A New Competitive Ratio for Network Applications with Hard Performance Guarantees

Han Deng
Department of ECE
Texas A&M University
College Station, TX 77840, USA
Email: hdeng@email.tamu.edu

I-Hong Hou
Department of ECE
Texas A&M University
College Station, TX 77840, USA
Email: ihou@tamu.edu

Abstract—**Network applications in highly mobile systems need to employ online algorithms that do not rely on precise predictions about future events. In order to meet hard performance guarantees in the presence of unknown future events, common practice is to add redundancy to the systems. In this paper, we define a new competitive ratio that reflects the amount of redundancy needed to ensure some given performance guarantees. We study two special applications, namely, online job allocations in cloud computing and online scheduling in delayed mobile offloading, and propose online algorithms for each of them. We prove that our algorithms are optimal. By comparing our algorithms with commonly used other policies, we also show that our policies need much less redundancy than those commonly used policies to provide the same degree of performance guarantees.**

I. Introduction

Many existing studies use *competitive ratio* to evaluate the performance of online algorithm that needs to make decisions without any knowledge of future events. Traditionally, competitive ratio is defined as the worst-case ratio between the performance of an online algorithm, and that of the optimal solution. For example, Moharir, and Sanghavi and Shakkottai [1] have studied the competitive ratios of online policies for job allocations in cloud computing, and proposed an online algorithm that guarantees to allocate $1-\frac{1}{e}$ as many jobs as the optimal policy. Mao, Koksal, and Shroff [2] have studied the packet scheduling problem in multi-hop networks with end-to-end deadline guarantees. They proposed an online scheduling policy that has a competitive ratio of $O(P \log P)$, meaning that the policy is guaranteed to deliver $1/O(P \log P)$ as many packets as the optimal policy, where P is the length of the longest path in the network.

While existing studies on competitive ratios reveal important insights about online algorithms, they are insufficient to capture service requirements of many emerging network applications. For example, by only guaranteeing to allocate $1-\frac{1}{e} \approx 63\%$ as many jobs as the optimal policy, the algorithm in [1] may need to drop up to 37% of jobs. In a network whose longest path has length 10, the policy in [2] may drop more than 90% of packets. Most emerging network applications cannot survive such high drop ratios.

In order to meet the stringent service requirement of network applications, current practice is to add more redundancy to the system, such as increasing the number of servers in data centers, and increasing network bandwidth. Motivated by this observation, we define a new competitive ratio that captures the minimum amount of redundancy needed to provide a certain degree of service guarantees.

Our recent studies [3], [4] have evaluated competitive ratios of online algorithms for two specific applications: online job allocation in cloud computing, and online packet scheduling for delayed mobile offloading. We demonstrate that both applications can be formulated as linear programming problems that bear some similarities with the well-studied online packing problem [5]. For both applications, adding redundancy into the system is equivalent to relaxing some constraints in their corresponding linear programming problems. We then propose online algorithms and demonstrate that they are optimal, or close to optimal, in view of our new definition of competitive ratio. This paper provides a brief overview of the studied applications and the technical approach of evaluating the new competitive ratio.

II. An Example of Online Packing Problem with Standard Competitive Ratio Analysis

We first give an overview of the basic primal-dual approach for studying the competitive ratio. We focus on the ad-auction problem, which is a special case of the online packing problem [5]. There are a set of n buyers, and each buyer i has a budget of $B(i)$. Seller has a collection of m products which arrives one by one. When a product j arrives, each buyer i will provide its bid $b(i,j)$ for this product. The seller chooses a buyer i^* and sells the product to it with price $b(i^*,j)$. We use $y(i,j)$ to be the indicator function that product j is sold to buyer i. The goal is to maximize the total revenue of the seller. There are two constraints of the problem. First, buyers cannot spend more than their budgets. Second, each product cannot be allocated to more than one buyer. Thus we have the following linear programming:

