Managing Cancellations and No-shows of Reservations with Overbooking to Increase Resource Revenue

Anthony Sulistio[1], Kyong Hoon Kim[2], and Rajkumar Buyya[1]
[1]Dept. of Computer Science and Software Eng.
The University of Melbourne, Australia
{anthony, raj}@csse.unimelb.edu.au
[2]Dept. of Information Science
Gyeongsang National University, Korea
khkim@gnu.ac.kr

Abstract

Advance reservation allows users to request available nodes in the future, whereas economy provides an incentive for resource owners to be part of the Grid, and encourages users to utilize resources optimally and effectively. In this paper, we use overbooking models from Revenue Management to manage cancellations and no-shows of reservations in a Grid system. Without overbooking, the resource owners are faced with a prospect of loss of income and lower system utilization. Thus, the models aim to find an ideal limit that exceeds the maximum capacity, without incurring greater compensation cost. Moreover, we introduce several novel strategies for selecting which bookings to deny, based on compensation cost and user class level, namely Lottery, Denied Cost First (DCF), and Lower Class DCF. The result shows that by overbooking reservations, a resource gains an extra 6–9% in the total net revenue.

1 Introduction

Grid [4] technology enables the aggregation of distributed resources for solving large-scale and computationally-intensive applications. However, managing various resources and applications in highly dynamic Grid environments is a complex and challenging process. Resource management is not only about scheduling large and compute-intensive applications, but also the manner in which resources are allocated, assigned, and accessed. In most scheduling systems, submitted jobs are initially placed into a queue if there are no available resources. Therefore, there is no guarantee as to when these jobs will be executed. This causes problems for time-critical or workflow applications, where jobs may have interdependencies.

To address these issues and to ensure the specified resources are available for application's consumption when required, several researchers have proposed the need for advance reservation (AR) [5, 11, 18]. Common resources that can be reserved or requested are compute nodes (CNs) and network bandwidth.

Buyya et al. [1] introduced a Grid economy concept that provides a mechanism for regulating demand and supply of resources, and calculates pricing policies based on these criteria. With this concept, it offers an incentive for resource owners to be part of the Grid, and encourages users to utilize resources optimally and effectively.

In our previous work [19], we studied the use of Revenue Management (RM) to determine pricing of reservations, and to increase total revenue of a resource. The main objective of RM is to maximize profits by providing the right price for every product to different customers, and to periodically update the prices in response to market demands. Therefore, the resource provider can apply RM techniques to *shift demands*, and to ensure that resources are allocated to applications that are highly valued by the users. The result shows an increase in total revenue for resources that utilize RM over those that price their resources statically [19].

In reality, users may cancel their reservations before starting time or by not submitting at all (*no-show*), due to reasons such as resource or network failures on the other end. Thus, during a period of high demands for example, the resource provider has no choice but to reject bookings from potential users, who are committing to use the resource and willing to pay for a higher price. As a result, the resource provider is faced with a prospect of loss of income and lower system utilization.

Overbooking offers a solution for the above problem, by allowing the resource provider to accept more reservations than the capacity. Hence, it can be effectively used to minimize the loss of revenue [10, 14]. However, the challenging issues in using overbooking are determining the appropriate number of excess reservations, minimizing total compensation cost, addressing legal and regulatory issues, and dealing with market acceptance, especially the *ill-will* or negative effects from users who have been denied access [21]. In this paper, we only consider the first two issues.

The contribution of this paper is as follows. We extend our previous work in RM [19] by using overbooking models to manage cancellations and no-shows of reservations in a Grid system [18]. The overbooking models aim to find an ideal overbooking limit that exceeds the maximum capacity, without incurring greater compensation cost. Moreover, we introduce several novel strategies for selecting which bookings to deny, based on compensation cost and user class level, namely Lottery, Denied Cost First (DCF), and Lower Class DCF. Finally, we investigate the impact of these models and strategies on the net resource revenue through performance evaluation.

The rest of the paper is organized as follows. Section 2 mentions some related works on the overbooking issues in airlines, hotels, networks and Grids. Section 3 gives an overview of RM techniques, whereas Section 4 explains the overbooking models. Section 5 integrates overbooking into a capacity allocation heuristic to estimate a suitable quota for different users. Section 6 calculates cost and penalties of each reservation, and presents several strategies for choosing which reservations to be denied. Section 7 conducts a performance evaluation. Finally, Section 8 concludes the paper and suggests some further work to be done.

2 Related Work

In a study done by Smith et al. [15] on American Airlines, 50% of the bookings were resulted in cancellations or no-shows. Moreover, the report found that 15% of the flight seats would be unused, if bookings were only limited to the capacity of a plane. Therefore, overbooking models were introduced to address the problem in unanticipated cancellations and no-shows, by several researchers in the airlines industry [2, 22]. The overbooking policies were also studied and applied to several industries, such as hotel [8], and car rentals [7]. Similarly, in this paper, we adopt these overbooking policies in the context of scheduling jobs by means of reservations in a Grid resource. Moreover, we propose several strategies for determining which excess reservations to deny, based on compensation cost and user class level.

In networks, overbooking is used to optimize throughput [12] and to address the issue of burst contentions in optical burst switched networks from a new domain [24]. Similarly, in Grids, Urgaonkar et al. [23] suggested overbooking as a way to increase resource utilization in shared hosting platforms, by specifying an overbooking tolerance on each component of an application running on one compute node. However, none of these works aim at maximizing revenue by charging the users with different prices, and calculating an ideal overbooking limit.

Table 1. An example of market segmentation in Grids for reserving jobs.

Class	User Category	Restrictions
1	Premium	none
2	Business	same VO, allow cancellation
3	Budget	same VO, non-refundable, only for a limited number of nodes

3 An Overview of Revenue Management

Revenue Management (RM) has been widely adopted in various industries, such as airlines, hotels, and car rentals [10], since they have a limited and perishable capacity. RM can also be applied to Grid computing, as computing powers or nodes are considered to be perishable. In this section, we briefly give an overview of RM and its adaptation in Grids before going into more depth about the proposed overbooking models. A more detail explanation can be found in [19].

