Part A

MORE FORMULAS DERIVATIONS FOR JOB SCHEDULING

A. State steady equation:

The stationary probability p_k for state k can be determined by solving the set of balance equations, which state that the flux into a state should be equal to the flux out of this state when the system is stationary [**?**]:

$$\begin{aligned}
&\text{When state } k=0, \lambda p_0=\mu p_1,\\
&\quad p_1=\rho_1 p_0=n\rho p_0;\\
&\text{When state } k=1, \lambda p_1=2\mu p_2,\\
&\quad p_2=\tfrac{\rho_1^2}{2!}p_0=\tfrac{n^2}{2!}\rho^2 p_0;\\
&\text{When state } k=2, \lambda p_2=3\mu p_3,\\
&\quad p_3=\tfrac{\rho_1^3}{3!}p_0=\tfrac{n^3}{3!}\rho^3 p_0;\\
&\quad \dots\\
&\text{When state } k=n-1, \lambda p_{n-1}=n\mu p_n,\\
&\quad p_n=\tfrac{\rho_1^n}{n!}p_0=\tfrac{n^n}{n!}\rho^n p_0;\\
&\text{When state } k=n, \lambda p_n=n\mu p_{n+1},\\
&\quad p_{n+1}=\tfrac{\rho_1^{n+1}}{n!n}p_0=\tfrac{n^n}{n!}\rho^{n+1} p_0;\\
&\quad \dots\\
&\text{When state } k=n+r-1, \lambda p_{n+r-1}=n\mu p_{n+r},\\
&\quad p_{n+r}=\tfrac{\rho_1^{n+r}}{n!n^r}p_0=\tfrac{n^n}{n!}\rho^{n+r} p_0.
\end{aligned} \tag{1}$$

In general,

$$p_k=\begin{cases}\frac{\rho_1^k}{k!}p_0=\frac{n^k}{k!}\rho^k p_0, & 0\le k<n\\ \frac{\rho_1^k}{n!n^{k-n}}p_0=\frac{n^n}{n!}\rho^k p_0; & k\ge n\end{cases} \tag{2}$$

B. Density functions and distribution functions of sojourn time and waiting time

The variance of mean number of jobs waiting in the queue shows as follows. Since,

$$\begin{aligned}
&E(\bar{L}_{wai}^2)=\sum_{k=n}^{\infty}(k-n)^2p_k=\sum_{h=1}^{\infty}h^2p_{h+n}\\
&=\sum_{h=1}^{\infty}\tfrac{h^2}{n!n^h}(n\rho)^{h+n}p_0\\
&=\tfrac{(n\rho)^n\rho^2p_0}{n!}\sum_{h=2}^{\infty}h(h-1)\rho^{h-2}+\tfrac{(n\rho)^n\rho p_0}{n!}\sum_{h=1}^{\infty}h\rho^{h-1}\\
&=\tfrac{2\rho^2\rho_1^np_0}{n!(1-\rho)^3}+\bar{L}_{wai}=\tfrac{1+\rho}{1-\rho}\bar{L}_{wai}
\end{aligned} \tag{3}$$

Thus,

$$\sigma^2(\bar{L}_{wai})=E(\bar{L}_{wai}^2)-[E(\bar{L}_{wai})]^2=\bar{L}_{wai}(\frac{1+\rho}{1-\rho}-\bar{L}_{wai}) \tag{4}$$

In addition,

$$E(L_{sys})=E(L_{wai})+E(L_{ser}) \tag{5}$$

$$E(W_{soj})=E(W_{wai})+E(T_{ser}) \tag{6}$$

we can easily get

$$E(T_{ser})=\frac{1}{\mu} \tag{7}$$

If we only consider servers of the system, without regarding the waiting queues outside the servers, it is easy to observe that there are no losses, and therefore the arrival rate in this cloud system is λ, and the mean waiting time of each customer is $E(T_{ser})=\frac{1}{\mu}$ [**?**], [**?**].

