Contents

1 Introduzione

Terminologia:

- Sistema: insieme di risorse hardware e software
- Metriche: criteri per confrontare le prestazioni del sistema, qualsiasi esso sia. Ad esempio:
  - il tempo di risposta
  - throughput: "produttività" del sistema per unità di tempo, in base a cosa il sistema produce (es richieste per unità di tempo)
- Workload: richieste che gli utenti sottomettono per un servizio che il sistema fornisce. Esempi:
  - istruzioni CPU che un certo server può ricevere
  - query al DB
- Tecnica: misure, simulazioni e modelli analitici

1.1 Importanza del perfomance modelling

Nonostante viviamo e sperimentiamo la sua importanza quotidianamente, ha una sensibilità poco importante. Ultimi anni:

1. Google down: 14/12/2020, problema per cui per circa 2h e 15 min tutti i servizi sono andati giù su scala mondiale. Didattica a distanza e smart working bloccati.
   Problema è stato il blocco di qualsiasi servizio per l'accesso tramite autenticazione e quindi tutte le applicazioni coinvolte. La capacità ridotta del sistema centrale di gestione delle identità e di autenticazione di Google
2. Cashbak IO Pagopa: 7-10/12/2020. Milioni di donwload e di accessi, fino a 14000/s ed un autenticazione molto lenta, le troppe richieste hanno saturato le porte disponibili per l'accodamento

delle richieste.
Blocco nell'inserimento dei metodi di pagamento, con annesso crollo dei servizi di push (che mette in coda le richieste che arrivano), dovuto alla lentezza dell'autenticazione.
Bottleneck nell'autenticazione e problema nella gestione non appropriata delle richieste

3. Signal: 16/01/2021, aumento improvviso dei download del circa 4200% in una settimana. Primo rallentamento del servizio, seguito da una parziale interruzione. La soluzione è stata di creare una replica del back-end su altri server.

Moltissimi altri casi di questo tipo, oggi i sistemi hanno un livello di complessità molto alta come anche la loro composizione che è più complessa, la loro evoluzione è sempre più rapida. Inoltre, sono sempre più essenziali per il business e questo richiede la necessità di strumenti e tecniche che assistano progettisti, service provider etc... che permettano di capire a pieno il comportamento dei sistemi, in tutte le fasi:

- Progetto ed implementazione
- Dimensionamento
- Vita ed evoluzione del sistema

Tutte le tecniche per la valutazione delle prestazioni consentono anche una visione ed una conoscenza del sistema che il sistema stesso non offre: studiare il comportamento mediante un modello consente di vedere aspetti che non avremo potuto vedere in altro modo.
Tutte le figure che hanno a che fare con sistemi di questo tipo devono avere un background di tecniche di valutazione delle prestazioni.

1.2 Valutazione delle performance

Non è importante solo a livello industriale, ma anche nella ricerca accademica, nel momento in cui si vuole valutare una nuova proposta. Anche in questo caso la modellazione può essere molto utile.

Nell'industria è essenziale per mantenere dei livelli di qualità alti, espressi negli SLA: il service provider deve poter identificare in maniera rapida dove c'è un problema che potrebbe portare al crollo delle performance garantite (per non andare in contro a penali) e di fare un tuning del sistema per ritornare al livello di qualità che doveva essere garantito.
Un buon modello di valutazione delle prestazioni ci da una conoscenza profonda del comportamento del sistema: perché il sistema si comporta in un certo modo e perché lo fa, quali sono i limiti di quel comportamento ed i punti critici nel caso ci fossero problemi e fosse necessario migliorare le performance.

1.2.1 Obiettivi della valutazione delle performance

Alcuni esempi e obiettivi per un sistema:

- Capicity planning: determinare il numero e la taglia dei componenti del sistema
- Tuining del sistema: messa a punto del sistema, quando c'è qualcosa che si evidenzia nel comportamento del sistema che porta al degrado delle performance, bisogna determinare l'ottimo per il valore del parametro
- Bottleneck identification: determinare le performance di un bottleneck, identificarlo per poter capire qual'è la risorsa che saturerà per prima.
- Caratterizzazione del carico: solitamente ben modellate con distribuzioni esponenziali (o al più a fase), quindi che hanno la maggior parte della loro probabilità distribuite su valori piccoli, ovvero con tempi piccoli. Le probabilità di valori grandi sono piuttosto basse. Heavy-tail, la coda della distribuzione che modella i valori grandi non è così trascurabile.
- Previsione del carico (forecasting): tende ad oscillare, può essere critica

Quale può essere l'approccio: cominciare con una visione completa del sistema, degli obiettivi dello studio e dell' applicazione. Il punto di partenza è questo, anche se poi le tecniche non devono essere condizionate da tale visione.
Una volta fatto ciò, ci sono molti approcci ed hanno due casi limite:

- Intuizione ed estrapolazione delle tendenze: richiede alto grado di esperienza e di capacità intuitiva. Rapido e flessibile, ma l'accuratezza dei risultati a cui si arriva va guardato con un certo sospetto, in quanto sono derivanti dall'esperienza
- Valutazione sperimentale delle alternative: sempre possibile, ma costosa in quanto non è detto che sia generalizzata. Un esperimento porta alla conoscenza accurata del sistema sotto determinate assunzioni.
  Accuratezza eccellente, ma la valutazione sperimentale è molto più complessa e poco flessibile.

Entrambe le strade hanno vantaggi e svantaggi, tra questi due approcci estremi si colloca la modellazione, che prende lati positivi di uno e dell'alto ed anche i limiti.

1.3 Modellazione

Un modello è un'astrazione del sistema, la modellazione è il tentativo di distillare dalla quantità enorme di dettagli di quel sistema esattamente quegli aspetti e non di più che sono essenziali al comportamento del sistema rispetto agli obiettivi posti.
Il modello va definito e per fare questo occorrono:

- capacità di astrazione
- parametrizzazione del modello
- valutazione

Rispetto ai pro e contro dei due approcci, la modellazione è:

- più affidabile dell'approccio intuitivo
- meno costoso dell'approccio sperimentale

1.3.1 Tecniche di performance

Le tecniche di performance sono tecniche matematiche e computazionali per analizzare le performance del sistema stocastico (non ha un comportamento puramente deterministico)
Modellare: racchiude tutto il framework concettuale che descrive il sistema, la soluzione di divide in:

- tecniche analitiche
- tecniche simulative: eseguire esperimenti usando l'implementazione del modello
- misure: nel processo di monitoraggio di un sistema si collezionano una quantità di misure che sono molto importanti per capire il comportamento del sistema.

Tutto ciò che verrà derivato dalle due tecniche (simulative e analitiche) andrà verificato.

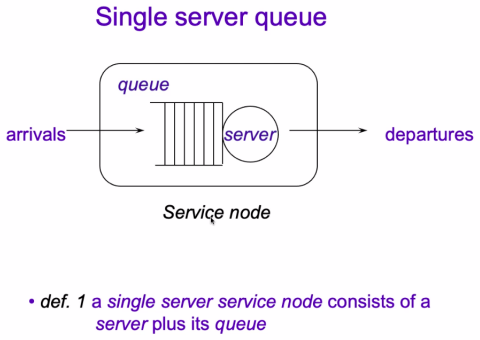
2 Introduzione alla modellazione-teoria delle code

Indipendentemente dall'approccio risolutivo che si sceglie di usare (analitico o simulativo), il modello usato è quello derivato dalla teoria delle code. La teoria delle code è un area della matematica che coinvolge l'analisi probabilistica/stocastica e consente di fare una predizione delle performance (in senso lato) e quindi capire che tipo di algoritmo usare per migliorare le prestazioni.
L'idea è quella di modellare la contesa per l'uso di risorse, quelle più tradizionali erano quelle hardware (CPU, dischi, etc...). Concetto di risorsa si applica a più ampio spettro: può essere la banda, le VM etc... La concorrenza sulle risorse può causare tempi di attesa lunghi, che degradano le prestazioni.

Mediante la teoria delle code è possibile calcolare i tempi di attesa e medi e come è possibile ridurli.

2.1 Single server queue

Modello un servizio per il quale ho un flusso di richiese con una cosa a servente singolo.



Ho due parti:

- server: la risorsa che deve essere assegnata, è singolo perché in questo modello semplice si assume che è possibile servire una singola richiesta
- queue: mantiene tutte le altre richieste che possono arrivare per il servizio. Quando termina il servizio, viene scelta la prima richiesta in coda

Spesso si assume che le richieste per il servizio arrivino ad istanti random: non c'è correlazione tra gli istanti di arrivo delle richieste. Questa caratteristica modella bene la stragrande maggioranza dei processi di arrivo.
Inoltre, si assume che la coda di buffer sia infinita (non ho la linea nera alla fine), ma spesso si è interessati alla loss probability: dato che la coda è finita, voglio conoscere la probabilità che nel momento in cui una richiesta arriva questa venga rifiutata in quanto non può essere messa in attesa. Ad esempio, in comunicazione tale probabilità di perdita è proprio il livello di connettività offerto da una rete.

2.2 Definizioni

La coda è un elemento importante di questa astrazione, è l'area di buffer in cui possono attendere i job che non possono essere serviti. Bisognerà decidere la politica di scelta con cui si serve il job successivo, una volta che ha finito quello corrente.
Disciplina di scheduling è quell'algoritmo secondo il quale si sceglie un job dalla coda:

- FIFO: serve il job arrivato per prima nel sistema: se arriva un job ed il servizio non è disponibile, questo prende la prima posizione in coda. Un secondo job andrà dietro l'ultimo entrato, quindi gli istanti di partenza dal server saranno ordinati nello stesso modo con cui sono stati ordinati gli istanti di arrivo
- LIFO: opposto della FIFO, i job vengono presi a partire dall'ultima posizione della coda
- random
- priority: i job non sono tutti uguali per il sistema, alcuni hanno valore maggiore di altri. Può esserci un criterio di priorità secondo il quale vengono scelti sempre (se ci sono) i job di priorità più alta
- processor sharing: rappresentazione ideale di quello che succede in un sistema time-sharing, in cui ad ogni job è assegnato un quanto di tempo per usare la risorsa. TCP ad esempio divide la banda fra i diversi job che sono attivi in un certo istante di tempo, quindi tale disciplina prevede la condivisione della capacità elaborativa (avendone assegnata $\frac{1}{n}$, se ho n job) come se fossero tutti attivi in simultanea

Si parla di job non-preemptive se un job che ha iniziato il servizio non può essere interrotto. Se invece il servizio è preemptive, allora

questo può essere interrotto per dare il servente ad un job a priorità maggiore (nel caso di algoritmo a priority). Quindi, la processor sharing, avendo questo meccanismo, entra nella classe delle discipline preemptive.
Altra caratteristica è quella del servizio conservativo: potrebbe avere senso, nel caso in cui ci siano conoscenza della caratteristica del flusso di job che arrivano, che nel caso in cui server divenga vuoto questo non prende un job e lo manda in esecuzione perché è a conoscenza del fatto che sta arrivando un flusso prioritario.
Nel caso in cui un server rimanga in attesa senza fare nulla, si parla di server non conservative. Faremo l'ipotesi di serventi conservativi.

2.2.1 Primo esempio: quanto conta lo scheduling

10 job, conosco i 10 istanti di arrivo dei 10 job: 15 47 71 111 123 152 166 226 310 320 e questi job richiedono delle unità di tempo di servizio: 43 36 34 30 38 40 31 29 36 30.
Tempo di risposta di un centro: tempo che passa da quando la richiesta arriva al centro a quando parte. Lo scheduling può avere un effetto enorme sulle prestazioni: calcolo la media dei tempi di servizi (del campione) = 34.7
Calcolo inoltre la varianza = 20.21 Il tempo di risposta medio è 26.70: qual'è la caratteristica che inficia molto negativamente su tale valore? La variabilità: prendo il campione di 10 tempi ed aumento la variabilità sommando e sottraendo 20: 63 16 54 10 18 60 51 9 56 10. La media rimane uguale, ma la varianza aumenta moltissimo, passando da 20.21 a 504.21
Se invece cambio lo scheduling, ordinando i tempi di servizio in modo che arrivino in ordine decrescente di richiesta (o crescente nel caso opposto). Cosa accade: da un punto di vista di media e varianza, non cambia nulla, ma cambia in termini di tempo di risposta. Infatti, nel caso di ordine decrescente (servo dal più piccolo al più grande), i tempi di risposta diminuiscono di molto. Se faccio il contrario, ovvero servo in ordine decrescente, allora ho il caso peggiore: tutti

hanno davanti a se job che occupano per più tempo di quello che chiedono.
L'algoritmo di scheduling non costa nulla, ma il modo in cui si schedula può agire molto sulle prestazioni. La variabilità è sempre un indice negativo a livello di prestazioni

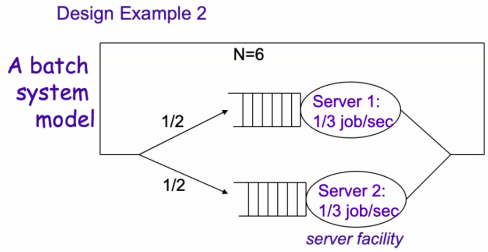
2.3 Altri esempi-applicabilità della modellazione

Ho un rete, conosco i route dei pacchetti, la loro frequenza di arrivo, il tempo di trasmissione e la lunghezza dei collegamenti. Con la teoria delle code, possiamo conoscere:

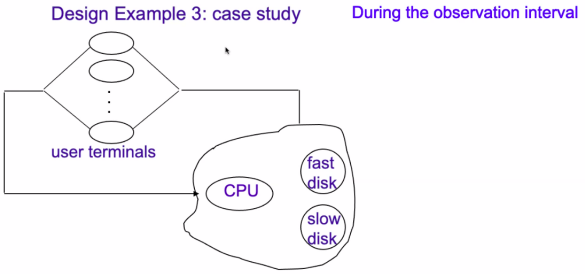
- valor medio del tempo speso nel router i
- la distribuzione dei pacchetti nella coda al router i
- tempo totale medio per poter andare dal punto A al punto B

L'approccio modellistico è anche usato come tool di progettazione: so che un server subirà un carico, dal punto di vista degli arrivi, raddoppiato. Dobbiamo fare in modo che gli utenti non si rendano conto di questo raddoppio, quindi che i tempi rimangano simili. Di quanto va aumentata la capacità operativa della CPU del server per poter mantenere lo stesso tempo di risposta medio?
La soluzione che viene subito in mente è prendere una CPU con velocità doppia, ma in realtà serve meno del doppio se l'obiettivo è quello di mantenere gli stessi tempi di risposta.
È possibile rendersene conto anche analiticamente, questo evidenzia quanto a volte l'intuizione può portare dalla parte sbagliata.
Se sostituissi la CPU con una di potenza doppia, il tempo di risposta sarebbe dimezzato rispetto a quello del giorno prima, quello che succede è che il sistema dal punto di vista del carico che sopporta e della capacità di elaborazione rimane identico, ma come se fosse velocizzato di un fattore 2. Quindi, questo comporta un tempo di risposta che è la metà.
Supponiamo ora di cambiare anche lo scheduling: da FIFO passiamo

a processor sharing. Cambiano le cose? No.
Supponiamo di avere un modello di un sistema batch con due server (carico fisso): ho 6 job che possono andare nel server 1 o 2, che sono identici.



Supponiamo che il server 1 sia potenziato con uno che abbia il doppio della potenza, il tempo di risposta migliora? Ed il thoughput? Il miglioramento è poco e tende al nulla quanto più il numero di job del sistema cresce. Il problema è che nel sistema ho un bottleneck: posso migliorare il server quanto voglio, ma essendo il carico diviso a metà questo pesa nel calcolo della media (e noi guardiamo il calcolo della media). Se invece avessimo meno di 6 job, il miglioramento trascurabile diventa più apprezzabile? Sì, nel caso limite di uno stand alone job calcolerei la media come: 0.5·tempo del server 1 (inverso della frequenza) + 0.5·tempo di risposta del server 2.
Se invece il sistema fosse aperto (tempi di arrivo indipendenti dal completamento del servizio)? Cambia molto, migliorando uno dei due server, l'$\frac{1}{2}$ di arrivi che va nel server migliore ne risente, mentre prima c'era il bottleneck del server meno potente.
Altro esempio tipico, navigazione in Internet, vorrei sapere il tempo di risposta medio per una richiesta. Una situazione di questo tipo è spesso modellata, quando il numero di utenti è illimitato, infinit server; mentre se il numero è limitato sarà un centro ad un certo numero di serventi paralleli. Non ci sono code: quando arriva una richiesta c'è sempre un utente libero che aspetta la risposta.



Osserviamo il sistema per un certo tempo (es T = 900 sec), possiamo sapere per ogni dispositivi:

- busy time: tempo in cui la risorsa è occupata
- numero di completamenti, fatti nella finestra T
- numero di completamenti del sistema
- "think time" medio degli utenti: tempo per quanto aspettano gli utenti prima di lanciare un nuovo comando

Quale modifica è più effettiva per aumentare il throughput globale del sistema? Possiamo:

- prendere CPU con velocità doppia
- fare load balancing dei due dischi
- prendere un secondo disco veloce

Ancora una volta, la risposta è contro-intuitiva, l'approccio che è possibile usare per rispondere è l'analisi operazionale ed è "molto facile". Inoltre, è indipendente da assunzioni sulla distribuzione e non necessita di conoscere l'intera topologia della rete. Occorre però l'osservazione del sistema reale, per poter derivare le 3 quantità introdotte sopra.
Un altro problema molto interessante è il seguente: ho un certo budget, ad esempio in termini di potenza di calcolo ho $4\frac{job}{sec}$. Potrei avere questa potenza con 4 macchine da $1\frac{job}{sec}$ o una da $4\frac{job}{sec}$. Qui, si punta a minimizzare il tempo medio di risposta, ma occorre fare ipotesi.

