Online server allocation in a server farm via benefit task systems

T.S. Jayram (IBM Almaden)

Tracy Kimbrel (IBM Watson)

Robert Krauthgamer (Hebrew University)

Baruch Schieber (IBM Watson)

Maxim Sviridenko (IBM Watson)

Optimization in the service industry

Optimization problems arise naturally not only in manufacturing but also in the service industry.

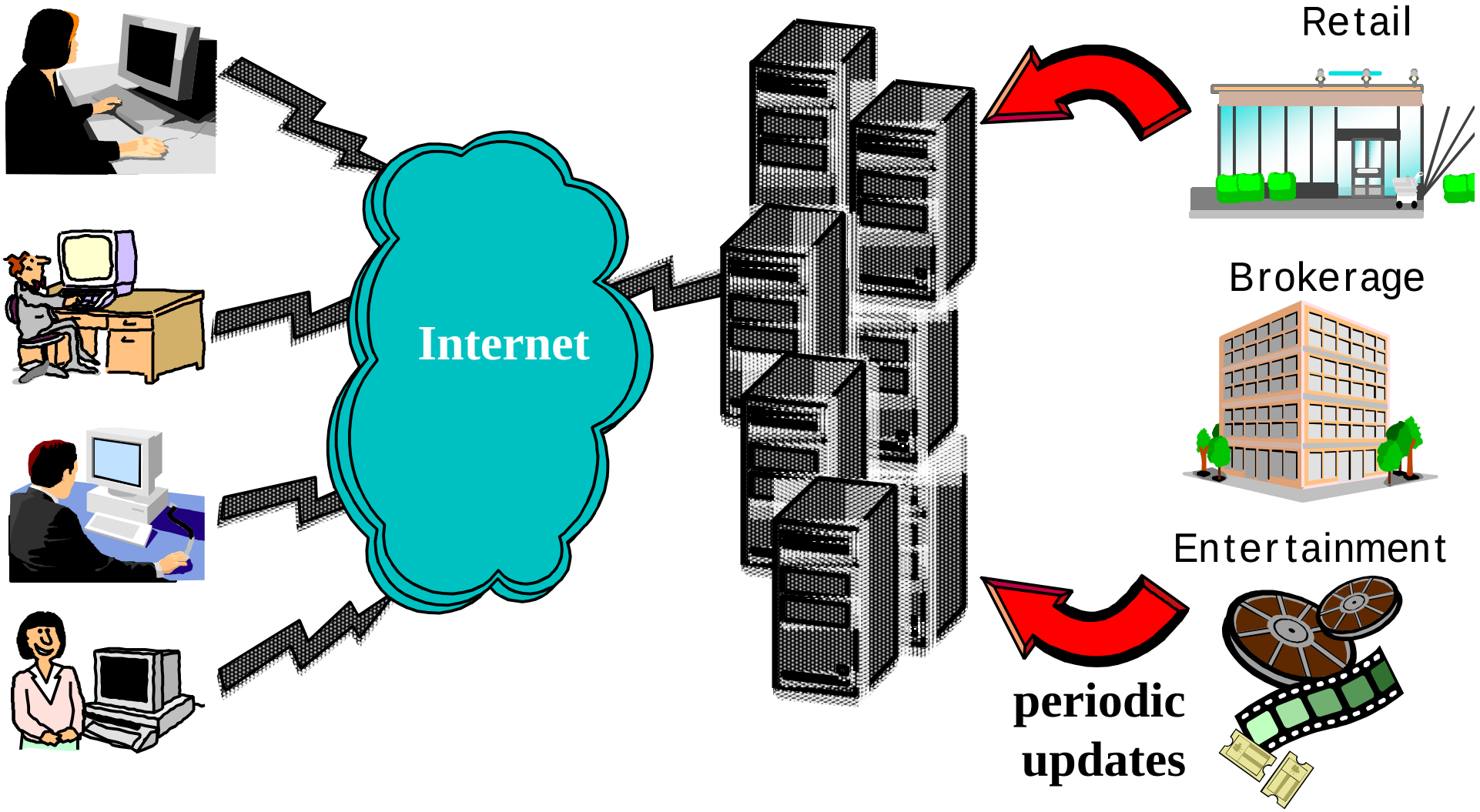
The outsourcing trend calls for highly efficient competitive service providers that

- Rely on economy of scale & sharing of resources and thus
- Wish to maximize utilization of available resources.

We illustrate this on web content hosting service

- That hosts and serves (clients) web pages from farms of commodity servers.

Web content hosting



Advantages of a server farm

Cost effective (economy of scale)

- Increases resources utilization.
- Improves availability.

Assumptions:

- Server is allocated to one site at a time (for security).
- The time it takes to reload a server with another site s data is relatively short (5-10 minutes).

