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 [1]:

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

In general,

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

According to regularity condition $\sum\limits_{k=0}^{\infty} p_k = 1$, when $\rho < 1$, we can get

$$1 = (\sum_{k=0}^{n-1} \frac{\rho_1^k}{k!} + \sum_{k=n}^{\infty} \frac{\rho_1^k}{n!n^{k-n}})p_0 = (\sum_{k=0}^{n-1} \frac{\rho_1^k}{k!} + \frac{\rho_1^n}{n!}\frac{1}{1-\rho})p_0$$

Thus, we can gain an equation for p_0

$$p_0 = (\sum_{k=0}^{n-1} \frac{\rho_1^k}{k!} + \frac{\rho_1^n}{n!}\frac{1}{1-\rho})^{-1} \tag{3}$$

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,

$$\begin{array}{l}
E(\bar{L}_{wai}^2) = \sum\limits_{k=n}^{\infty} (k-n)^2 p_k = \sum\limits_{h=1}^{\infty} h^2 p_{h+n} \\
= \sum\limits_{h=1}^{\infty} \frac{h^2}{n!n^h}(n\rho)^{h+n} p_0 \\
= \frac{(n\rho)^n \rho^2 p_0}{n!} \sum\limits_{h=2}^{\infty} h(h-1)\rho^{h-2} + \frac{(n\rho)^n \rho p_0}{n!} \sum\limits_{h=1}^{\infty} h\rho^{h-1} \\
= \frac{2\rho^2 \rho_1^n p_0}{n!(1-\rho)^3} + \bar{L}_{wai} = \frac{1+\rho}{1-\rho}\bar{L}_{wai}
\end{array} \tag{4}$$

Thus,

$$\sigma^2(\bar{L}_{wai}) = E(\bar{L}_{wai}^2) - [E(\bar{L}_{wai})]^2 = \bar{L}_{wai}(\frac{1+\rho}{1-\rho} - \bar{L}_{wai}) \tag{5}$$

In addition,

$$E(L_{sys}) = E(L_{wai}) + E(L_{ser}) \tag{6}$$

$$E(W_{soj}) = E(W_{wai}) + E(T_{ser}) \tag{7}$$

we can easily get

$$E(T_{ser}) = \frac{1}{\mu} \tag{8}$$

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 $E(T_{ser}) = \frac{1}{\mu}$ [2], [3].

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,

$$E(L_{wai}|q \ge n) = \frac{\rho/n}{1-\rho/n} = \frac{\rho}{n-\rho} \tag{9}$$

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

$$\begin{array}{l}
f_w(x) = \frac{p_0(\frac{\lambda}{\mu})^n}{n!} n\mu e^{-n\mu x} \sum\limits_{j=0}^{\infty} \frac{(\rho n\mu x)^j}{j!} \\
= \frac{(\frac{\lambda}{\mu})^n}{n!} p_0 n\mu e^{-(n\mu-\lambda)x} \\
= \frac{(\frac{\lambda}{\mu})^n}{n!} p_0 n\mu e^{-n\mu(1-\rho)x} \\
= \frac{(\frac{\lambda}{\mu})^n}{n!} p_0 \frac{1}{1-\rho} n\mu(1-\rho) e^{-n\mu(1-\rho)x} \\
= P(Waiting) n\mu(1-\rho) e^{-n\mu(1-\rho)x}
\end{array} \tag{10}$$

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

$$\begin{array}{l}
P(W > x) = \int\limits_x^{\infty} f_w(u)du = P(Waiting)e^{-n\mu(1-\rho)x} \\
= C(n,\rho) \bullet e^{-\mu(n-\frac{\rho}{n})x}
\end{array} \tag{11}$$

The distribution function of waiting time can be written as:

$$\begin{array}{l}
F_w(x) = 1 - P(Waiting) + P(Waiting)(1 - e^{-n\mu(1-\rho)x}) \\
= 1 - P(Waiting)e^{-n\mu(1-\rho)x} \\
= 1 - C(n,\rho) \bullet e^{-\mu(n-\frac{\rho}{n})x}
\end{array} \tag{12}$$