Partiamo dall'ipotesi che i job non siano preemtible. La soluzione migliore dipende dal carico, in particolare dalla variabilità del carico in termini di taglia del job.
Se c'è una alta variabilità, può accadere che un job piccolo venga bloccato da uno grande, quindi l'alternativa multi-server paralleli offre la possibilità di distribuire il carico di job piccoli in modo che non siano bloccati (ovviamente, tutti i confronti vanno fatti a parità di potenza computazionale). Viceversa, se la variabilità è bassa, conviene il singolo servente.
Se invece assumiamo che la classe sia preemtible, è sempre preferibile la configurazione ad un servente, perché possiamo togliere il job grande e mettere quello piccolo.
Il problema ha un'applicazione enorme, considerato che il server può rappresentare diverse risorse, problema è spesso il trade-off tra comprare una macchina più costosa che consuma di meno da un punto di vista energetico, oppure comprare un certo numero di macchine meno costose ma che dissipano più energia. Un altro problema molto diffuso riguarda il task assignment in una server farm o data center. Ho un dispatcher che si preoccupa di distribuire il workload fra diversi server. Ancora una volta, l'obiettivo del dispatcher può condizionare il tipo di politica adottata: per siti web, il dispatcher fa in modo di mantenere il carico di lavoro bilanciato fra i vari server. Nell'ambito dei super-computer la politica di bilanciamento può non essere utile. Il modello è un certo numero di code ed il dispatcher davanti che deve distribuire fra le diverse code il carico che riceve. Facciamo delle assunzioni:

- server omogenei fra loro
- risorsa singola per ciascun job, ogni richiesta che arriva al dispatcher userà una singola risorsa, sarebbe possibile che una richiesta possa chiedere l'uso di più di una risorsa, ma assumiamo di no.
- scheduling delle code FIFO non-preemptible

Tra le politiche applicabili abbiamo:

- random: ogni job viene assegnato "lanciando una moneta equa"
- round-robin: i job i-esimo va all'host *imodn*, dove n è il numero di host
- shortest-queue: il dispatcher cerca di vedere il carico dell'host, manda il job all'host che ha il numero minimo di richieste
- central-queue: unica coda, appena uno degli host diventa libero va sulla coda prende il primo job che trova
- size-interval-task-assignment: assumiamo di conoscere la size delle richieste (in termini di carico di lavoro richiesto). I job vengono suddivisi in base alle size (si può anche avere una partizione più fine):
  - short
  - medium
  - long

  Si indirizzano i job in base alla taglia
- least-work-left: si sceglie il server col carico minore, ogni job che arriva viene mandato al server dove a quell'istante di tempo c'è il minor carico, dove il carico è la somma delle size che si trovano in coda. Il server con valore minore sarà quello designato per ricevere il job

Le ultime due politiche sono size-based (non è sempre possibile conoscerla), mentre le altre no.
Supponiamo che si voglia scegliere la politica che comporta il tempo di risposta medio: dipende dalla variabilità della size:

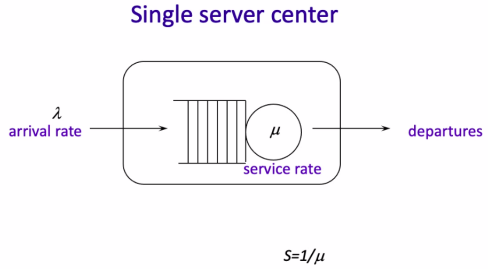
- se poco variabili, la LWL comporta il tempo minimo
- se molto variabili, la SITA comporta il tempo minimo

Quanto sarebbe quindi importante conoscere la size? In realtà, la maggior parte delle politiche usate non richiedono la conoscenza della size e si può dimostrare che in media la LWL è equivalente alla politica della central-queue: in qualche modo nella seconda politica si usa la nozione di size, in quanto il primo server che si libera all'istante di arrivo di un job è quello che aveva meno size.
Un'altra distinzione è fra classi di job preemptible e non: se non è preemptible non bisogna fare load balancing (non porta vantaggi) e bisogna fare altre considerazioni e studi. Il secondo è una open issue e dipende da cosa si vuole fare, se minimizzare il tempo di risposta o minimizzare il tempo di slow-down.
Nell'ipotesi che non vengano fatte assunzioni sulla taglia dei job e che ci siano job non-preemptive, è possibile dimostrare che finché non si mette in mezzo al taglia, è dimostrabile che non cambia nulla. Immagino ora di avere un algoritmo di scheduling FIFO preemptive, ovvero appena arriva un nuovo job ottiene subito il servizio. Per un carico che presenta almeno un minimo di variabilità si ha un miglioramento delle prestazioni enormi. Mentre invece, per un carico molto variabile, si ha un peggioramento delle prestazioni di quasi il doppio.

3 Modelli analitici

3.1 Modelli a risorsa singola

Ho una singola cosa, l'analisi guarda prevalentemente i valori medi (anche se potrei calcolare la distribuzione di probabilità)

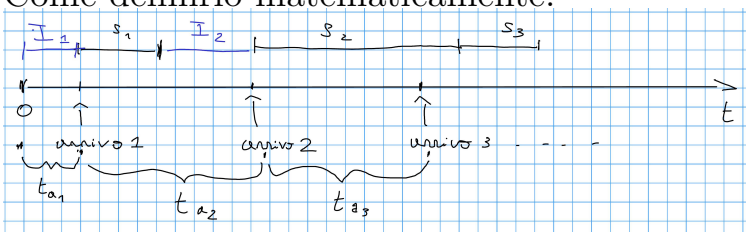


i parametri λ e μ sono valori medi.

Si parte dalla definizione del modello concettuale (è sempre il primo passo che va fatto). Terminologia:

- S è il tempo di servizio ed è pari a $\frac{1}{\mu}$
- T_q è il tempo di coda
- T_s è il tempo nel sistema
- N_s è il numero nel sistema
- N_q è il numero in coda
- U, ρ è il numero nel servizio

Tipicamente, si considera il valore medio di tutte queste grandezze, un'altra misura importante, in presenza di SLA e QoS, è $\mathrm{P}(\{T_s > t\})$, che è la coda della distribuzione, mentre $\mathrm{E(n)}_t$ è il numero medio di job serviti in un intervallo temporale t.
Quanto più λ aumenta, tanto più cresceranno tutte le misure medie. Così come al crescere di μ, che rappresenta la capacità di smaltire il traffico che arriva, le misure decresceranno.
Se rapportiamo $\mathrm{E(n)}_t$ al tempo unitario 1, quindi $\mathrm{E(n)}_1$ (possono essere secondi, minuti, ore etc...), stiamo guardando il throughput.
Altro fattore importante è l'utilizzazione, che ci da il rapporto tra il periodo di occupazione del servizio ed il tempo totale di osservazione.
Come definirlo matematicamente:

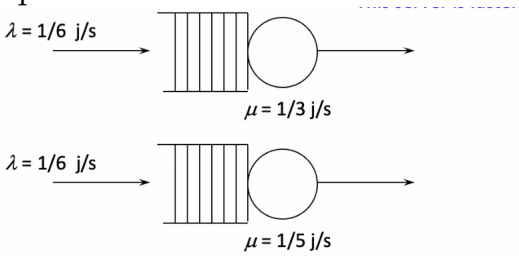


Denotiamo con I gli intervalli in cui il servente è idle (non ha job in servizio), il tempo di occupazione totale sarà dato o da $t - \sum_{i=1}^{n} I_i$ oppure da $\sum_{i=1}^{n} s_i$.

Definiamo l'utilizzazione come $\rho = lim_{t->\inf}\frac{B}{T}$.A questo punto, andiamo a sostituire i valori medi con le quantità per cui si considera il limite ottenendo $\cong \frac{E(\sum_{i=1}^{n} s_i)}{E(\sum_{i=1}^{n} t_{a_i})}$.
Siccome gli s_i sono indipendenti ed identicamente distribuite, così come le t_{a_i}, che sono tra l'altro esponenziali. Al di là della distribuzione, siccome hanno stessa media e stessa distribuzione, otteniamo $\frac{n \cdot E(s)}{n \cdot E(t_a)} = \frac{E(s)}{E(t_a)} = \frac{\lambda}{\mu}$ ed è anche $1 - P(n = 0)$, se n è il numero di job in coda. Quindi, possiamo dire che l'utilizzazione è il rapporto $\frac{frequenza\_di\_arrivo}{frequenza\_di\_servizio}$.
Possiamo anche dire che:

- $E(N_s) = E(N_q) + E(number\_in\_service)$, questo non è altro che $E(N_q) + \rho$

esempio:



By assuming *job flow balance*, the throughput is the same !!
For both systems $X = \lambda = 1/6$ j/s

Il server più veloce è comunque utile, in quanto ci saranno dei tempi di risposta più brevi. Però, il minimizzare il tempo di risposta non è detto che migliori il thorughtput, come avviene in questo caso.
Se i centri sono in equilibrio stazionario, ovvero il flusso che entra è pari al flusso che esce (se $\lambda < \mu$ o $\rho < 1$), allora il thoughput è pari a λ. Se invece μ è maggiore di λ, il throughput è pari a μ, ma questo vuole anche dire che la coda cresce indeterminatamente, perché il centro non riesce a smaltire le richieste in entrata.
Un altra metrica interessante è il tempo medio di servizio, definito in termini di C = della capacità operativa del server ($\frac{op}{s}$) e di Z = domanda media dei job (op): $\frac{E(Z)}{C}$.

3.1.1 Server singolo a buffer finito

Cosa accade nel caso di buffer finito: quando la coda è piena, l'arrivo si perde (c'è tutto un altro tema quando l'arrivo alla coda piena viene mantenuta dove c'è una certa area). Qual è in questo caso il throughput? Non è più λ e cambia anche ρ in quanto non entra λ nella coda, bensì λ'. Dove c'è un buffer finito, c'è sempre l'equilibrio perché più di tanto non entra in coda, il sistema sarà sempre in grado di servire le richieste. Per poter calcolare il throughput, avremo bisogno dei processi di Markov, calcolare la probabilità di avere lo stato vuoto e quello saturo, la probabilità di perdita sarà la probabilità di avere la coda piena.

3.2 Server a coda multipla

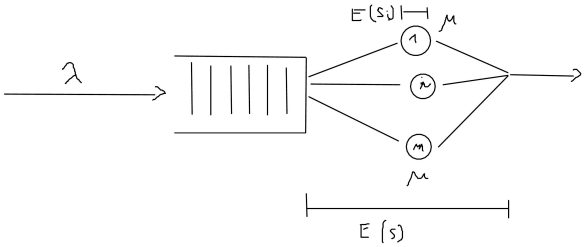
Nel caso di un centro a servente singolo, vale che:

$$N_s = \begin{cases} 0 \\ 1 \end{cases}$$

se $N_q = 0$. Mentre $N_s = 1 + N_q$se $N_q > 0$.
Nel caso in cui abbiamo serventi multipli vale che:

$$N_s = \begin{cases} 0 \\ 1 \\ . \\ . \\ m \end{cases}$$

se $N_q = 0$, mentre invece $N_s = N_q + m$ se $N_q > 0$.
In questo caso, cambiano tutte le misure che abbiamo effettuato: nell'immagine, consideriamo ogni servente identico



il tempo di servizio medio, nel caso di un singolo server era $E(s)$, ora lo denotiamo con $E(s_i)$ e ci rappresenta il tempo che passa da quando un job si mette in coda a quando esce, quindi è il tempo di servizio medio su un server.
$E(s)$ è invece il tempo di servizio medio dall'istante in cui entra un job a quando esce un job (non per forza lo stesso), quindi è il tempo medio che occorre per liberare un server. I due tempi sono sicuramente diversi fra loro:

- $E(s_i) = \frac{1}{\mu}$
- $E(s)$ è invece l'inverso del tasso globale, il multi-server ha una frequenza globale pari a $m \cdot \mu$, quindi $E(s) = \frac{1}{m \cdot \mu} = \frac{E(s_i)}{m}$

Per quando riguarda $E(N_s)$, abbiamo che:

$$E(N_s) = \begin{cases} E(N_q) + \rho & \text{if } m = 1 \\ E(N_q) + m \cdot \rho & \text{if } m > 1 \end{cases}$$

Come viene definito ρ nel caso del multi-server: in generale, sappiamo che $\rho \cong \frac{frequenza\_di\_ingresso}{max\_frequenza\_d'uscita}$, quindi andiamo a fare un prima distinzione fra:

- ρ_i, che è l'utilizzazione del singolo server
- ρ_{glob} che è l'utilizzazione globale del multi-server

Per rho_i, abbiamo che il tasso d'ingresso λ viene diviso equamente su tutti gli m centri, quindi avremo $\rho_i = \frac{\lambda}{m \cdot \mu}$; per ρ_{glob}, il tasso d'ingresso è λ, ma la frequenza del multi-server è pari a $m \cdot \mu$, quindi abbiamo $\rho_{glob} = \frac{\lambda}{m \cdot \mu}$. Nonostante i due fattori di utilizzazione siano

uguali, cambia il significato che hanno: se ad esempio consideriamo un $\rho = 0.7$, nel caso del singolo servente, e quindi di ρ_i, questo corrisponde al fatto che il centro sarà utilizzato per il 70%, per un periodo di osservazione lungo. Per quando riguarda invece ρ_{glob}, avere un valore di 0.7 vuol dire che in media il 70% dei server saranno pieni. Per quanto riguarda le relazioni medie introdotte in precedenza, abbiamo che:

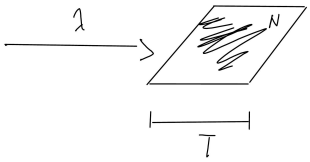
- $E(T_s) = E(T_q) + E(s_i)$
- $E(T_q) = f(\lambda, \rho, E(s))$

3.3 Legge di Little

La legge di Little permette di fare delle considerazioni sui valori medi di un sistema a code (anche per l'intera rete), vale sotto determinate assunzioni:

1. coda con disciplina FIFO
2. capacità del servente infinita
3. flussi bilanciati

Nonostante ciò, è possibile applicare il teorema di Little anche se non valgono alcune delle ipotesi, ad esempio se il servente non ha capacità infinita o se per un periodo di tempo di osservazione del sistema non c'è il bilanciamento dei flussi.



Legge di Little: sia N il numero medio di popolazione in una "black box", λ il tasso di arrivi medio e T il tempo medio di percorrenza

della box. Allora vale che $N = \lambda \cdot T$.
Possiamo ipotizzare che la black box sia qualunque coda:

- la coda + il servente: la legge ci fornisce $E(N_s) = \lambda \cdot E(T_s)$
- la singola coda: $E(N_q) = \lambda \cdot E(T_q)$
- il solo servente: $\rho = \lambda \cdot E(S)$
- l'intera rete: $N = \lambda \cdot T$

È quindi possibile riscrivere tutte le relazioni trovate in precedenza per servente singolo e multi-server in funzione della legge di Little, ad esempio per il singolo centro:

1. $E(N_s) = \lambda \cdot E(T_s)$
2. $E(N_q) = \lambda \cdot E(T_q)$

e vale lo stesso per il multi-server

3.4 Risultati del modello analitico

Abbiamo visto alcune relazioni per i valori medi, riscritte anche con la legge di Little. Le soluzioni note partono tutte da $E(N_q)$, cominciamo con il servente singolo.

Notazione di Kendall : notazione del tipo $A/S/m/B/N/D$, dove:

- A si riferisce alla distribuzione degli inter-arrivi
- S alla distribuzione del servizio
- m al n° server
- B alla capacità della coda
- N alla taglia della popolazione
- D alla disciplina della coda

Le distribuzioni più classiche sono $D, M.E_k, H_2, G$.
Per ora assumiamo scheduling astratto e non preemptive, quindi FIFO, LIFO non -preemptive, Random. Sembrerebbe che la FIFO, poiché rispetta l'ordine di arrivo, potrebbe essere quella che produce il tempo di risposta minimo (in quanto segue l'ordine di arrivo delle richieste), viceversa con la LIFO sembrerebbe che i job si vedano sempre bloccati da altri con tempi maggiori.
Ma tutte le politiche danno lo stesso **tempo di risposta medio**.
La notazione che utilizzeremo sarà $M/G/1$ con scheduling astratto.

Equazione di Khinchin Pollaczek (1930): da una espressione per il livello di congestione nella coda, ovvero quanti job in media si trovano nella coda. $E(N_q) = \frac{\rho^2}{2\cdot(1-\rho)} \cdot [1 + \frac{\sigma^2(S)}{E(S)^2}]$. $C^2 = \frac{\sigma^2(S)}{E(S)^2}$ vuol dire che la variabilità incide moltissimo sulle prestazioni: C^2 misura la dispersione dei tempi di servizio attorno alla media, quindi potrei avere distribuzioni con la stessa media, ma se la varianza è differente le cose cambiano di molto (rivedi esempi di introduzione. La congestione della coda, ovvero la popolazione media, è proporzionale a C^2; la formula fa riferimento ad un sistema all'equilibrio e tutti i risultati analitici fanno riferimento a sistemi all'equilibrio (o $E(N_q) -> \infty$.

3.5 Distribuzioni a fasi

Facciamo la distinzione fra le diverse distribuzioni a fasi,ovvero in cui si combinano più fasi esponenziali per modellare il tempo di servizio, **consideriamo sempre un solo servente, che serve un solo job**:

- esponenziale: il tempo di servizio che viene caratterizzato da una fase, tasso μ
- k-Erlang: il tempo di servizio che viene caratterizzato con la k-Erlang è la successione di k tempi, dove ciascuna fase è esponenziale. Le fasi sono uguali fra di loro, con tasso μk, un job che prende servizio termina tutte le k fasi e poi esce

- distribuzione iper-esponenziale a 2 fasi: c'è una alternanza con probabilità p fra due tempi esponenziali. I due tempi sono legati da una relazione, che dipende da p:
  - tasso del 1° stadio è $2p\mu$
  - 2° stadio ha tasso $2(1-p)\mu$

  quindi cambieranno le frequenze di servizio.
  Quando $p = 0.5$, i due stadi sono equiprobabili, hanno entrambi tasso μ, quindi l'iper-esponenziale diventa una esponenziale. Tanto più p si allontana da 0.5, tanto più cresce la variabilità fra i due diversi possibili tempi
- distribuzione di Cox: può modellare qualsiasi distribuzione. Può avere un numero arbitrario di fasi, ciascuna di esse è esponenziale ma al contrario della Erlang, dopo ciascuna fase possono accadere due cose (esclusa l'ultima fase):
  - o si esce, questo accade nella fase i con probabilità b_i
  - o si passa all fase successiva e questo accade nella fase i con probabilità a_i

  ovviamente devono valere le classiche regole, ovvero $a_i = 1-b_i, \forall i$ ed inoltre per lo stato finale k abbiamo $b_k = 1, a_k = 0$.

