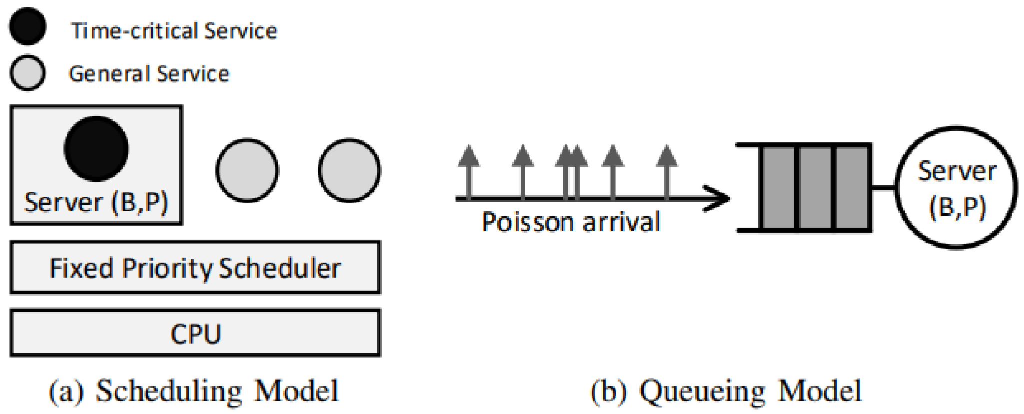
مقدمه

در مقاله «پیشبینی توزیع تاخیر در سرویس‌های نامتناوب زمان-بحرانی»[1] به مسئله پیشبینی زمان انتظار taskها در یک صف با دو نوع سرور بررسی شده است: ۱. سرور متناوب (PS): این سرور با دو پارامتر P (تناوب) و B (بودجه) مدل می‌شود، در این نوع، سرور در فاصله $t \in [kP, kP + (P - B))$ خاموش است و در فاصله $t \in [kP + (P - B), (k+1)P)$ سرور روشن است و تسک‌ها را اجرا می‌کند. ۲. Deferrable Server (DS): اینجا هم از دو پارامتر P و B برای مدلسازی سرور استفاده می‌کنیم. با این تفاوت که در این نوع سرور، بودجه سرور در ابتدای هر تناوب، به مقدار B بازگردانی می‌شود. هر زمانی که سرور در حال اجرای یک تسک باشد، بودجه مصرف می‌کند و به طور خطی با زمان بودجه کاهش می‌یابد. اما در صورتی که تسکی نباشد بودجه ثابت می‌ماند. اگر بودجه به ۰ برسد، سرور تا تناوب بعدی خاموش می‌شود.

در این مسئله می‌خواهیم دو نوع سرور DS و PS را در یک صف با یک سرور و ورودی با توزیع پوآسن تحلیل کنیم و توزیع زمان انتظار در سیستم برای تسک‌ها را در حالت دائم محاسبه کنیم:



شکل ۱، مدل مسئله مورد بررسی

با توجه به پیچیدگی مسئله ارائه پاسخ بسته برای سیستم با سختی‌های زیادی همراه است؛ در این مقاله روشی عددی برای محاسبه توزیع ارائه شده است و تلاش شده است تا این روش برای طراحی و نیز پیشبینی عملکرد قابل استفاده باشد. این ابزار به ما این امکان را می‌دهد تا بررسی کنیم که آیا برای مثال سیستم طراحی شده می‌تواند برای ۹۰٪ تسک‌های ورودی تاخیر کمتر از 10ms داشته باشد یا نه. در

[1] predicting latency distribution of aperiodic time-critical services

نهایت دو نوع صف $M/D(PS)/1$ و $M/D(DS)/1$ برای تحلیل در نظر گرفته شده است. که در اینجا M نشان‌دهنده بی‌حافظه بودن فرایند ورود تسک‌ها، D نشان‌دهنده ثابت بودن زمان اجرای تسک‌های ورودی در سیستم و 1 نشان‌دهنده‌ی این است که صف فقط ۱ سرور دارد. با توجه به دشواری ارائه روش تحلیلی برای مسئله یاد شده، در اینجا سیستم پیوسته را در ابتدا با یک سیستم گسسته تقریب زده می‌شود و سپس تحلیل بر روی این سیستم تقریبی گسسته انجام می‌شود. برای اینکار باید توجه شود که با توجه به قضیه Poisson Limit Theorem ورودی پوآسن را می‌توان با فرایند برنولی تخمین زد. با کوچک کردن زمان نمونه برداری به اندازه کافی می‌توان به دقت کافی رسید. با توجه به لزوم ثابت بودن پهنای باند مصرفی و utilization سیستم، اگر نرخ ورود به سیستم گسسته λ و زمان اجرای تسک d' باشد، و پارامترهای سیستم پیوسته (λ, d', P', B') باشد، با در نظر گرفتن طول هر بازه گسسته به صورت $\frac{d'}{N}$، می‌توان پارامترهای مدل گسسته را بدست آورد:

$$\lambda d' = \eta N, \frac{B'}{P'} = \frac{B}{P} \Rightarrow \begin{cases} \eta = \dfrac{\lambda d'}{N} \\ d = N \\ P = \dfrac{NP'}{d'} \\ B = \dfrac{NB'}{d'} \end{cases}$$