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

$$f_s(x) = P(No\ waiting)\mu e^{-\mu x} + f_{w+ser}(x) \tag{13}$$

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

$$\begin{array}{l}
f_{w+ser}(z) = \int\limits_0^z f_w(x)\mu e^{-\mu(z-x)}dx \\
= P(Waiting)n\mu(1-\rho)\mu \int\limits_0^z e^{-n\mu(1-\rho)x} e^{-\mu(z-x)}dx \\
= \frac{(n\rho)^n}{n!} p_0 \frac{1}{(1-\rho)} n\mu(1-\rho)\mu e^{-z\mu} \int\limits_0^z e^{-\mu(n-1-\frac{\lambda}{\mu})x}dx \\
= \frac{(n\rho)^n}{n!} p_0 n\mu \frac{1}{(n-1-\frac{\lambda}{\mu})} e^{-z\mu}(1 - e^{-\mu(n-1-\frac{\lambda}{\mu})z})
\end{array} \tag{14}$$

Therefore,

$$\begin{array}{l}
f_s(x) = (1 - (\frac{\lambda}{\mu})^n \frac{p_0}{n!(1-\rho)})\mu e^{-\mu x} + \\
\qquad \frac{(\frac{\lambda}{\mu})^n}{n!} n\mu p_0 \frac{1}{(n-1-\frac{\lambda}{\mu})} e^{-\mu x}(1 - e^{-\mu(n-1-\frac{\lambda}{\mu})x}) \\
= \mu e^{-\mu x}(1 - \frac{(\frac{\lambda}{\mu})^n p_0}{n!(1-\rho)} + \frac{(\frac{\lambda}{\mu})^n}{n!} n p_0 \frac{1}{(n-1-\frac{\lambda}{\mu})}(1 - e^{-\mu(n-1-\frac{\lambda}{\mu})x})) \\
= \mu e^{-\mu x}(1 + \frac{(\frac{\lambda}{\mu})^n p_0}{n!(1-\rho)} \frac{1-(n-\frac{\lambda}{\mu})e^{-\mu(n-1-\frac{\lambda}{\mu})x}}{(n-1-\frac{\lambda}{\mu})})
\end{array} \tag{15}$$

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

$$\begin{array}{l}
P(S > x) = \int\limits_x^{\infty} f_s(y)dy = \\
\int\limits_x^{\infty} \mu e^{-\mu y} + \frac{(\frac{\lambda}{\mu})^n p_0}{n!(1-\rho)} \frac{1}{(n-1-\frac{\lambda}{\mu})}(\mu e^{-\mu y} - \mu(n-\frac{\lambda}{\mu})e^{-\mu(n-\frac{\lambda}{\mu})y})dy \\
= e^{-\mu x} + (\frac{\lambda}{\mu})^n p_0 \frac{1}{n!(1-\rho)(n-1-\frac{\lambda}{\mu})}(e^{-\mu x} - e^{-\mu(n-\frac{\lambda}{\mu})x}) \\
= e^{-\mu x}(1 + \frac{(\frac{\lambda}{\mu})^n p_0}{n!(1-\rho)} \frac{1-e^{-\mu(n-1-\frac{\lambda}{\mu})x}}{(n-1-\frac{\lambda}{\mu})})
\end{array} \tag{16}$$

Therefore the distribution function can be presented as

$$F_s(x) = 1 - P(S > x) \tag{17}$$

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 an improved DRF allocation strategy proposed in work [4]. The improved DRF <CPU, memory, vdisk> can allocate the corresponding resource to jobs, based on different types of dominant shared resources and this enables all tenants to share the cloud system's resources reasonably and efficiently. More detailed theory of this resource allocation are presented in work [4].

