```
:- use_module(library(clpfd)).

%Define all container facts.
container(a,2,2).
container(b,4,1).
container(c,2,2).
container(d,1,1).
%Define contianer stacking hierarchy
on(a,d).
on(b,c).
on(c,d).

%Main clause for finding the best scheduling
schedule(StartList, EndList, Sum) :-
    %Convert facts to list
    container_to_list(ContainerList),
    %Define length to be equal to number of containers
    length(ContainerList,N),
    length(StartList,N),
    length(EndList,N),
    %Set domains to limit search space.
    domain(StartList, 0, 10),
    domain(EndList, 0, 10),
    WorkersRequired in 1..10,
    MaxEndTime in 1..10,
    %Make sure start and end times are concistent with the stacking hierachy
    check_constraints(StartList,EndList,ContainerList),
    %Find highest end time
    max_end_time(EndList,MaxEndTime),
    %Generate tasks for cumulative
    task(StartList,EndList, ContainerList, Tasks),
    %Solve the schelduing while limiting number of workers
    cumulative(Tasks,[limit(WorkersRequired)]),
    %Assign solution cost given the problem description
    Sum #= WorkersRequired * MaxEndTime,
    %Find the lowest sum and start times
    labeling([minimize(Sum)], [Sum|StartList]).

%Find the largest value in EndList
max_end_time([],_).
max_end_time([End|EndList],MaxEndTime) :-
        MaxEndTime #>= End,
        max_end_time(EndList,MaxEndTime).

%Converts from container fact to list.
container_to_list(List) :- findall([Id,Workers,Duration], container(Id,Workers,Durat
ion),List).

%Check problem constraints
check_constraints([],[],[]).
check_constraints([Start|StartList],[End|EndList],[[Id,_,_]|ContainerList]) :-
        check_forward_constraints(Start,StartList,End,EndList,Id,ContainerList),
        check_constraints(StartList,EndList,ContainerList).

%Verify stacking consistency for the rest of the container list.
check_forward_constraints(_,[],_,[],_,[]).
check_forward_constraints(Start1,[Start2|StartList],End1,[End2|EndList],Id1,[[Id2,_,
_]|ContainerList]) :-
        constraint(Start1,Start2,End1,End2,Id1,Id2),
        check_forward_constraints(Start1,StartList,End1,EndList,Id1,ContainerList).

% Make sure the containers are unstacked in the right order
% according to their start and end times.
constraint(_, Start2, End1, End2, Id1,Id2) :-
        on(Id1,Id2),
        Start2 #>= End1,
        End2 #>= Start2.
constraint(Start1, _, End1, End2, Id1, Id2) :-
        on(Id2,Id1),
        Start1 #>= End2,
        End1 #>= Start1.
% If there is no dependency between the containers,
% then there is no need to constraint this container.
```

```
constraint(_, _, _, _, Id1, Id2) :-
        findall(_,on(Id1,Id2),[]),
        findall(_,on(Id2,Id1),[]).

% Format the different values to allow cumulative to solve the scheduling problem fo
r us.
task([],[],[],[]).
task([Start|StartList], [End|EndList], [[Id,Workers,Duration]|ContainerList], [T|Tas
kList]) :-
        T = task(Start,Duration,End,Workers,0),
        task(StartList,EndList,ContainerList,TaskList).

% Example run
%
% | ?- schedule(S,E,Sum).
% S = [1,0,1,3],
% E = [3,1,3,4],
% Sum = 16 ? ;
% no
```