3.1 Market Segmentation and Price Differentiation

An initial step of RM is to identify various customer segments for a single product, and to apply different prices to each of them. The airlines industry is a well-known example that segments customers into different classes, such as first, business, and economy. With this approach, the airline can offer users from different classes with various fares and restrictions, based on their flexibility, price sensitivity, and time of bookings prior to departure times [10]. Hence, the airlines can set prices for seats in a same flight to be: $p_1 > p_2 > p_3$, where p_1 denotes the price paid by class 1 (*Premium*) users, and so on.

In Grids, resources can be part of one or more *virtual organizations* (VOs). The concept of a VO allows users and institutions to gain access to their accumulated pool of resources to run applications from a specific field [6], such as high-energy physics or aerospace design. Table 1 shows an example of market segmentation in Grids. The classifications are based on VOs and time of bookings.

3.2 Booking Limits and Protection Levels

A *booking limit*, b, denotes a quota that is allocated to each user class. As listed in Table 1, each user class has a set of restrictions, which leads to a different price. In this paper, we assume that class 3 (*Budget*) users reserve *before* class 2 (*Business*) users *before* class 1 (*Premium*) users, as shown in Figure 1. This assumption is used so that once a

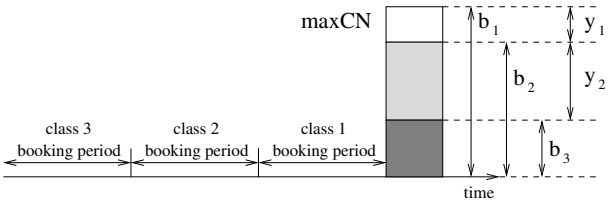

Figure 1. Protection levels (y_1, y_2) and nested booking limits (b_1, b_2, b_3) for each time slot.

booking limit for class 3, b_3, is reached, then users will be offered a fare class of the next one, i.e. class 2, and so on.

To prevent high-fare bookings are being rejected in favor of budget ones, *protection levels* y_1 and y_2 are required to make some compute nodes (CNs) available for class 1 and 2 users respectively, that may book later. Moreover, a nested approach is used to determine b_i, where b_i denotes the booking limit for class i, as shown in Figure 1. With this approach, the booking limits are always nonincreasing, i.e. $b_1 \geq b_2 \geq b_3$. In addition, every class has access to all of the bookings available to lower classes. Hence, b_1 equals to $maxCN$, where $maxCN$ denotes the maximum capacity to be reserved.

In this paper, we use an array-based structure for administering reservations efficiently [17]. It is a *time-slotted* data structure, where each slot contains rv, the number of already reserved CNs, and $bookingList$, a list for storing reservations that start at this time. Thus, it partitions each reservation duration time dur into slots based on a fixed time interval δ. If dur spans multiple slots, rv on each of them is updated accordingly. Moreover, each time slot contains b_1, b_2, and b_3 denoting the booking limit for class 1, 2 and 3 respectively.

4 Overbooking Policies

Figure 2 illustrates an example on how overbooking can protect a resource provider against unanticipated cancellations and no-shows. We define a *cancellation* as a reservation that is terminated by a user *before* the service or starting time t_s, as shown in Figure 2. Moreover, we describe a *no-show* as a reservation that *fails* to arrive and run on the resource on t_s (without a cancellation notice).

By setting the overbooking limit ob to be greater than $maxCN$, the resource provider can still accept more reservations (after t_1) until total number of reservations tot_{AR} equals to ob (on t_2 and t_3), as shown in Figure 2. In contrast, a resource without overbooking has to deny potential reservations starting from t_1, since the capacity is full.

The overbooking limit itself needs to be updated and evaluated frequently as t_s approaches. Thus, as tot_{AR} increases, ob decreases, as shown in Figure 2. Then, the resource provider takes an advantage of the cancellation and no-show periods to reduce tot_{AR}. In the best-case scenario, the resource may not need to deny any excess reservations due to a large number of no-shows. In the end, on t_s, a resource with overbooking yields more reservations that show up than without overbooking.

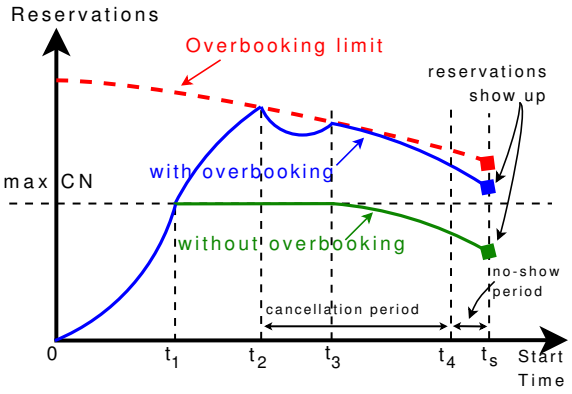

Figure 2. An example of total number of reservations with and without overbooking.

In this section, we adopt several *static* overbooking policies, introduced in the RM literature [14, 21], into our work. These static policies only calculate the ideal overbooking limit periodically prior to t_s, when the state and probabilities change over time. Thus, we assume the following things:

- cancellations and no-shows are independent of the number of total bookings.
- the probability of a cancellation is Markovian, i.e. it only depends on the current time.
- no-shows are treated as cancellations on t_s. Hence, we can define $q(t)$ as a show rate or a probability that reservations show up from the time remaining until t_s.

4.1 A Probability-based Policy

This is a simple overbooking policy, where ob is determined statistically based on the probability of shows. Equation 1 determines the overbooking limit at time t. For example, if $maxCN = 100$ and $q(t) = 0.80$, then the amount of overbooking capacity is 125. Therefore, the lower the probability of shows, the higher the overbooking limit becomes.

$$ob = \frac{maxCN}{q(t)} \tag{1}$$

4.2 A Risk-based Policy

A risk-based policy aims to balance the expected cost of denied service with the revenue by accepting more bookings. The cost of denied service refers to the compensation money given to users who got rejected or *bumped* at the service time. This cost of denied service is denoted as $cost_{ds}$ and is usually higher than the resource price p. Thus, a risk analysis is required in order to calculate a threshold at which the overbooking is allowed.