محاسبه توزیع در صف M/D(PS)/1

ابتدا توزیع در سرور متناوب محاسبه می‌شود و با توجه به قضیه‌ای که اثبات شده است، از جواب این قسمت برای محاسبه پاسخ سرور DS استفاده می‌کنیم. همانطور که ذکر شد باید مسئله گسسته را برای صف P/D(PS)/1 با پارامترهای (η, d, B, P) حل کنیم. اگر پس از به تعادل رسیدن داشته باشیم:

$$p_{l,n} = \Pr\{\bar{V}_{PS} = l | T = n\}, l, n \in N$$

که در آن $\overline{V_{PS}}$ نشان‌دهنده تعداد slotهای مورد نیاز پردازنده برای انجام تسک‌های موجود در صف در زمان n است. مشخص است که دنباله تعداد slot های مورد نیاز در ابتدای هر تناوب، یک فرایند مارکوف است. در اینجا با توجه به توضیحات داده شده اگر سرور خاموش باشد، پس از گذشت ۱ واحد زمان با احتمال η یک تسک جدید وارد می‌شود و با احتمال $1\text{-}\eta$ هیچ تسکی وارد نمی‌شود، بنابراین برای $0 \leq n \leq P - B - 1$ داریم:

$$\begin{cases} p_{l,n+1} = (1-\eta)p_{l,n} & 0 \leq l < d, \\ p_{l,n+1} = (1-\eta)p_{l,n} + \eta p_{l-d,n} & l \geq d. \end{cases} \quad (3)$$

دقت شود که اگر تسک جدید وارد شود، میزان slotهای مورد نظر به تعداد d تا افزایش می‌یابد (d تعداد slot های لازم برای انجام هر ۱ تسک است).

همچنین برای $P - B \leq n \leq P$ داریم:

$$\begin{cases} p_{0,n+1} = (1-\eta)(p_{0,n} + p_{1,n}) & l = 0, \\ p_{l,n+1} = (1-\eta)p_{l+1,n} & 1 \leq l < d-1, \\ p_{l,n+1} = (1-\eta)p_{l+1,n} + \eta p_{l-d+1,n} & l \leq d-1. \end{cases} \quad (4)$$

حالت اول نشان‌دهنده تمام گذارها به سیستم خالی است: یا سیستم خالی بوده و هیچ تسکی وارد نشده، یا یک slot در پردازنده مورد نیاز بوده و کسی وارد نشده و ۱ واحد تامین شده است. دلیل تفکیک بین حالت دوم و سوم نیز در این است که برای حالت‌هایی که تعداد slot مورد نیاز پردازنده از d-1 بیشتر باشد، دو مسیر برای رسیدن به آن استیت وجود دارد. یک مسیر اینکه کسی وارد نشود، و یک واحد کار از استیت بزرگتر تامین شود، و حالت دوم زمانی است که سیستم l-d+1 اسلات نیاز داشته باشد و یک تسک وارد شود و نیاز را d واحد افزایش دهد. این گذارها در شکل ۲ قابل مشاهده است. در نهایت با ترکیب این روابط با متناوب بودن p و نیز برابری مجموع احتمال با ۱، پاسخ نهایی را به ما خواهد داد:

$$p_{l,n+P} = p_{l,n}, \quad \forall l \in \mathbb{N}. \quad (5)$$

$$\sum_{l=0}^{\infty} p_{l,n} = \sum_{l=0}^{\infty} Pr\{\overline{V}_{PS} = l | T = n\} = 1, \quad 0 \leq n \leq P. \quad (6)$$

با توجه به رابطه ۶، سری pl,n همگرا است، در نتیجه می‌توان با انتخاب M به اندازه کافی بزرگ تنها M جمله اول را به صورت بازگشتی محاسبه کرد و توزیع را بدست آورد: از $p_{0,0} = 1$شروع می‌کنیم، در هر مرحله با استفاده از روابط گذار داده شده یک لایه پیش می‌رویم و در نهایت احتمال را به ۱ نرمالیزه می‌کنیم. اینکار را چند بار انجام می‌دهیم تا در نهایت تغییرات کاهش یابد و به دقت مورد نظر برسیم (شکل ۳).