Packing

$$Max \quad \sum_{i,j} b(i,j)y(i,j) \tag{1}$$

$$s.t. \quad \sum_{j} b(i,j)y(i,j) \leq B(i), \forall 1 \leq i \leq n, \tag{2}$$

$$\sum_{i} y(i,j) \leq 1, \forall 1 \leq j \leq m, \tag{3}$$

$$y(i,j) \geq 0, \forall i,j. \tag{4}$$

The dual of **Packing** is:

Covering

$$Min \quad \sum_{i} B(i)x(i) + \sum_{j} z(j) \tag{5}$$

$$s.t. \quad b(i,j)x(i) + z(j) \geq b(i,j), \forall i,j, \tag{6}$$

$$x(i), z(j) \geq 0, \forall i,j. \tag{7}$$

When a new product arrives, a new set of constants $b(i,j)$ appear for the **Packing** problem, and the seller needs to determine a new set of variables $y(i,j)$. At the mean time, a new set of constraints in the form of (6) appear for the **Covering** problem, and the seller needs to determine a new set of variables $z(j)$ and updates existing ones $x(i)$.

It is well-known that any feasible solution of **Covering** has an objective function, $\sum_i B(i)x(i)+\sum_j z(j)$, that is no smaller than that of the optimal solution of **Packing**. The standard primal-dual approach aims to determine and update variables $x(i), y(i,j)$ and $z(j)$ so as to maintain a desirable ratio between its solutions to **Covering** and **Packing**. A detailed description of the approach is shown in Algorithm 1.

Algorithm 1 Algorithm for Online Ad-auction Problem

```
1: Initially, $x_i = 0$, $y_j = 0$.
2: for each new product $j$ do
3:     $i^* \leftarrow \text{argmax}_i\, b(i,j)(1 - x(i))$
4:     if $x(i^*) < 1$ then
5:         $z(j) \leftarrow b(i^*,j)(1 - x(i^*))$
6:         $x(i^*) \leftarrow x(i^*)(1 + \frac{b(i^*,j)}{B(i^*)}) + \frac{b(i^*,j)}{(c-1)\cdot B(i^*)}$
7:         $y(i^*,j) \leftarrow 1$
8:         Product $j$ is assigned to buyer $i^*$, charge the buyer the minimum between $b(i^*,j)$ and its remaining budget.
```

Theorem 1 ([5]): Algorithm 1 is $(1-1/c)(1-Q_{max})$-competitive, where $c = (1+Q_{max})^{(\frac{1}{Q_{max}})}$, $Q_{max} = \max_{i,j}\{\frac{b(i,j)}{B(i)}\}$. When $Q_{max} \to 0$, the competitive ratio goes to $(1-1/e)$.

III. Online Job Allocation with Hard Allocation Ratio Requirement

In this section, we discuss the online job allocation problem for cloud computing. For example, in YouTube, or Netflix, each video is replicated only in a small number of servers, and each server can only serve a limited number of streams simultaneously. When a user makes a request to watch a video, a server will be selected for this request. If no server can process this request, the request is dropped, which can significantly impact user satisfaction. The system performance depends on the number of requests that can be allocated. For cloud computing, merely achieving a competitive ratio of $(1-1/e)$ is not acceptable, as even a small portion of dropped requests can result in significant revenue loss. Therefore, we need a different concept of competitive ratio to evaluate online policies.

A. System Model

We formulate the problem of maximizing the number of allocated requests as a linear programming problem.

We use $\mathcal{J}$ to denote the set of servers, and $\mathcal{I} = \{1,2,\dots\}$ to denote the arrival sequence of jobs. A server j has a total amount of C_j units capacity. Each job i can be served by a subset of servers. If $K_{ij}=1$, then job i can be served by server j; otherwise, i cannot be served by j. The corresponding server subset is revealed when job i arrives. The system either allocates it to a server or drop it. If job i is allocated to server j, then job i takes one unit of capacity in j. We use X_{ij} to denote the assignment of the jobs. If $X_{ij}=1$, then job i is assigned to server j. If $X_{ij}=0$, job i is not assigned to server j.