For computing the threshold, we need to find out the probability distribution of users demand and number of shows. Let $A(x)$ denotes the probability that the demand of users is less than or equal to x, where x denotes the number of bookings. Moreover, we define $F_x(y)$ as the probability that the number of bookings that will show up at the time of service is less than or equal to y.

Let us assume that the current booking limit and capacity are b and C, respectively. Then, we derive the expected revenue change by increasing the booking limit from b to $b+1$. By doing this, we are faced with three possible cases:

1. $demand < b + 1$, which means there are no changes in the forecasted revenue.
2. $demand \geq b+1$ and the number of shows $\leq C$. Since the resource provider can serve users at t_s, the profit of p is obtained by accepting an additional reservation.
3. $demand \geq b+1$ and the number of shows $> C$, which means the resource provider has to deny one user with a compensation cost. As a result, there is a loss of $p - cost_{ds}$, where $cost_{ds} > p$.

Thus, we can derive the expected revenue change by increasing the booking limit from b to $b + 1$ as follows.

$$\begin{aligned} &E[R|b+1] - E[R|b] \\ &= (1 - A(b))\{pF_{b+1}(C) + (p - cost_{ds})(1 - F_{b+1}(C))\} \\ &= (1 - A(b))\{p - cost_{ds}(1 - F_{b+1}(C))\} \end{aligned} \tag{2}$$

Algorithm 1: Overbooking Limit using a Risk Policy

```
1 ob ← C;
2 IR ← (1 − A(ob)){p − cost_ds(1 − F_{ob+1}(C)) ;
3 while IR > 0 do
4 |   ob ← ob + 1 ;
5 |   IR ← (1 − A(ob)){p − cost_ds(1 − F_{ob+1}(C)) ;
6 end
7 return ob ;
```

As long as the expected revenue change is greater than zero, the overbooking limit can be increased, as shown in Algorithm 1. The booking limit ob starts from the maximum capacity C (line 1), and is incremented until the expected revenue change becomes zero or negative (line 3).

In multiple fare classes, the increased revenue from having an additional booking can not be easily calculated. Therefore, a suitable approach is to determine a weighted average price $\hat{p}$, based on the mean demands μ in each user class [14]. More specifically,

$$\hat{p} = \sum_{i=0}^{n} \mu_i p_i \tag{3}$$

We derive the show distribution $F_x(y)$ at time t, under the assumption that each customer's showing probabilities are independent. The show probability of each customer at time t is denoted as q. Then, the number of shows, as investigated by Thompson [22], follows a binomial distribution:

$$F_x(y) = \sum_{k=0}^{y} \binom{x}{k} q^k (1-q)^{x-k} \tag{4}$$

4.3 A Service-Level Policy

Although the risk-based policy enhances the expected revenue of the resource, users who got denied at the service time, tend to submit their jobs to other resources in the future. Thus, by using this policy, a resource may lose some of these users in the long term. Moreover, this policy may increases the negative impact of overbooking towards users' satisfaction.

A service-level policy addresses the above issues by defining a specified level or fraction of denied users. For example, American Airlines and United Airlines have an involuntary denied boarding (DB) ratio of 0.84 and 0.51 per 10,000 passengers respectively, due to oversales in 2006 [13]. The data were taken from flights within and originated in the United States. With the service-level policy, the airlines may set a threshold of involuntary DB ratio to be 0.50 as an example. Accordingly, the airlines could determine the overbooking limit based on this threshold.

Suppose that the number of shows for a given x bookings is denoted as $B(x)$. Then, the service level of x bookings, $s(x)$ is defined by Equation 5, where $(B(x) - C)^+ = \max(0, B(x) - C)$. The equation implies the fraction of the expected denied service over the expected number of shows.

$$s(x) = E[(B(x) - C)^+] \;/\; E[B(x)] \tag{5}$$

If we use a binomial distribution for show demands, then the service level of x bookings can be defined as follows.

$$s(x) = \frac{1}{xq} \times \sum_{k=C+1}^{x} (k - C) \binom{x}{k} q^k (1-q)^{x-k} \tag{6}$$

For a given service level $ds\_threshold$, the overbooking limit for this policy is calculated in Algorithm 2. Initially, the overbooking limit ob starts from the maximum capacity C (line 1), and is incremented until $s(x)$ equals to or less than $ds\_threshold$ (line 3).

Algorithm 2: Overbooking Limit using Service Policy

```
1 ob ← C;
2 s(x) ← E[(B(x) − C)+] / E[B(x)] ;
3 while s(x) ≤ ds_threshold do
4 |   ob ← ob + 1 ;
5 |   s(x) ← E[(B(x) − C)+] / E[B(x)] ;
6 end
7 return ob ;
```

4.4 Overbooking Limit Calculation

In this subsection, we give a brief example on the calculation of the overbooking limit for the above policies. We consider the price of a single time slot in a resource is fixed, with $p = 100$ and $C = 50$ for simplicity. However, the denied cost $cost_{ds}$ and the show-rate q are varied from 125 to 175 and from 0.60 to 0.95, respectively.

$$\begin{aligned} ENR &= pE[B(ob)] - cost_{ds} * E[(B(ob) - C)^+] \qquad (7) \\ &= p * ob * q - cost_{ds} * \sum_{k=C+1}^{ob} \binom{ob}{k} q^k (1-q)^{ob-k} \end{aligned}$$

Table 2 shows ob, expected net revenue (ENR in G$), and service level (SL) for each q, according to the probability-based policy. In contrast, the risk-based policy adaptively selects ob with a consideration of both the show rate and the denied cost, as shown in Table 3. The ENR and SL in both tables are calculated using Equation 7 and 6 respectively.

