Resources in a cloud are allocated in bundles, allowing users get maximum benefit from a specific

combination of resources. Indeed, along with CPU cycles, an application needs specific amounts ofmain

memory, disk space, network bandwidth, and so on. Resource bundling complicates traditional resource

allocation models and has generated interest in economic models and, in particular, auction algorithms.

In the context of cloud computing, an auction is the allocation of resources to the highest bidder.

Combinatorial Auctions. Auctions in which participants can bid on combinations of items, or packages,

are called combinatorial auctions [93]. Such auctions provide a relatively simple, scalable, and

tractable solution to cloud resource allocation. Two recent combinatorial auction algorithms are the simultaneous clock auction [29] and the clock proxy auction [30]. The algorithm discussed in this chapter

and introduced in [333] is called the ascending clock auction (ASCA). In all these algorithms the

current price for each resource is represented by a “clock” seen by all participants at the auction.

We consider a strategy in which prices and allocation are set as a result of an auction. In this auction,

users provide bids for desirable bundles and the price they are willing to pay. We assume a population

of U users, u = {1, 2, . . . ,U}, and R resources, r = {1, 2, . . . , R}. The bid of user u is Bu = {Qu, πu}

withQi = (q1

u , q2

u , q3

u , . . . ) an R-component vector; each element of this vector, qiu

, represents a bundle

of resources user u would accept and, in return, pay the total price πu. Each vector component qiu

is a

positive quantity and encodes the quantity of a resource desired or, if negative, the quantity of the resource

offered. A user expresses her desires as an indifference set I = (q1

u XOR q2

u XOR q3

u XOR . . . ).

The final auction prices for individual resources are given by the vector p = (p1, p2, . . . , pR) and

the amounts of resources allocated to user u are xu = (x1

u , x2

u, . . . , x R

u ). Thus, the expression [(xu)T p]

represents the total price paid by user u for the bundle of resources if the bid is successful at time T .

The scalar [minq∈Qu (qT p)] is the final price established through the bidding process.

The bidding process aims to optimize an objective function f (x, p). This function could be tailored

to measure the net value of all resources traded, or it can measure the total surplus – the difference

between the maximum amount users are willing to pay minus the amount they pay. Other optimization

functions could be considered for a specific system, e.g., the minimization of energy consumption or

of security risks.

Pricing and Allocation Algorithms. A pricing and allocation algorithm partitions the set of users

into two disjoint sets, winners and losers, denoted as W and L, respectively. The algorithm should:

1. Be computationally tractable. Traditional combinatorial auction algorithms such as Vickey-Clarke-

Groves (VLG) fail this criteria, because they are not computationally tractable.

2. Scale well. Given the scale of the system and the number of requests for service, scalability is a

necessary condition.

3. Be objective. Partitioning in winners and losers should only be based on the price πu of a user’s bid.

If the price exceeds the threshold, the user is a winner; otherwise the user is a loser.

4. Be fair.Make sure that the prices are uniform. All winners within a given resource pool pay the same

price.

5. Indicate clearly at the end of the auction the unit prices for each resource pool.

6. Indicate clearly to all participants the relationship between the supply and the demand in the system.

The function to be maximized is

max

x,p

f (x, p). (6.25)

The constraints in Table 6.4 correspond to our intuition: (a) the first one states that a user either

gets one of the bundles it has opted for or nothing; no partial allocation is acceptable. (b) The second

constraint expresses the fact that the system awards only available resources; only offered resources

can be allocated. (c) The third constraint is that the bid of the winners exceeds the final price. (d) The

fourth constraint states that the winners get the least expensive bundles in their indifference set. (e) The

fifth constraint states that losers bid below the final price. (f) The last constraint states that all prices are

positive numbers.

The ASCA Combinatorial Auction Algorithm. Informally, in the ASCA algorithm [333] the participants

at the auction specify the resource and the quantities of that resource offered or desired at the

price listed for that time slot. Then the excess vector

z(t) =

u

xu(t) (6.26)

is computed. If all its components are negative, the auction stops; negative components mean that the

demand does not exceed the offer. If the demand is larger than the offer, z(t)   0, the auctioneer

increases the price for items with a positive excess demand and solicits bids at the new price. Note that

the algorithm satisfies conditions 1 through 6; from Table 6.3 all users discover the price at the same time

and pay or receive a “fair” payment relative to uniform resource prices, the computation is tractable, and

the execution time is linear in the number of participants at the auction and the number of resources. The

computation is robust and generates plausible results regardless of the initial parameters of the system.

There is a slight complication as the algorithm involves user bidding in multiple rounds. To address

this problem the user proxies automatically adjust their demands on behalf of the actual bidders, as

shown in Figure 6.6. These proxies can be modeled as functions that compute the “best bundle” from

each Qu set given the current price

Qu =

ˆ qu if ˆ qT

u p   πu with ˆ qu ∈ argmin (qT

u p)

0 otherwise .

The input to the ASCA algorithm: U users, R resources,   p the starting price, and the update increment

function, g : (x, p)  → RR. The pseudocode of the algorithm is:

1: set t = 0, p(0) =  p

2: loop

3: collect bids xu(t) = Gu(p(t)), ∀u

4: calculate excess demand z(t) =

u xu(t)

5: if z(t) <0 then

6: break

7: else

8: update prices p(t + 1) = p(t) + g(x(t), p(t))

9: t ← t + 1

10: end if

11: end loop

In this algorithm g(x(t), p(t)) is the function for setting the price increase. This function can be

correlated with the excess demand z(t), as in g(x(t), p(t)) = αz(t)

+ (the notation x+ means max (x, 0))

with α a positive number. An alternative is to ensure that the price does not increase by an amount larger

than δ. In that case g(x(t), p(t)) = min (αz(t)

+

, δe) with e = (1, 1, . . . , 1) is an R-dimensional vector

and minimization is done component wise.

The convergence of the optimization problem is guaranteed only if all participants at the auction are

either providers of resources or consumers of resources, but not both providers and consumers at the same

time. Nevertheless, the clock algorithm only finds a feasible solution; it does not guarantee its optimality.

The authors of [333] have implemented the algorithm and allowed internal use of it within Google.

Their preliminary experiments show that the system led to substantial improvements. One of the most

interesting side effects of the new resource allocation policy is that users were encouraged to make their

applications more flexible and mobile to take advantage of the flexibility of the system controlled by

the ASCA algorithm.

An auctioning algorithm is very appealing because it supports resource bundling and does not require

a model of the system. At the same time, a practical implementation of such algorithms is challenging.

First, requests for service arrive at random times, whereas in an auction all participants must react

to a bid at the same time. Periodic auctions must then be organized, but this adds to the delay of

the response. Second, there is an incompatibility between cloud elasticity, which guarantees that the

demand for resources of an existing application will be satisfied immediately, and the idea of periodic

auctions.