Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters

Presenter: **Hongxing Li**

Authors: S. T. Maguluri, R. Srikant and L. Ying

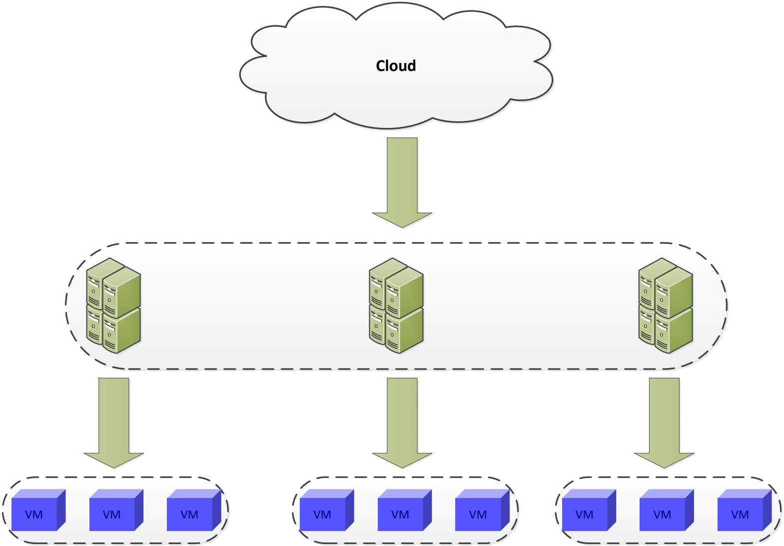
Sept. 19, 2012

Cloud computing platforms:

- Infrastructure as a service
- Platform as a service;
- Software as a service.

Cloud computing platforms:

- Infrastructure as a service: virtual machines (VMs);
- Platform as a service;
- Software as a service.


Cloud
VM
VM
VM
VM
VM
VM
VM
VM
VM

- What is the **capacity region** of a cloud in handling jobs;
- What is the **throughput optimality** in a cloud;
- What is **load balancing** and **scheduling** for cloud in this paper;
- How to design **throughput optimal algorithms** for load balancing and scheduling.

A preliminary definition

Capacity: the maximum rate at which jobs can be processed.

- How to define a VM;
- How to define a job.

Example of VMs

Instance Type	Memory	CPU	Storage
Standard Extra Large	15 GB	8 EC2 units	1,690 GB
High-Memory Extra Large	17.1 GB	6.5 EC2 units	420 GB
High-CPU Extra Large	7 GB	20 EC2 units	1,690 GB

Table: Three representative instances in Amazon EC2.

Cloud model

- K different resources, *e.g.*, CPU, memory and storage;
- M distinct VM types:
  - Resource requirement R_{mk}: amount of type-k resource required by type-m VM;
- L different servers;
  - C_{ik}: amount of type-k resource at server i
- One Job j:
  - VM type: m;
  - Size $S_j \leq S_{max}$: time slots.

VM configuration

- VM configuration $N^{(i)}$: an M-dimensional vector combination of different resources;
- A feasible VM configuration:

$$\sum_{m=1}^{M} N_m^{(i)} R_{mk} \leq C_{ik}, \ \forall i \in [1, L].$$

Here,

- R_{mk}: Type-k resource requirement of type-m VM;
- C_{ik}: Available type-k resource at server i.

Example: feasible VM configuration

Instance Type	Memory	CPU	Storage
Standard Extra Large	15 GB	8 EC2 units	1,690 GB
High-Memory Extra Large	17.1 GB	6.5 EC2 units	420 GB
High-CPU Extra Large	7 GB	20 EC2 units	1,690 GB

Table: Three representative instances in Amazon EC2.

One server:

- 30 GB of memory;
- 30 EC2 compute units;
- 4,000 GB of storage space.

Example: feasible VM configuration

Instance Type	Memory	CPU	Storage
Standard Extra Large	15 GB	8 EC2 units	1,690 GB
High-Memory Extra Large	17.1 GB	6.5 EC2 units	420 GB
High-CPU Extra Large	7 GB	20 EC2 units	1,690 GB

Table: Three representative instances in Amazon EC2.

One server:

- 30 GB of memory;
- 30 EC2 compute units;
- 4,000 GB of storage space.