Table 4 shows the ob and ENR for a given service level, i.e. from 0.01 (1%) to 0.0001 (0.01%). Note that the ENR is also calculated by using Equation 7. From Table 4, it can be concluded that as SL decreases, ob and ENR become smaller for the same q. Moreover, for the same $cost_{ds}$ of 150 in the service-level policy, the SL of the risk-based policy varies from 0.0162 (q = 0.95) to 0.0600 (q = 0.60), as shown in Table 3. This means that about 162 – 600 out of 10,000 reservations are denied by a resource provider when using the risk-based policy. Hence, from this example, the service-level policy produces a lower denied-service rate compared to the probability- and risk-based policy.

Table 2. ob **using a probability-based policy**

q	ob	Expected Net Revenue (G$)			Service Level
		$cost_{ds} = 125$	$cost_{ds} = 150$	$cost_{ds} = 175$	
0.60	83	4,770.5	4,728.6	4,686.7	0.0337
0.65	76	4,769.1	4,734.9	4,700.8	0.0277
0.70	71	4,796.7	4,762.1	4,727.4	0.0279
0.75	66	4,805.6	4,776.7	4,747.8	0.0233
0.80	62	4,828.4	4,802.1	4,775.8	0.0212
0.85	58	4,836.5	4,817.8	4,799.1	0.0152
0.90	55	4,870.7	4,854.9	4,839.0	0.0128
0.95	52	4,898.9	4,890.7	4,882.4	0.0067

Table 3. ob **using a risk-based policy**

q	$cost_{ds} = 125$			$cost_{ds} = 150$			$cost_{ds} = 175$		
	ob	ENR	SL	ob	ENR	SL	ob	ENR	SL
0.60	90	4,836.9	0.0834	87	4,750.4	0.0600	85	4,689.9	0.0459
0.65	83	4,846.7	0.0813	80	4,766.8	0.0555	78	4,711.1	0.0405
0.70	76	4,858.8	0.0693	74	4,784.2	0.0509	73	4,729.6	0.0425
0.75	71	4,870.4	0.0683	69	4,802.4	0.0480	68	4,753.2	0.0389
0.80	66	4,884.2	0.0600	64	4,824.3	0.0385	63	4,782.2	0.0292
0.85	62	4,898.4	0.0564	60	4,847.9	0.0330	59	4,811.5	0.0232
0.90	58	4,916.7	0.0465	57	4,873.1	0.0334	56	4,846.4	0.0219
0.95	54	4,941.4	0.0294	53	4,912.3	0.0162	53	4,891.9	0.0162

Table 4. ob **using a service-level policy (with** $cost_{ds} = 150$**)**

q	SL = 0.01		SL = 0.001		SL = 0.0001	
	ob	ENR	ob	ENR	ob	ENR
0.60	77	4,555.3	70	4,194.9	66	3,959.4
0.65	71	4,563.3	66	4,283.7	62	4,029.5
0.70	67	4,628.8	62	4,334.6	59	4,129.4
0.75	63	4,667.1	59	4,418.9	56	4,199.6
0.80	60	4,731.7	56	4,475.3	54	4,319.5
0.85	57	4,779.0	54	4,584.0	52	4,419.6
0.90	54	4,813.7	52	4,675.1	50	4,500.0
0.95	52	4,890.7	50	4,750.0	50	4,750.0

5 Capacity Allocation with Overbooking

The capacity allocation problem in RM is to decide the booking limit for each class user, in order to maximize the overall expected total revenue. If too many CNs are allocated to lower-class users during peak periods, we may lose a chance to earn more revenue from accepting future bookings from higher-class users. On the contrary, an insufficient quota for the lower-class users in off-peak periods, may lead to a lower resource utilization and revenue. Thus, finding an appropriate capacity allocation to each user class at different time period is an important factor in RM.

We use an expected marginal seat revenue (EMSR) heuristic to determine the booking limits of three user classes, as shown in Algorithm 3. The overbooking limit needs to be calculated according to one of the models we previously discussed (line 1). Then, a virtual capacity C^+ can be found (line 2), where $C^+ \geq maxCN$. In order to

determine b_3 (line 5), y_1 and y_2 need to be calculated first (line 3–4). Then, b_2 can found by using the *BookingLimit* algorithm with $C^+ - b_3$ as the current capacity (line 6).

Algorithm 3: Capacity Allocation with Overbooking

1 $ob \leftarrow$ *OverbookingLimit(q, maxCN)* ;
2 $C^+ \leftarrow \max(maxCN,\ ob)$;
3 $y_1 \leftarrow C^+ -$ *BookingLimit*$(C^+,\ p_1,\ p_3,\ F_1)$;
4 $y_2 \leftarrow C^+ -$ *BookingLimit*$(C^+,\ p_2,\ p_3,\ F_2)$;
5 $b_3 \leftarrow \max(0,\ C^+ - y_1 - y_2)$;
6 $b_2 \leftarrow b_3 +$ *BookingLimit*$(C^+ - b_3,\ p_1,\ p_2,\ F_1)$;
7 $b_1 \leftarrow C^+$;

Let p_i denotes the price of class i, and $F_i(x)$ denotes the probability that the demand of class i user is less than or equal to x. The *BookingLimit* algorithm finds the booking limit of a lower class user, based on the user's expected revenue and the demand of a higher class user. In Algorithm 3 line 3 and 6, we consider a class 1 user to be the higher class one. A detailed description of the the *BookingLimit* algorithm and the expected revenue of a class user have been omitted in this paper. However, they can be found on [19].

6 Reservation, Penalty and Denied Cost

Apart from overbooking and capacity allocation, the next important point in RM is to determine the pricing of each reservation. Moreover, if a cancellation or no-show occurs, a penalty fee needs to be introduced to discourage users from misusing it, and to cover some operational cost associated with managing reservations. Finally, the denied cost due to overbooking needs to be addressed.

