclusioni

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

6 Conclusioni

Oggetto di Studio

- Quando le variazioni di carico sono persistenti nel tempo (long-term) la soluzione dello Spike Server non è più efficace.
  - Necessità di sovra-dimensionare il sistema.
  - Sottoutilizzo delle risorse allocate nei periodi di bassa intensità di traffico.

Obiettivo dello Studio

- La soluzione proposta implementa una strategia di **scaling orizzontale** che permette di aumentare o ridurre dinamicamente il numero di Web Server attivi in base al carico effettivo.
- Vengono monitorati degli **indicatori di scaling** per prendere le decisioni di (de)allocazione delle risorse.
- L'obiettivo è analizzare quale metrica sia più adatta e affidabile per guidare le politiche di scaling orizzontale, bilanciando al meglio l'utilizzo delle risorse e la qualità del servizio offerto.

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

6 Conclusioni

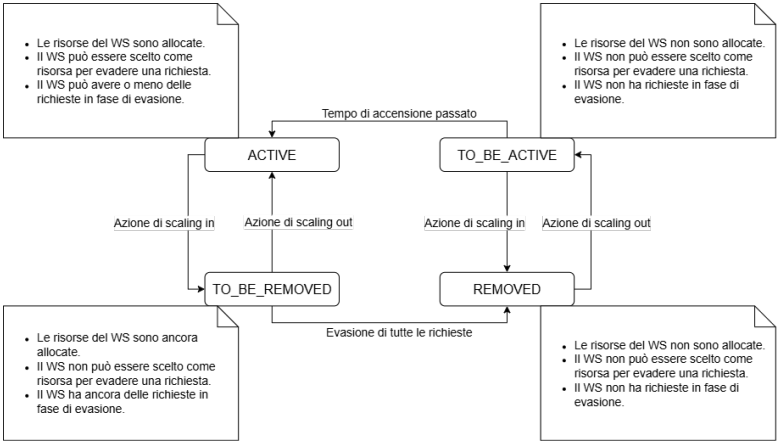


Figure 1: State diagram dei WS nel modello migliorativo

- Il **Load controller** è esteso per prendere le decisioni di scaling in funzione dell'indicatore scelto
- Con lo scaling orizzontale vengono aggiunti anche i relativi **eventi** di:
  - **Scaling out**: si aggiunge il server, ma l'aggiunta non è immediata
  - **Scaling in**: si sceglie il server da rimuovere, il quale verrà rimosso solo una volta che avrà smaltito le richieste rimanenti

- Basato sul **tempo di risposta**.
  - Si calcola la media delle finestre scorrevoli di ciascun Web Server, e il valore dell'indicatore è calcolato aggregando questi risultati in una media finale. L'indicatore viene ricalcolato ad ogni completamento.
- Basato sul **numero di job nel sistema** (sia del Web Server che Spike Server).
  - Il valore dell'indicatore viene calcolato semplicemente come la somma dei job presenti nei Web Server ACTIVE e TO_BE_REMOVED e dei job presenti nello Spike Server. L'indicatore viene ricalcolato ad ogni arrivo e completamento.

1. Utilizzare la soglia impostata sull'indicatore scelto per determinare se c'è la necessità di una azione di scaling.
2. Se viene stabilito di attivare un'azione di scaling, si controlla che l'azione sia possibile
   - scale-out $\Rightarrow$ devono esserci Web Server allocabili (TO_BE_REMOVED o REMOVED).
   - scale-in $\Rightarrow$ devono esserci più di 2 Web Server allocati.
3. Si stabilisce il Web Server su cui applicare l'azione di scaling con le seguenti preferenze
   - scale-out $\Rightarrow$ WS TO_BE_REMOVED in modo da non soffrire dei tempi di accensione.
   - scale-in $\Rightarrow$ WS TO_BE_ACTIVE così da non aspettare che un server si scarichi.