Similarly to section II, we can use linear programming to formulate this problem.

Allocation:

$$Max \sum_{ij} K_{ij}X_{ij} \tag{8}$$

$$s.t. \sum_{i} K_{ij}X_{ij} \leq C_j, \forall j \in \mathcal{J}, \tag{9}$$

$$\sum_{j} K_{ij}X_{ij} \leq 1, \forall i \in \mathcal{I}, \tag{10}$$

$$X_{ij} \geq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{11}$$

In order to allocate a high ratio of all requests without knowing any information about future requests, current practice is to increase the capacity of servers by adding redundancy. Suppose that the system capacity is increased by R times. Then the problem of maximizing the number of served jobs can be formulated as follows:

Allocation(R):

$$Max \sum_{ij} K_{ij}X_{ij} \tag{12}$$

$$s.t. \sum_{i} K_{ij}X_{ij} \leq RC_j, \forall j \in \mathcal{J}, \tag{13}$$

$$\sum_{j} K_{ij}X_{ij} \leq 1, \forall i \in \mathcal{I}, \tag{14}$$

$$X_{ij} \geq 0, \forall i \in \mathcal{I}, j \in \mathcal{J}. \tag{15}$$

We evaluate the performance of online policies by comparing the number of allocated jobs under online policy with R times the servers' capacity against that under offline policy with unit capacity. Specifically, given K_{ij} and $\{C_j\}$, let Γ_{opt} be the optimal value of $\sum_{ij} K_{ij}X_{ij}$ in **Allocation**, and $\Gamma_\eta(R)$ be the value of $\sum_{ij} K_{ij}X_{ij}$ in **Allocation(R)** under policy η. We define the *competitive ratio* as follows:

Definition 1: An online policy η is said to be (R, θ)-*competitive* if there exists a function $\Theta(C_{min})$ such that for all $\{K_i\}$ and $\{C_j\}$ with $C_j \geq C_{min}$, $\Gamma_{opt}/\Gamma_\eta(R) \leq \Theta(C_{min})$, and $\Theta(C_{min}) \to \theta$ as $C_{min} \to \infty$.

B. Scheduling Policy

In this section, we will propose an online policy for the online allocation problem using primal dual method as introduced in Section II.

The dual problem of **Allocation** can be written as:

Dual(Allocation):

$$Min \sum_j C_j\alpha_j + \sum_i \beta_i, \tag{16}$$

$$s.t.\ \alpha_j + \beta_i \geq 1, \forall i \in \mathcal{I}, K_{ij} = 1 \tag{17}$$

$$\alpha_j \geq 0, \forall j, \tag{18}$$

$$\beta_i \geq 0, \forall i, \tag{19}$$

Initially, the system sets α_j, β_i, and X_{ij} to be 0. When a job i arrives, the policy checks the values of α_j for all server j with $K_{ij} = 1$, and selects server j^* which has the minimum value of α_j. If $\alpha_{j^*} < 1$, job i is allocated to server j^*, and α_{j^*} will be increasing y ba certain ratio as $\alpha_{j^*}(1 + \frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*} - 1)C_{j^*}}$, where $d_j = (1 + 1/C_j)^{RC_j}$, for all j. The value of d_j is chosen to achieve the optimal competitive ratio. If $\alpha_{j^*} \geq 1$, job i is discarded. The policy is shown as Algorithm 2 [3].

Algorithm 2 PD Algorithm for Online Job Allocation

```
1: Initially, $\alpha_j = 0$, $\beta_i = 0$, $X_{ij} = 0$.
2: $d_j \leftarrow (1 + 1/C_j)^{RC_j}, \forall j$.
3: for each arriving job $i$ do
4:     $j^* \leftarrow \text{argmin}_{K_{ij}=1}\, \alpha_j$.
5:     if $\alpha_{j^*} < 1$ then
6:         $\beta_i \leftarrow 1 - \alpha_{j^*}$.
7:         $\alpha_{j^*} \leftarrow \alpha_{j^*}(1 + \frac{1}{C_{j^*}}) + \frac{1}{(d_{j^*} - 1)C_{j^*}}$.
8:         $X_{ij^*} \leftarrow 1$.
9:         Job $i$ is assigned to server $j^*$.
10:    else
11:        Discard job $i$.
```

C. Competitive Ratio

We have the following theorems on the competitive ratios of online policies. The detailed proof can be found in [3].