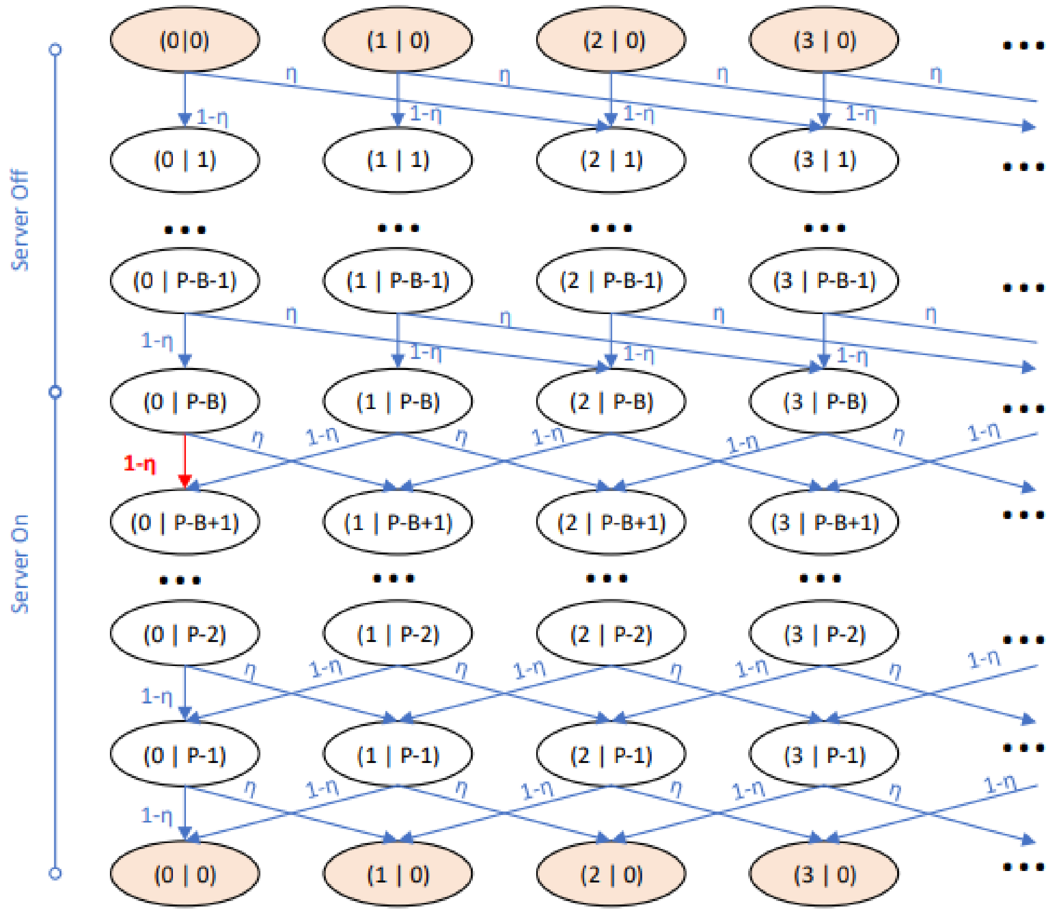


Fig. 8: State Transition Diagram for a B/D(PS)/1 queue: $Pr\{\overline{V}_{PS} = l|T = n\}$, $d = 2$

شکل ۲

Algorithm 1: Compute first M terms of $p_{l,0}$

Input: Vector Length: M, Expected Error: Δ

Output: $M \times 1$-Vector: $\overrightarrow{V} = (p_{0,0}, p_{1,0}, ..., p_{M-1,0})$

$M \times 1$-Vector: $\overrightarrow{V} \leftarrow (1, 0, 0, ...)$; $\quad \delta \leftarrow \infty$;
while $\delta \geq \Delta$ **do**
    $\overrightarrow{V'} \leftarrow \overrightarrow{V}$;
    for $i \leftarrow 0$ **to** $P - B - 1$ **do**
        $M \times 1$-Vector: $\overrightarrow{V_T} \leftarrow (0, 0, 0, ...)$;
        Compute $\overrightarrow{V_T} = (p_{0,i+1}, p_{1,i+1}, ..., p_{M-1,i+1})$ from $\overrightarrow{V} = (p_{0,i}, p_{1,i}, ..., p_{M-1,i})$, using Eq.(3);
        $\overrightarrow{V} \leftarrow \overrightarrow{V_T}$;
    end
    for $i \leftarrow P - B$ **to** $P - 1$ **do**
        $M \times 1$-Vector: $\overrightarrow{V_T} \leftarrow (0, 0, 0, ...)$;
        Compute $\overrightarrow{V_T} = (p_{0,i+1}, p_{1,i+1}, ..., p_{M-1,i+1})$ from $\overrightarrow{V} = (p_{0,i}, p_{1,i}, ..., p_{M-1,i})$, using Eq.(4);
        $\overrightarrow{V} \leftarrow \overrightarrow{V_T}$;
    end
    $\overrightarrow{V} \leftarrow \overrightarrow{V} / \|\overrightarrow{V}\|_1$; $\quad \delta \leftarrow \|\overrightarrow{V} - \overrightarrow{V'}\|_1$;
end
return $\overrightarrow{V}$;

شکل ۳

محاسبه توزیع در صف M/D(DS)/1

در اینجا پیچیدگی با توجه به بستگی احتمال به بودجه در هر لحظه پیچیده‌تر می‌شود. با در نظر گرفتن بودجه می‌توان احتمال زیر را در محاسبه کرد:

$$q_{l,g,n} = Pr\{\tilde{V}_{DS} = l, G = g | T = n\}, \quad l, n \in \mathbb{N}, 0 \leq g \leq B. \tag{8}$$

در واقع ما باید احتمال بودن در هر سطح بودجه ممکن، در هر میزان slot مورد نیاز پردازنده را در زمان حساب کنیم. دی اینجا می‌توان دقیقا مانند قسمت قبل عمل‌کرد اما با اینکار پیچیدگی مسئله زیاد می‌شود با توجه با افزایش تعداد حالت‌های ممکن به صورت ضرب در فاکتور B. برای ارائه راه حل ساده‌تر باید توجه

کرد که سرور DS، در ابتدای هر تناوب بودجه خود را به B بازمیگرداند. در واقع نکته‌ای که به آسان شدن مسئله کمک می‌کند این است که اگر توزیع زمان مورد نیاز پردازنده در ابتدای تناوب nام را برای سرور PS و DS به ترتیب با $V_{PS}[n]$ و $V_{DS}[n]$ نشان دهیم، می‌توان نشان داد (اثبات در پیوست):

$$Pr\{V_{DS}[n] = V_{PS}[n]\} = 1.$$

با توجه به این ۲ نکته داریم:

$$\begin{cases} q_{l,B,0} = p_{l,0}, & l \in \mathbb{N}, \\ q_{l,g,0} = 0, & l \in \mathbb{N}, g < B. \end{cases} \quad (9)$$