Riscrivendo la KP in termini di C^2, otteniamo $E(N_q) = \frac{\rho^2}{2\cdot(1-\rho)} \cdot [1 + C^2]$, dove C^2 dipende dalla distribuzione :

- D: $C^2 -> 0$
- E_k: $C^2 = 1 + \frac{1}{k}, k \geq 1$
- M: $C^2 = 1$
- H_2: $C^2 = g(p) = \frac{1}{2\cdot p\cdot(1-p)} - 1$. Per $p = 0.5$, riotteniamo la C^2 di una M, mentre al crescere di p cresce anche il termine C^2.

esempio: utilizziamo la KP per ricavare il tempo medio di attesa in coda. $E(T_q) = \frac{E(N_q)}{\lambda} = \frac{\rho^2}{\lambda \cdot 2 \cdot (1-\rho)} \cdot (1 + C^2)$, scrivo uno dei ρ al numeratore come $\lambda \cdot E(S)$ ed ottengo $\frac{\lambda \cdot E(S) \cdot \rho}{\lambda \cdot 2 \cdot (1-\rho)} \cdot (1 + C^2) = \frac{E(S) \cdot \rho}{1-\rho} \cdot \frac{(1+C^2)}{2}$. Anche per un fattore ρ basso, ad esempio 0.5, il coefficiente C^2 per le heavy tails deve essere almeno pari a 25, quindi $\frac{1+C^2}{2} = 13$, ovvero il tempo di attesa medio può diventare 13 volte il tempo di servizio. Nella tabella sono riportati in base al service time, $E(N_q)$ ed $E(T_q)$:

Service time	$E(N_q)$ $\frac{p^2}{2 \cdot (1-\rho)} \cdot [1 + C^2]$	$E(T_q)$ $\frac{E(S) \cdot \rho}{1-\rho} \cdot \frac{(1+C^2)}{2}$
Deterministic M/D/1	$\frac{\rho^2}{2 \cdot (1-\rho)}$	$\frac{\rho \cdot E(S)}{2 \cdot (1-\rho)}$
Markovian M/M/1	$\frac{\rho^2}{1-\rho}$	$\frac{\rho \cdot E(S)}{1-\rho}$
k-Erlang M/E_k/1 $\sigma^2(S) = \frac{E(S)}{k}$	$\frac{\rho^2}{2 \cdot (1-\rho)} \cdot (1 + \frac{1}{k})$	$\frac{\rho \cdot E(S)}{2 \cdot (1-\rho)} \cdot (1 + \frac{1}{k})$
Iper-esponenziale M/H_2/1 $\sigma^2(S) = E^2(S) \cdot g(p)$	$\frac{\rho^2}{2 \cdot (1-\rho)} \cdot (1 + g^2(p))$	$\frac{\rho \cdot E(S)}{2 \cdot (1-\rho)} \cdot (1 + g(p))$

Al di là delle formule, è interessate notare come tutti i valori siano indipendenti da C^2.
esempio: richiamiamo l'esempio del provider che deve aumentare la potenza di calcolo per far si che gli utenti, nonostante il tasso di arrivi sia raddoppiato, sperimentino lo stesso tempo di risposta. Mostriamo in particolare come, se raddoppia la potenza il tempo di risposta risulta dimezzato:

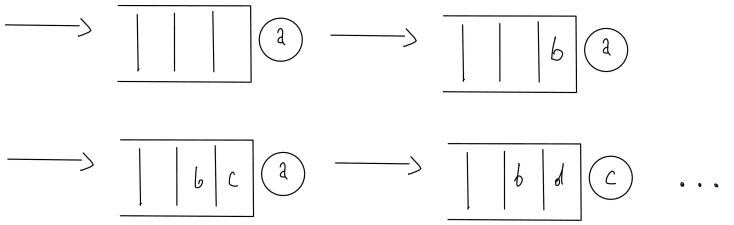
- in partenza avevamo λ, μ, ρ
- ora, abbiamo $\lambda' = 2 \cdot \lambda$, $\mu' = 2 \cdot \mu$, ma $\rho' = \rho$

$E(S)'$ diviene pari ad $\frac{E(S)}{2}$, quindi ora consideriamo $E(T_s') = E(T_q') +$

$E(S')$.Per quanto riguarda $E(T_q')$, vedendo le formule sopra, questo viene dimezzato (in quanto $E(S)$ è la metà), ed è quindi pari a $\frac{E(T_q)}{2}$. Quindi, abbiamo che $E(T_s') = \frac{E(T_q)}{2} + \frac{E(S)}{2} = \frac{E(T_s)}{2}$, quindi raddoppiando la potenza si dimezza il tempo di risposta.
Valgono le relazioni:

- $E(N_q)_D \leq E(N_q)_{E_k} \leq E(N_q)_M \leq E(N_q)_{H_2}$
- $\sigma^2(N_q)_D \leq \sigma^2(N_q)_{E_k} \leq \sigma^2(N_q)_M \leq \sigma^2(N_q)_{H_2}$

C'è sensitività al tempo di servizio, visto che $E(N_s)+\rho$, vale lo stesso ordinamento. Con Little è possibile vedere la stessa cosa sui tempi, ma vale solo sulla media; per il momento del II ordine invece non vale.
Per quanto riguarda la sensitività allo scheduling, sono uguali sia media e varianza perché KP vale per ogni servizio astratto. Lo stesso vale per $E(N_s)$ ed ancora per $E(T_q)$, ma non per $\sigma^2(T_q)$ perché come abbiamo visto, lo scheduling conta.
Ad esempio, per uno scheduling LIFO, c'è la maggiore variabilità:



in questo caso, b si vede arrivare sempre dei job e quindi risente del tempo di esecuzione di questi. L'esempio fa riferimento alla sola varianza, dato che nella media di tutti i possibili casi non conta, ed inoltre vale se ρ è grande, in quanto questo corrisponde ad un'alta probabilità che al suo arrivo b trovi la coda piena.
Se consideriamo una esponenziale di parametro $\frac{1}{\mu}$:

- la k-Erlang ha $f_i(x) = k\mu e^{-k\mu x}$, media e varianza saranno divise per k
- per l'iper-esponenziale, abbiamo che il singolo stadio è una esponenziale: una sarà di param $2p\mu$ e l'altra $2(1-p)\mu$. Per la me-

dia si fa la somma pesata delle due medie, e viene $= \frac{1}{\mu}$; per quanto riguarda la varianza abbiamo $\sigma(X) = g(p)(\frac{1}{\mu})^2$, dove $g(p) = \frac{1}{2p(1-p)} - 1$ e tanto più ci allontaniamo da 0.5, tanto più p decresce e $g(p) -> \infty$

- per distribuzioni generiche, abbiamo la distribuzione di Cox (può venire una distribuzione complessa). Ogni stadio è esponenziale di media $\frac{1}{\mu_i}$, dove i vari μ_i possono essere differenti e se $t_1, t_2, ..., t_k$ sono i tempi spesi in ciascuno stadio il tempo totale t è:
  - t $= t_1$ con probabilità b_1
  - t $= t_1 + t_2$ con probabilità $a_1 \cdot b_2$
  - t $= t_1 + t_2 + t_3$ con probabilità $a_1 \cdot a_2 \cdot b_3$
  - ...

3.5.1 Distribuzioni arbitrarie

È sempre possibile trovare una distribuzione di Cox che approssima bene la funzione arbitraria, occorre guardare alla trasformata di Laplace e distinguere i casi in cui sia razionale o no:

a) se è razionale, possiamo approssimare $C_k(t) = f(t)$ per un certo k, esatto o con una certa precisione nota

b) altrimenti, serve una $f(t) \approx g(t)$, in cui l'errore è noto ed a questo punto $C_k(t) \approx g(t)$, ma c'è un errore di sicuro

3.5.2 Confronti

Consideriamo un esempio con distribuzione esponenziale, $E(S) = 1s$, $\mu = 1\frac{job}{s}$, se andiamo a graficare $E(T_q)$ in funzione di ρ vediamo come dopo un valore di 0.7, la curva cresce velocemente a ∞.
Se confrontiamo con una k-Erlang (con k = 3), a parità di $E(S)$ (o il confronto non avrebbe senso), con $\mu = 2\frac{job}{s}$, abbiamo come frequenza del singolo stadio $\frac{\mu}{k}$, $E(S_i) = \frac{0.5}{3} = 0.1666...s$ (tempo medio nello stato i) ed un $\sigma^2(S) = \frac{1}{k} \cdot (\frac{1}{\mu})^2 = 0.08333...$, un'esponenziale di stessa

media avrebbe varianza pari a 0.25 (la varianza * 3).
Considerando infine una iper-esponenziale, con stessa $E(S)$, $\mu = 2\frac{job}{s}$, $p = 0.2$ abbiamo che il primo stato ha un tasso pari a $2 \cdot \rho \cdot \mu = 0.8\frac{job}{s}$, mentre il secondo stato pari a $2 \cdot (1 - \rho) \cdot \mu = 3.2\frac{job}{s}$, quindi in media il 20% del traffico riceve un servizio con tempo pari ad $\frac{1}{3.2}$, molto più basso di quello ricevuto in media dal restante 80% del traffico.
esercizio:
un sistema TP accetta e processa uno stream di transazioni, mediate tramite un buffer grande.

- Le transazioni arrivano in modo random (distribuzione dei tempi di inter-arrivo random)

- il server TP è in grado di servire transazioni ad un certo service rate

Se il tasso di arrivi e di servizio raddoppiano, sappiamo che il tempo di risposta dimezza. Supponiamo che $\lambda = 15tps$ ed il tempo di servizio medio per transazione sia $E(S) = 58.37ms = 0.05837s$. Cosa succede al tempo di risposta medio se la frequenza di arrivo cresce del 10%?
$\lambda' = 16.5tps$, $\rho = 87.56\%$, ora il $\rho' = 96.31\%$. Cosa possiamo dire su $E(S)$: siccome t. medio di servizio non è cambiato possiamo ragionare sul tempo medio di attesa. $E(T_q) = \frac{\rho}{1-\rho} \cdot (\frac{C^2+1}{2}) \cdot E(S)$, al cambiamento di ρ, i due termini che non vi dipendono non cambiano, quindi valutiamo il rapporto fra $E(T_q)$ ed $E(T'_q)$: $\frac{E(T_q)}{E(T'_q)} = \frac{\frac{\rho}{1-\rho}}{\frac{\rho'}{1-\rho'}} = \frac{7.0354}{26.1039} \simeq \frac{1}{3.7}$. Tempo di attesa quasi quadruplicato.

3.6 Proprietà di mancanza di memoria e distribuzioni di probabilità

La classe delle distribuzioni a fasi è molto importante, perché permette di modellare una distribuzione generale. La distribuzione esponenziale gode della memoryless property: informalmente, la RV non

si ricorda del passato e si comporta come se fosse una variabile nuova, quindi il comportamento dipende solo dal presente.
esempi: supponiamo che la rv X sia il tempo trascorso in un negozio in un certo giorno della settimana dall'istante di apertura all'istante in cui entra il primo cliente. Oppure X sia il tempo da quando un server viene acceso a quando arriva la prima richiesta al server. La proprietà di mancanza di memoria fa un confronto fra la distribuzione di probabilità che valuta il fenomeno dall'istante 0 e la distribuzione che valuta il fenomeno aleatorio prima del momento in cui avviene per la prima volta un evento. Le due distribuzioni sono identiche.
Le uniche distribuzioni che soddisfano questa proprietà sono l'esponenziale e la geometrica; questa proprietà semplifica moltissimo l'analisi.
esempio: due utenti in fila alla posta, arriva un terzo utente. Qual è la probabilità che A sia l'ultimo ad uscire? $\frac{1}{2}$, uno fra B e C se ne andrà, quindi poi uscirà uno fra A ed il rimanente. Siccome i tempi di servizio sono esponenziali, la probabilità che finisca A o il rimanente è il 50%, non c'è influenza della storia passata.
Supponiamo di avere una v.a esponenziale di parametro $\frac{1}{\mu}$, quale è $P(X \leq t)? = 1 - e^{-\mu t}$. Se condiziono invece rispetto al fatto che la v.a non ha terminato a t_0 e voglio scoprire la probabilità che termini a $t + t_0$, $X - t_0$ è **tempo di servizio rimanente**.
$P(X \leq t_0 + t | X > t_0) = \frac{P(X \leq t_0 + t \cap X > t_0)}{X > t_0} = \frac{P(X \leq t_0 + t) - P(X > t_0)}{P(X > t_0)} = ... = 1 - e^{-\mu t}$

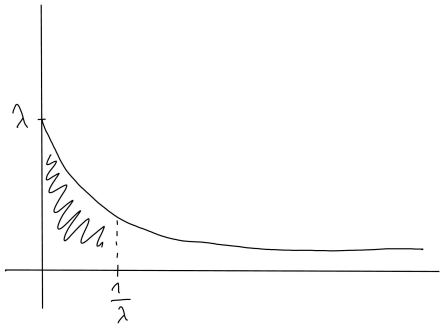
3.6.1 Costante di decadimento dell'esponenziale

L'esponenziale ha un'altra proprietà particolare: $f(t) = \lambda e^{-\lambda t}$, la calcolo in alcuni punti:

- $f(0) = \lambda$
- $f(1) = \lambda e^{-\lambda} = f(0)e^{-\lambda}$
- $f(2) = \lambda e^{-2\lambda} = f(1)e^{-\lambda}$
- ...

- $f(n) = f(n-1)e^{-\lambda}$

La media è $\frac{1}{\lambda}$, ed $f(\frac{1}{\lambda}) = \lambda e^{-1}$ ovvero: l'esponenziale parte da un punto che è il suo parametro e crolla, valutando f nella media $\frac{1}{\lambda}$ scopro che il suo valore iniziale è stato ridotto dell'$e^{-1}\%$, quindi la media è il tempo per ridurre la quantità iniziale del $1 - e^{-1}\%$. Questa viene detta costante di tempo ($\frac{1}{\lambda}$): se valuto la funzione di distribuzione nella media $\frac{1}{\lambda}$: $F(\frac{1}{\lambda}) = \int_0^{\frac{1}{\lambda}} f(t)dt = 1 - e^{-1} = 0.6321$.



È indipendente da λ: qualunque sia la costante di tempo, questo equivale al tempo necessario affinché il valore iniziale si riduca di circa il 63%.

3.7 Memoryless come tempo di vita

Una v.a X è detta memoryless se $P(X > s + t|X > s) = P(X > t), \forall s, t > 0$, ovvero il fatto che sia passato già un tempo s non conta nulla. esempio: sia X il tempo di vita di una lampadina. La memoryless direbbe che la probabilità che la lampadina duri ancora t secondi prima che si fulmini, dato che aveva funzionato per s secondi, è la stessa che la lampadina funzioni per almeno t.
Vediamo in termini di distribuzioni, non considerando la memoryless: considero tutte le distribuzioni per cui questo tempo di vita trascorso conta. Le distribuzioni per cui $P(X > s + t|X > s)$ decresce al crescere di s vengono dette a *failure rate crescente*. Mentre, quelle per cui vale il contrario (ovvero al crescere di s la probabilità decresce), vengono dette a *decreasing failure rate*.

esempi: il tempo di vita di una automobile è caratterizzato da un tempo di vita a increasing failure rate, nel caso chi ci riguarda di più:

- i tempi di vita dei job UNIX sono caratterizzati da distribuzioni decreasing failure rate: tanta più CPU ha usato un job, quanto più è probabile che ne debba usare altra
- stesso vale per i chip: (test spesso eseguito per i chip) se non falliscono per un certo tempo, la probabilità di difetto è bassa (solitamente compaiono in un primo tempo).

Hazard rate function: supponiamo di avere una v.a X continua con densità di probabilità $f(t)$ e funzione di distribuzione $F(t) = PX < t$, chiamiamo $r(t) = \frac{f(t)}{\bar{F}(t)}$, dove $\bar{F}(t)$ è il complementare della $F(t)$, consideriamo la probabilità che un oggetto di t anni fallirà nei prossimi dt secondi $= P(X \in (t, t+dt)|X > t) \approx \frac{f(t)dt}{\bar{F}(t)} = r(t)dt$, quindi la hazard rate function ci da il failure rate istantaneo.
Quando $r(t)$ è costante, allora la distribuzione è esponenziale.

3.7.1 Importanza del tempo di vita rimamente

Consideriamo un load balacing per la CPU in una rete di workstations: può essere utile migrare un job sulla workstation meno carica, per migliorare i tempi medi di risposta. Ma questo ha un costo, bisogna portarsi dietro lo stato del processo, ci sono due tipi di migrazioni:

- non-preemptive migration: re-alloca solo processi "nuovi", non ancora attivi
- preemptive migration (active process migration): possibile migrare anche processi attivi

Ci poniamo delle domande: può essere utile la migrazione P, o basta la NP? E se scegliamo di usare la P, quale è una buona politica di migrazione (quale processo è meglio scegliere)?
Richiamiamo ancora una volta il lifetime, ricordiamo la terminologia:

- taglia di un job: domanda totale di CPU
- vita di un job: utilizzo di CPU fino ad ora
- tempo di vita di un job: richiesta totale di CPU di un job, dipende sia dalla capacità operativa che da quanto richiede (coincide sostanzialmente con la size
- tempo di vita rimanente di un job: si riferisce ad un certo istante di tempo, la rimanente richiesta di CPU

Non per tutte le applicazioni è possibile conoscere tutti i termini, ma è sempre possibile su un sistema misurare quanta elaborazione è stata fatta, quindi è sempre possibile conoscere l'age (vita di un job).