Theorem 2: Algorithm 2 is $(R, \frac{e^R}{e^R - 1})$-competitive.

Proof: We prove Theorem 2 by three steps:

First, we show that all constraints in **Dual(Allocation)** are satisfied. Initially, α_j and β_i are 0. By step 7 in Algorithm 2, α_j is non-decreasing, and thus (18) holds. From step 5 and step 6 we know that $\beta_i \geq 0$ constaint (19) holds. If $\alpha_j \geq 1$, constraint (17) holds. If $\alpha_j < 1$, $\beta_i = 1 - \alpha_{j^*} \geq 1 - \alpha_j$. Thus $\alpha_j + \beta_i \geq 1$, constrraint (17) holds.

Second, we show that all constraints in **Allocation(R)** are satisfied. By induction, we can prove that $\alpha_j[n] = (\frac{1}{d_j - 1})(d_j^{n/RC_j} - 1)$, where $\alpha_j[n]$ is the value of α_j after n jobs have been allocated to server j. Then after RC_j jobs have been allocated to server j, $\alpha_j = 1$. Since step 5 in Algorithm 2 shows that jobs can be allocated to servers when $\alpha_j < 1$, we know that constraint (13) is satisfied.

Next, we derive the ratio between $\sum_{ij} X_{ij}$ and $\sum_j C_j\alpha_j + \sum_i \beta_i$. Both formulas are initially 0. We now consider the amounts of change of these two formulas when a job i arrives. We use $\Delta P(R)$ to denote the change of $\sum_{ij} X_{ij}$, and ΔD to denote the change of $\sum_j C_j\alpha_j + \sum_i \beta_i$.

If job i is discarded, then $\{X_{ij}\}$, $\{\alpha_j\}$ and $\{\beta_i\}$ remain unchanged, and therefore $\Delta P(R) = \Delta D = 0$.

On the other hand, consider the case when job i is assigned to server j. We have $X_{ij} = 1$ and $\Delta P(R) = 1$. We also have

$$\begin{aligned}\frac{\Delta D}{\Delta P(R)} = \Delta D =& C_j(\frac{\alpha_j}{C_j} + \frac{1}{(d_j - 1)C_j}) + 1 - \alpha_j \\ =& 1 + \frac{1}{d_j - 1} = \frac{d_j}{d_j - 1} \\ =& \frac{(1 + 1/C_j)^{RC_j}}{(1 + 1/C_j)^{RC_j} - 1}.\end{aligned}$$

When we imposes a lower bound on C_j by requiring $C_j \geq C_{min}$, for all j, and let $C_{min} \to \infty$, we have $\frac{\Delta D}{\Delta P(R)} \to \frac{e^R}{e^R - 1}$, whenever a job i is allocated to some server. Therefore, we have,

$$\frac{\sum_j C_j\alpha_j + \sum_i \beta_i}{\sum_{ij} X_{ij}} = \frac{e^R}{e^R - 1}.$$

Finally, by weak duality, Algorithm 2 is $(R, \frac{e^R}{e^R - 1})$-competitive. ■

Theorem 3: Any online policy cannot be better than $(R, \frac{e^R}{e^R - 1})$-competitive.

From Theorem 3 and 2, we know that our proposed policy achieves the optimal bound.

We also evaluate the new competitive ratio for two commonly used allocation policies, namely, join the shortest queue (JSQ) and join the most residue queue(JMQ). JSQ allocates each arriving job to the server with the fewest jobs. JMQ allocates each arriving job to the server with most remaining space, which is the server capacity minus the number of allocated jobs in this server.