- Se il server scelto per l'operazione di scaling out è completamente spento, cioè nello stato REMOVED, la sua accensione richiederà un certo tempo.
- Il tempo di accensione è stato modellato utilizzando una variabile aleatoria Normale di media $5\ s$ e varianza $0.5\ s^2$.
- Il tempo è modellato per tutti i WS con questa stessa distribuzione. Ci si aspetta infatti una bassa variabilità poichè, assumendo che i Web Server siano omogenei, il tempo di accensione mediamente sarà lo stesso.

- Si aggiorna l'evento di **arrivo** per modificare gli indicatori che vengono utilizzati per prendere le decisioni di scaling, e in caso, schedularle.
- Si aggiorna l'evento di **completamento** per modificare gli indicatori che vengono utilizzati per prendere le decisioni di scaling, e in caso, schedularle.
  - Se un Web Server che si trova nello stato TO_BE_REMOVED risulta essere senza job assegnati, allora questo viene portato allo stato REMOVED.

- Nella nuova `planScaling` è stata configurata una soglia `SCALING_OUT_THRESHOLD` finita.
- In base a qusta soglia di scaling si pianifica un evento di scaling in oppure si fa una richiesta di scaling out.

```
private void planScaling(SystemState s, double endTs, double scalingIndicator){
    IServerInfrastructure servers = s.getServers();
    if (SCALING_OUT_THRESHOLD != INFINITY){
        ...
        int activatedServer = servers.getNumWebServersByState(ServerState.ACTIVE) +
                              servers.getNumWebServersByState(ServerState.TO_BE_ACTIVE);

        scalingOutPossible = activatedServer < MAX_NUM_SERVERS;
        scalingInPossible = servers.getNumWebServersByState(ServerState.ACTIVE) > 1;

        if (SCALING_INDICATOR_TYPE.equals("r0")) {
            scalingOutCondition = scalingIndicator > SCALING_OUT_THRESHOLD * 1.5;
            scalingInCondition = scalingIndicator < SCALING_OUT_THRESHOLD * 0.5;
        } else if (SCALING_INDICATOR_TYPE.equals("jobs")) {
            int expectedServers = (int) (Math.ceil(scalingIndicator / SCALING_OUT_THRESHOLD));
            scalingOutCondition = activatedServer < expectedServers;
            scalingInCondition = activatedServer > expectedServers;
        }

        if (scalingOutCondition && scalingOutPossible)
            s.addEvent(new ScalingOutReqEvent(endTs));
        else if (scalingInCondition && scalingInPossible)
            s.addEvent(new ScalingInEvent(endTs));
    }
}
```

Figure 2: Codice planScaling

- Si aggiunge l'evento **Richiesta di Scaling out**.
  - Modella la necessità di attendere che un server si accenda.
  - Necessaria perchè il tempo di accensione non è deterministico, quindi in seguito ad una decisione di scaling out serve ricalcolare il prossimo server che si accende.
- Si aggiunge l'evento **Scaling out**.
  - Il server in accensione viene effettivamente portato nello stato ACTIVE.
  - Si schedula il prossimo evento di Scaling out (se esiste un server TO_BE_ACTIVE).
- Si aggiunge l'evento **Scaling in**.
  - Il server da spegnere viene portato nello stato TO_BE_REMOVED.
  - Quando avrà evaso tutte le sue richieste verrà portato nello stato REMOVED.

- Si genera la v.a. Normale Standard Z tramite la trasformazione di **Box-Muller** a partire da due v.a. Uniformi.
- Si genera la v.a. Normale desiderata Z' tramite la formula

$$Z' = \mu + Z\sigma$$

- μ e σ sono rispettivamente media e deviazione standard di Z'

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

6 Conclusioni

Validazione

- Domanda: l'autoscaler alloca i server in corrispondenza dei picchi di carico?
- Come si può vedere in figura, quando gli arrivi si intensificano l'autoscaler alloca più risorse, mentre nei periodi di bassa intensità di carico le dealloca.