3.8 Distribuzioni di Pareto

(vedi slide) Vedo le raccolte dei tempi di vita di job Unix, sembrano distribuiti esponenzialmente ma non è così: per t=2 abbiamo una decrescenza di $\frac{1}{2}$, per t=4 $\frac{1}{4}$ etc..., classe delle distribuzioni di Pareto: $f(x) = \alpha \cdot k^{\alpha} \cdot x^{-\alpha-1}$ con $k \leq x < \infty, 0 \leq \alpha \leq 2$, con code la cui misura è data da α:

- $\alpha -> 0$, più variabilità e più pesantezza della coda
- $\alpha -> 2$, il contrario

problema: hanno momento finito di ordine i sono se $\alpha > i$, quindi non possiamo valutarne la varianza.
Proprietà:

- sono a decreasing failure rate: più CPU viene usata, più continua ad usarne
- hanno varianza infinita, la heavy tail property ci dice che una una minuscola frazione dei job molto grandi contribuisce alla metà del carico del sistema. Per esempio, con $\alpha = 1.1$, se migro i job grandi, tolgo la metà del carico dal sistema.

3.8.1 Bounded Pareto

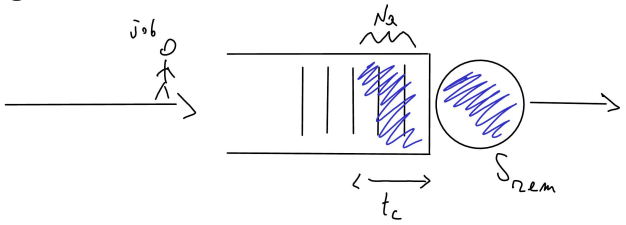
L'infinito che limita la x si può togliere, si può facilmente trovare un limite finito molto grande alla size, si può passare alla classe bounded Pareto: $f(x) = \alpha \cdot x^{-\alpha-1} \cdot \frac{k^\alpha}{1-(\frac{k}{p})^\alpha}$ con $k \leq x \leq p, 0 < \alpha < 2$, tutti i momenti sono finiti.
$C^2 = 25$ è il minimo coefficiente di variazione misurato per distribuzioni di questo tipo, solitamente varia tra 25 e 49, quindi le prestazioni possono degradare moltissimo (con la variabilità), nel caso di distribuzione di questa classe.
Quindi, rispondendo alle due domande precedenti, la caratteristica del decreasing failure rate (quindi tutti i casi in cui l'applicazione viene ben modellata da questa caratteristica) ci porta a pensare che la migrazione è utile e può essere più sensato migrare i job vecchi. Sebbene i job vecchi potrebbe aver maturato uno stato più significativo di un job "giovane" (appena nato"), quel costo di migrazione verrà ammortizzato dal tempo in cui quel job continuerà a chiedere risorse di computazione.

3.8.2 Pareto study

Consideriamo una distribuzione di Pareto: $f(x) = \alpha \cdot k^\alpha \cdot x^{-\alpha-1}$ con $k \leq x < \infty, 0 \leq \alpha \leq 2$, $E[X] = \frac{\alpha k}{\alpha-1}, \alpha > 1$, combinando media e varianza si ottiene il tempo di attesa $E(T_Q)$. Dalla media posso ricavare k a seconda dell'α che vogliamo: man mano che ci si allontana da $\alpha = 2$ ci si allontana dal caso peggiore (vogliamo $\alpha < 2$ per una heavy tail). Tutti i casi sono peggiori dell'iper-esponenziale, che è il caso peggiore di tutte le distribuzioni viste, andando verso le heavy tail e con fattori di utilizzazione alti il tempo di attesa passa dal 7 dell'iper-esponenziale a 144 per la Pareto; la variabilità ha quindi un impatto pesante.

3.9 Risultati analitici ulteriori della KP

Vediamo la KP nell'ottica del tempo di servizio rimanente. Ad un certo istante di tempo, la situazione di un server può essere la seguente:



supponiamo che lo scheduling sia FIFO, il job che arriva dovrà attendere il tempo necessario ad elaborare i job nella coda ed il tempo di servizio rimanente del job in servizio. Il tempo di attesa sarà quindi una qualche combinazione $T_Q = t_c \oplus S_{rem}$, si può dimostrare che per **qualunque distribuzione** $E(S_{rem} = \frac{\lambda}{2} \cdot E(S^2)$. Nel caso dell'esponenziale, poiché $E(S^2) = 2 \cdot E(S)^2 \rightarrow E(S_{rem}) = \frac{\lambda}{2} 2E(S)^2 = E(S_{rem} = \rho E(S))$, quindi $E(T_Q) = \frac{\rho E(S)}{1-\rho} = \frac{E(S_{rem})}{1-\rho}$, quindi l'operazione per l'esponenziale è un prodotto di $E(S)$ con $\frac{1}{1-\rho}$ (che non è il tempo per completare, **ma è un numero puro**).
Nel caso generale (KP): $E(T_Q) = \frac{\rho}{1-\rho} \cdot \frac{C^2+1}{2} \cdot E(S) = \frac{1}{1-\rho}[\frac{\sigma(S)^2}{E(S)^2} + 1] = \frac{\rho}{2(1-\rho)} \cdot [\frac{E(S^2)-E(S)^2}{E(S)^2} + 1] = \frac{\lambda E(S)}{2(1-\rho)}[\frac{E(S^2)}{E(S)^2} - 1 + 1] = \frac{\lambda}{2(1-\rho)}[\frac{E(S^2)}{E(S)^2}]E(S)^2 = \frac{\lambda}{2} \cdot \frac{E(S^2)}{1-\rho}$.
Il termine $\frac{1}{1-\rho}$ rappresenta il fattore legato al t_c (NON È UN TEMPO), siccome i due termini sono combinati con un prodotto:

- se uno è 0, il risultato è 0. Se $E(S_{rem}) = 0$, allora non c'è coda. Il job che arriva non deve attendere e viene subito servito
- se uno è 1, abbiamo un invariante per il prodotto. Se il coefficiente $\frac{1}{1-\rho}$ fosse 1, il job che arriva trova solo il job in servizio, quindi in pratica è come se non attendesse nulla

Tempo di risposta per il tempo di risposta $E(T_S)$ abbiamo:

- M/G/1: $E(T_S) = E(T_Q) + E(S) = \frac{\frac{\lambda}{2} \cdot E(S^2)}{1-\rho}$
- M/M/1: $E(T_S) = \frac{\rho E(S)}{1-\rho} + E(S) = \frac{E(S)}{1-\rho}$.

3.10 Scheduling astratto non-preemptive

Riprendiamo le definizioni sullo scheduling, abbiamo dei risultati molto più forti di quelli della KP. Ricordiamo che:

Definizione 1: una politica prevede preemptive se un job può essere fermato durante la sua esecuzione e ripreso più tardi, dal punto in cui era stato fermato. Una politica è invece non preemptive se i job non possono mai essere fermati mentre sono in servizio.

Definizione 2: lo scheduling si dice work conserving quando il server non è mai fermo nel caso in cui ci sia qualcuno nella coda. Può avere senso, per un server, attendere un job di taglia piccola (se sa che questo arriverà a breve), quindi il work non-conserving ha un senso. C'è un risultato molto più forte della KP:

Teorema di Conway, Maxwell, Miller: tutto lo scheduling astratto non preemptive mostra avere la stessa distribuzione del numero di job nel sistema: vale per $E(N_S), E(N_Q)$, quindi con Little anche per $E(T_Q)$ ed $E(T_S)$; quindi, vale per i momenti di ogni ordine (per i tempi invece, vale solo per le medie).
In generale $E(T_Q) = \frac{\frac{\lambda}{2} E(S^2)}{1-\rho}$, può essere alto quando il quadrato è alto.
Cosa non ci piace: l'attesa in coda non ha nessuna relazione con la size del job, Mentre un job grande la tollera, in quanto l'attesa è proporzionale a quanto lui richiede, ma per job piccoli questo è poco fair. Ciò che vorremmo vedere è un tempo di risposta proporzionale alla size: chi chiede molto attende molto, chi poco attende poco.

3.10.1 Tempo di slowdown

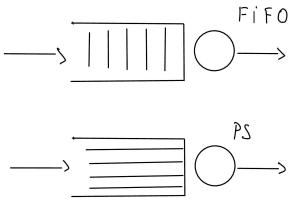
Consideriamo il tempo di risposta medio per un job di taglia x: $E(T_S(x)) = E(x + T_Q(x))$, ATTENZIONE: si mette la taglia del job (x), non il tempo di servizio medio $E(S)$, perché dopo aver atteso in coda non vogliono vedere il tempo $E(S)$, bensì x. Quindi, siccome x non è una media, ma una size, inoltre $E(T_Q)$ è indipendente dalla size x, quindi otteniamo $E(T_S(x)) = x + E(T_Q)$. Definiamo una nuova misura di prestazione chiamata *slowdown* (rallentamento): non valuto solo il tempo di risposta, che una misura non fair, ma anche lo slow down per size che mi dice quanto il sistema è stato fair con i job. Lo slowdown medio per job di taglia x è il tempo medio osservato rispetto all taglia del job: $E(sd(x)) = \frac{E(T_S(x))}{x} = 1 + \frac{\frac{\lambda}{2}E(S^2)}{x \cdot (1-\rho)}$ quindi lo slowdown diventa tanto più grande, quanto più il job è piccolo.

3.10.2 Slowdown vs response time

Il tempo di risposta tende ad essere rappresentativo per la performance di pochi job, i più grandi tendono ad enfatizzare la performance di quelli molto grandi, visto che contano di più in media il loro tempo di risposta tendere ad essere il più grande.
Il tempo di slowdown tende ad essere rappresentativo per la performance della maggior parte dei job perché domina la performance del grande numero di piccoli job; vorremmo rendere $E(T_S(x))$ piccolo per x piccoli. Come fare se non si conosce la taglia x: due ragioni storiche per cui lo scheduling delle CPU è circa processor sharing :

- parliamo di sistemi con molte risorse, quindi è utile avere in esecuzione molteplici job perché questi richiedono diverse risorse
- una divisione di tempo o processor sharing, in termini di modelli, fa andare avanti job piccoli senza necessariamente conoscere la taglia. Se do un n-simo di quanti (o quanto di tempo) a ciascuno, i job piccoli finiranno subito, quelli grandi continueranno a stare lì

dovrebbe essere migliore il processor sharing rispetto alla FIFO, in termini di tempi di risposta medi,perché manda via i job piccoli velocemente, ed in termini di slowdown dovrebbe essere molto meglio, perché non rallenta i job piccoli. $P(N_S = n)^{M/G/1/PS} = \rho^n(1-\rho) = P(N_S = n)^{M/M/1/FIFO}$, riusciamo ad avere le stesse prestazioni, anche in caso di distribuzione generale. In particolare: $E(N_S)^{M/G/1/PS} = \frac{\rho}{1-\rho} = E(N_S)^{M/M/1/FIFO}$ ed $E(T_S)^{M/G/1/PS} = \frac{E(S)}{1-\rho} = E(T_S)^{M/M/1/FIFO}$, quindi il processor sharing, che tra l'altro modella bene molte situazioni reali, ha degli indici di prestazioni ottimi quando la variabilità è maggiore di 1. Inoltre, è insensibile alla variabilità del tempo di servizio, perché vale per distribuzione generale (infatti la variabilità non compare nelle formule), mentre nella FIFO le prestazioni sarebbero peggiori.
Se guardassimo delle sequenze particolari di job, potrebbero esserci prestazioni differenti:



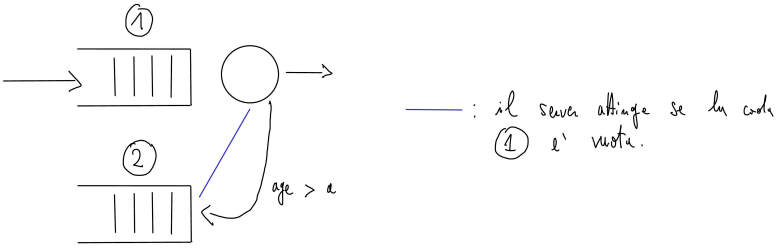
arrivano 2 job in simultanea con una taglia di 1s, per quale dei due scheduling abbiamo $E(T_S)$ più piccolo;

- nel caso FIFO, arrivano i due job in simultanea, il primo prende subito servizio e ci mette 1 sec. Il secondo attende 1 sec + 1 sec per il suo servizio, quindi in totale 2. In media, $1.5s$
- nel caso della PS, i due job dovranno stare nel centro 2 sec per poter essere entrambi serviti, quindi $E(T_S) = 2s$, simultaneamente riceveranno la metà della capacità, nel 1° secondo $\frac{1}{2}$ della capacità a testa e nel 2° secondo l'altro $\frac{1}{2}$

quindi su alcune sequenza particolari di job la Processor Sharing può dare dei risultati peggiori.

In termini di slowdown abbiamo che $E(T_S(x))^{M/G/1/PS} = \frac{x}{1-\rho}$ e quindi $E(sd(x))^{M/G/1/PS} = \frac{1}{1-\rho}$, quindi è indipendente dalla size: tutti i job vengono rallentati dello stesso fattore, piuttosto dipende dal carico del sistema; quindi la processor sharing è lo scheduling fair per eccellenza, spesso viene usato per confronti con altri scheduling. Nel caso astratto abbiamo $E(T_S(x))^{M/G/1/abstract} = 1 + \frac{\frac{\lambda}{2}E(S^2)}{x(1-\rho)}$. Tutti gli scheduiling preemptive non basati size hanno lo stesso tempo di slowdown $\frac{1}{1-\rho}$. Noi vorremmo rallentare i job grandi a favore dei piccoli, ma senza conoscere la size dei job come facciamo? Sappiamo sempre di sicuro l'age dei job, ed è un indicazione della domanda rimanente: se le distribuzioni hanno decreasign failure rate, allora i job più grandi hanno con una certa probabilità una domanda rimanente più alta, quindi potremmo dare preferenza ai job con age bassa che **potrebbero essere piccoli** (per le heavy tail, consultare paragrafo 20.7).

Scheduling Unix la disciplina si chiama foreground-background:



due code gestite in PS, tutto il traffico in I va nella prima coda che viene servita con priorità. Il server passa alla seconda coda solo quando la prima è libera, il passaggio di coda avviene mediante l'age: superato un valore a di age (parametro del sistema) di uso della risorsa, il job viene messo in coda 2. Non appena arriva qualcuno in coda 1, il server torna a servire questa coda, in modo da dare la precedenza ai job piccoli pur senza conoscerne la size, ma "giocando" sull'age.
Il tempo di risposta medio è abbastanza condizionato dai job con tempi di servizio alti, viceversa nel calcolo dello slowdown la metrica

tiene conto di una grande parte di job piccoli. Nello slowdown, se non ci fosse attesa (perché ρ è piccolo) lo slowdown sarebbe 1, la curva tende ad 1.Il tempo di risposta condizionato è invece lineare ad x. Nel caso della processo sharing, abbiamo $E(T_S(x))^{PS} = \frac{1}{1-\rho}$: confrontando con FIFO, il tempo di risposta è sempre lineare rispetto ad x, ma lo slowdown nel caso della PS è costante e dipende solo da ρ.

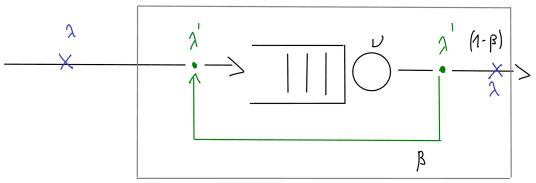
3.11 Utilizzazione e bilancio dei flussi

L'utilizzazione si può vedere studiando nel tempo l'occorrere degli eventi, otteniamo $\frac{\lambda}{\mu}$. Vediamo ora il feedback

3.11.1 Servente a coda singola con feedback

Un job che ha completato potrebbe ancora avere necessità del server, quindi il flusso di arrivo al server è mischiato:

- richieste nuove
- richieste che hanno terminato e vogliono richiedere di nuovo il servizio



a questo punto, le partenze hanno significati diversi dai completamenti: il throughput del server può essere più alto del numero di completamenti, in quanto alcuni possono essere parziali.

denotiamo con β la probabilità di feedback, ovvero quella con cui un job si rimette in coda (verrà servito in base alla disciplina della coda). Occorre fare delle considerazioni:

- il feedback è indipendente dalla storia passata, sarebbe diverso se un job sapesse quanti "giri" deve fare quando arriva in coda, assumiamo che sia probabilistico
- in teoria, un job può fare feedback un numero arbitrario di volte
- job si mischiano fra quelli che fanno feedback, ma quando calcolo il tempo di risposta i giri fatti da un job devono essere inclusi

Vediamo bilanciamento del flusso, throughput etc... L'utilizzazione del centro dipenderà da λ e da β (non è più $\frac{\lambda}{\nu}$).