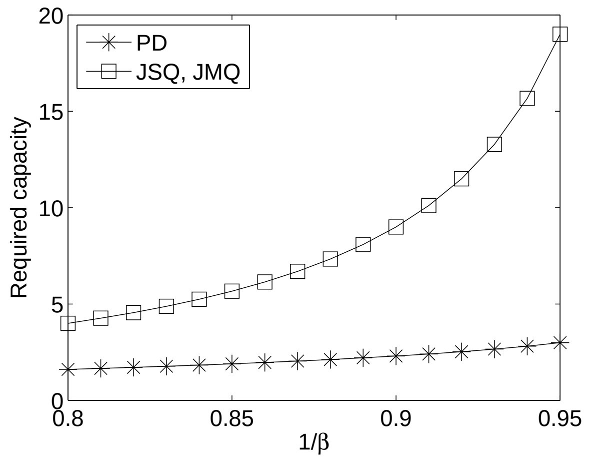

Fig. 1. Capacity requirements of job allocation problem.

Theorem 4: JSQ and JMQ cannot be better than $(R, 1+\frac{1}{R})$-competitive-per-sample-path.

Our results can be interpreted as follows: Suppose the offline optimal policy can allocate all jobs. To guarantee that at least $1-\frac{1}{\theta}$ of the jobs are allocated by online policies, our policy needs to increase the capacity by $R = \ln\theta$ times so that $\frac{e^R}{e^R-1} \leq 1/(1-\frac{1}{\theta})$. On the other hand, JSQ and JMQ both need to increase at least $(\theta-1)$ times, which is an order higher capacity to achieve the same allocation ratio requirement than our policy. We illustrate the capacity requirement for different allocation ratios in Fig. 1.

IV. Online Scheduling for Delayed Mobile Offloading

In this section, we study another problem called delayed mobile offloading. Delayed mobile offloading aims to offload mobile traffic from cellular networks to WiFi networks so as to reduce the congestion of cellular networks.

A. System Model

We consider a cellular network system in which WiFi hotspots are deployed by the cellular operator to reduce the network congestion. Mobile users are moving within the area of the network. They may enter the system at any time and any locations. We use $\mathcal{M}$ to denote the set of WiFi APs and $\mathcal{I}$ to denote the set of mobile users. User information is revealed only after it enters the system, that is, a mobile user i specifies the amount of data, denoted by C_i, that it needs to obtain, and a deadline T_i. Here we assume that time is slotted and numbered as $t = 1, 2, \ldots$.

Users move within the system and tries to download as much data from WiFi APs as possible before the deadline. After deadline, users downloads the remaining data from cellular network. As the users are moving in the system, the connectivity and channel capacity between APs and users may change from time to time. A user is only able to connect to one AP at any point of time. To represent the connectivity of a client i and AP m at time slot t, we use K_{imt} to denote the channel capacity. At time t, if there is no connection between i and m, we set $K_{imt} = 0$. Since user i cannot connect to any server before its entrance time and after deadline T_i, we set those corresponding K_{imt} to be 0. We normalize the system so that $0 \leq K_{imt} \leq 1$ for all i, m, t.

When client i is connected to AP m at time t, the AP determines the portion of time it spends on transmitting to i in time slot t, denoted by X_{imt}. Thus, the amount of data that mobile user i obtains from m during time t is $K_{imt}X_{imt}$. Our goal is to maximize the total data delivered by WiFi, which is the total amount of $K_{imt}X_{imt}$ of all clients from all APs before their respective deadlines.