به این معنی که در لحظه صفر، احتمال فقط در بودجه B غیر صفر است و برابر است با $p_{l,0}$. با استفاده از رابطه می‌توانیم ابتدا با استفاده از الگوریتم ۱ در شکل (۳) $q_{l,B,0}$ را محاسبه کرد. با شروع از انی نقطه می‌توان تمام استیت‌های ممکن را حساب کرد. زیرا در هر زمان ما می‌دانیم که با احتمال η یک تسک جدید می‌رسد و با احتمال $1-\eta$ تسک جدید وارد نمی‌شود. پس هر استیت در زمان n حداکثر به دو استیت دیگر در زمان n+1 گذار می‌تواند بکند. اگر بودجه صفر باشد، میدانیم که میزان کار موجود در سرور کاهش نمی‌یابد و فقط در صورت ورود تسک جدید ممکن است d واحد افزایش پیدا کند. در صورتی که g مثبت باشد، پس از گذشت یک واحد از کار سرور کاهش می‌یابد و با توجه به احتمال η در ۲ حالت ممکن است که کار جدیدی به سیستم اضافه شود یا نشود. همچنین اگر سیستم خالی باشد طبیعتا حالت‌های ممکن برای آینده سیستم خالی ماند آن و یا اضافه شدن بار آن به اندازه d است. با این توضیحات می‌توان الگوریتم ۲ را ارائه داد برای محاسبه توزیع $q_{l,g,n}$ (شکل ۴). در نهایت کافیست توزیع $q_{l,g,n}$ که نشان‌دهنده احتمال وجود بار سیستم به اندازه l و با بودجه g در لحظه d است را به توزیع میزان تاخیر برای تسک تبدیل کرد. برای اینکار با استفاده از قضیه احتمال کل:

$$F_{DS}(t) = Pr\{R_{DS} = t\} = \sum_{l=0}^{\infty}\sum_{g=0}^{B}\sum_{n=0}^{P-1} Pr\{R_{DS} = t | \overline{V}_{DS} = l, G = g, T = n\} Pr\{\overline{V}_{DS} = l, G = g, T = n\}. \quad (10)$$

البته در اینجا از نسخه گسسته قضیه PASTA نیز استفاده شده است که بیان می‌کند یک job در هنگام ورود به سیستم با ورودی برنولی، سیستم را در یک زمان تصادفی در steady state و بدون وجود خودش می‌بینند. اما با داشتن l و g و n، می‌توان response time را تعیین کرد. اگر تعریف کنیم $\gamma_1 =$

$\min\{P - n, g\}$ و $\gamma_2 = P - n - \gamma_1$ و $h = l + d$، γ_1 نشان دهنده میزان زمان سرویس باقی‌مانده تا پایان پریود و h نشان‌دهنده میزان سرویس مورد نیاز برای انجام موفقیت آمیز تسک‌ها خواهد بود.

Algorithm 2: Compute $q_{l,g,n}$

```
Input: M × 1-Vector: V⃗ = (p_{0,0}, p_{1,0}, ..., p_{M−1,0})
Output: M × B × P-Matrix: A = {q_{l,g,n}}, List:
        nz_list
A = zeros(M, B, P); nz_list ← [ ]; tset ← { };
for l ← 0 to M − 1 do
    A[l][B][0] ← p_{l,0};
    tset.union({(l, B)});
end
nz_list.append(tset);
for n ← 0 to P − 2 do
    iterset ← nz_list[n]; tset ← { };
    for (l, g) in iterset do
        if g > 0 then
            if l > 0 then
                A[l − 1][g − 1][n + 1] += (1 − η)A[l][g][n];
                tset.union({(l − 1, g − 1)});
                if l + d − 1 ≤ M then
                    A[l + d − 1][g − 1][n + 1] +=
                     ηA[l][g][n];
                    tset.union({(l + d − 1, g − 1)});
                end
            else
                A[l][g][n + 1] += (1 − η)A[l][g][n];
                tset.union({(l, g)});
                if l + d − 1 ≤ M then
                    A[l + d − 1][g − 1][n + 1] +=
                     ηA[l][g][n];
                    tset.union({(l + d − 1, g − 1)});
                end
            end
        else
            A[l][g][n + 1] += (1 − η)A[l][g][n];
            tset.union({(l, g)});
            if l + d > M then
                A[l + d][g][n + 1] += ηA[l][g][n];
                tset.union({(l + d, g)});
            end
        end
    end
    nz_list.append(tset);
end
return (A, nz_list);
```

شکل ۴

در نتیجه واضح است که اگر h کمتر از γ_1 باشد، زمان اتمام برابر h خواهد بود. در غیر این صورت اگر بتوان در پریود بعدی کارا به اتمام رساند، یعنی اگر بودجه فعلی به اضافه بودجه B که در پریود بعد اضافه می‌شود کافی باشد $(h < \gamma_1 + B)$ در این صورت، اتمام کار به اندازه $h + (P - n - \gamma_1) = h + \gamma_2$ صبر کرد. زیرا γ_1 تا در ابتدا تعیین می‌شود، سپس تا پریود دوم صبر می‌کنیم و ادامه سرویس را در پریود دوم دریافت می‌کنیم. در نهایت اگر زمان بیشتری نیاز باشد، مشابه تعمیم این حالت زمان $h + \gamma_2 + \left(\frac{h-(\gamma_1+B)}{B}\right)(P - B)$ نیاز است. بنابراین با داشتن l, g, n زمان پاسخ به صورت زیر است:

$$f_{DS}(l,g,n) = \begin{cases} h, & h \leq \gamma_1, \\ h + \gamma_2, & \gamma_1 < h \leq \gamma_1 + B, \\ h + \gamma_2 + \lceil \frac{h - (\gamma_1 + B)}{B} \rceil (P - B), & h > \gamma_1 + B. \end{cases} \quad (11)$$
$$h = l + d, \quad \gamma_1 = \min\{P - n, g\}, \quad \gamma_2 = P - n - \gamma_1.$$