Bilanciamento del flusso deve valere $\lambda = \lambda'(1-\beta)$, quindi $\rho = \frac{\lambda'}{\nu} = \frac{\lambda}{(1-\beta)\nu}$ ed il sistema satura quando $\rho \longrightarrow 1$ ovvero quando $\frac{\lambda}{(1-\beta)\nu} \longrightarrow 1$, in termini di beta quando $\beta \longrightarrow 1 - \frac{\lambda}{\nu}$

3.12 Analisi del multi-server

Soluzione del 1917 di Erlang, il modello è M/M/m con scheduling astratto. Anche questa soluzione parte dal determinare la popolazione media nella coda $E(N_Q)_{Erlang}$: si parte con la distribuzione dei job, ovvero la probabilità che nel sistema ci siano N job. Il sistema presenta delle particolarità:

- il ρ dell'intero sistema è pari al ρ del singolo centro, dove il ρ globale rappresenta la percentuale (sempre a livello stazionario) media di server occupati su un certo periodo di tempo di osservazione
- il tempo di servizio medio cambia significato: da una parte abbiamo quello del singolo centro, dall'altra quello che passa da quando un job entra in servizio all'istante in cui uno (non è detto che sia lo stesso) esce. I due tempi sono diversi fra loro, il tempo globale è m volte più piccolo del tempo che si spende nel singolo centro.

la probabilità che nel centro (coda + m serventi) ci siano n job:

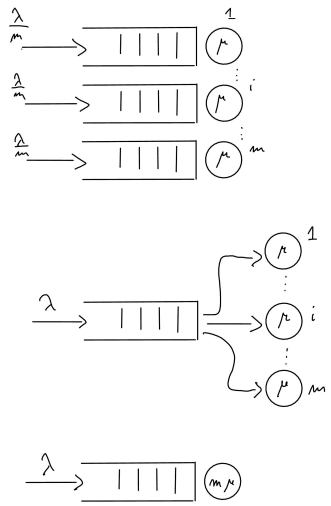
$$\begin{cases} p(n) = \frac{1}{n!}(m\rho)^n p(0) & \text{for } n = 1, ..., m \\ \frac{m^m}{m!}\rho^n p(0) & \text{for } n > m \end{cases}$$

, dove $p(0) = [\sum_{i=0}^{m-1} \frac{(m\rho)^i}{i!} + \frac{(m\rho)^m}{m!(1-\rho)}]^{-1}$.
L'altra grandezza è $P_Q \cong P(n \geq m)$, ovvero la probabilità che tutti i serve siano pieni, quindi la misura di probabilità che inizi a formarsi la coda: $P_Q = \sum_{n=m}^{\infty} p(n) = \sum_{n=m}^{\infty} \frac{m^m}{m!}\rho^n p(0) = \frac{m^m}{m!p(0)} \sum_{n=m}^{\infty} \rho^n = \frac{m^m}{m!}p(0) \sum_{n=}^{\infty} \rho^{n+m} = \frac{m^m}{m!}p(0)\rho^m \cdot \sum_{n=0}^{\infty} \rho^n$, ma $\sum_{n=0}^{\infty} \rho^n = \frac{\rho}{1-\rho}$ e quindi $P_Q = \frac{(m\cdot\rho)^m}{m!(1-\rho)}p(0)$
Quindi la popolazione media viene espressa come $E(N_Q)_{Erlang} = P_Q \cdot \frac{\rho}{1-\rho}$ ed $E(N_S) = P_Q \cdot \frac{\rho}{1-\rho} + m\rho$. Vale ovviamente la legge di Little, quindi possiamo ottenere $E(T_Q) = \frac{E(N_Q)}{\lambda} \Rightarrow E(T_Q) = P_Q \frac{\rho}{\lambda(1-\rho)} = \frac{P_Q E(S)}{1-\rho}$.
Confrontando con la formula del servente singolo (M/M/1), notiamo una sostanziale somiglianza: qui, il tempo di servizio rimanente in media è il tempo necessario a liberare uno degli m server, quindi NON È $E(S_i)$, perché sarebbe come mettersi in attesa esattamente di uno degli m centro (l'i-esimo specificatamente), IMPORTANTE:è un errore grave fare questo. Si verifica con probabilità P_Q e dura fino ad $E(S)$, ovvero il tempo per liberare uno degli m server.
Un'altra grandezza che studiamo è il numero medio di server occupati: chiamiamo la media c: $E(c) = \sum_{n=0}^{m-1} np(n) + \sum_{n=m}^{\infty} mp(n) = m\rho$, ma quindi possiamo riscrivere ρ in altri termini: $\rho = \sum_{n=0}^{m-1} \frac{n}{m}p(n) + \sum_{n=m}^{\infty} p(n)$ ed è ancora uguale a $\sum_{n=0}^{m-1} \frac{n}{m}p(n) + P_Q$ da cui deriviamo che $\rho \geq P_Q$.
Da un punto di vista dell'attesa, confrontando multi-server con ser-

vente singolo di uguale capacità (tasso $m\mu$, stesso λ), vediamo che l'attesa è più piccola nel caso del multi-server: nel momento in cui ho m server, è chiaro che l'attesa diminuisce rispetto al singolo; quindi una configurazione del genere consente di avere un tempo medio di attesa più piccola

3.12.1 Organizzazioni di server

Possiamo avere diverse configurazioni:



i ρ delle configurazioni sono le stesse: $\frac{\lambda}{m\mu}$.
Le applicazioni sono molteplici:
linee di comunicazione, m linee diverse ed m flussi di Poisson, ognuna con una frequenza di arrivo $\frac{\lambda}{m}$ pacchetti per secondo. La distribuzione dei servizi è una $Exp(\frac{1}{\mu})$. Nella gestione dei canali di comunicazione ci sono due diversi approcci:

- frequency division multiplexing (o time): sporziono la frequenza in m frequenza di uguale capacità, quindi è il primo modello visto
- statistical multiplexing: condivido la capacità comunicativa, quindi c'è il merge degli m originali in un unico flusso, quindi il modello sarà il 3°.

la domanda può essere come i due diversi approcci si comportano da un punto di vista del tempo medio di risposta: nel caso di una

M/M/1, abbiamo $E(T_S) = \frac{\rho E(S)}{1-\rho} + E(S) = \frac{E(S)}{1-\rho}$, da cui $E(T_S) = \frac{1}{\mu-\lambda}$.
Nei 3 casi,tutti i centri sono M/M/1, quindi per $E(T_S) = \frac{m}{m\mu-\lambda}$ nel caso del FDM. Nel caso dello statistical multiplexing abbiamo $E(T_S) = \frac{1}{m\mu-\lambda}$, quindi FDM ha un tempo medio di risposta m volte più grande dello SM.
Ma il FDM da una garanzia di qualità per ogni flusso, quindi se abbiamo una QoS da rispettare, con il FDM siamo tranquilli di poter garantire ad ogni flusso una frequenza di servizio che è μ, mentre nel caso dello SM non c'è questa sicurezza.
Inoltre, se noi avessimo dei flussi molto regolari (non di Poisson), andando a fare il merge si perde questa regolarità: andiamo ad introdurre unac erta variabilità, che potrebbe non andare bene per applicazioni che richiedono una bassa variabilità (es: traffico voce o video).
Rimane il confronto fra il 2° ed il 3° modello, ricorda uno degli esempi fatti inizialmente, sui server di capacità differenti, in cui volevamo un tempo medio di risposta basso. Dipende dalle assunzioni: se i job sono non preemptible:

- se la variabilità è alta, si preferisce il 2° modello. Vogliamo evitare che job grandi blocchino i piccoli
- se la variabilità è bassa, sarebbe da preferire il 3° modello. Se ρ è basso, con una capacità non contentata le risorse sarebbero sprecate

indipendentemente dall'assunzione, se l'obiettivo è minimizzare il tempo medio di risposta ed è possibile fare preemption, allora possiamo sempre simulare il comportamento del 2° sistema col 3°: possiamo interrompere un job grande all'arrivo di uno piccolo, diamo servizio a quest'ultimo e poi riprendiamo; tutti i job usufruiscono di capacità elaborativa maggiore e c'è anche fairness.
Vediamo tutto questo nell'analisi:

- tempo di attesa: $E(T_Q)_{Erlang} = ¶_Q frac E(S) 1 - \rho$, mentre $E(T_Q)_{KP} =$

$\frac{\rho E(S)}{1-\rho}$

- da un punto di vista del tempo di risposta $E(T_S)_{Erlang}$, dobbiamo sommare il tempo di servizio che è $E(S_i)$, perché $E(T_S)_{Erlang}$ modella il tempo che passa da quando il job arriva a quando prende servizio presso uno dei job; a quel punto sarà servito con tasso $\frac{1}{\mu}$. Otteniamo quindi: $E(T_S)_{Erlang} = \frac{P_Q E(S)}{1-\rho} + E(S_i)$ ed $E(T_S)_{KP} = \frac{\rho E(S)}{1-\rho} + E(S)$, notare che $E(S_i) = \frac{1}{\mu} = m\frac{1}{m\mu} = mE(S)$. Confrontando i due termini, il risultato dipenderà dal valore di ρ:
  - $\rho \longrightarrow 1$, le prestazioni tendono a convergere, i server saranno sempre molto pieni quindi approssimativamente hanno lo stesso tempo di risposta
  - $\rho \longrightarrow 0$ la configurazione 2° da prestazioni m volte più lente

dipende quindi tutto dall'obiettivo che si ha.
Dal confronto fra servente singolo e multi-server (con m = 5), emergono alcuni risultati:

- Tempo di attesa: al tendere di ρ ad 1, le due curve tendono allo stesso asintoto verticale.
- Tempo di risposta, le cose cambiano completamente: ATTENZIONE: nel tempo di risposta si somma la tempo di attesa il tempo di servizio che il job dovrà mediamente spendere. Nel caso del multi-server, per avere una capacità globale equivalente, questo tempo è m volte più lento ed è quello che pesa di più nel requisito di qualità[1]

3.12.2 Fattore di scala

Cosa succede se le specifiche del nostro problema vengono aumentate entrambe di un fattore a: $\lambda \to a\lambda$ e $\mu \to a\mu$, Abbiamo visto che i

[1] in questo caso il tempo di risposta più piccolo

tempi si riducono dello stesso fattore a. Consideriamo sempre i due sistemi di prima, siamo sempre a parità di ρ:

- tempo medio di attesa: $E(T_Q)_{m,a} = \frac{P_Q E(S)_{m,a}}{1-\rho} = \frac{P_Q}{ma\mu(1-\rho)} = \frac{1}{a} \cdot \frac{P_Q E(S)_{m,1}}{1-\rho} = \frac{1}{a} \cdot E(T_Q)_{m,1}$
- tempo medio di risposta

3.12.3 Confronti numerici

Confrontiamo ancora tutti e 3 i modelli, per confermare con i numeri le prestazioni:
FDM: $\lambda = 4j/s, m = 4, \mu = 1.5j/s, E(S) = 0.666667s$, abbiamo un $\rho = 0.666667$, assumiamo tempo di servizio esperienziale, otteniamo un $E(T_S) = \frac{1}{\mu-\lambda} = 2s$[2], mentre per $E(T_Q) = \frac{\rho E(S)}{1-\rho} = 1.3336s$.
Confrontando con il multi-server, abbiamo un $E(S)00.1667$, stesso ρ, con la Erlang otteniamo $p(0) = [(\sum_{i=0}^{3} \frac{(4\rho)^i}{i!}) + \frac{(4\rho)^4}{4!(1-\rho)}]^{-1} = 0.059857$.
La $p(0)$ è la probabilità che il centro sia completamente vuoto, sia in coda che in servizio, quindi da un punto di vista stazionario se osserviamo il sistema per un periodo di tempo abbastanza lungo, la percentuale di tempo in cui lo osserviamo vuoto è circa del 6%. Otteniamo quindi una $P_Q = \frac{(4\rho)^4}{4!(1-\rho)} p(0) = 0.37847$, ovvero la probabilità che tutti i centri siano pieni è del 38% circa, $E(T_S) = \frac{P_Q \cdot E(S)}{1-\rho} + E(S_i)$, sommo $E(S_i)$, misura da quando un job arriva in coda a quando prenderà servizio e quando lo farà spenderà in media $E(S_i)$ e non $E(S)$, infine $E(T_Q) = 0.189292s$.
Quindi ora concentriamo anche gli mμ: $m\mu = 4 \cdot 1.5j/s, E(S) = 0.166667s$, usando la KP calcoliamo $E(T_S) = \frac{1}{m\mu-\lambda} = 0.5$, $E(T_Q) = 0.3334s$. Quindi nel confronto finale, il tempo minimo di attesa è nel multi-server, poi si passa al concentrato ed infine al FMD, in cui abbiamo garanzie sul QoS a scapito di prestazioni decisamente peggiori.

[2] dove il λ è quello del singolo centro, quindi $\frac{\lambda}{4}$

Da un punto di vista del tempo di attesa le cose cambiano: il 3° modello ha il minimo, poi abbiamo il multi-server e poi il 1° caso.

4 Processi di Markov

4.1 Storia di Google

L'algoritmo dietro al motore di ricerca è PageRank: l'idea è stata quella di risolvere il problema ricorsivamente, usando un formalismo modellistico che assegnava un rank (peso) ad una pagina a seconda di quant'era la somma dei punteggi che avevano le pagine che la puntavano. Ogni pagina aveva un rank, che dipendeva dai ranking di quelle che vi puntavano. Rank j $\approx prob\pi_j$, dove $\pi_j = \sum_{i=1}^{n} \pi_i$.
L'algoritmo prevedeva la creazione di una catena di Markov a stati discreti dove ogni stato era un pagina web con connessioni per le pagine linkate.
Se una pagina i è collegata a k pagine, la probabilità di spostarsi da quella pagina ad una delle k, la probabilità di sposarsi era uniforme $\frac{1}{k}$.
Risolvendo la catena, le pagine erano presentate in base alle loro probabilità limite, quindi in ordine in base al loro rank. Ci sono una serie di problemi se la cosa fosse risolta solo così:

- le pagine possono essere facilmente create, quindi si potrebbero creare 1000 pagine che puntano alla loro
- pagine trap, ovvero che non puntano ad altre pagine

ci sono quindi una serie di raffinamenti che tengono conto di tutti i problemi

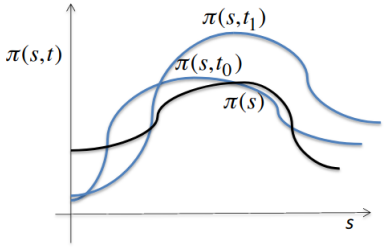
4.2 Elementi per un processo di Markov

Processo stocastico: possiamo definirlo come un insieme di variabili aleatorie che ha come indice il parametro tempo, ne consideriamo

varie istanze (outcome): $\{X(t_1), X(t_2), ...\}$
Lo spazio degli stati ci dice i valori ammissibili che la famiglia di v.a può assumere: $\{s_0, s_1, ...\}$
La catena di Markov è un particolare processo stocastico che gode della stessa memoryless property per cui l'evoluzione delle v.a nel tempo non determinano nulla rispetto all'evoluzione futura, ma l'insieme informativo è tutto contenuto nello stato all'istante t. Ne definiamo la probabilità come : $P\{X(t_{n+1}) = x_{n+1}|X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, ..., X(t_0) = x_0\} = P\{X(t_{n+1}) = x_{n+1}|X(t_n) = x_n\}$. Siamo molto interessati alle distribuzioni a fasi, ad esempio alla Cox, in quanto abbiamo diversi stati che ci rappresentano le fasi di servizio, che è esponenziale e quindi può essere modellata con Markov. Se il job esce, bisogna vedere cosa accade, comunque il sistema transita in un nuovo stato, se invece il jbo continua, il processo stocastico che modella l'evoluzione del servizio, passa da uno stato esponenziale ad un altro sempre esponenziale; quindi il processo stocastico continua ad essere di Markov, pur in presenza di distribuzioni del servizio così generali.

Distribuzione di probabilità istantanea definita come $P\{X(t) = s_i\} = \pi(s_i, t)$. Si dimostra che quando lo spazio degli stati è finito, il processo è irriducibile[3] ed ergodico, esiste $\lim_{t\to\infty} \pi(s_i, t) = \pi(s_i)$.