Similar to the previous section, we assume that the cellular operator may be able to increase the capacity of WiFi hotspots. When the capacity of WiFi APs are increased by R times, we can also describe the system as increasing the serving time by R times with the original capacity. The problem of maximizing total offloaded traffic can be written as the following linear programming problem:

Offload(R):

$$Max \sum_{imt} K_{imt}X_{imt} \tag{20}$$

$$s.t. \sum_{mt} K_{imt}X_{imt} \leq C_i, \forall i \in \mathcal{I}, \tag{21}$$

$$\sum_{i} X_{imt} \leq R, \forall m \in \mathcal{M}, t, \tag{22}$$

$$X_{imt} \geq 0, \forall i \in \mathcal{I}, m \in \mathcal{M}, t. \tag{23}$$

We use Γ_{opt} to denote the offline optimal value of $\sum_{imt} X_{imt}K_{imt}$ when $R = 1$. Let $\Gamma_\eta(R)$ be the value of $\sum_{imt} X_{imt}K_{imt}$ for the Offload(R) problem under policy η. We define the *competitive ratio* below:

Definition 2: A policy η is said to be (R, β)-competitive if $\Gamma_{opt}/\Gamma_\eta(R) \leq \beta$, as $\min_{i\in\mathcal{I}} C_i \to \infty$, for all systems.

B. Scheduling Policy

In this section, we first propose an online policy using PD method. The dual problem to **Offload(R)** when $R = 1$ is:

Dual(Offload):

$$Min \sum_{mt} Y_{mt} + \sum_{i} C_i Z_i, \tag{24}$$

$$s.t.\ Y_{mt} + K_{imt}Z_i \geq K_{imt}, \forall i, m, t, \tag{25}$$

$$Y_{mt} \geq 0, \forall m, t, \tag{26}$$

$$Z_i \geq 0, \forall i, \tag{27}$$

Initially, X_{imt}, Y_{mt}, and $Z_i = 0$ are set to be 0. APs keep track of and update Z_i for each client i. At each time t, each AP m chooses a client which has maximum value of $K_{imt}(1-Z_i)$. Let this client be i^*_m. If $K_{imt}(1-Z_i) > 0$, then we set $X_{i^*_m mt}$ to be R. AP m updates $Z_{i^*_m}$ by setting it to be $Z_{i_m^*}(1+\frac{K_{i_m^* mt}}{C_{i^*}}) + \frac{K_{i_m^* mt}}{(d-1)C_{i_m^*}}$. Here d is set to be $(1+1/C_{min})^{C_{min}}$. AP m broadcasts the updated $Z_{i_m^*}$ to all APs. The detailed aglorithm is described as Algorithm 3.

Algorithm 3 PD Algorithm for Online Mobile Offloading

```
1:  Initially, $X_{imt} = 0$, $Y_{mt} = 0$, $Z_i = 0$.
2:  $C_{min} \leftarrow \min_i C_i, d \leftarrow (1+1/C_{min})^{C_{min}/R}$.
3:  for each time slot $t$ do
4:      for each AP $m$ do
5:          $i_m^* \leftarrow \underset{i}{\mathrm{argmax}}\{K_{imt}(1-Z_i)\}$.
6:          if $K_{i_m^* mt}(1-Z_{i_m^*}) > 0$ then
7:              $Y_{mt} \leftarrow K_{i_m^* mt}(1-Z_{i_m^*})$.
8:              $Z_{i_m^*} \leftarrow Z_{i_m^*}(1+\frac{K_{i_m^* mt}}{C_{i_m^*}}) + \frac{K_{i_m^* mt}}{(d-1)C_{i_m^*}}$.
9:              $X_{i_m^* mt} \leftarrow R$.
10:             AP $m$ transmits to client $i_m^*$ at time $t$.
11:             if $\sum_{p,s\le t} X_{i_m^* ps} K_{i_m^* ps} > C_{i_m^*}$ then
12:                 $X_{i_m^* mt} \leftarrow \frac{C_{i_m^*} - \sum_{p,s<t} X_{i_m^* ps} K_{i_m^* ps}}{K_{i_m^* mt}}$.
```

C. Competitive Ratio

Our previous work [4] has established the new competitive ratios of Algorithm 3 and other commonly used policies.

Theorem 5: Algorithm 3 is $(R, \frac{e^{1/R}}{R[e^{1/R}-1]})$-competitive. It is approximately $(R, 1+\frac{1}{2R})$-competitive.