اگر این مقدار در رابطه ۱۰ با مقدار t برابر باشد، در این صورت $\Pr\{R_{DS} = t | V_{DS} = l, G = g, T = n\}$ برابر ۱ خواهد بود و در غیر این صورت صفر. در نهایت داریم:

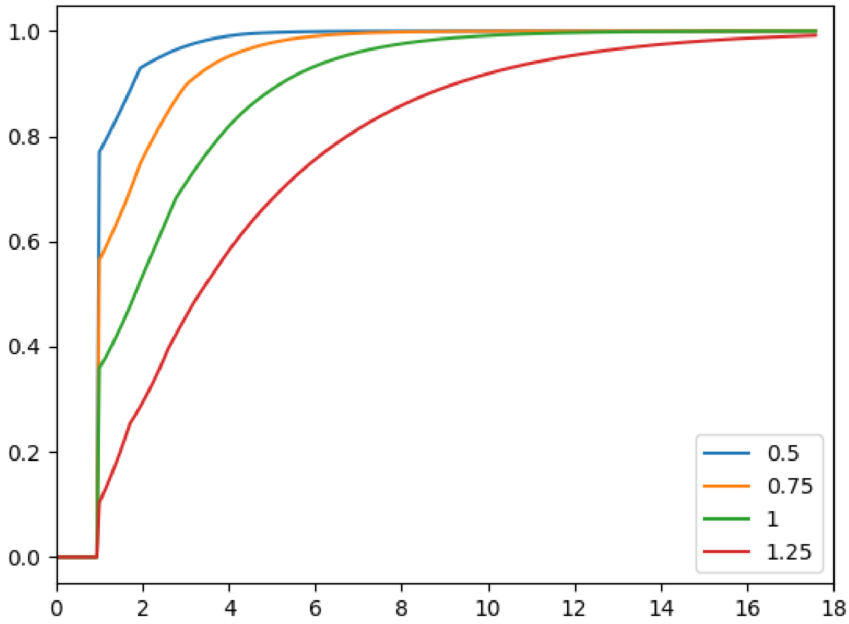
$$F_{DS}(t) = \frac{1}{P} \sum_{\substack{l \geq 0, 0 \leq g \leq B, 0 \leq n \leq P-1, \\ f_{DS}(l,g,n) = t}} q_{l,g,n}. \qquad (13)$$

زیرا:

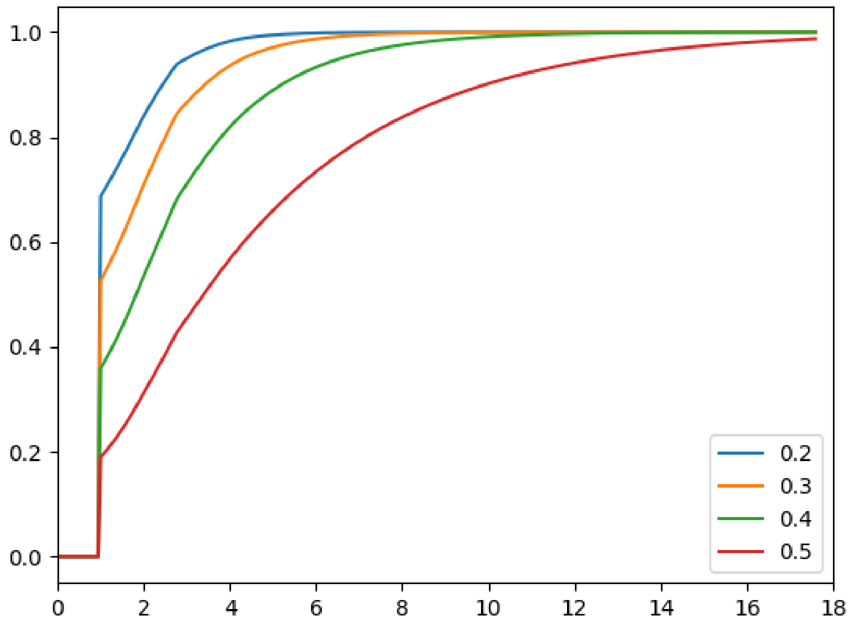
$$Pr\{\overline{V}_{DS} = l, G = g, T = n\} = \frac{q_{l,g,n}}{P}.$$

نتایج

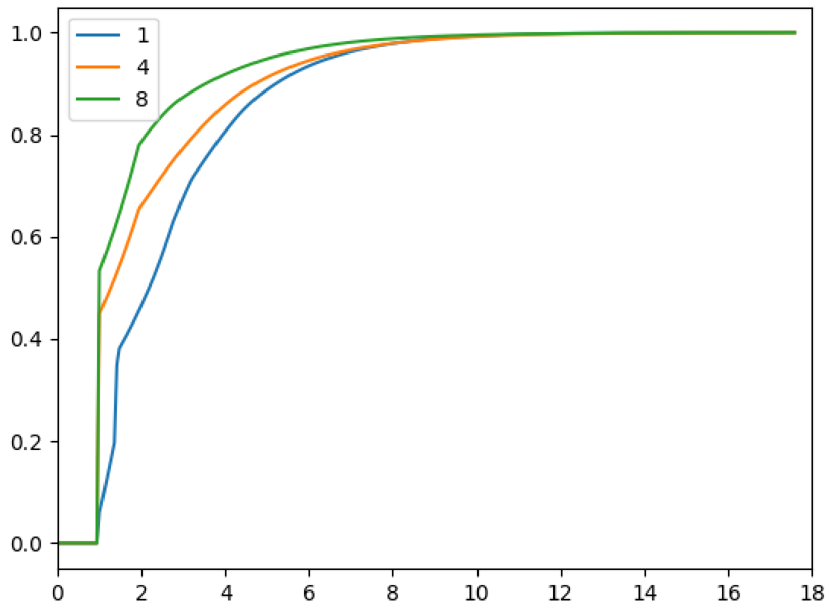
الگوریتم داده شده در مسئله در پایتون پیاده‌سازی شده است و نتایج بدست آمده در شکل ۵ تا ۸ با نتایج مقاله مقایسه شده است (شکل ۹). مشاهده می‌شود که نتایج انطباق دارد. دی این شبیه سازی حالت پایه به صورت: λ = 0.4, d = 1.0, P = 2.0, B = 1.2 است.