Feasible VM configurations:

- $N = (2, 0, 0)$;
- $N = (0, 1, 1)$

Job arrival and service model

- $\mathcal{A}_m(t)$: set of type m jobs arrive at time slot t;
- $A_m(t) = |\mathcal{A}_m(t)|$: number of type m jobs' arrival at t.
- $W_m(t) = \sum_{j \in \mathcal{A}_m(t)} S_j$: overall size of jobs in $\mathcal{A}_m(t)$;
  - $W_m(t)$: i.i.d. process with $E[W_m(t)] = \lambda_m$ and $Pr(W_m(t)) = 0 > \epsilon_W, \exists \epsilon_W > 0$.
- $D_m(t)$: number of served type-m jobs at slot t.

Job arrival and service model

- $\mathcal{A}_m(t)$: set of type m jobs arrive at time slot t;
- $A_m(t) = |\mathcal{A}_m(t)|$: number of type m jobs' arrival at t.
- $W_m(t) = \sum_{j \in \mathcal{A}_m(t)} S_j$: overall size of jobs in $\mathcal{A}_m(t)$;
  - $W_m(t)$: i.i.d. process with $E[W_m(t)] = \lambda_m$ and $Pr(W_m(t)) = 0 > \epsilon_W, \exists \epsilon_W > 0$.
- $D_m(t)$: number of served type-m jobs at slot t.

Note: the size of each $D_m(t)$ job reduces one at the end of slot t.

Job arrival and service model

- $\mathcal{A}_m(t)$: set of type m jobs arrive at time slot t;
- $A_m(t) = |\mathcal{A}_m(t)|$: number of type m jobs' arrival at t.
- $W_m(t) = \sum_{j \in \mathcal{A}_m(t)} S_j$: overall size of jobs in $\mathcal{A}_m(t)$;
  - $W_m(t)$: i.i.d. process with $E[W_m(t)] = \lambda_m$ and $Pr(W_m(t)) = 0 > \epsilon_W, \exists \epsilon_W > 0$.
- $D_m(t)$: number of served type-m jobs at slot t.

Note: the size of each $D_m(t)$ job reduces one at the end of slot t.
Queue of type-m jobs:

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i N_m^{(i)} \right)^+.$$

Here, $(x)^+ = \max\{x, 0\}$.

Cloud stability

The cloud system is stable if

$$\limsup_{t\to\infty} E[\sum_m Q_m(t)] \leq \infty.$$

Capacity region

Capacity of a cloud: the set of traffic loads under which the queues in the system can be stabilized.

$$\mathcal{C} = \left\{ \lambda : \lambda = \sum_{i=1}^{L} \lambda^{(i)} \text{ and } \lambda^{(i)} \in Conv(\mathcal{N}_i) \right\}.$$

Here, $Conv(\cdot)$ is the convex hull. N_i: set of feasible VM-configurations on a server i.

- 3 servers and 2 resource types:
  - $\mathcal{N}^{(1)} = \mathcal{N}^{(2)} = \{(0,0),(1,0),(0,1)\}$;
  - $\mathcal{N}^{(3)} = \{(0,0),(1,0),(2,0),(0,1)\}$

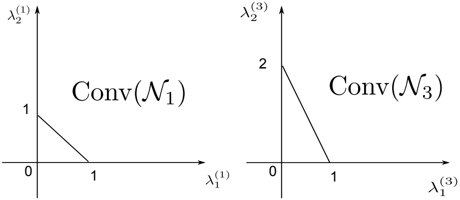


Figure: Regions $Conv(\mathcal{N}_1)$ and $Conv(\mathcal{N}_3)$

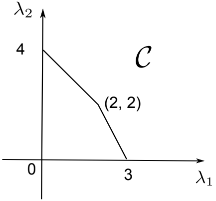


Figure: Capacity region $\mathcal{C}$

Throughput optimality

Definition

A job scheduling algorithm is throughput optimal if the algorithm can support any λ such that $(1+\epsilon)\lambda \in \mathcal{C}$ for some $\epsilon > 0$.

Load balancing and scheduling problem

Load balancing:

- To which server, the jobs should be routed;
- How many jobs should be routed.

Scheduling:

- How to decide the VM configuration on each server at each time slot to handle the jobs.

Centralized scheduler without load balancing

- Jobs are queued at the central job scheduler;
- there is no queue at each server;
  - No load balancing problem;
- Centralized job scheduler makes the VM configuration.

Algorithm 1 with preemption

- VM configuration: server-by-server MaxWeight allocation,

$$N^{(i)*}(t) \in \text{arg} \max_{N \in \mathcal{N}^{(i)}} \sum_m Q_m(t) N_m, \ \forall i \in [1, L].$$

- Queue update:

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i N_m^{(i)}(t) \right)^+, \ \forall m \in [1, M].$$

- Unfinished and preempted jobs are put back to the queues.