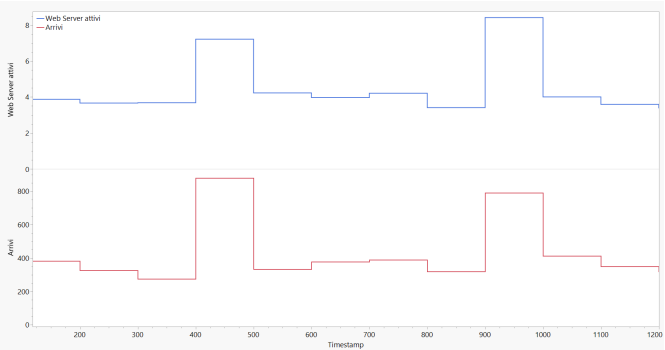


Figure 3: Num. WS attivi in corrispondenza dei picchi di arrivi

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

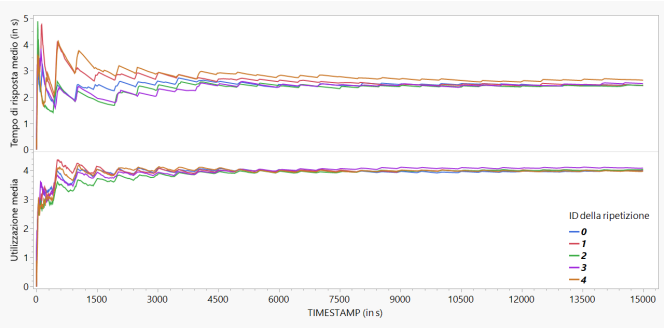
6 Conclusioni

Studio del Transitorio

- Lo studio del transitorio è stato eseguito con la tecnica delle **Replicazioni**, usando 64 replications, e intervalli di confidenza al 95%.
- Si è dimostrato che il sistema **converge** con entrambi gli indicatori di scaling se la capacità massima allocabile per secondo è maggiore della richiesta per secondo dei job.

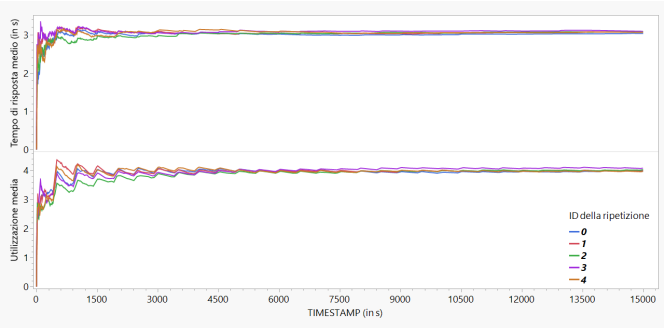
- Scaling basato sul tempo di risposta.

$$\text{scaling}_{thr} = 3\ s$$



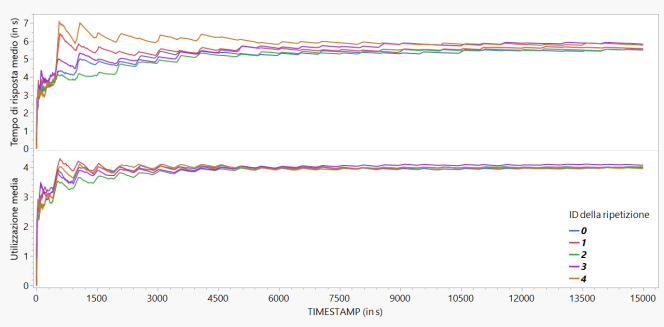
- Scaling basato sul numero di job nel sistema.

$$\text{scaling}_{thr} = 3 \ \frac{\text{job}}{\text{server attivo}}$$



- Scaling basato sul numero di job nel sistema.