Probabilità stazionaria immagino che lo spazio degli stati sia finito, li poniamo sull'asse x, sulle ordinate abbiamo la probabilità dello stato al tempo t

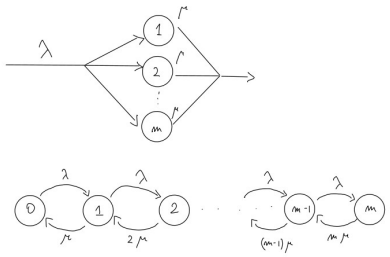


[3]irriducibile: non ci sono stati da cui non è possibile uscire, quindi al limite la probabilità dello stato sarebbe 1

Studiamo il sistema e capiamo le condizioni iniziali[4], poi al tempo t_1, notando che la probabilità è cambiata, perché magari il sistema evolve in un certo modo. Possiamo vedere dopo quanto tempo si tende ad una probabilità stazionaria, ovvero dalla quale non ci si muove più al trascorrere del tempo: anche se passa altro tempo, il sistema non cambia probabilità. Questo è fondamentale nella simulazione, nel caso in cui siamo interessati alla stazionarietà: se dobbiamo raccogliere un campione, occorre capire da quale punto in poi il campione rappresenta lo stato stazionario

4.3 Server a singolo centro con coda finita

Se la coda è piena, alcuni arrivi vengono perduti, quindi il throughput e l'utilizzazione cambiano, è necessario un processo di Markov. Da un punto di vista della modellazione, le cose cambiano: supponiamo che la capacità del sistema sia C[5], il processo X(t) rappresenta in numero di job nel centro. Siccome ho capacità finita, questo **garantisce che la stazionarietà c'è sempre**: la coda è finita, quindi del tasso di arrivo si perde ed il server si troverà a smaltire C, ed al tendere del tempo all'infinito riuscirà sempre a smaltire C:



da ogni stato si passa nel successivo con tasso λ, mentre si torna indietro di stato con tasso μ.
Come avviene la risoluzione: scriviamo stato per stato **l'equazione di bilanciamento globale**, stiamo considerando la stazionarietà:

- $\pi_0 \cdot \lambda = \pi_1 \cdot \mu$, ci deve essere un equilibrio fra i flussi che entrano e quelli che escono , altrimenti non avrei la stazionarietà, da qui

[4] ovvero al tempo t_0
[5] coda + servizio

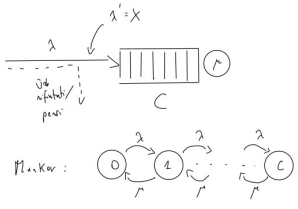
possiamo ricavare $\pi_1 = \frac{\lambda}{\mu}\pi_0$

- $\pi_1(\lambda+\mu) = \pi_0\lambda+\pi_2\mu$, qui le cose cambiano, in quanto nello stato 1 posso entrare ed uscire in diversi modi. Anche qui, possiamo ricavare $\pi_2 = (\frac{\lambda}{\mu})^2\pi_0$

quindi avremo: $\pi_C = (\frac{\lambda}{\mu})^C\pi_0$, e per il generico i $\pi_i = (\frac{\lambda}{\mu})^i\pi_0$. Dobbiamo avere che $\sum_{i=1}^{C}\pi_i = 1$ (condizione di normalizzazione), da qui possiamo ricavare $\pi_0 = \frac{1}{\sum_{i=0}^{C}(\frac{\lambda}{\mu})^i}$. Qual è la probabilità di perdita: $\pi_C = (\frac{\lambda}{\mu})^C\pi_0 = \frac{(\frac{\lambda}{\mu})^C}{\sum_{i=0}^{C}(\frac{\lambda}{\mu})^i}$, quindi il tasso di ingresso reale è $\lambda' = \lambda(1 - p_{loss})$ da cui $\rho = \frac{\lambda'}{\mu}$.
esempio: C = 4, $\lambda = \mu = 5j/s$, la capacità finita fa sì che il sistema si stabilizzi sempre. Da quanto visto prima, abbiamo che $\pi_i = (\frac{\lambda}{\mu})^i\pi_0$, tutti i π_i saranno pari ad $\frac{1}{5}$ (andando ad intersecare con la condizione iniziale $\sum_{i=0}^{4}\pi_i = 1$. La $p_{loss} = \pi_4 = \frac{1}{5}$, da cui $\lambda' = \lambda(1-p_{loss}) = 4 = X$ e $\rho = 0.8$

4.4 M/M/m/m - m server loss system

Abbiamo un certo numero di risorse parallele, senza alcun buffer delle richieste: quindi, o le richieste trovano subito un server libero, oppure si perdono (ad esempio: chiamate telefoniche o connessioni con circuiti virtuali). Ipotizziamo una distribuzione del servizio esponenziale, la soluzione (di Erlang) usa le catene di Markov:



a seconda del livello di occupazione del server, il tasso di servizio

sarà pari a $i\mu$. Abbiamo $\pi_i = (\frac{\lambda}{\mu})^i \frac{1}{i!}\pi_0$, unendo con la somma delle probabilità $\sum_{i=0}^{m} \pi_i = 1$ otteniamo $\pi_0 = \frac{1}{\sum_{i=0}^{m} (\frac{\lambda}{\mu})^i \frac{1}{i!}}$, da cui otteniamo $\pi_i = \frac{(\frac{\lambda}{\mu})^i}{i! \sum_{j=0}^{m} (\frac{\lambda}{\mu})^j \frac{1}{j!}}$, tale formula prende il nome di Erlang-B, lo stato m è la probabilità di perdita, ovvero la percentuale di tempo in cui il sistema perderà richieste in ingresso. Confrontando con la Erlang-C, a parità di λ, μ, m: $\pi_m = (\frac{\lambda}{\mu})^m \frac{1}{m!}\pi_0$ vs $P_Q = \frac{(m\rho)^m p(0)}{(\frac{\lambda}{\mu})^m m!(1-\rho)} = \frac{(\frac{\lambda}{\mu})^m}{m!(1-\rho)p(0)}$. Ora, la parte $\frac{(\frac{\lambda}{\mu})^m}{m!(1-\rho)}$ è sicuramente maggiore di $\frac{(\frac{\lambda}{\mu})^m}{m!}$, ed intuitivamente, la presenza della coda rende $\pi_0 > p(0)$.
esempio: $\lambda = 5j/s$, $\mu = \frac{1}{300}j/ms = 3.33333j/s$, $m = 4 \Rightarrow m\mu > 12j/s$, abbiamo una $p_{loss} = \pi_4 = 0.048$, $\lambda' = \lambda(1 - p_{loss}) = 4.76j/s$ con un $\rho = \frac{\lambda'}{\mu} = 0.357$.

5 Introduzione alla simulazione

Cambia il tipo di approccio per la valutazione delle prestazioni, per caratterizzare un modello di simulazione ci sono diverse tipologie, tutti i sistemi hanno una qualche componente stocastica, differenza fra modello statico e dinamico: nel primo la variabile tempo non è significativa, mentre nel secondo si. Il caso di nostro interesse è un sistema con componente stocastica, dinamica e discreta. Come si sviluppa un modello: non ci sono tecniche automatiche, ce ne sono per delle classi particolari e limitate, in generale si parla di "arte" dello sviluppo dei modelli di valutazione. Occorre seguire una serie di passi, saltarne alcuni porta al fallimento dello sviluppo della valutazione. Quali sono i primi passi:

- obiettivi dello studio: sono fondamentali, possono essere espressi come decisioni booleane, numeriche etc...
- sviluppo del modello concettuale

- convertire il modello concettuale in modello delle specifiche
- convertire il modello delle specifiche nel modello computazionale: il modello computazionale sarà lo sviluppo del simulatore se si parla di modello simulativo, altrimenti un solutore della catena di Markov se siamo in un modello analitico
- verifica
- validazione

Sul primo punto, gli obiettivi non devono condizionare le scelte nella fase di sviluppo del modello.
Ci sono 3 livelli di modelli:

i modello concettuale: a livello alto, occorre cercare di definire quali sono le variabili di stato essenziali per quel sistema sotto quegli obiettivi. Quali sono le variabili di stato possono essere ignorate per semplificare il modello, come le variabili sono collegate fra loro

ii modello delle specifiche: viene fatta sostanzialmente su carta, può coinvolgere pseudocode, equazioni etc... Cosa riceverà il modello come input? Raccogliamo ed analizziamo l'input usando dei modelli stocastici rappresentativi. Vedremo più che altro le tecniche, occorre capire

iii modello computazionale: sviluppo del codice che serve per la soluzione, un'altra filosofia è se usare linguaggi general purpose o specifici per la simulazione: il simulatore è un oggetto relativamente complesso, ha una serie di debolezze a cui occorre prestare attenzione, gli strumenti "black box" sono rischiosi; quindi si sceglie un linguaggi general purpose.

Verifica vs validazione la verifica occorre per capire se, rispetto al modello che avevo, ciò che ho implementato è corretto o no. Ma ciò non mi dice nulla rispetto al sistema reale, ovvero se il modello è una

buona rappresentazione di quel caso di studio oppure no. Verifica: ho costruito il modello correttamente? Validazione: ho costruito il modello giusto rispetto a quel sistema?
Il processo è iterativo, alcune osservazioni:

- cerchiamo di avere modelli quanto più semplici possibili, ovvero che non ci sia un modello più semplice di quello trovato, che comunque non deve né ignorare delle caratteristiche, né includerne troppe
- lo sviluppo non è sequenziale, se si lavora in team si possono fare degli step in parallelo. ATTENZIONE: non si mischiano verifica e validazione
- NON si saltano i passi per passare alla programmazione

C'è una seconda parte dell'algoritmo, che riguarda la progettazione degli esperimenti:

- progettazione di esperimenti di simulazione
- fare delle run, conservando dati di input e condizioni iniziali
- analisi dell'output: dai risultati ottenuti, occorre tirare fuori delle linee guida, verso che tipo di soluzioni andare. L'analisi statistica dei risultati della simulazione è più difficile della statistica classica: viene meno l'indipendenza fra le v.a, occorre che il campione abbia delle caratteristiche che permettono di applicare i metodi statistici
- prendere delle decisioni: dal passo precedente, bisogna decidere cosa fare. Prima di mettere in atto le decisioni, potremmo simulare quali sarebbero le conseguenze delle azioni che stiamo pensando di applicare, potremmo modificare il simulatore per poter includere le politiche che migliorano le prestazioni del sistema
- documentazione dei risultati: la simulazione è uno studio prezioso, costoso, quindi è un know-how prezioso. **Se non è ben documentato, so cazzi:** importante avere una documentazione ben

fatta sia del codice, sia sulle linee guida generali che sono state raggiunte

5.1 Machine stop model

Problema di failure di macchine. 150 macchine identiche che operano continuativamente:

- 8 h/giorno
- 250 g/anno

Operano indipendentemente, vengono riparate se falliscono. Guadagno è di 50000 €/h per macchina. Per la riparazione, viene assunto un servizio tecnico:

- contratto di 2 anni, 60000 €/anno
- ogni tecnico lavora 230 g/anno per 8 h/giorno

Quanti tecnici dovrebbero essere assunti per massimizzare il profitto? È utile pensare ai casi limite:

- unico tecnico: l'overhead del servizio tecnico è minimizzato, ma i tempi di attesa per la riparazione cresceranno moltissimo, quindi ci doverebbe essere una grande perdita di profitto
- numero di tecnici pari al numero di macchine: overhead tecnico enorme, è probabilmente il minimo caso di donwtime, il profitto dovrebbe massimizzarsi, ma c'è l'overhead dei tecnici

Modello concettuale il modello concettuale: deve sicuramente includere:

- lo stato di ogni macchina
- lo stato di ogni tecnico
- dovrà dare lo stato del sistema a ciascun istante di tempo

modello ibrido: ci sono degli elementi del modello analitico
A livello di specifiche, occorre chiedersi alcune cose:

- sappiamo dopo quanto tempo una macchina si guasta di nuovo? I guasti sono random? Se lo fossero, il tempo inter-guasto sarebbe esponenziale
- sappiamo qualcosa sui tempi di riparazione? C'è una distribuzione che rappresenta bene i tempi?
- come verrà simulata la distribuzione temporale

Modello computazionale nel modello computazionale, dovrà esserci una struttura dati per le macchine guaste ed anche per i tecnici disponibili. Dovranno anche essere incluse tutte le strutture dati per collezionare la variabili di output.

Verifica la fase di verifica è una tipica attività di ingegneria del software, solitamente viene fatta mediante testing, è possibile anche usare dei controlli di consistenza.

Validazione la fase di validazione tende a stabilire quanto il modello deciso ed implementato è rappresentativo del sistema reale: la maniera migliore di validare un modello, sarebbe quella di confrontarsi con il caso reale ma non è sempre possibile. Quindi, si possono eseguire controlli di consistenza, per verificare che il simulatore abbia lo stesso comportamento del sistema reale.
Se il caso di studio lascia pensare (rispetto al modello delle specifiche) che la soluzione analitica è coerente, quindi ha un certo livello di affidabilità, lo stesso modello analitico potrebbe anche costituire in parte una validazione: se il simulatore da la stessa soluzione, possiamo fidarci che il simulatore è abbastanza rappresentativo.

Progetto degli esperimenti l'obiettivo è quello di trovare l'ottimo numero di tecnici, quindi occorrerà fare riferimento a valori parametrici. Partiamo da alcune condizioni iniziali, ad esempio che inizialmente tutte le macchine all'istante 0 sono operative.
A questo punto, per ogni set di parametri bisogna capire quanti run di simulazione occorrono: un unico run rappresenta uno scenario particolare.

Runs la gestione dell'output può diventare un problema, una simulazione produce una quantità di dati molto grande. Evitare di memorizzare dati grezzi, siccome la simulazione è replicabile, è possibile ripetere un particolare esperimento di simulazione.

Analisi dell'output è più complicata dell'analisi statistica classica (dove avevo il mio simple random sample con le sue proprietà), se stiamo guardando il numero di macchine guaste durante una simulazione su un intervallo di tempo significativo è abbastanza probabile che due osservazioni successive siano correlate: rispetto a quello che dovrebbe essere il valore medio, potrebbero essere entrambe sopra o sotto la media.
Due osservazioni saranno vicine temporalmente ed avranno una certa correlazione

Fase di decisione la rappresentazione grafica dell'output della simulazione aiuta, ad esempio abbiamo un grafico in cui da un lato c'è il profitto, dall'altro il fatto che avere pochi tecnici porterebbe ad avere più macchine guaste. Ci immaginiamo una curva che parte bassa, poi sale e poi scende: possiamo vedere l'ottimo (se è un massimo assoluto) ed anche la sensitività negli intorni di quest'ottimo

Documentazione dovrà includere una visione del diagramma del sistema, che sia comunicabile anche a non esperti, tutte le tabelle

grafici, il software, le assunzioni e la descrizione dell'analisi dell'output. Un'altra caratteristica interessante della modellazione è che ci permette di capire delle cose che altrimenti non sarebbe possibile ottenere.

6 Simulazione trace-driven

Vediamo l'applicazione al caso di studio 1, introdotto nella 5.1. Partiamo con terminologia:

- coda/centro/nodo sono sinonimi
- job/richiesta/utente sono sinonimi

da un punto di vista di tempi:

- tempo di attesa, come usato fin ora, rappresenta il tempo che passa dall'istante in cui arriva finché non prende servizio. Nel libro, il tempo di attesa è indicato con *delay*, ovvero in ottica del ritardo che subisce la richiesta
- il tempo di risposta viene chiamato *wait*

Per un job i, in un centro con singolo server e singola coda teniamo traccia di:

- tempo di arrivo a_i
- tempo di servizio s_i
- delay in coda d_i
- istante in cui comincia il servizio: $b_i = a_i + d_i$
- tempo di wait: $w_i = d_i + s_i$
- tempi di partenza: $c_i = a_i + w_i$ (completamento)

a_i, s_i sono variabili di input, tutte le altre sono variabili di output. Il tempo di inter-arrivo fra i job j_{i-1} e j_i è indicato come $r_i = a_i - a_{i-1}$, dove per definizione $a_0 = 0$, assumiamo che non ci siano bulk arrivals, ovvero che ad ogni istante di tempo ci sia un solo arrivo (probabilità di attivi simultanei prossima allo 0), tutti gli $r_i > 0 \forall i$.

6.1 Trace driven simulation

Il modello è guidato dai dati esterni di input, noi abbiamo a_i ed s_i, come possiamo calcolare il tempo di coda d_i: per alcune discipline di coda, non è semplice. Ma se ci limitiamo al caso FIFO, d_i è determinato da quando a_i avviene rispetto a c_{i-1}, ovvero rispetto all'ultimo completamente. A seconda se avvenga prima o dopo, ci saranno risultati diversi:

- il job arriva prima che il precedente completi, ovvero $a_i < c_{i-1}$
- se invece $a_i > c_{i-1}$, il job i-esimo prende subito servizio (perché trova il centro vuoto) e quindi il tempo di delay è $d_i = 0$.

Queste equazioni vengono determinate a livello di specifiche e modello concettuale, in cui si capiscono le relazioni.

6.1.1 Statistiche di output

Da un punto di vista di definizione analitico, le statistiche di output si dividono in due tipologie:

- job average: il tempo medio di inter-arrivo $\bar{r} = \frac{1}{n}\sum_{r_i} = \frac{a_n}{n}$, quindi il tasso di arrivo (λ) è $\frac{1}{\bar{r}}$. Da un punto di vista empirico, è abbastanza chiaro quant'è il tempo di inter-arrivo medio: prendo il numero di arrivi avuti fino a quel momento e divido per n.
  Analoga operazione per il tempo di servizio medio: $\bar{s} = \frac{1}{n}\sum_{s_i}$.
  L'attesa e la risposta media: $\bar{d} = \frac{1}{n}\sum_{d_i}$ e $\bar{w} = \frac{1}{n}\sum_{w_i}$. Poiché $w_i = d_i + s_i$, la relazione vale anche per le medie. Quindi, dai dati in input è possibile calcolare le medie, e viste le relazioni di dipendenza basta calcolarne due e derivare la terza. Ma sarebbe buona norma calcolare tutti i valori indipendentemente e poi fare la verifica come controllo di consistenza usare la relazione.
- time average: statistiche legate alle popolazioni:

– l(t): popolazione media nel sistema al tempo t
– q(t): popolazione media nella coda al tempo t
– x(t): numero medio di job in servizio al tempo t

per definizione: l(t) = q(t) + x(t) $\forall t$. Sono tutte funzioni a gradino, siccome assumono valori interi.

Supponiamo di guardare un intervallo $(0, \tau)$:

- il numero medio nel tempo in coda è dato da $\bar{l} = \frac{1}{\tau} \int_0^\tau l(t)dt$
- media nel tempo in coda $\bar{q} = \frac{1}{\tau} \int_0^\tau q(t)dt$
- media nel tempo in servizio $\bar{x} = \frac{1}{\tau} \int_0^\tau x(t)dt$, è anche l'utilizzazione.

  Per definizione di utilizzazione, è chiaramente minore di 1

sono tutte collegate con la legge di Little: come visto nella 3.3, le ipotesi valgono e quindi abbiamo: $\int_0^{c_n} l(t)dt = \sum_{i=1}^{n} w_i$, dove $c_n = \tau$. Lo stesso si può dire a livello di coda e di servente: $\int_0^{c_n} q(t)dt = \sum_{i=1}^{n} d_i$, $\int_0^{c_n} x(t)dt = \sum_{i=1}^{n} s_i$, dove $c_n = \tau$. Se noi andiamo a scriverci la funzione cumulativa degli arrivi e la funzione cumulativa delle partenze, abbiamo una corrispondenza con il tempo che ogni job richiede: quando calcoliamo l'aera, stiamo calcolando il tempo di risposta del job. Quindi integrando su tutto il tempo equivale a fare la somma di tutti i tempi di risposta per tutti i job.
τ corrisponde a c_n, quindi sostituendo: $\bar{l} = \frac{1}{\tau} \int_0^\tau l(t)dt = c_n\bar{l} = \int_0^{c_n} l(t)dt$, usando Little abbiamo $\sum_{i=1}^{n} w_i = n\bar{w}$, da cui $\bar{l} = \frac{n}{c_n}\bar{w}$ lo stesso vale per $\bar{q} = \frac{n}{c_n}\bar{d}$, $\bar{x} = \frac{n}{c_n}\bar{s}$.