Algorithm 1 PDSonQueue Scheduler: Initialisation phase

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

Algorithm 2 PDSonQueue Scheduler: Resource allocation phase

```
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.
// job_i is a regular job or under W_i^{wai} of deadline constraint job_i
while (W_i^{wai} > 0) do
    select user z with lowest dominant share Dos_z
    Dem_i  → demand of user z's next job
    if Com + Dem_i ≤ Res then
        Com = Com + Dem_i  → update consumed resources
        All_z = All_z + Dem_i  → update user z's resource allocation
        Res_rem = Res − Com  → update available resources
        Dos[] = sort_{n=1}^{p}(a_{z,n}/r_n)  → calculate the dominant share of each user
        Dos_z = Dos[Dos.length − Lev]  → determine dominant share degree
    else
        return  → the cloud cluster is full
    end if
end while
preempt resource and allocate required resource Dem_i to job_i
Com = Com + Dem_i
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:

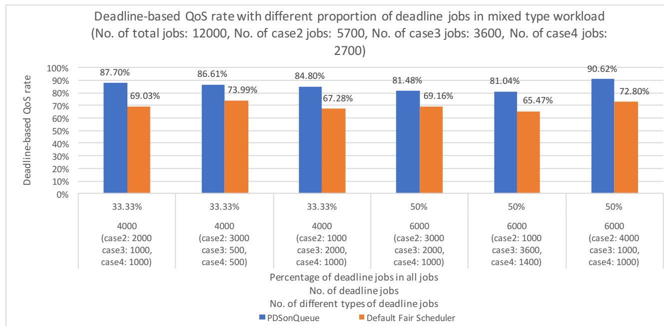

Fig. 1. Deadline-based QoS rate on a mixed use cases workload

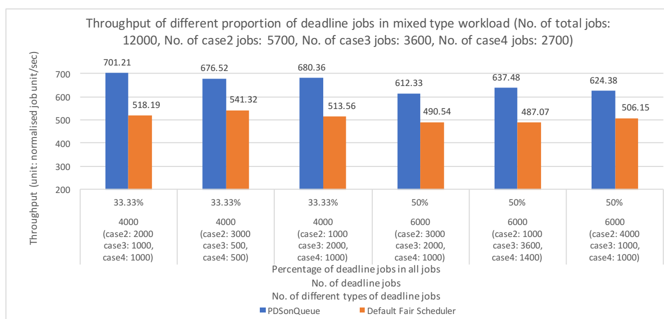

Fig. 2. Throughput on a mixed use cases workload

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.

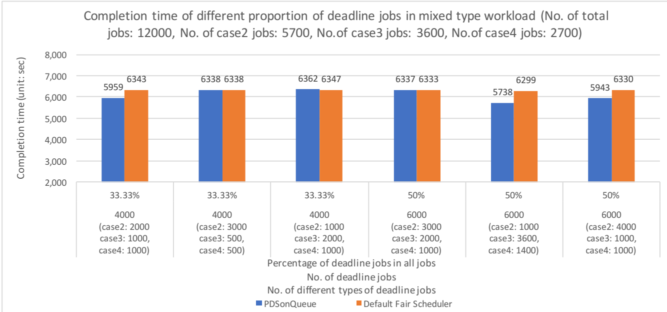

Fig. 3. Completion time on a mixed use cases workload

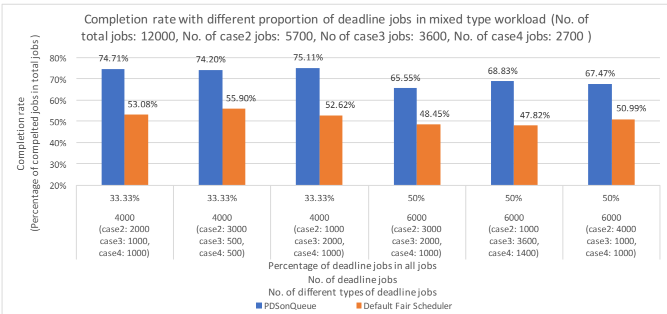

Fig. 4. Completion rate on a mixed use cases workload

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.

Part D
THRESTS TO VALIDITY

Construct threats include the wrong choice of evaluation metrics, incorrect collection of metrics, and incorrect inference of performance from these metrics. We chose well-known and accepted metrics in cloud computing system performance and job scheduling evaluation. We collected data using YARN provided APIs and scheduler statistics, and ran jobs using the different schedulers with the same experimental set-up, data collection and data analysis.

Internal threats include our choice of job mixes, interaction of different resource needs in mixed-jobs, and our cloud platform resources available. We tried to mitigate using a variety of job mixes, a representative cloud platform and configuration, and running different sets of experiments with different workload, resource, job type and job number mixes.

The key **external** threat validity of our experiments is how generalisable the results are due to the limited number of benchmark and real-world applications we have run them on. We chose a mix of benchmarks, mix of quite different real-world applications, and mix of job types to mitigate this.

REFERENCES

[1] W. Ellens, J. Akkerboom, R. Litjens, H. van den Berg *et al.*, "Performance of cloud computing centers with multiple priority classes," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 245–252.

[2] L. Guo, T. Yan, S. Zhao, and C. Jiang, "Dynamic performance optimization for cloud computing using m/m/m queueing system," *Journal of Applied Mathematics*, vol. 2014, 2014.

[3] J. Sztrik, "Basic queueing theory," *University of Debrecen, Faculty of Informatics*, vol. 193, 2012.

[4] J. Ru, J. Grundy, Y. Yang, J. Keung, and L. Hao, "Providing fairer resource allocation for multi-tenant cloud-based systems," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*. IEEE, 2015, pp. 306–313.