To obtain $E(L_{wai}|q\ge n)$, noted that the evolution of the M/M/n queue during the time when $q\ge n$ is equal to that of M/M/1 queue with the arrival rate λ and the service rate $n\mu$. Therefore, the mean queue length of this kind of M/M/1 queue is equivalent to $\frac{1}{1-\rho}$, where $\rho=\frac{\lambda}{n\mu}$. Therefore,

$$E(L_{wai}|q\ge n)=\frac{\rho/n}{1-\rho/n}=\frac{\rho}{n-\rho} \tag{8}$$

Substitute the distribution for the density function of the waiting time, we get

$$\begin{aligned}
f_w(x)&=\tfrac{p_0(\frac{\lambda}{\mu})^n}{n!}n\mu e^{-n\mu x}\sum_{j=0}^{\infty}\tfrac{(\rho n\mu x)^j}{j!}\\
&=\tfrac{(\frac{\lambda}{\mu})^n}{n!}p_0n\mu e^{-(n\mu-\lambda)x}\\
&=\tfrac{(\frac{\lambda}{\mu})^n}{n!}p_0n\mu e^{-n\mu(1-\rho)x}\\
&=\tfrac{(\frac{\lambda}{\mu})^n}{n!}p_0\tfrac{1}{1-\rho}n\mu(1-\rho)e^{-n\mu(1-\rho)x}\\
&=P(Waiting)n\mu(1-\rho)e^{-n\mu(1-\rho)x}
\end{aligned} \tag{9}$$

Thus for the complement of the the distribution function, we have

$$\begin{aligned}
P(W>x)&=\int_x^{\infty}f_w(u)du=P(Waiting)e^{-n\mu(1-\rho)x}\\
&=C(n,\rho)\bullet e^{-\mu(n-\frac{\rho}{n})x}
\end{aligned} \tag{10}$$

The distribution function of waiting time can be written as:

$$\begin{aligned}
F_w(x)&=1-P(Waiting)+P(Waiting)(1-e^{-n\mu(1-\rho)x})\\
&=1-P(Waiting)e^{-n\mu(1-\rho)x}\\
&=1-C(n,\rho)\bullet e^{-\mu(n-\frac{\rho}{n})x}
\end{aligned} \tag{11}$$

By applying the law of total probability for the density function of the sojourn time, $f_s(x)$ is given as follow:

$$f_s(x)=P(No\ waiting)\mu e^{-\mu x}+f_{w+ser}(x) \tag{12}$$

Whereas, the density function of sojourn time for the job that needs to wait first $f_{w+ser}(x)$:

$$\begin{aligned}
&f_{w+ser}(z)=\int_0^z f_w(x)\mu e^{-\mu(z-x)}dx\\
&=P(Waiting)n\mu(1-\rho)\mu\int_0^z e^{-n\mu(1-\rho)x}e^{-\mu(z-x)}dx\\
&=\tfrac{(n\rho)^n}{n!}p_0\tfrac{1}{(1-\rho)}n\mu(1-\rho)\mu e^{-z\mu}\int_0^z e^{-\mu(n-1-\frac{\lambda}{\mu})x}dx\\
&=\tfrac{(n\rho)^n}{n!}p_0n\mu\tfrac{1}{(n-1-\frac{\lambda}{\mu})}e^{-z\mu}(1-e^{-\mu(n-1-\frac{\lambda}{\mu})z})
\end{aligned} \tag{13}$$

Therefore,

$$\begin{aligned}
&f_s(x)=(1-(\tfrac{\lambda}{\mu})^n\tfrac{p_0}{n!(1-\rho)})\mu e^{-\mu x}+\\
&\quad\tfrac{(\frac{\lambda}{\mu})^n}{n!}n\mu p_0\tfrac{1}{(n-1-\frac{\lambda}{\mu})}e^{-\mu x}(1-e^{-\mu(n-1-\frac{\lambda}{\mu})x})\\
&=\mu e^{-\mu x}(1-\tfrac{(\frac{\lambda}{\mu})^np_0}{n!(1-\rho)}+\tfrac{(\frac{\lambda}{\mu})^n}{n!}np_0\tfrac{1}{(n-1-\frac{\lambda}{\mu})}(1-e^{-\mu(n-1-\frac{\lambda}{\mu})x}))\\
&=\mu e^{-\mu x}(1+\tfrac{(\frac{\lambda}{\mu})^np_0}{n!(1-\rho)}\tfrac{1-(n-\frac{\lambda}{\mu})e^{-\mu(n-1-\frac{\lambda}{\mu})x}}{(n-1-\frac{\lambda}{\mu})})
\end{aligned} \tag{14}$$