$\frac{n}{c_n}$: n job completano in c_n, quindi otteniamo il throughput in c_n; nel caso della coda infinita, corrisponde alla frequenza media di arrivo.

Intensità di traffico il rapporto fra la frequenza di arrivo e la frequenza di servizio $\frac{\frac{1}{\bar{r}}}{\frac{1}{\bar{s}}} = \frac{\bar{s}}{\bar{r}} = \frac{\bar{s}}{\frac{a_n}{n}} = (\frac{c_n}{a_n})\bar{x}$ questo poiché $\bar{x} = \frac{x}{c_n}\bar{s}$. Questo per dire che la frequenza di servizio coincide con l'utilizzazione soltanto quando il rapporto $\frac{c_n}{a_n}$ tende ad 1, ma per periodi di tempo piccoli (quindi piccole finestre temporali) l'utilizzazione coincide con l'intensità di traffico quando il tempo di simulazione è sufficientemente lungo.

6.1.2 Algoritmo

Algoritmo:

```
c_0 = 0.0;
i = 0;
while (more jobs to process){
        i++;
        ai = GetArrival();
        if (a_i < c_{i-1}) d_i = c_{i-1} - a_i;
        else  d_i = 0.0;
        s_i = GetService();
        c_i = a_i + d_i + s_i;
        }
        n = 1;
        return  d_1, d_2, ..., d_n;
```

GetArrival() e *GetService()* leggeranno da file, prendendo i primi tempi. Possiamo calcolare il tempo di inter-arrivo medio, il tempo di servizio medio etc... a partire dai dati e dai valori dei d_i risultanti dopo la simulazione. È possibile calcolare la media dei delay $\bar{d}_i$, siccome occorre dover effettuare la verifica e questo può essere

complicato, un approccio può essere quello di calcolare i valori e poi usare l'equazione $\bar{w} = \bar{d} + \bar{s}$ per fare la verifica.

6.1.3 Case study - gelateria

Una gelateria vuole offrire delle varianti di gusti, ma vuole sapere quanto il tempo in più necessario a prepararli possa impattare sui tempi di coda. Viene usato un modello con singolo servente, il file dati ssq1.dat rappresenta i tempi di attesa per 1000 clienti, contenente le coppie istante di arrivo - tempo di servizio. I dati danno il così detto punto di stima (con le relative statistiche), ma è uno scenario: i 1000 clienti sono arrivati in quel preciso tempo ed hanno chiesto quel determinato tempo. Per lo studio, i tempi di servizio vengono incrementati e decrementati sistematicamente di un fattore moltiplicativo. Per il file di partenza, abbiamo un fattore di utilizzazione minore di 1, infatti 1-x (che è la probabilità che il centro sia vuoto) è circa del 28%. A dispetto di questa probabilità, il numero medio in coda $\bar{q} = 2$, quindi non trascurabile: c'è una relazione fra l'aumento del tempo di servizio e la dimensione della coda.
Linee guida:

- se i dati non hanno delle incertezze ed il risultato è regolare, è possibile unire i punti, ma i dati originali vanno mantenuti
- se la curva non è così regolare probabilmente occorrono più punti
- se i punti corrispondono a dati incerti, non fare interpolazione, è approssimativo
- se i dati sono inerentemente discreti, mai fare interpolazione. È un errore grave

6.2 Caso di studio 2

L'oggetto di studio è un magazzino, in particolare l'inventario: il magazzino riceve una serie di domande da utenti per il tipo di merci

che vende e che, nel momento in cui le merci fossero poche per soddisfare gli ordini, sarebbero rifornite. Gli utenti acquistano un certo numero di articoli, i quali arrivano dai fornitori una volta che gli ordini vengono evasi e le merci vengono inviate.
La domanda di merci è discreta, consideriamo un unico tipo di merce per semplificare il case study.

Obiettivi trovare una politica per l'inventario per minimizzare i costi. Vi sono due categorie di politiche:

- quando il numero di merci nel magazzino varia, viene rifatta un analisi dello stato dell'inventario
- politica periodica, indipendente dalla variazione delle merci

I pro ed i contro: nel primo caso, il costo è maggiore in quanto occorre verificare il livello dell'inventario ed agire di conseguenza. Il pro è che in questo modo verranno minimizzati i periodi in cui il livello merci è basso, la *short age*[6] dovrebbe essere minimizzato. Nel caso periodico, l'epoca di revisione è periodica, per esempio settimanale. Gli articoli vengono ordinati se necessario solo all'epoca della revisione, ad esempio il lunedì mattina. Si usano due parametri: il livello minimo di merci s ed il livello massimo S(che il magazzino può ospitare), chiaramente avremo $0 \leq s \leq S$.
Scegliamo la politica due, l'obiettivo dello studio diventa la ricerca della coppia (s,S) che minimizza i costi.

Modello concettuale definiamo tutti i costi di sistema:

- Holding cost: costo del mantenimento delle merci nel magazzino, legato al costo dello spazio
- Shortage cost: costo da pagare se non si risponde alla domanda
- Setup cost: costo fisso quando si effettua un ordine

[6] mancanza di merci

- Item cost: costo per articolo
- Ordering cost: somma dei costi di setup e di items

A livello concettuale, lo stato del sistema è dato:

- dal livello dell'inventario l
- dalla quantità di merce ordinata o
- dalla quantità di merce richiesta dall'utente d

CI sono poi una serie di assunzioni da fare:

- arriva un ordine che ha una richiesta maggiore del livello merci, per essere evaso dovrebbe portare il livello ad un valore negativo; viene chiamato back ordering
- delivery lag: tempo che passa da quando qualcuno ordina le merci a quando queste arrivano. Lo assumiamo nullo (ma manco per nulla è nullo)
- il livello di scorte iniziale è S
- il livello dell'inventario al termine dello studio è ancora S, questo fa si che abbiamo un bilanciamento dei flussi per tutt il tempo dello studio

Modello delle specifiche fissiamo le variaibli a livello di specifica:

- il tempo comincia all'istante 0
- gli istanti di revisione sono t=0,1,2... che sono le settimane
- l_{i-1} è il livello dell'inventario all'inizio della settimana i
- o_{i-1} è la quantità ordinata dal tempo t= i-1
- alla fine dell'intervallo i-esimo, l'inventario è diminuito di d_i: $I_i = I_{i-1} + o_{i-1} - d_i$
- l'inventario alla fine della settimana può essere negativo

Il livello dell'inventario viene revisionato al tempo t = i-1, se la quantità è almeno s non viene fatto nessun ordine, altrimenti viene fatto un ordine per riportare il livello di scorte ad S. Le statistiche che calcoliamo sono $\bar{d}_i$ e $\bar{o}_i$, la domanda media e l'ordine medio devono essere uguali per definizione. Per quanto riguarda i costi, ci sono dei punti in cui il livello del magazzino è negativo e quindi si paga la penalità per lo short age. Abbiamo due casi:

$$o_{i-1} = \begin{cases} 0 & \text{se } I_{i-1} \geq s \\ S - I_{i-1} & \text{se } I_{i-1} < s \end{cases}$$

$\bar{l_i^+}$ è la parte positiva di $\bar{l}$ nell'intervallo i-esimo, data da $\bar{l_i^+} = \int_{i-1}^{i} l_i(t)dt$, se c'è una parte negativa (dovuta allo short age) ed una positiva, queste andranno sommate: $\bar{l_i^+} = \int_{i-1}^{\tau} l(t)dt - \bar{l_i^-} = \int_{\tau}^{i} l(t)dt$ Abbiamo le rispettive statistiche: la parte positiva sarà la somma su tutti gli intervalli, stesso vale per la parte negativa. Possiamo calcolare tutte le statistiche: $\bar{d}, \bar{o}, \bar{l^+}, \bar{l^-}$, possiamo poi fare la verifica dei valori.

6.2.1 Caso di studio - concessionario d'auto

La facility qui è costituita dallo showroom e dalle automobili. Sono macchine nuove, il fornitore sarà il produttore delle macchine e per semplicità consideriamo un solo tipo di macchina.
Nel caso di studio la quantità massima S = 80, l'inventario viene rinnovato ogni lunedì, se si scende sotto s = 20, viene fatto un ordine per tornare a 80.
La politica ottima è quella che porta ad un costo medio minimo. Notiamo che $\bar{o} = \bar{d}$ e $\bar{d}$ dipende solo dalla domanda, quindi il costo di item è indipendente da (s, S). La parte che dipende da s sarà sicuramente il costo di setup medio, il costo medio di immagazzinamento ed il costo medio di shortage (andranno tutti sommati). Fissato S e la sequenza di domande, facciamo variare s e ci aspettiamo che:

- il costo di immagazzinamento cresca ed anche il costo di setup
- il costo medio di shortage decrescerà

La funzione dei costi totali avrà un andamento ad "U" e potremo osservare il minimo valore.

7 Random number generators

Tutti i case study visti fin ora hanno bisogno di dati persi dall'esterno, quindi l'utilità dei programmi è limitata dalla disponibilità dei dati:

- se non ci fossero dati?
- se il modello cambiasse?
- se i dati di input non fossero abbastanza?

Occorre un random number generator: per generare i numeri caratterizzati da un particolare distribuzione di probabilità si parte da un generatore di numeri random fra 0 ed 1.
Tra tutte le classi di generatori, siamo interessati a quelli software, che hanno diversi vantaggi. Un generatore ideale dovrebbe estrarre il valore fra 0 ed 1 (che è una probabilità), in modo che tutti i possibili infiniti valori siano equiprobabili. Un generatore "buono" produce un output che è statisticamente (slides): scegliamo un m grande, positivo $m > 0$, denotiamo un insieme $\chi_m = \{1, 2, ..., m-1\}$, immaginiamo di estrarre a caso un intero x e di dividerlo per m, ottenendo $u = \frac{x}{m}$. I possibili valori sono $\frac{1}{m}$, $\frac{2}{m}$,... È importante che m sia grande in modo che i possibili valori siano molti.
Per definizione, i due estremi 0 ed 1 sono esclusi, inoltre vorremmo che ogni volta che estraiamo dall'urna un particolare valore vorremmo avere sempre lo stesso valore di probabilità ovvero poter "rimettere" nell'urna il numero estratto. Vorremmo simulare un'estrazione con rimpiazzo, per ragioni pratiche dovremmo accontentarci di un generatore senza rimpiazzo ma se scegliamo m grande ed il gruppo

di numeri usati è più piccolo di m, il fatto che non ci sia rimpiazzo è quasi non percepibile.

7.1 Generatore di Lehmer

Definito sulla base di due parametri:

- modulo m: un primo molto grande
- moltiplicatore a: uno dei numeri in χ_m

I possibili valori $x_0, x_1, ...$ vanno da $\frac{1}{m}$ ad $\frac{m-1}{m}$, la sequenza dei numeri generati è definita da un'equazione iterativa: $x_{i+1} = g(x_i)$ con $g(x) = axmodm$; $x_0 \in \chi_m$ è chiamato seme (spesso, per convenzione, si sceglie 1). Se a ed m sono scelti propriamente, un generatore id Lehmer è statisticamente non distinguibile dal pescare da χ_m con rimpiazzo. Una buona scelta di m è l'INTMAX su sistemi a 32 bit, su sistemi a 64 bit la scelta non è così ovvia e quindi $2^{31} - 1$ continua ad essere una buona scelta. La scelta di a deve essere fatta con grande cura:

- vogliamo una sequenza full period
- inoltre, se abbiamo una sequenza full period, possono esserci sequenza migliori di altre a seconda della caratteristica di randomicità
- l'operazione $axmodm$ non può essere fatta in modo banale, perché ax potrebbe causare integer overflow.

7.1.1 Implementazione dell'algoritmo

Come abbiamo detto, è possibile che ax vada in overflow: su una macchina a 32 bit, tutti gli interi maggiori di $2^{32} - 1$ verranno persi, ma è possibile trasformare la funzione di partenza col calcolo di due funzioni, e ricavare $d(x) = \gamma(x) + m \cdot \delta(x)$, dove il prodotto ax non è solo. Inoltre, c'è un teorema che da una strada implementativa semplice:

Teorema 2.2.1 se $m = aq + r$ è primo ed $r < q$ vale che per $x \in \chi_m$: $\delta(x) = 0$ o $\delta(x) = 1$, dove $\delta(x) = \lfloor \frac{x}{q} \rfloor - \lfloor \frac{ax}{m} \rfloor$, inoltre

$$\delta(x) = \begin{cases} 0 & \text{if } \gamma(x) \in \chi_m \\ 1 & \text{if } -\gamma(x) \in \chi_m \end{cases}$$

dovremo scegliere una coppia (a,m) per cui vale la relazione del teorema.
Da qui, nell'algoritmo sapremo come generare il numero, in base al fatto che $\gamma(x)$ sia positiva o negativa in modo tale da non avere problemi di overflow.
La caratteristica per cui riusciamo a trovare m primo per cui vale il Teorema è detta modulo-compatibilità, il modulo scelto è $2^{31} - 1$, un moltiplicatore considerato lo standard è a = 48271.

7.1.2 Proprietà

- i moltiplicatori modulo-compatibili sono tutti minori di $\frac{m-1}{2}$
- sono più densi verso lo 0

quindi, basta scegliere un moltiplicatore piccolo per avere un alta probabilità che sia MC. Un moltiplicatore è "piccolo" quando il suo quadrato è minore di m, quindi tutti i moltiplicatori da 1 a $\lfloor \sqrt[2]{m} \rfloor$ sono MC. Da un punto di vista algoritmico, bisognerebbe partire da un moltiplicatore piccolo e vedere se è full period.
esempio: con $m = 2^{31} - 1$ ed un FPMC a = 7 è possibile generare altri 23093 FPMC, in particolare a = 16807 è uno standard "minimale", mentre lo standard consigliato è 48271.
Da un punto di vista della randomicità, dovremmo scegliere dei FPMC che danno delle sequenze quanto più randomiche possibili: non c'è una definizione universale di randomicità, inoltre il generatore mostra una struttura a lattice (tutti i punti si trovano su rette parallele). **WARNING:** rand(), della libreria ANSI-C fa schifo, non usarla per lavori scientifici. Non fidarsi di tool che non danno informazioni sull'algoritmo con cui vengono generati i numeri

7.2 Esempio - single server queue

Sappiamo solo che i tempi di servizio sono distribuiti fra 1 e 2 minuti, non sappiamo nulla sulla distribuzione e quindi assumiamo che sia distribuito uniformemente fra 1 e 2 sia $Unif(1,2)$. Abbiamo un generatore di Lehmer che genera valori fra 0 ed 1, abbiamo bisogno fi una funzione che ci trasformi questo valore in uno che modella il tempo di servizio. Partendo dal valore, otteniamo x per cui il valore della densità di probabilità è proprio u, la trasformazione è $x = -\mu ln(1-u)$, ovvero della trasformazione esponenziale e dove μ è la media campionaria. Potremmo usare quindi in questo esempio l'esponenziale per i tempi di inter-arrivo, ovvero generando gli istanti di arrivo come $a_i = a_{i-1} + Exponential(\mu); i = 1, 2, 3..., n$, per i tempi di servizio s_i scegliamo la $Uniform(1.0, 2.0)$. L'uniformità de tempi di servizio non è molto diffusa, molto più spesso ci sono distribuzioni monotòne decrescenti, come le esponenziali.
esercizio: calcoliamo teoricamente le statistiche di output che vengono prodotte con la simulazione: abbiamo un centro a coda infinita, servente singolo. Scriviamo i parametri:

- $\lambda = \frac{1}{2} j/s$, di tipo esponenziale
- E(S) = 1.5 s, unif (1,2)
- con Little, abbiamo ricavato $E(T_Q) = \dfrac{\frac{\lambda}{2}E(S^2)}{1-\rho}$

per l'uniforme, in generale, la media è $E(S) = \frac{b+a}{2} = \frac{3}{2}s$, la varianza è $\sigma^2(S) = \frac{(b-a)^2}{12} = \frac{1}{12}$.
Per conoscere $E(S^2)$, poiché conosciamo sempre media e varianza delle v.a, possiamo scrivere $E(S^2) = \sigma^2(S) + E(S)^2$, da cui $E(S^2) = \frac{7}{3}$.
Possiamo quindi ricavare $\rho = \lambda E(S) = 0.75$ ed $E(T_Q) = 2.\bar{3}$.
Ora possiamo ricavare $E(T_S) = E(T_Q) + E(S) = 1.5 + 2.\bar{3} = 3.8333s$ e possiamo usare Little per le popolazioni:

- $E(N_Q) = 1.1667$

- $E(N_S) = 1.9167$

dalla simulazione, otteniamo gli stessi risultati. Da un punto di vista delle prestazioni, nonostante il servente sia occupato per il 75% del tempo, in media ci sono 2 job nel centro, quindi in tutto un job che mediamente chiede 1.5 spende più del doppio: 2.33 in coda ed 1.5 nel servizio.
Il vantaggio della simulazione è che possiamo mandare avanti la simulazione per molti job e vedere quando le statistiche tendono a questi valori asintotici, in quanto il calcolo effettuato usa la KP, quindi per valori stabili. Notiamo come la convergenza di w al valore 3.83 è lenta e dipende dal seme iniziale.
La simulazione può essere usata sia per usare il comportamento allo stato stazionario, ma anche per studiare la fase transiente e quindi la parte iniziale delle curve: si decide il numero di job e si fanno diverse repliche, usando semi differenti ma mantenendo fisso lo stato iniziale del sistema.
Cerchiamo di simulare dall'inizio lo stato di stazionarietà. Sappiamo che il tempo di attesa medio in coda è 2.33, piuttosto che far partire il sistema come se fosse vuoto, partiamo con un sistema in uno stato stabile (con departure = 3):

- $a_1 = a_0 + expo(2) = 0 + 0.8 = 0.8$, confrontiamo poi arriva con departure. Il primo job va quindi in coda per un tempo $d_1 = 3 - 0.8 = 2.2$. Abbiamo un $s_1 = Uniform(1, 2) = 1.3$, con $w_1 = 2.2 + 1.3 = 3.5$ e $c_1 = 0.8 + 3.5 = 4.3$

C'è un bias nel calcolo dei dati: nella media campionaria non è così evidente, mentre nella varianza sì e andrà corretto matematicamente nel calcolo della varianza.
Vorremmo un indipendenza fra i valori di x_i, che però non c'è: è presente una correlazione seriale, i DES time-sequenced spesso presentano correlazione seriale che introduce bias sui valori.
esempio 2: i job arrivano on lo stesso tasso λ, ma stavolta i tempi di servizio modellano una situazione più articolata:

- ogni job può avere pari ad 1 + Geom(0.9)
- ogni task richiede un tempo (minuti) che è Unif(0.1, 0.2)

quindi, mediamente ci saranno 10 task per job (la media della geometrica è $\frac{1}{0.9}$ ed il tempo di servizio per singolo job sarà 1.5 (0,15·10). Sembra lo stesso caso di prima, ma ora è cambiata la varianza: abbiamo la varianza di 10 uniformi, quindi è più alta. Cambiano quindi tutte le statistiche, tranne $\bar{r}, \bar{x}, \bar{s}$, quindi le misure di performance sono sensibili alla scelta della distribuzione dei tempi di servizio.