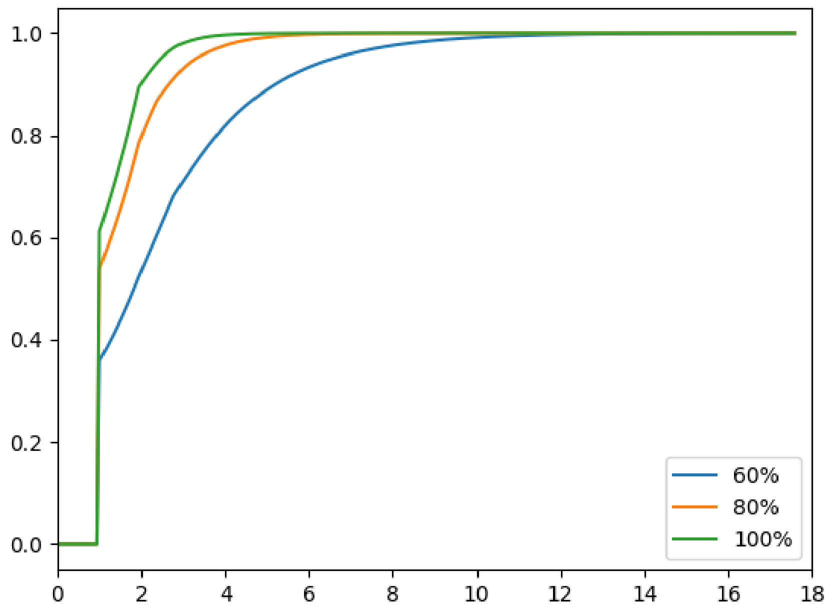
شکل ۵، تغییر پارامتر **d** در ۴ مقدار



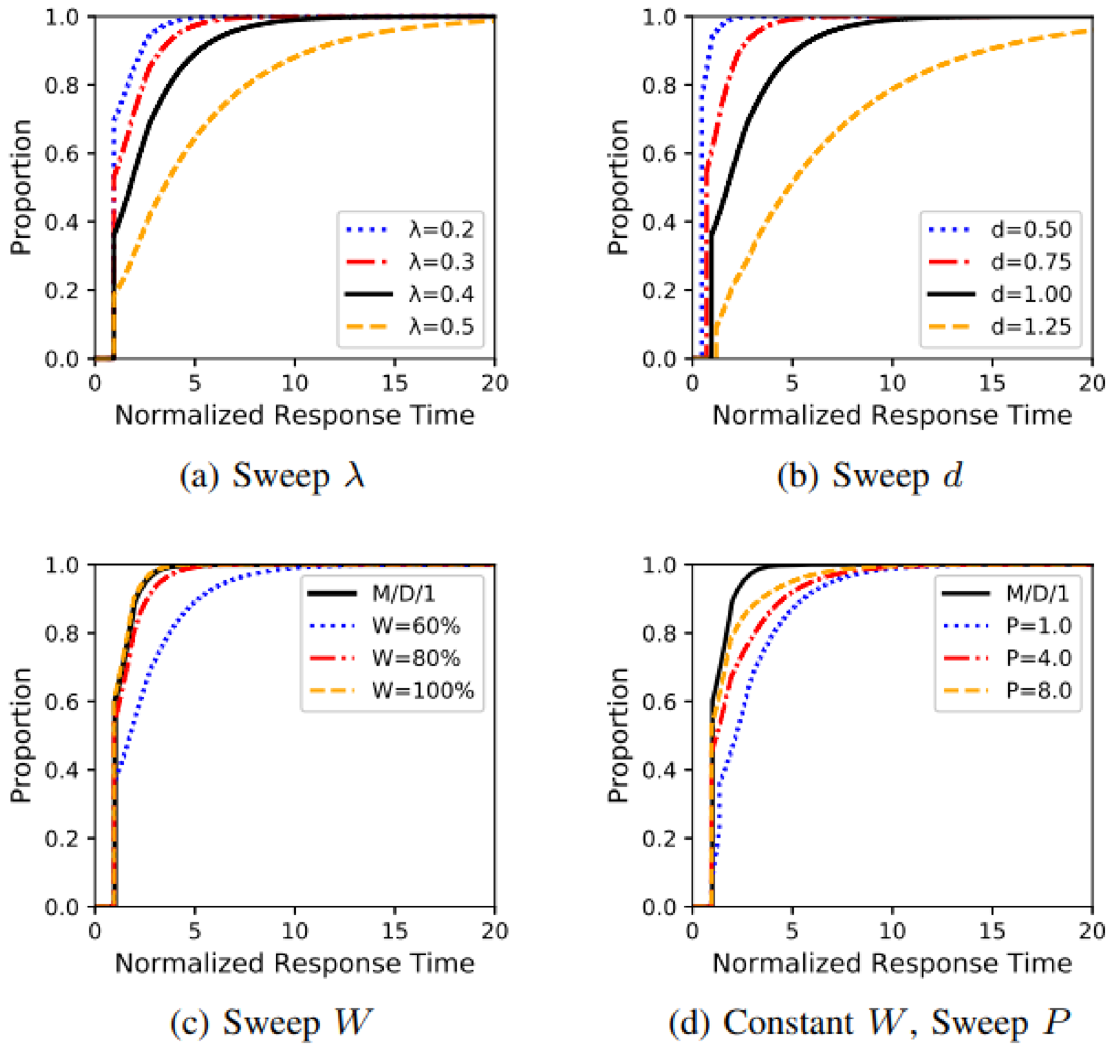
شکل ۶، تغییر پارامتر λ در ۴ مقدار



شکل ۷، تغییر پارامتر **P**



شکل ۸، تغییر پارامتر پهنای باند در **P** ثابت



شکل ۹، نتایج مقاله

اثبات برابری توزیع کار باقیمانده در ابتدای هر تناوب در سرور PS و DS

اگر $V_{DS}[n]$ و $V_{PS}[n]$ به ترتیب نشان‌دهنده توزیع میزان کار باقیمانده در سرور در ابتدای پریود nام باشد (زمان nP)، در این صورت می‌خواهیم نشان دهیم در دو سیستم $G/G(DS)/1$ و $G/G(PS)/1$ همواره هر تجسمی از این فرایندهای تصادفی که آن‌ها را با $v_{DS}[n]$ و $v_{PS}[n]$ نشان می‌دهیم همواره با هم برابرند:

$$v_{PS}[n] = v_{DS}[n]$$

برای اثبات، یک دنباله از ورود تسک‌ها با زمان ورود و زمان اجرای مشخص $\{(a_i, c_i)\}$ را در نظر می‌گیریم و به هر دو سیستم ورودی می‌دهیم. می‌خواهیم نشان دهیم در این صورت تساوی یاد شده برقرار

خواهد بود. پریود $[0, P)$ را برای اینکار در نظر می‌گیریم و فرض می‌کنیم در شروع کار x واحد کار در سرور باقیمانده باشد. در این صورت کافیست *H(n)* تعریف شده در زیر را برای n های صحیح اثبات کنیم:

Hypothesis H(n). In the described period, $\forall x \geq 0, \forall P > 0, \forall B \in [0, P]$, if exactly n jobs arrive within the period, then $y_{DS} = y_{PS}$. We can prove that $\forall n \in \mathbb{N}$, $H(n)$ is true.

اما می‌دانیم *H(0)* درست است. زیرا *H(0)* به این معنی است که هیچ تسکی وارد نشود. در این صورت هر در انتهای پریود در هر دو حالت، *x-B* واحد کار باقی می‌ماند (یا هر دو صفر می‌شود اگر x کوچک باشد).