For the complement of the distribution function of the response time, we get

$$\begin{aligned}
&P(S>x)=\int_x^{\infty}f_s(y)dy=\\
&\int_x^{\infty}\mu e^{-\mu y}+\tfrac{(\frac{\lambda}{\mu})^np_0}{n!(1-\rho)}\tfrac{1}{(n-1-\frac{\lambda}{\mu})}(\mu e^{-\mu y}-\mu(n-\tfrac{\lambda}{\mu})e^{-\mu(n-\frac{\lambda}{\mu})y})dy\\
&=e^{-\mu x}+(\tfrac{\lambda}{\mu})^np_0\tfrac{1}{n!(1-\rho)(n-1-\frac{\lambda}{\mu})}(e^{-\mu x}-e^{-\mu(n-\frac{\lambda}{\mu})x})\\
&=e^{-\mu x}(1+\tfrac{(\frac{\lambda}{\mu})^np_0}{n!(1-\rho)}\tfrac{1-e^{-\mu(n-1-\frac{\lambda}{\mu})x}}{(n-1-\frac{\lambda}{\mu})})
\end{aligned} \tag{15}$$

Therefore the distribution function can be presented as

$$F_s(x)=1-P(S>x) \tag{16}$$

Part B
MORE DETAILED ALGORITHMS

Algorithm 1 presents the initialisation in PDSonQueue, Algorithm 2 shows the pseudo code of our resource allocation strategy. Algorithm 2 refers our previous work [**?**]. More detailed theory of this resource allocation are presented in work [**?**].

Algorithm 1 PDSonQueue Scheduler: Initialisation phase

1: Res = $(r_1, r_2, ..., r_p)$ → total resources capacities
2: Com = $(c_1, c_2, ..., c_p)$ → consumed resources, initial value = 0
3: Res_{rem} = $(rem_1, rem_2, ..., rem_p)$ → remaining available resources, initial value = Res, $Rem_{rem} = Res - Com$
4: Dem_i = $(de_1, de_2, ..., de_p)$ → the demand resource of job_i
5: Dos_z $(z = 1, 2, ...q)$ → user z 's dominant shares, initial value = 0
6: All_z = $(a_{z,1}, a_{z,2}, ..., a_{z,p})$ $(z = 1, 2, ..., q)$ → the resources allocated to user z, initial value = 0

Algorithm 2 PDSonQueue Scheduler: Resource allocation phase

1: $Lev = i$ $(i = 1, 2, 3)$ → receive the resource amount according to the level. The lower the level is, the more dominant resource. $Level = 1$ indicates the most dominant resource value.
2: // job_i is a regular job or under W_i^{wai} of deadline constraint job_i
3: **while** $(W_i^{wai} > 0)$ **do**
4: select user z with lowest dominant share Dos_z
5: Dem_i → demand of user z's next job
6: **if** $Com + Dem_i \leq Res$ **then**
7: $Com = Com + Dem_i$ → update consumed resources
8: $All_z = All_z + Dem_i$ → update user z's resource allocation
9: $Res_{rem} = Res - Com$ → update available resources
10: $Dos[] = sort_{n=1}^{p}(a_{z,n}/r_n)$ → calculate the dominant share of each user
11: $Dos_z = Dos[Dos.length - Lev]$ → determine dominant share degree
12: **else**
13: **return** → the cloud cluster is full
14: **end if**
15: **end while**
16: preempt resource and allocate required resource Dem_i to job_i
17: $Com = Com + Dem_i$
18: **return** job_i begins to run

Part C
MORE EXPERIMENTAL RESULTS

We evaluate our PDSonQueue vs YARN fair scheduler performance using 4 real world applications. The first case has been presented in the main paper. Here we describe cases 2, 3 and 4 and their experimental results.

C. Real world applications

Case 2: Number plate image recognition: This License Plate Recognition System (LPRS) recognises a vehicle plate license from images with edge detection used to identify points in digital images with discontinuities. A 40G image data file is loaded and read once from disk. This application is predominantly ***CPU-intensive***.

Case 3: Hadoop log file text search: This application tracks Hadoop's logs to search for error information using a simple lambda expression based on the "error" string, identifying a cluster's health status and weakness. Its complexity is low and it consumes little CPU resource. A 40G log file is buffered in memory, read and searched. Hence, this application is ***memory-intensive***.

Case 4: Hadoop data migration: In a Hadoop cluster, the input file is split into one or more blocks stored in a set of DataNodes (running on commodity machines). When data volume is huge, tasks split from jobs are deployed on one node, however the needed data may be stored on different nodes and even different racks. Thus the system needs to copy other nodes' data to this destination node. A 40G telecommunications data file is copied and transmitted among nodes. It is ***I/O-intensive***.