7.2.1 Revisione dell'inventario

Come cambia il case study nel caso in cui: viene sostituita la Equilikely(a,b), usando sempre gli stessi dati. Per studiare al variare di s come cambiano le cose, non possiamo cambiare troppe cose insieme: facciamo variare s e lasciamo il resto invariato, così da capire come cambiano le statistiche di output. Qui, l'utilizzo di uno studio stazionario può sembrare inappropriato: già i due anni sembrano un ipotesi eccessiva, in quanto le cose tendono a cambiare. Il senso è che la curva è molto più smooth e quindi mostra meglio qual è il minimo per s. La variabilità nella fase di transizione ha assolutamente senso e non va eliminata, per quanto riguarda studi come quest'ultimo, in cui vogliamo solo vedere la variazione di s, ridurre la varianza vuole dire scegliere sempre lo stesso seme e cambiare solo s. Una stima viene detta robusta quando piccole oscillazioni nell'introno della stima non spostano gli obiettivi posti dallo studio.

7.2.2 Modifiche alla coda singola

Partiamo dall'esempio della coda singola, dove la *GetService()* sceglieva uniformemente fra 1.0 ed 2.0 il valore. Cambiamo il codice in modo da:

- inizializzare una volta per tutte la variabile del seme (che per questo è **static**)

- generare il tempo di servizio s
- ricavare lo stato del generatore, ovvero l'ultimo valore da cui bisogna partire

Analogo per la funzione che generava i tempi di arrivo, *GetArrival()* che potremmo cambiare in maniera similare alla *GetService()*.
Il problema è che noi vogliamo che i flussi non si sovrappongano, non sappiamo di quanti tempi di servizio abbiamo bisogno, di quanti tempi di arrivo, e quindi se poi generando questi due tempi non si sovrappongano. Non sappiamo la distanza in termini di numeri generati fra due seed, ad esempio fra 12345 e 54321: i primi valori sono diversi, ma se poi i tempi sono così tanti da superare la distanza fra i due seed, come servizi usiamo numeri fra 0 ed 1 che sono già stati usati come tempi di servizio, causando quindi un fenomeno di accoppiamento fra i servizi, che dovrebbero essere indipendenti.

7.2.3 Jump multipliers

Dividiamo il flusso in pezzi, in modo che le partizioni siano non sovrapposti, in modo che il numero di generazioni all'interno di ogni pezzo dovrà esser sufficiente per gestire i valori da generare. Prima ci siamo mossi a caso, ma occorre avere una scelta dei semi non casuale, ma fatta in modo che i flussi siano disaccoppiati: abbiamo il generatore di Lehmer $axmodm$, viene detta **funzione salto** quella che usa come moltiplicatore $a^j modm$, con $1 < j < m - 1$. $g^j(x) = (a^j modm)xmodm$ genera i valori distanziati di j, scelto j opportunamente fra 1 ed m-1, quindi a partire da x_0 genera $x_j, x_{2j}, ...$, quindi consente di passare da un flusso all'altro di lunghezza j. Quindi se j è piuttosto grande, possiamo essere sicuri che ai processi vengano assegnati flussi lunghi j che non si sovrappongono. Dobbiamo avere dei seed che siano full-period e modulo compatibili, per standard il moltiplicatore è 48271, supponiamo di dividerlo in 256 flussi. Partizioniamo la ruota dei valori in 256 flussi diversi, dobbiamo assicurarci che j sia $< \frac{2^{31}}{2^8} = 2^{23}$ in modo che il moltiplicatore salto sia

modulo-compatibile. Dobbiamo scegliere il moltiplicatore salto più grande possibile, in modo che sia modulo-compatibile.

7.3 Disaccoppiamento di processi stocastici

Possiamo provare a riscrivere il processo dei servizi come la somma di due uniformi, entrambe fra 0 ed 1.5: la media rimarrà uguale, mentre la varianza cambia. In questo modo, possiamo considerare gli stessi arrivi del caso della Unif(1.0, 2.0), possiamo prendere esattamente lo stesso flusso ed usarne un altro per i servizi "nuovi" ma sempre di media 1.5, con una variabilità che è 4.5x quella di prima. Se con un unico flusso avessimo variato anche la frequenza di arrivo, la variabilità sarebbe dovuta a più elementi che cambiavano, che non potevamo attribuire alla sola componente che volevamo cambiare. Vediamo come tutti i valori dei tempi di risposta e della popolazione media crescono, perché la variabilità influisce negativamente.

7.4 Altri esempio

7.4.1 Feedback model

Abbiamo una frequenza di arrivo al centro che viene aumentata, dato il numero di job che faranno feedback una volta finito il servizio e questo avviene con una certa probabilità. Per aggiungere il feedback, ci basta definire la probabilità con cui un job fa feedback ed a seconda del valore di ritorno, occorre gestire i job: se non contiamo due volte il job che fa feedback, ma manteniamo il vecchio indice, il calcolo delle statistiche job-avereaged non cambiano. I tempi cambiano: se un job riceve un nuovo tempo di servizio di media $\frac{1}{\nu}$, quindi tempo di attesa, di risposta, e di servizio medio del job cambiano. Anche la semplice aggiunta di un feedback rende problematica la gestione degli arrivi nella coda che si mischiano con il feedback. Occorre quindi una struttura dati che mantenga le informazioni sui job che fanno feedback.

7.4.2 Inventory system con delivery lag

Assumiamo di avere del delivery lag, che sia generato randomicamente ma che sia minore di 1, ovvero che l'ordine sia evaso prima della settimana successiva. Si assume inoltre (per avere bilanciamento dei flussi) che nella settimana finale non ci sia delivery lag.
Le statistiche $\bar{o}, \bar{d}, \bar{u}$ saranno pressoché uguali al variare del delivery lag, mentre la statistica $\bar{l}^+$ tenderà a diminuire, mentre la parte negativa $\bar{l}^-$ crescerà o rimarrà simile. Dalle curve globali che rappresentano il costo globale, si può vedere che con delivery lag l'ottimo viene shiftato a sinistra.

7.4.3 Machine shop model

Partendo dalle assunzioni fatte in precedenza, abbiamo che ogni singola macchina si guasta in media ogni 100 unità di tempo (tempi distribuiti esponenzialmente), il tecnico è unico. Il programma è sostanzialmente lo stesso, dove però gli arrivi sono le failure, possiamo valutare $M - \bar{l}$, ovvero il numero di macchine operanti come totale - media di macchine guaste, contro il numero totale di macchine. Si nota che, oltre un certo valore, il numero di macchine attive si schiaccia contro un asintoto orizzontale, quindi occorre aggiungere tecnici e non macchine.

7.4.4 Controlli di consistenza

In generale, quando si parte da un modello e si aggiunge qualcosa, è importante verificare il modello esteso con quello di partenza tramite dei controlli di consistenza. Ad esempio, per il modello con feedback, il controllo più semplice da fare può essere impostare $\beta = 0$ e verificare che i valori siano gli stessi. Poi, facendo variare di poco il valore di β, si può vedere che i valori cominciano ad aumentare. In questo caso, avere un multi-stream facilita il lavoro, vogliamo controllare l'output di due sequenze diverse senza cambiare il modello.
Anche nel caso dell'inventory system, si può verificare che con de-

livery lag pari a 0 i valori trovati corrispondano a quelli del sistema originale.

8 Next Event simulation

Fin ora, abbiamo visto una simulazione in cui il tempo non c'è: abbiamo delle tracce, generate a random o lette da un file, su cui vengono effettuati dei calcoli. Non c'è la simulazione temporale, ed è questo che rende gli algoritmi difficili: ogni volta, occorre verificare se i tempi di completamento sono più o meno grandi di quelli di arrivo etc...
L'approccio next event fa una simulazione del tempo e degli eventi che accadono a mano a mano che il tempo passa. Gli elementi chiave di una next event simulation sono:

- stato del sistema
- eventi
- un clock di simulazione
- scheduling di eventi, un ordine di eventi
- lista di eventi, in cui ci sono tutti gli eventi correnti nel sistema in quell'istante di tempo

Stato lo stato del sistema è una sua caratterizzazione completa in un istante di tempo. Sappiamo che nel definire il modello ci sono 3 livelli, lo stato va definito per ognuno dei livelli:

- livello concettuale: lo stato è un insieme di elementi astratti, anche espressi in linguaggio naturale. L'insieme di variabile è l'evoluzione del sistema nel tempo
- livello di specifiche: le variabili diventano matematiche e le relazioni matematiche fra di esse

- livello computazionale: le variabili diventano variabili di programma

per esempio in ssq lo stato è il numero di job nel sistema, per l'inventory system lo stato è la quantità di merci nel magazzino.

Eventi un evento è una occorrenza che può cambiare lo stato, per definizione lo stato può cambiare solo in presenza di eventi definiti per quel caso; ogni evento ha un suo tipo. Possiamo aggiungere agli eventi propri degli eventi artificiali, ovvero che non cambiano lo stato del sistema, ma che servono per scopi di simulazione, ad esempio un evento di campionamento per raccogliere le statistiche. Oppure, decidiamo che la simulazione deve essere un certo tempo τ, quindi arrivati a quel tempo limite si chiude il processo degli arrivi; anche questo è un evento, ma aggiunto ai fini della simulazione

Clock orologio di sistema, i simulatori visti fin ora mancavano completamente del concetto di tempo. Qui, il tempo sarà il clock di simulazione

Scheduler meccanismo di avanzamento del tempo, nel nostro caso si segue l'ordine del tempo degli arrivi, l'orologio di simulazione passa dal tempo corrente a quello dell'accadimento del prossimo evento.

Lista degli eventi struct che mantiene tutti gli istanti di tempo di occorrenza dei prossimi eventi, uno per ogni tipo.

Per costruire un simulatore next-event occorre:

- individuare l'insieme di variabili di stato
- identificare i tipi di evento
- costruire un insieme di algoritmi che definiscono il cambiamento di stato per ogni tipo di evento

Algoritmo:

- inizializzazione: si impostano il clock e una prima occorrenza per ogni tipo di evento
- passo iterativo: si scandisce la lista degli eventi, scegliendo il primo, si aggiornano clock e stato
- schedulazione di nuovi eventi: un evento di arrivo genera un evento di completamento, se il primo va subito in esecuzione, che non è detto che sia il prossimo perché magari prima c'è un altro arrivo. Quindi si schedulano nuovi eventi se sono stati generati
- termina: il clock avanza finché non si arriva alla terminazione

quindi, il clock avanza in maniera asincrona.

8.1 Estensione del modello - multi-server

Da un punto di vista simulativo, a livello concettuale il multi-server è un centro con una coda singola (potrebbe non averla, ricordando l'M/M/m loss system) ed un certo numero di serventi che operano in parallelo. Quindi, per lo stato:

- ogni server può essere libero o occupato
- la coda può essere vuota o non vuota
- se ci sono dei job in coda, questi devono prendere servizio. Non è possibile avere server vuoti e code non vuote
- se ogni coda è vuota, allora tutti i server sono pieni

Abbiamo dato per scontato che la scelta del server libero è indipendente, ma nella simulazione potrebbe aver senso cercare di vedere cosa accade se i server sono diversi fra di loro e ci sia una politica di scelta del server libero:

- random
- in ordine, numeriamo i serventi paralleli

- cyclic, vengono scelti in ordine ma dopo l'ultimo si torna al primo
- equity, si usa il meno usato in modo tale da non sovraccaricare i server.
- priority, si sceglie il migliore server idle ma bisogna definire cosa vuol dire "migliore"

8.2 Esempi di simulatori next-event

8.2.1 Single server queue

Lo stato è il numero di job nel centro al tempo t, $l(t)$. Nel tempo, lo stato varia per due eventi possibili:

- arrivo di un job, lo stato viene incrementato di 1
- completamento di un job, lo stato viene incrementato di 1

A livello di specifiche, la variabile di stato viene messa in relazione con $q(t)$ ed $x(t)$, che sono le altre variabili a livello di specifica. Quindi,

- se $l(t) = 0$, allora $q(t) = 0$ ed $x(t) = 0$
- se $l(t) > 0$ allora $q(t) = l(t) - 1$ ed $x(t) = 1$

Scegliamo di inizializzare lo stato $l(0)$ come valore non negativo, tipicamente pari a 0; lo stato terminale sarà un qualunque valore non negativo, assumiamo che al tempo τ il processo degli arrivi si fermi, in modo che i processi rimasti vengano evasi prima della terminazione per riportare il sistema allo stato finale come quello allo stato iniziale. Occorre usare una codifica per l'evento impossibile: ad esempio, per chiudere il processo degli arrivi, occorre denotare il processo come chiuso. In termini astratti, potremmo dire che un evento è impossibile se il suo tempo è ∞.
Nei due approcci, trace-driven e next-event, a parità di condizioni dobbiamo ottenere gli stessi risultati. Nel caso di ssq1 e 2, viene sempre generato un tempo arrivo e poi un tempo servizio, mentre

nel caso del next-event non è detto che venga subito evaso un completamento dopo un arrivo, in quanto può darsi che il prossimo evento sarà un nuovo arrivo, occorre quindi avere un multi-stream.
Se ci fossero discipline di code diverse dalla FIFO, sarebbe utile avere una lista collegata per gestire le diverse politiche. Nel caso con coda finita, consideriamo ora un sistema con perdita, verifichiamo la saturazione nell'algoritmo nel momento in cui arriva un nuovo job.

8.2.2 Simple delivey system con delivery lag

Erano stati fatti i cambiamenti per introdurre il delivery lag, inoltre per ogni settimana veniva generato (o letto dalla traccia) un valore, che era globale per la settimana. Di fatto, il livello di scorte che parte da l_{i-1} e che scende al valore $l'_{i-1} - d_i$ cala di un item alla volta, come se la domanda venisse spalmata per tutta la settimana. Un modello più realistico prevede di utilizzare un esponenziale di parametro $\frac{1}{\lambda}$ per i tempi inter-domanda, in questo modo la domanda sarà un processo di arrivo e la domanda media sarà λ per ogni intervallo di tempo.

8.2.3 Multi-server

Il servente multiplo ha una coda ed m server paralleli. Lo stato è:

- il numero di job nel sistema al tempo t
- m variabili che si riferiscono allo stato del server

Quali sono i tipi di eventi che possono cambiare lo stato del sistema:

- gli arrivi
- i completamenti, che stavolta sono m

il numero di diversi eventi sono quindi m+1. Facciamo sì che al tempo τ si spenga il processo degli arrivi, il sistema servirà i job presenti nel sistema per tornare allo stato iniziale. Per questo motivo, l'ultimo evento è sempre un completamento, mentre il primo è sempre un arrivo. Per semplicità consideriamo che i server siano

tutti uguali e che la selezione del server venga fatta secondo equity, ovvero scegliendo il server meno utilizzato.

9 Feedback 1

9.1 Distribuzioni heavy tail

Le distribuzioni esponenziali hanno diverse proprietà: consideriamo le distribuzioni a fasi esponenziali, in tutte quante le code vanno presto a 0, ovvero la probabilità dei valori grandi è estremamente piccola. Nella Pareto, anche la parte della distribuzione dei valori grandi ha delle probabilità non trascurabili, quindi le cose cambiano completamente.
Nella singola fase dell'esponenziale, la caratteristica principale è la memoryless, ovvero il failure rate è costante. Le distribuzioni con heavy tail hanno comportamenti opposti:

- più passa il tempo, meno è probabile che "la vita finisca", ovvero sono a failure rate decrescente.
- ad esempio, nella distribuzione di Pareto, la failure rate (ovvero la hazard rate function) è $e(x) = \frac{\alpha}{x}$, con $x > 1$. Quanto più la size x è grande, tanto più è piccola la frequenza di fallimento

Le heavy tail sono presenti in moltissimi aspetti:

- job Unix
- dimensione dei file web, con $\alpha \approx 1.1$
- topologia Internet: la maggior parte dei nodi ha un grado di uscita piccolo, pochi nodi hanno un grado di uscita molto grande
- n° di flussi di pacchetti IP: l'1% dei flussi più grandi comportano il 50% dei bytes della somma di tutti i flussi. Quindi, se indirizzo un flusso verso un certo routing, spostando solo l'1% dei flussi più grandi si toglie una grande quantità di carico
- fenomeni naturali