اکنون اگر نشان دهیم از درستی *H(n-1)* می‌توان درستی *H(n)* را نتیجه گرفت اثبات تکمیل است. ابتدا نشان می‌دهیم اگر *H(n)* یا صحیح است، یا همان مقدار *H(n-1)* را دارد. چند خالت در نظر می‌گیریم:

1. اگر x از B بزرگتر باشد: در این صورت به سادگی مشخص است که در هر دو حالت سرور، در انتهای سرور B واحد از x تامین می‌شود و باقی کارها باقی‌میماند:

$$y_{DS}[n] = y_{PS}[n] = x - B + sum(c_i)$$

2. اگر x از B کوچکتر باشد و $a_n \geq x$:

در سرور *DS*، مشخص است که تا قبل از ورود اولین $job\,(a_n)$، کل x تامین شده است. در نتیجه با سرور خالی مواجه می‌شود. بنابراین می‌توان سیستم را یک سیستم جدید تعریف کرد با *n-1* ورودی، و بار a_n، یعنی c_n را برای آن به صورت x جدید تعریف کرد. سیستم جدید پارامترهای $P' = P - a_n$ و $B' = \min\{B - x, P - a_n\}$ را دارد. همچنین سرور *PS* نیز به همین صورت قابل کاهش به سیستمی با *n-1* ورودی با پارامترهای یکسان است. در نتیجه:

$$H(n) = H(n-1)$$

3. اگر x از B کوچکتر باشد و $a_n < x$:

در این حالت اگر $x + c_n \geq B$ باشد، کل بودجه صرف x و اولین job خواهد شد در هر دو صورت و خواهیم داشت:

$$y_{DS} = y_{PS} = x + c_n + \sum_{i=1}^{n-1} c_i - B = x + \sum(c_i) - B$$

در نتیجه *H(n)* درست است.

اما اگر $x + c_n < B$ باشد در این صورت سیستم معادل است با ورود *job n-1* و مقدار اولیه برابر $x + c_n$. که در این سیستم جدید مقدار y_{DS} و یا y_{PS} ثابت می‌ماند. در نتیجه مقدار *H(n)* با مقدار *H(n-1)* برابر است.