6.1 Initial Cost of A Reservation

As mentioned previously, in this model, we differentiate jobs based on whether they are using reservations or not. For non-AR jobs, we calculate the running cost as

$$cost = dur * numCN * bcost \tag{8}$$

where $numCN$ denotes the number of CNs used, and $bcost$ is the base cost of running a job at one time unit. Intuitively, the cost for jobs that use AR will incur higher due to the privilege of having guaranteed resources at a future time. Hence, the running cost for AR jobs is charged based on the number of reserved slots in the data structure:

$$cost_{AR} = numSlot * numCN * bcost_{AR} \tag{9}$$

$$bcost_{AR} = \tau * bcost * \delta \tag{10}$$

where $numSlot$ is the total number of reserved slots, $bcost_{AR}$ is the cost of running the AR job at one time slot, and τ is a constant factor ($\tau \geq 1$) to differentiate the pricing. Table 5 shows an example of setting different τ of equation (10), according to demands or daily arrival rate from several parallel and Grid workload traces [3, 9]. Note that τ_1, τ_2, and τ_3 denote τ for user class 1, 2 and 3 respectively.

Table 5. An example of variable pricing with different τ_1, τ_2, and τ_3 during the week.

Pricing Name	Day Period	Time Period	τ_1	τ_2	τ_3
Super Saver	Weekdays	12 am – 06 am	1.88	1.56	1.25
Peak	Weekdays	06 am – 06 pm	3.38	2.81	2.25
Off-Peak	Weekdays	06 pm – 12 am	2.63	2.19	1.75
Super Saver	Weekends	06 pm – 06 am	1.88	1.56	1.25
Off-Peak	Weekends	06 am – 06 pm	2.63	2.19	1.75

6.2 Cancellation & No-Show Penalty Cost

In this paper, we use a simple penalty cost function, where the resource provider charges a user with a penalty rate α_p times the price for each canceled or no-show reservation, where $0 \leq \alpha_p \leq 1$. $\alpha_p = 0$ means the reservation is fully refundable, and $\alpha_p = 1$ means it is not refundable. In multiple fare classes, we have $\alpha_{p1} < \alpha_{p2} < \alpha_{p3}$.

6.3 Denied or Compensation Cost

In this paper, we use Equation 9 to determine $cost_{ds}$, i.e. the denied service or compensation cost for each reservation. The value of τ_{ds} depends on the agreement or policy set by the resource provider to a particular user class, with $\tau_{ds} > \tau$. Moreover, we present several strategies for addressing which excess reservations to deny at the starting time t_s, based on $cost_{ds}$ and user class level, namely Lottery, Denied Cost First (DCF), and Lower Class DCF, as shown in Algorithm 4, 5, and 6 respectively.

Algorithm 4: Lottery drawing

Input: t_s **and** C

1 $bookingList \leftarrow get\_booking\_list(t_s)$;
2 $overbookedCN \leftarrow get\_total\_CN(bookingList) - C$;
3 $denyCN \leftarrow 0$; `// total nodes to be denied`
4 **while** $denyCN < overbookedCN$ **do**
5 $data \leftarrow get\_booking(bookingList,\ LOTTERY)$;
6 $calculate\_denied\_cost(data)$;
7 $remove(data,\ bookingList)$;
8 $denyCN \leftarrow denyCN + get\_total\_CN(data)$;
9 **end**

The simplest way to deny existing reservations is by conducting a lottery drawing, as depicted in Algorithm 4. Initially, a list of bookings, $bookingList$, that start at time

t_s is withdrawn from the data structure (line 1). Since a booking may require more than one node, we also need to find out the number of overbooked CNs, $overbookedCN$, based on the current capacity C, and the total CNs required from $bookingList$ (line 2). Then, the algorithm performs a lottery drawing on $bookingList$ (line 5), with the unlucky booking is compensated and removed from the list and the data structure altogether (line 6–7). Next, the total CNs to be denied, $denyCN$, is incremented (line 8). Finally, this algorithm keeps ejecting more bookings from the list as long as $denyCN < overbookedPE$ (line 4–9).

Algorithm 5: Denied Cost First (DCF)

Input: t_s **and** C

1 $bookingList \leftarrow get\_booking\_list(t_s)$;
2 $overbookedCN \leftarrow get\_total\_CN(bookingList) - C$;
3 $denyCN \leftarrow 0$; `// total nodes to be denied`
4 $sort(bookingList,\ GLOBAL\_DENIED\_COST)$;
5 **while** $denyCN < overbookedCN$ **do**
6 $\quad data \leftarrow get\_booking(bookingList,\ HEAD)$;
7 $\quad calculate\_denied\_cost(data)$;
8 $\quad remove(data,\ bookingList)$;
9 $\quad denyCN \leftarrow denyCN + get\_total\_CN(data)$;
10 **end**

In contrast, to minimize the total compensation cost on t_s, the Denied Cost First (DCF) strategy chooses which bookings to be denied based on $cost_{ds}$, as shown in Algorithm 5. Thus, DCF sorts $bookingList$ based on the lowest $cost_{ds}$ globally, regardless of any class types (line 3). Afterwards, DCF removes this booking from the head of $bookingList$ (line 6), since the list is sorted from lowest to highest $cost_{ds}$. The rest of the operations are similar to the Lottery strategy.

Lower Class Denied Cost First (LC-DCF), as shown in Algorithm 6, has a similar strategy as DCF. However, LC-DCF aims at protecting higher-class bookings from being denied in the first place. Hence, LC-DCF sorts $bookingList$ based on $cost_{ds}$ for each class type (line 3). Similar to DCF, LC-DCF removes a booking from the head of $bookingList$ (line 6), but this booking is from a lower-class user that has the lowest $cost_{ds}$. If there are no more bookings from a lower class, then LC-DCF continues removing bookings from a higher class. The rest of the operations are similar to the Lottery strategy.

7 Performance Evaluation

We carried out performance evaluation by using the GridSim toolkit [20] because we need to conduct *repeatable* and *controlled* experiments that would otherwise be difficult to perform in real Grid testbeds. The details of simulation parameters are discussed next.