D. Real application evaluation results

Deadline-based QoS, throughput, completion time and completion rate when running use cases 2, 3 and 4 together:

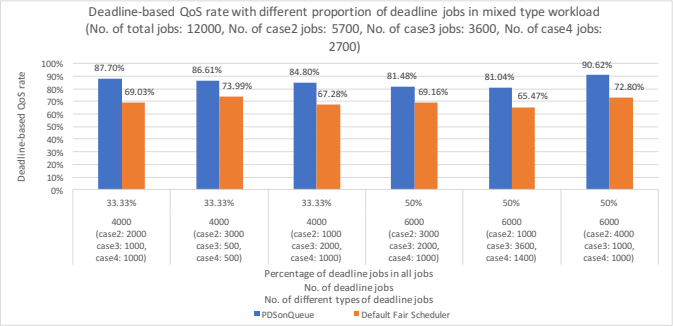

Fig. 1. Deadline-based QoS rate on a mixed use cases workload

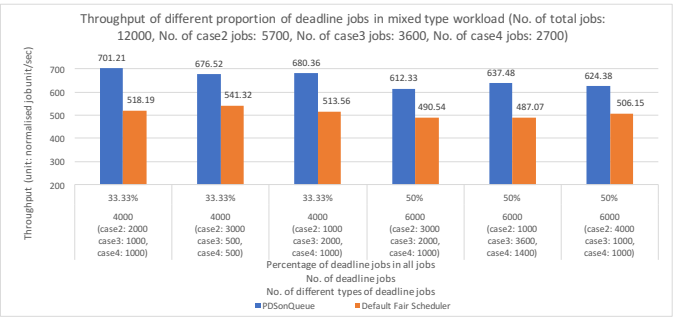

Fig. 2. Throughput on a mixed use cases workload

There are 6 different group combinations. Group 1 has 4,000 deadline jobs including 2,000 use case 2 jobs, 1,000 use case 3 jobs and 1,000 use case 4 jobs and other group combinations are presented in horizontal axis in Fig. 1. Fig. 1 presents QoS rate, and our PDSonQueue's QoS rate is 85.37% and fair scheduler' QoS rate is 69.62%, which has 15.75% improvement. When deadline jobs occupy 33% of total jobs, for both schedulers, the QoS achievement rate is higher than that of the deadline jobs occupying 50% of the total jobs.

When the number of deadline jobs is high, there are more preemptions from regular jobs and more failed deadline jobs.

In Fig. 2, PDSonQueue's average throughput is 655.37 job units/s and fair scheduler's is only 509.47 job units/s. PDSonQueue can improve by 28.63% for throughput. When the rate of deadline jobs is 33%, PDSonQueue's throughput is 686.02 job units/s and fair scheduler's throughput is only 524.35 job units/s. Yet, when the rate of deadline jobs is 50%, PDSonQueue's throughput reduces to 624.72 job units/s, and fair scheduler's throughput also cuts down to 494.58 job units/s. When deadline jobs are less, the throughput is higher.

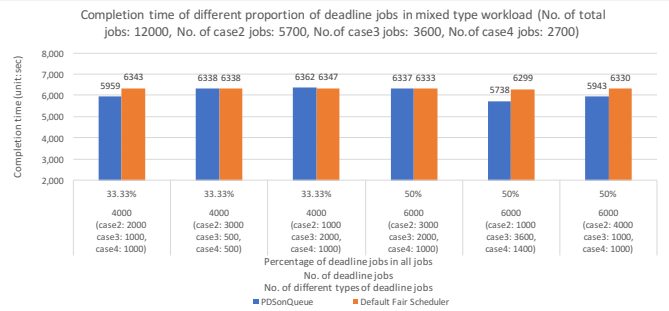

Fig. 3. Completion time on a mixed use cases workload

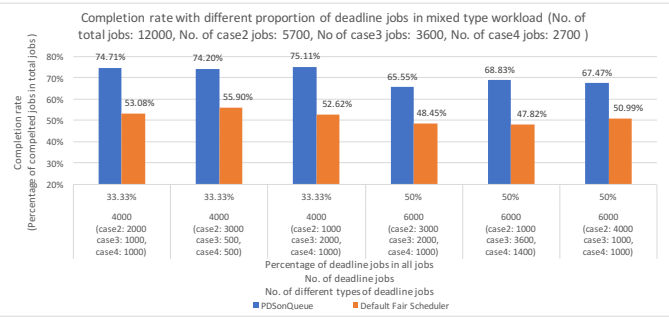

Fig. 4. Completion rate on a mixed use cases workload

Fig. 3 shows completion times of different job combinations. The average completion time is 6112.83s using PDSonQueue and average completion time is 6331.45s. PDSonQueue can reduce completion time of 218.62s on average, compared with YARN scheduler.

In Fig. 4, PDSonQueue's completion rate is 70.98% and fair scheduler's is 51.48%, which is a near 20% improvement.