Problem: How to allocate the servers optimally (especially with unknown usage pattern) ?

The web server farm model

Time is divided into intervals of fixed length.

Each site s demand is assumed to be uniform throughout an interval.

Each server has a rate of requests it can serve in an interval (normalized to be 1).

During an interval, a server can either

- Serve requests (demand) of its allocated site, or
- Change its allocation (and serve no requests).

The model (cont.)

Each site s demand is associated with a benefit.

- Benefit (and demand) can vary across servers and time.

Servers allocated for a site gain the benefit of the demand that they satisfy (at the same time interval).

Demand not satisfied within the interval is lost.

Objective: Find a time-varying server allocation that maximizes the total benefit.

Three versions: offline, fully online, lookahead.

Web server farm problem

Input:

s web sites that are to be served by k servers,

a nonnegative demand matrix $d_{i,t}$, $i \in [1..s]$, $t \in [1..\tau]$,

a nonnegative benefit matrix $b_{i,t}$, $i \in [1..s]$, $t \in [1..\tau]$.

Output:

Time-varying allocation $a_{i,t}$, $i \in [1..s]$, $t \in [1..\tau]$,

that satisfies $\sum_i a_{i,t} \le k$ for every t, and

maximizes the total benefit gained:

$$\sum_t \sum_i b_{i,t} \cdot \min\{d_{i,t}, a_{i,t}, a_{i,t-1}\}$$

Benefit task system problem

Input:

a set of possible states Y_t for all $t \hat{I} [1..\tau]$,

the benefit accrued by each transition, given by a nonnegative benefit function $B : \bigcup_t (Y_t \times Y_{t+1}) \to R_+$

Output:

a sequence of states $\{y_t\}$, that

maximizes the total benefit gained $\sum_t B(y_t, y_{t+1})$.

Includes benefit maximization versions of the k-server problem and metrical task system.

Results for web server farm problem

Offline: Can be solved in polynomial time.

Competitive ratio = worst-case ratio of optimal offline benefit to the benefit accrued by online (with or without lookahead) algorithm. (≥ 1)

Fully online (no lookahead): Unbounded comp ratio.

Online with lookahead L: nearly tight bounds for deterministic and randomized algorithms. Even tighter bounds for two special cases ($L=1$ $s=2,*$).

Results for lookahead L (both problems)

	Competitive ratio	
	Lower bound	Upper bound
Deterministic L=1	$4-\varepsilon$	4
Deterministic L>1	$1+\frac{1}{L}+\frac{1}{L^2+1}$	$\min\{1+\frac{4}{L-7}, (1+\frac{1}{L})\sqrt[L]{L+1}\}$
Randomized	$1+\frac{1}{L}-e$	$1+\frac{1}{L}$

Offline web server farm problem

Can be solved efficiently by dynamic programming when $k = O(1)$ or $s = O(1)$.

Can be reduced to min-cost flow problem.

- Each server is represented by a unit of flow.
- Recall that (minimum cost) maximum flow is integral.
- The network size is linear in demands matrix size.
- Hence, can be solved in polynomial time.

Used by lookahead algorithms to compute optimal moves sequence for the foreseeable future.

The future discounting algorithm

Motto: Bird in the hand is worth a in the bush .

I.e., discount the future exponentially: Repeat every step

1. Let $b_0, b_1, \ , b_L$ be possible benefits in the next L moves.
2. Follow the step that maximizes $b_0 + b_1/a + \ + b_L/a^L$.

   Using offline (min-cost flow) algorithm

Theorem: When $a = \sqrt[L]{L+1}$, competitive ratio is

$$(1+1/L)\sqrt[L]{L+1} = 1+\Theta\left(\log L / L\right) .$$

(E.g. L=1 $\Rightarrow$ a = 2 $\Rightarrow$ competitive ratio = 4 is optimal.)

The intermittent reset algorithm

Motivation (from a randomized algorithm):

Sacrifice one out of (roughly) L steps to gain optimal benefit in all other steps.

Iteratively,

Choose a time $T \in [\text{now}+L/2, \quad ,\text{now}+L]$, whose maximum benefit (over all transitions) is minimal.

Follow the optimal path from now to step T.

Theorem: Competitive ratio is $1+4/(L-7)=1+\Theta(1/L)$

(Nearly optimal for large L.)

Experimental results

We tested experimentally the offline, discount and reset algorithms.

We used inputs generated at random and inputs generated from commercial retail web servers logs.

Surprisingly, all algorithms performed very well.

In general, the discount algorithm performed better than the reset algorithm.

In case of underload, the discount algorithm was optimal (not surprising).

Concluding remarks

Is the discount better than the reset algorithm?

The discount algorithm is more suitable when demands are steady (naive forecasting).

Can we make the model more realistic?

A restricted, more realistic, adversary.

Consider errors in forecasting of future demands.