در نتیجه *H(n)* برای تمام *n*های صحیح درست است. با اعمال *H(n)* به پیرودهای متوالی می‌توان نتیجه گرفت که در ابتدای همه پریودها:

$$v_{PS}[n] = v_{DS}[n]$$

و قضیه اثبات می‌شود.

```python
import numpy as np
import math
import sys
from matplotlib import pyplot as plt
np.set_printoptions(threshold=sys.maxsize)

def algorithm1(M, Delta, eta, P, B, d):

    V = np.zeros(M)
    V[0] = 1
    delta = math.inf

    while delta >= Delta:
        V_prime =   np.copy(V)
        for i in range(P-B-1):
            V_T = np.zeros(M)
            V_T[0:d] = (1-eta)*V[0:d]
            V_T[d:] = (1-eta)*V[d:] + eta*V[0:M-d]
            V = V_T

        for i in range(P-B, P):
            V_T = np.zeros(M)
            V_T[0] = (1-eta)*(V[0] + V[1])
            # print(V_T.shape)
            # print(V.shape)
            V_T[1:d-1] = (1-eta)*V[2:d]
            V_T[d-1:] = (1-eta)*np.append(V[d:], np.zeros(1)) + eta*V[0:M-d+1]
            V = V_T

        V = V / sum(np.abs(V))
        delta = sum(abs(V-V_prime))
        # print(delta)
    return V
```

```python

def algorithm2(M, V, eta, P, B, d):

    A = np.zeros((M, B, P))
    nz_list = []
    tset = []

    for l in range(M):
        A[l,B-1,0] = V[l]
        tset.append((l, B-1))

    nz_list.append(tset)

    for n in range(P-1):

        iterset = nz_list[n]
        tset = []
        for (l, g) in iterset:
            # print( (l, g, n) )
            if g > 0:

                if l > 0:

                    A[l-1,g-1,n+1] += (1-eta)*A[l, g, n]
                    tset.append((l-1, g-1))
                    if l+d-1 < M:
                        A[l+d-1, g-1, n+1] += eta*A[l,g,n]
                        tset.append((l+d-1, g-1))

                elif l == 0:

                    A[l, g, n+1] += (1-eta)*A[l, g, n]
                    tset.append((l, g))
                    if l+d-1 < M:
                        A[l+d-1, g-1, n+1] += eta*A[l, g, n]
                        tset.append((l+d-1, g-1))
```

```
            elif g == 0:

                A[l, g, n+1] += (1-eta)*A[l, g, n]
                tset.append((l, g))
                if l+d < M:
                    A[l+d,g,n+1] += eta*A[l, g, n]
                    tset.append((l+d, g))
        tset = list(set(tset))
        nz_list.append(tset)

    return A, nz_list


def fds(l, g, n, d, P, B):
    h = l + d
    gamma1 = min(P-n, g)
    gamma2 = P-n-gamma1
    if h <= gamma1:
        return h
    elif h > gamma1 and h <= gamma1 + B:
        return h + gamma2
    else:
        return h + gamma2 + math.ceil((h-(gamma1+B))/B)*(P-B)

    print("error")
    return -1

def get_distribution(lambda_, d_prime, B_prime, P_prime, N=20, M=200):

    eq_sense = 0.1

    d = N
    B = math.floor(N*B_prime/d_prime)
    P = math.floor(N*P_prime/d_prime)
    eta = lambda_*d_prime/N
    print(f"{M}*({B}+1)*{P}, eta={eta}, d={d}")
```

```python
    Delta = 0.01

    q_l_b_0 = algorithm1(M, Delta, eta, P, B, d)
    A, nz_list = algorithm2(M, q_l_b_0, eta, P, B, d)
    # print(q_l_b_0)
    print("finish alg 2")
    # print(A)
    # print( sum(sum( A[:, :, 0]) ) )
    # print( sum(sum( A[:, :, 1] )) )
    # print( sum(sum( A[:, :, 2] )) )
    # print( sum(sum( A[:, :, 3] )) )
    # print( sum(sum( A[:, :, 4] )) )
    # print( sum(sum( A[:, :, 5] )) )
    # print( sum(sum( A[:, :, 6] )) )
    # print( sum(sum( A[:, :, 7] )) )
    F_DS = np.zeros(M)
    # print(np.sum(A))
    for i in range(M):
        for l in range(A.shape[0]):
            for g in range(B):
                for n in range(P):
                    if abs( fds(l, g, n, d, P, B) - i ) < eq_sense:
                        F_DS[i] += 1/P*(A[l, g, n])

    return F_DS


if __name__ == "__main__":
    lambda_ = [0.2, 0.3, 0.4, 0.5]
    d_ = [0.5, 0.75, 1, 1.25]
    P_ = [1, 4, 8]
    W_ = [0.6, 0.8, 1]
    dist = []
    N = 17
    M = 300
```

```python
sweep_type = "W"
# for lamb_ in lambda_:
#     x = get_distribution(lamb_, 1, 1.2, 2, N, M)
#     dist.append(x)
#     plt.plot(np.arange(M)/N ,np.cumsum(x))
#     plt.xlim((0, math.ceil(M/N)))

# for d in d_:
#     x = get_distribution(0.4, d, 1.2, 2, N, M)
#     dist.append(x)
#     plt.plot(np.arange(M)/N ,np.cumsum(x))
#     plt.xlim((0, math.ceil(M/N)))

# for P in P_:
#     x = get_distribution(0.4, 1, 1.2/2*P, P, N, M)
#     dist.append(x)
#     plt.plot(np.arange(M)/N ,np.cumsum(x))
#     plt.xlim((0, math.ceil(M/N)))


for W in W_:
    x = get_distribution(0.4, 1, 2*W, 2, N, M)
    dist.append(x)
    plt.plot(np.arange(M)/N ,np.cumsum(x))
    plt.xlim((0, math.ceil(M/N)))

plt.legend(["60%", "80%", "100%"])
plt.savefig(f"{sweep_type}-sweepp,N={N},M={M}.png")
```