$$\text{scaling}_{thr} = 5 \ \frac{\text{job}}{\text{server attivo}}$$



Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

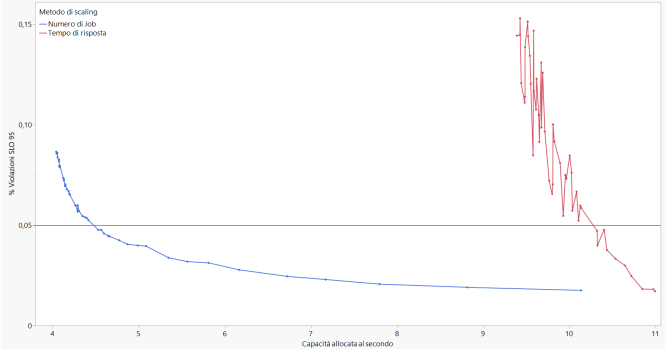
5 Studio dello Stazionario

6 Conclusioni

Studio dello Stazionario

- Gli esperimenti ad orizzonte infinito sono stati eseguiti con la tecnica dei **Batch Means**, usando 128 batch di dimensione 512,, e intervalli di confidenza al 95%.
- 2 gruppi di esperimenti.
  - Gruppo 1: ricerca del miglior indicatore di scaling.
  - Gruppo 2: ricerca della migliore configurazione.

- Fissati gli altri parametri, si sono fatti variare il tipo di indicatore di scaling e i rispettivi valori.
- L'indicatore basato su $E(N_S)$ (in blu) porta i risultati migliori, allocando meno risorse e rispettando di più gli SLO.



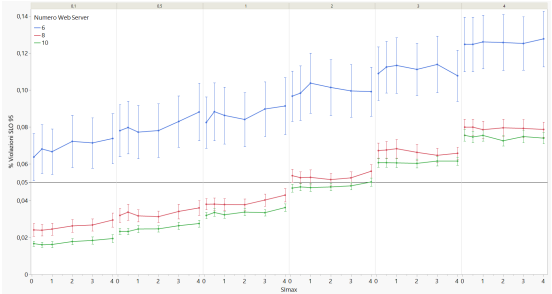


Figure 4: Esperimenti sulla percentuale di violazioni dello SLO 95

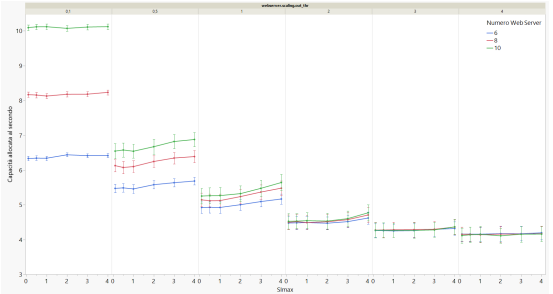


Figure 5: Esperimenti sulla capacità media allocata al secondo

Roadmap

1 Introduzione

2 Modellazione

3 Verifica e Validazione

4 Studio del Transitorio

5 Studio dello Stazionario

6 Conclusioni

Conclusioni

Meccanismo di routing dinamico e indicatore di scaling

- Il **meccanismo di routing dinamico** proposto nello studio originale presentava gravi instabilità per soglie basse
  - Instabilità dovuta alla congestione dello Spike Server
  - La soluzione proposta risolve questo problema, tenendo anche conto dello stato di congestione dello Spike Server
- L'**indicatore di scaling** migliore risulta essere quello basato su $E(N_S)$
  - Fornisce aggiornamenti più regolari
  - In fase di congestione un nuovo campione di $E(T_S)$ può impiegare molto ad arrivare dal momento che corrisponde ad un completamento

- Dagli esperimenti si vede che i valori migliori per la soglia $SI^{\max}$ sono quelli più bassi.
- Questo denota che, a parità di capacità, è sempre meglio avere un Web Server in più che uno Spike Server adibito specificatamente per la gestione delle congestioni.

- La migliore configurazione per entrambi i modelli base e migliorato sono le seguenti

Esp	SI^{max}	scaling$_{thr}$	#WS	% Viol. SLO	Cap. al sec
base-g2-219	0,4	/	7	0,035309 $\pm$ 0,006651	8,0 $\pm$ 0,0
adv-g2-291	0,1	$2\ s$	10	0,046814 $\pm$ 0,002108	4,52705 $\pm$ 0,217157

- Il sistema migliorato permette il rispetto dello SLO 95 usando solo il 56.6% della capacità allocata dal sistema base.

GRAZIE PER L'ATTENZIONE

Link al repository GitHub