Algorithm 6: Lower Class Denied Cost First (LC-DCF)

Input: t_s **and** C

1 $bookingList \leftarrow get\_booking\_list(t_s)$;
2 $overbookedCN \leftarrow get\_total\_CN(bookingList) - C$;
3 $denyCN \leftarrow 0$; `// total nodes to be denied`
4 $sort(bookingList,\ CLASS\_DENIED\_COST)$;
5 **while** $denyCN < overbookedCN$ **do**
6 $\quad data \leftarrow get\_booking(bookingList,\ HEAD)$;
7 $\quad calculate\_denied\_cost(data)$;
8 $\quad remove(data,\ bookingList)$;
9 $\quad denyCN \leftarrow denyCN + get\_total\_CN(data)$;
10 **end**

Table 6. mean CPU rating for Grid and VO level, and their jobs' inter-arrival rates (λ).

Level	μ Rating	μ runtime	λ_{peak}	λ_{off}	λ_{saver}
Grid	56,000	2 hours	0.13812	0.02290	0.01979
VO 1	56,000	5 hours	0.05087	0.02092	0.01913
VO 2	20,000	5 hours	0.05954	0.00537	0.00295
VO 3	60,000	5 hours	0.15901	0.00097	0.00046
VO 4	68,000	5 hours	0.07098	0.00672	0.00257

7.1 Simulation Setups

Table 6 and 7 summarizes all relevant information of the traces and resources used for our experiments. In GridSim, a CPU rating of one node is modeled in the form of MIPS (Million Instructions Per Second) as devised by Standard Performance Evaluation Corporation (SPEC) [16]. To determine $bcost_{AR}$ on each resource, we use τ from Table 5 for different time periods. A detailed explanation of these settings can be found in [19].

We model incoming job traffic at three levels: Grid (with all 10 resources), VO and resource, by using a Poisson model with different lambdas for peak (λ_{peak}), off-peak (λ_{off}) and super saver (λ_{saver}) period, as depicted in Table 6 and 7. Thus, we can set the peak period to be arriving more frequently than the off-peak period, and so on. For handling no-show of reservations, we use binomial distribution with the probability of no-shows (q_{ns}) sets to 0.05, 0.10 and 0.15 for peak, off-peak, and super saver periods respectively. We use an exponential distribution to model the mean runtime (μ) of jobs. Finally, we set the number of reserved CNs to 1 for all jobs.

We identify the Grid-, resource-, and VO-level trace to be *Premium*, *Business*, and *Budget* users respectively. Moreover, all traces have the same booking period of 2 hours, and they use exponential distribution to calculate the number of cancellations. Table 8 lists the job's canceled rate (λ_c), the penalty rate (α_p), and τ_{ds} for the denied service cost for each user class.

Table 7. Resource specifications and their jobs' inter-arrival rates (λ).

Resource Name (Location)	ID	# Nodes	CPU Rating	VO	$bcost$ (G$)	μ runtime	λ_{peak}	λ_{off}	λ_{saver}
RAL (UK)	$R1$	41	49,000	1	0.49	3 hours	0.01670	0.00835	0.004175
Imperial College (UK)	$R2$	52	62,000	1	0.62	3 hours	0.01670	0.00835	0.004175
NorduGrid (Norway)	$R3$	17	20,000	2	0.20	3 hours	0.00835	0.004175	0.0020875
NIKHEF (Netherlands)	$R4$	18	21,000	2	0.21	3 hours	0.00835	0.004175	0.0020875
Lyon (France)	$R5$	12	14,000	3	0.14	3 hours	0.00835	0.004175	0.0020875
CERN (Switzerland)	$R6$	59	70,000	3	0.70	3 hours	0.03340	0.00167	0.000835
Milano (Italy)	$R7$	5	7,000	4	0.07	3 hours	0.00418	0.0020875	0.00104375
Torino (Italy)	$R8$	2	3,000	4	0.03	3 hours	0.00167	0.000835	0.0004175
Rome (Italy)	$R9$	5	6,000	4	0.06	3 hours	0.00418	0.00209	0.001045
Bologna (Italy)	$R10$	67	80,000	4	0.80	3 hours	0.03340	0.0167	0.00835

Table 8. Simulated users' characteristics.

User Class	λ_c	α_p	τ_{ds}
Premium	0.25	0%	5 τ_1
Business	0.45	10%	4 τ_2
Budget	0.85	25%	3 τ_3

Table 9. The impact of unanticipated cancellations and no-shows (CNS) on net revenue.

Resource Name	No CNS (x 1000)	Allow CNS (x 1000)	% loss
RAL	G$ 31,523	G$ 2,321.44	-92.64
Imperial	G$ 61,645	G$ 7,038.12	-88.58
Nordu	G$ 4,638	G$ 413.90	-91.08
NIKHEF	G$ 5,171	G$ 421.90	-91.84
Lyon	G$ 742	G$ 94.08	-87.32
CERN	G$ 103,529	G$ 10,400.91	-89.95
Milano	G$ 171	G$ 7.46	-95.63
Torino	G$ 13	G$ 0.58	-95.43
Rome	G$ 119	G$ 5.56	-95.31
Bologna	G$ 147,279	G$ 7,606.47	-94.84

The main objective of our performance evaluation is to examine the impact of the overbooking policies (Probability (Pr), Risk, and Service-Level (SL)), and the denied-booking strategies (Lottery, DCF, and LC-DCF) on the net revenue of a resource, where cancellations and no-shows are allowed. For the Service-Level (SL) policy, we set the $ds\_threshold$ to be 0.01 or 1%. Overall, we simulate 15 traces in this evaluation for a period of 14 days.

7.2 Experiment Results

Table 9 shows the negative effect of unanticipated cancellations and no-shows (CNS) on the net revenue of each resource. By allowing CNS and without any overbooking policies, all resources experienced a significant drop in revenue, i.e. by more than 87%. However, if we set RAL and Bologna to use overbooking policies instead, they both reported around 6–9% increase in net profits from their previous evaluation (without overbooking), as shown in Figure 3 and 4 respectively. Thus, this finding provides a financial incentive for other resources to overbook. Note that RAL has zero denied bookings for all the overbooking policies. Hence, in this section, we mainly discuss the impact of overbooking policies and denied-booking strategies in Bologna.