Algorithm 1 with preemption

- VM configuration: server-by-server MaxWeight allocation,

$$N^{(i)*}(t) \in \text{arg} \max_{N \in \mathcal{N}^{(i)}} \sum_m Q_m(t) N_m, \ \forall i \in [1, L].$$

- Queue update:

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i N_m^{(i)}(t) \right)^+, \ \forall m \in [1, M].$$

- Unfinished and preempted jobs are put back to the queues.

Achievable capacity region: the set of λ with $(1+\epsilon)\lambda \in \mathcal{C}$ for some ϵ.

- Throughput optimal if $\epsilon \to 0$.

Algorithm 1 is throughput-optimal in an ideal scenario, where

- Jobs can be preempted: costly;
- Jobs can migrate among servers: costly;
- Servers can be reconfigured at each time instant.

Algorithm 1 is throughput-optimal in an ideal scenario, where

- Jobs can be preempted: costly;
- Jobs can migrate among servers: costly;
- Servers can be reconfigured at each time instant.

Non-preemptive case: once a job is scheduled, it will be processed without preemption on one sever.

Algorithm 2 without preemption

- Group T slots into one super time slot;
- VM configuration: server-by-server MaxWeight allocation,
  - $t = nT$: the beginning of one super time slot,

$$N^{(i)*}(t) \in \arg \max_{N \in \mathcal{N}^{(i)}} \sum_m Q_m(t) N_m, \ \forall i \in [1, L].$$

  - $t \neq nT$: within the super time slot,

$$\tilde{N}^{(i)*}(t) \in \arg \max_{N : N + N^{(i)}(t^-) \in \mathcal{N}^{(i)}} \sum_m Q_m(t) N_m, \ \forall i \in [1, L].$$

    Here, $N^{(i)}(t^-)$: scheduled and unfinished jobs by the end of slot $t-1$.
  - Constraint on $\tilde{N}^{(i)}(t)$: all served jobs can be finished by the end of super time slot.

Algorithm 2 without preemption (Cont.)

- Queue update:

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i (N_m^{(i)}(t^-) + \tilde{N}_m^{(i)}(t)) \right)^+ .$$

- Unfinished and preempted jobs are put back to the queues.

Algorithm 2 without preemption (Cont.)

- Queue update:

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i (N_m^{(i)}(t^-) + \tilde{N}_m^{(i)}(t)) \right)^+ .$$

- Unfinished and preempted jobs are put back to the queues.

Achievable capacity region: the set of λ with $(1+\epsilon)\frac{T}{T-S_{max}}\lambda \in \mathcal{C}$ for some ϵ.

- Throughput optimal if $\epsilon \to 0$ and $T \to \infty$.

No centralized scheduler

- No centralized queues, *i.e.*, $Q_m(t)$;
- Each server maintains the queues for each VM type: $Q_{mi}(t)$;
- Load balancing: for one job arrival, to which server should it be routed?
- Scheduling: VM configuration on each server.

Algorithm 3 without preemption

- Load balancing: Join-the-Shortest-Queue (JSQ) algorithm;
  1. Find the server with shortest queue for type m:

$$i_m^*(t) = \arg \min_{i \in \{1,2,\ldots,L\}} Q_{mi}(t).$$

  2. Route all arrivals of type m to server i_m^*,

$$W_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & \text{otherwise} \end{cases}.$$

Algorithm 3 without preemption (cont.)

- Scheduling: Non-preemptive Myopic MaxWeight algorithm,
  - $t = nT$: the beginning of one super time slot,

$$N^{(i)*}(t) \in \arg\max_{N \in \mathcal{N}^{(i)}} \sum_m Q_{mi}(t) N_m, \ \forall i \in [1, L].$$

  - $t \neq nT$: within the super time slot,

$$\tilde{N}^{(i)*}(t) \in \arg\max_{N: N + N^{(i)}(t^-) \in \mathcal{N}^{(i)}} \sum_m Q_{mi}(t) N_m, \ \forall i \in [1, L].$$

  - Constraint on $\tilde{N}^{(i)}(t)$: all served jobs can be finished by the end of super time slot.
- Queue update:

$$Q_{mi}(t+1) = \left(Q_{mi}(t) + W_{mi}(t) - \sum_i (N_m^{(i)}(t^-) + \tilde{N}_m^{(i)}(t)) \right)^+ .$$

Drawbacks of Algorithm 3 and other alternatives

JSQ keeps track of queue lengths at all servers: computation- and communication-prohibitive if

- Number of servers: large;
- Arrival rates of jobs: large.