Theorem 6: For any given $\epsilon > 0$, there exists a R_0, such that, when $R > R_0$, no policy has better competitive ratio than $(R, 1+\frac{1}{2R}-\frac{\epsilon}{R})$.

Since $\frac{e^{1/R}}{R[e^{1/R}-1]} \approx 1+\frac{1}{2R}$. Theorems 6 and 5 show that our proposed policy is close to the theoretical bound of the new competitive ratio.

We evaluate three commonly used policies, including round robin (RR) policy, max-weight (MW) policy, and proportional fair (PF) policy as comparison. With round robin policy (RR), each AP evenly distributes its capacity among all connected clients. With max-weight (MW) policy, of all connected clients, each AP selects the one with maximum value of the product of K_{imt} and the client's remaining data. With proportional fair (PF) policy, of all connected clients, each AP selects the one with the maximum value of $\frac{K_{imt}}{Throughput\ of\ client\ i}$. We have the following theorem.

Theorem 7: RR, MW, and PF scheduling policies cannot have better competitive ratio than $(R, 1+\frac{1}{R})$.

Suppose one needs to guarantee to offload at least $1-\frac{1}{\theta}$ as much data as the optimal offline policy, PD algorithm needs to increase capacity by approximately $\frac{\theta-1}{2}$ times, while RR, MW, and PF policies need at least $\theta-1$ capacity to achieve the same requirement. In other words, PD

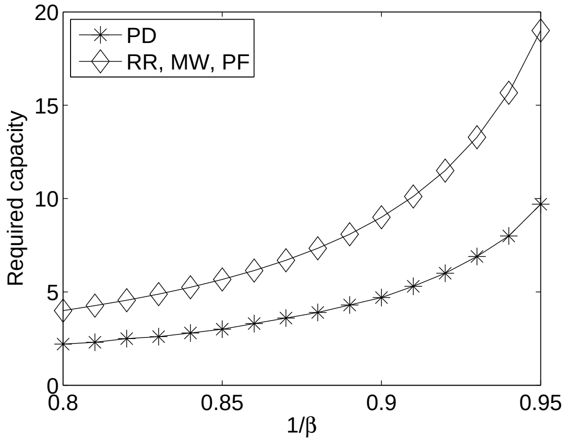

Fig. 2. Capacity requirements of mobile offloading problem.

needs about half as much capacity to provide the same guarantee as the other three policies. Fig. 2 illustrates the differences between these policies..

V. Conclusion

In this paper, we use a new definition of competitive ratio to evaluate online policies. Instead of simply considering the performance ratio of an online policy and the offline optimal policy, we consider how increasing resource can affect the performance ratio. We use two special cases to explain this question. In job allocation problem, we show that the commonly used JSQ and JMQ polices need an order higher capacity to achieve the same allocation ratio requirement than our policy. In mobile offloading problem, we show that the commonly used RR, MW, and PF policies need at least double the capacity of our policy to offload the same amount of data.

References

[1] S. Moharir, S. Sanghavi, and S. Shakkottai, "Online load balancing under graph constraints," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, pp. 363–364, ACM, 2013.

[2] Z. Mao, C. E. Koksal, and N. B. Shroff, "Optimal online scheduling with arbitrary hard deadlines in multihop communication networks," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 177–189, Feb 2016.

[3] H. Deng and I.-H. Hou, "Online job allocation with hard allocation ratio requirement," in *Computer Communications (INFOCOM), 2016 IEEE Conference on*, p. to appear, April 2016.

[4] H. Deng and I.-H. Hou, "Online scheduling for delayed mobile offloading," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pp. 1867–1875, April 2015.

[5] N. Buchbinder, K. Jain, and J. S. Naor, "Online primal-dual algorithms for maximizing ad-auctions revenue," in *Proceedings of the 15th Annual European Conference on Algorithms*, ESA'07, (Berlin, Heidelberg), pp. 253–264, Springer-Verlag, 2007.