When looking at the performance of each overbooking policy in Figure 3 and 4, all policies produce about the same amount of net revenue. However, the main difference between them is the overbooking limit, ob, at each time slot in the data structure, as shown in Figure 5 and 6. Note that we omit figures using Lottery and LC-DCF in Bologna, since they are similar.

For RAL in Figure 5, the maximum ob percentage gain from $maxCN$ is 7%, 12% and 27% for SL, Risk and Pr policies respectively. For Bologna in Figure 6, it is 8%, 12% and 24% for SL, Risk and Pr policies respectively. Thus, in both cases, the SL policy is the most conservative of all, since it estimates the lowest ob. This is consistent with the calculation that we performed in Section 4.4. However, with $cost_{ds}$ can be up to five times more expensive than $cost_{AR}$, the Risk policy sets a lower limit than the Pr policy in both Figure 5 and 6.

In this evaluation, we found that a lower ob leads to a smaller the total number of denied bookings and compensation cost, as shown in Figure 7 and 8 for Bologna respectively. In both figures, on average, the Risk and SL policies are 49% and 74% lower than the Pr policy respectively.

Apart from estimating ob, another important issue is selecting which excess bookings to deny. In terms of total net revenue, the denied-booking strategies (Lottery, DCF, and LC-DCF) in Bologna produced a similar income, i.e. within 0.1–2% of each other, as shown in Figure 4. On average, DCF gives the highest total amount of income, followed by LC-DCF and then Lottery.

Surprisingly, the Lottery strategy has the lowest total denied bookings compared to DCF and LC-DCF in the Pr and Risk policies, as shown in Figure 7. The Lottery strategy

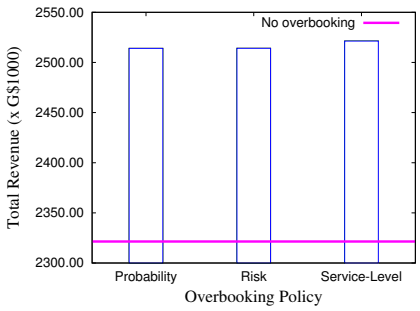

Figure 3. Total net revenue for RAL.

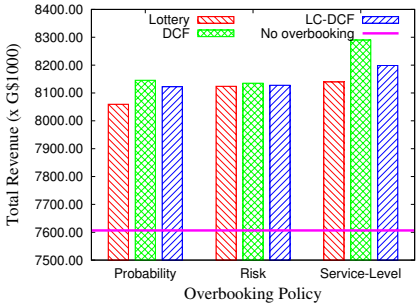

Figure 4. Total net revenue for Bologna.

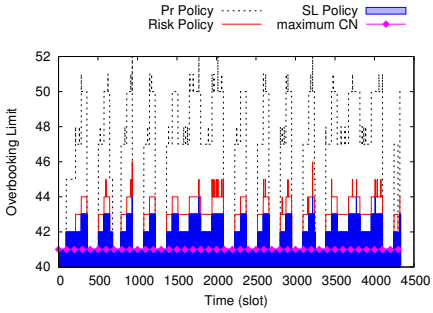

Figure 5. Overbooking Limit in RAL.

Table 10. Total denied bookings for the Service-Level policy.

	Premium Users	*Business* Users	*Budget* Users
Lottery	14	23	1
DCF	15	33	2
LC-DCF	6	38	7

is 4% and 35% lower than DCF in the Pr and Risk policies respectively. Moreover, it is 12% and 40% lower than LC-DCF in the Pr and Risk policies respectively. For the SL policy, the Lottery strategy is 2% higher than DCF, but 27% lower than LC-DCF, as depicted in Figure 7. As a result, the Lottery strategy works best in reducing total denied bookings. Moreover, it is the simplest and easiest to implement.

However, each denied booking has a different value in terms of the job duration time, user class level, and more importantly $cost_{ds}$. Thus, due to its randomness, the Lottery strategy pays the most amount of money to denied users, by up to 16%, 10% and 65% in the Pr, Risk and SL policies respectively, compared to DCF and LC-DCF, as depicted in Figure 8. Hence, from the compensation cost's point of view, the Lottery strategy is the least desirable.

Measuring DCF against LC-DCF, DCF has the lowest number of denied bookings by 7%, 3% and 30% in the Pr, Risk and SL policies respectively, as highlighted in Figure 7. Hence, in terms of total $cost_{ds}$, DCF is about 5% and 94% lower than LC-DCF in the Pr and SL policies respectively, as indicated in Figure 8. For the Risk policy, both DCF and LC-DCF have a similar cost, less than 0.5% of each other. Moreover, when combining with the SL policy, the total net revenue with DCF is the highest of all, as shown in Figure 4. Overall, from these findings, DCF seems to be a better choice than LC-DCF.

In our previous work [19], we found out that *Premium* and *Business* users contribute more than 60% on smaller and medium-sized resources (e.g. Torino and NorduGrid), and 50% on large-sized resources (e.g. CERN and Bologna) respectively. The main disadvantage of DCF is that this strategy does not take into consideration which user class level each booking belongs to. In contrast, LC-DCF removes bookings from lower-class users first, based on their $cost_{ds}$. As a result, LC-DCF has the lowest number of denied *Premium* users, as shown in Table 10. Therefore, to minimize the negative effects from high-paying users who have been denied access, the combination of SL and LC-DCF policies is a better solution in the long run.

8 Conclusion and Future Work

Advance reservation allows users to request available resources in the future. In this paper, we adopt several static overbooking policies, such as Probability (Pr), Risk, and Service-Level (SL), in managing unexpected cancellations and no-shows of reservations in a Grid system. Moreover, we present several strategies for selecting which excess reservations to deny, based on compensation cost and user class level, namely Lottery, Denied Cost First (DCF), and Lower Class DCF (LC-DCF). By overbooking, a resource provider is able to accept more reservations than the current capacity. As a result, it can be effectively used to increase the total net revenue of a resource by up to 9%.