Drawbacks of Algorithm 3 and other alternatives

JSQ keeps track of queue lengths at all servers: computation- and communication-prohibitive if

- Number of servers: large;
- Arrival rates of jobs: large.

Two alternatives with low complexity

- Power-of-two-choices routing for identical servers;
- Pick-and-compare routing for for non-identical servers.

Power-of-two-choices

Key idea: when a job arrives, two servers are sampled at random, and the job is routed to the server with the smaller queue for that job type.

For jobs of each type m,

1. Uniformly randomly choose two servers: $i_1^m(t)$ and $i_2^m(t)$;
2. Find the one with shorter queue length for type-m jobs:

$$i_m^*(t) = \text{arg} \min_{i \in \{i_1^m(t), i_2^m(t)\}} Q_{mi}(t);$$

3. Route the all type-m jobs to the server with shorter queue length:

$$W_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & \text{otherwise} \end{cases}.$$

Power-of-two-choices

Key idea: when a job arrives, two servers are sampled at random, and the job is routed to the server with the smaller queue for that job type.

For jobs of each type m,

1. Uniformly randomly choose two servers: $i_1^m(t)$ and $i_2^m(t)$;
2. Find the one with shorter queue length for type-m jobs:

$$i_m^*(t) = \text{arg} \min_{i \in \{i_1^m(t), i_2^m(t)\}} Q_{mi}(t);$$

3. Route the all type-m jobs to the server with shorter queue length:

$$W_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & \text{otherwise} \end{cases}.$$

- Throughput optimal;
- Drawback: throughput optimal only if all servers are identical.

Pick-and-compare

Key idea: when a job arrives, one server is sampled at random and compared with the decision in previous slot, and the job is routed to the server with the smaller queue for that job type.

For jobs of each type m,

1. Uniformly randomly choose one server: $i_m(t)$;
2. Compare $i_m(t)$ with the server to which jobs are routed in previous slot $i_m^*(t-1)$, find the one with shorter queue length:
$$i_m^*(t) = \arg \min_{i \in \{i_m(t), i_m^*(t-1)\}} Q_{mi}(t);$$
3. Route the all type-m jobs to the server with shorter queue length:
$$W_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & \text{otherwise} \end{cases}.$$

Note: applicable to cloud system with non-identical servers.

Delay performance

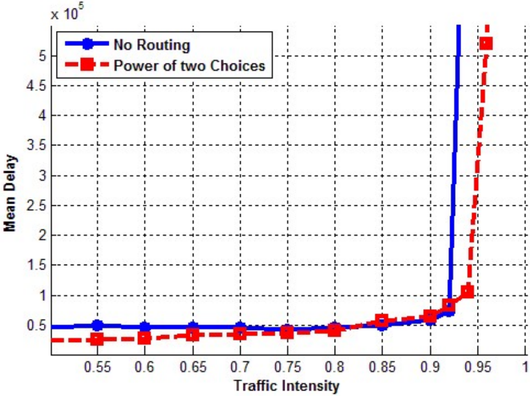


Figure: Comparison of Mean delay in in the cloud computing cluster in the case with a common queue and in the case with power of two choices routing when frame size is 4000

Impact of super time slot T

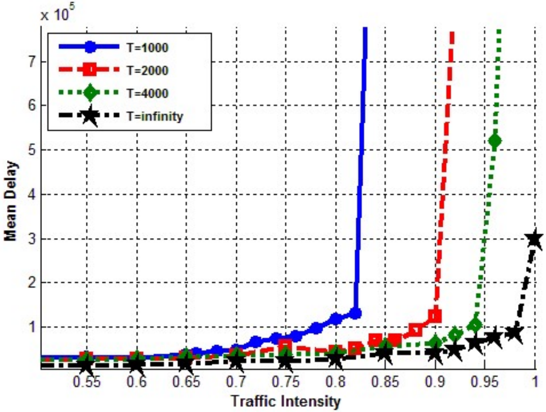


Figure: Comparison of power-of-two choices routing algorithm for various frame lengths T

Conclusion

- Define the capacity region and throughput optimality of cloud computing systems;
- One preemptive algorithm and three non-preemptive algorithms with throughput optimality.

Remarks

- Back-pressure routing and scheduling:
  - JSQ routing;
  - Max-weight scheduling;
- Proof based on Lyapunov theory for network stability:
  - Design our own stochastic optimization framework with: i), utility function; ii) revised analysis;
  - Impact of T is different!

Thank You!

Q&A