From the performance evaluation, the Pr policy suffers from excessive denied bookings and compensation cost ($cost_{ds}$), since it calculates the overbooking limit (ob) based only on user demands at that particular time. The Risk policy manages to balance the show rate and $cost_{ds}$ well. However, it tends to set a more conservative ob, when the compensation cost is much higher than the weighted average price of all class users. Finally, the SL policy defines a specified level or fraction of denied users. This approach has the advantage of having the lowest denied bookings and $cost_{ds}$ compared to other policies.

With regards to the denied-booking strategies, we consider DCF to be the best as it has both the lowest $cost_{ds}$ compared to Lottery and LC-DCF, and the highest net revenue when associated with the SL policy. However, to prevent high-paying users from submitting their jobs to other resources due to overbooking, the

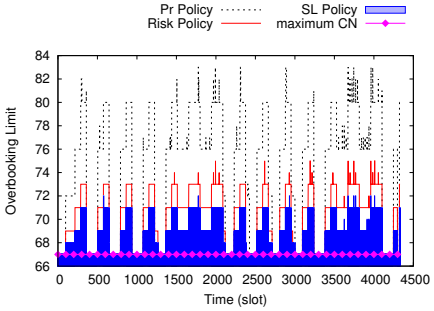

Figure 6. Overbooking Limit using DCF in Bologna.

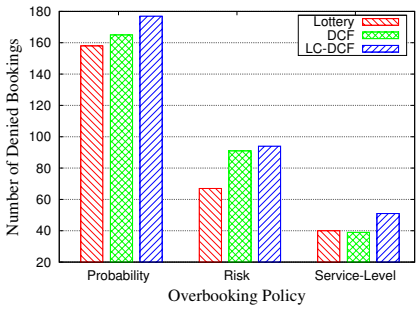

Figure 7. Denied bookings for Bologna (lower is better).

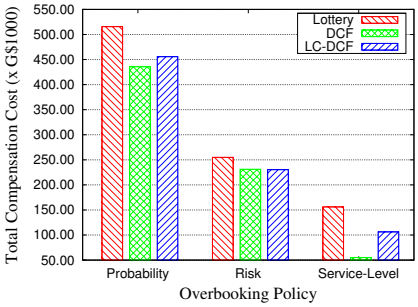

Figure 8. Total $cost_{ds}$ for Bologna (lower is better).

combination of the SL and LC-DCF policies is the better option.

As for future work, we need to consider a dynamic scenario where cancellations and no-shows are dependent of the number of total bookings. Moreover, we need to consider handling a group cancellation for batch reservations.

Acknowledgment

This work is partially supported by research grants from the Australian Research Council (ARC), and Australian Department of Education, Science and Training (DEST). We would like to thank C. S. Yeo for his comments on the paper.

References

[1] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[2] J. Coughlan. Airline Overbooking in the Multi-Class Case. *Operational Research Society*, 50(11):1098–1103, 1999.

[3] D. Feitelson. Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload, 2007.

[4] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[5] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proc. of the 7th Intl. Workshop on Quality of Service*, London, 1999.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications*, 15(3), 2001.

[7] M. K. Geraghty and E. Johnson. Revenue Management Saves National Car Rental. *Interfaces*, 27:107–127, 1997.

[8] G. C. Hadjinicola and C. Panayi. The Overbooking Problem in Hotels with Multiple Tour Operators. *Operations and Production Management*, 17(9):874–885, 1997.

[9] H. Li and M. Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Performance Evaluation Review*, 34(4):59–70, 2007.

[10] J. I. McGill and G. J. V. Ryzin. Revenue Management: Research Overview and Prospects. *Transportation Science*, 33(2):233–256, 1999.

[11] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow enactment in ICENI. *UK e-Science All Hands Meeting*, pages 894–900, Sep 2004.

[12] J. Milbrandt, M. Menth, and J. Junker. Experience-based Admission Control with Type-Specific Overbooking. In *Proc. of the 6th Intl. Workshop on IP Operations and Management (IPOM)*, Dublin, Ireland, Oct. 23–25 2006.

[13] D. of Transportation. Air Travel Consumer Report. In *Office of Aviation Enforcement and Proceedings (OAEP)*, Washington, DC, USA, April 2007.

[14] R. L. Phillips. *Pricing and Revenue Optimization*. Stanford University Press, 2005.

[15] B. C. Smith, J. F. Leimkuhler, and R. M. Darrow. Yield Management at American Airlines. *Interfaces*, 22:8–31, 1992.

[16] SPEC. Standard Performance Evaluation Corporation. http://www.spec.org, 2007.

[17] A. Sulistio, U. Cibej, S. Prasad, and R. Buyya. GarQ: An Efficient Scheduling Data Structure for Advance Reservations of Grid Resources. *Intl. Journal of Parallel, Emergent and Distributed Systems*, (in press, accepted on Jan. 19, 2008).

[18] A. Sulistio, K. H. Kim, and R. Buyya. On Incorporating an On-line Strip Packing Algorithm into Elastic Grid Reservation-based Systems. In *Proc. of the 13th Intl. Conference on Parallel and Distributed Systems (ICPADS)*, Hsinchu, Taiwan, Dec. 5–7 2007.

[19] A. Sulistio, K. H. Kim, and R. Buyya. Using Revenue Management to Determine Pricing of Reservations. In *Proc. of the 3rd Intl. Conference on e-Science and Grid Computing (e-Science)*, Bangalore, India, Dec. 10–13 2007.

[20] A. Sulistio, G. Poduval, R. Buyya, and C.-K. Tham. On Incorporating Differentiated Levels of Network Service into GridSim. *Future Generation Computer Systems*, 23(4):606–615, May 2007.

[21] K. T. Talluri and G. J. V. Ryzin. *The Theory and Practice of Revenue Management*. Springer Science + Bussiness Media, Inc., 2004.

[22] H. R. Thompson. Statistical Problems in Airline Reservation Control. *Operations Research Quaterly*, 12:167–185, 1961.

[23] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proc. of the 5th USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, Dec. 9–11 2002.

[24] Y.-X. Zhao and C.-J. Chen. A Redundant Overbooking Reservation Algorithm for OBS/OPS Networks. *Computer Networks*, 51(13):3919–3934, 2007.