A Success-History Based Learning Procedure to Optimize Server Throughput in Large Distributed Control Systems

Yuan Gao

Department of Electrical and Computer Engineering

Stony Brook University, USA

Outlines

PART 1:
Client-Server Problem with Dynamic Server Capacity

PART 2:
A Regret-Based Procedure and Success-History based Parameter Adaptation Scheme

PART 3:
Simulation Results

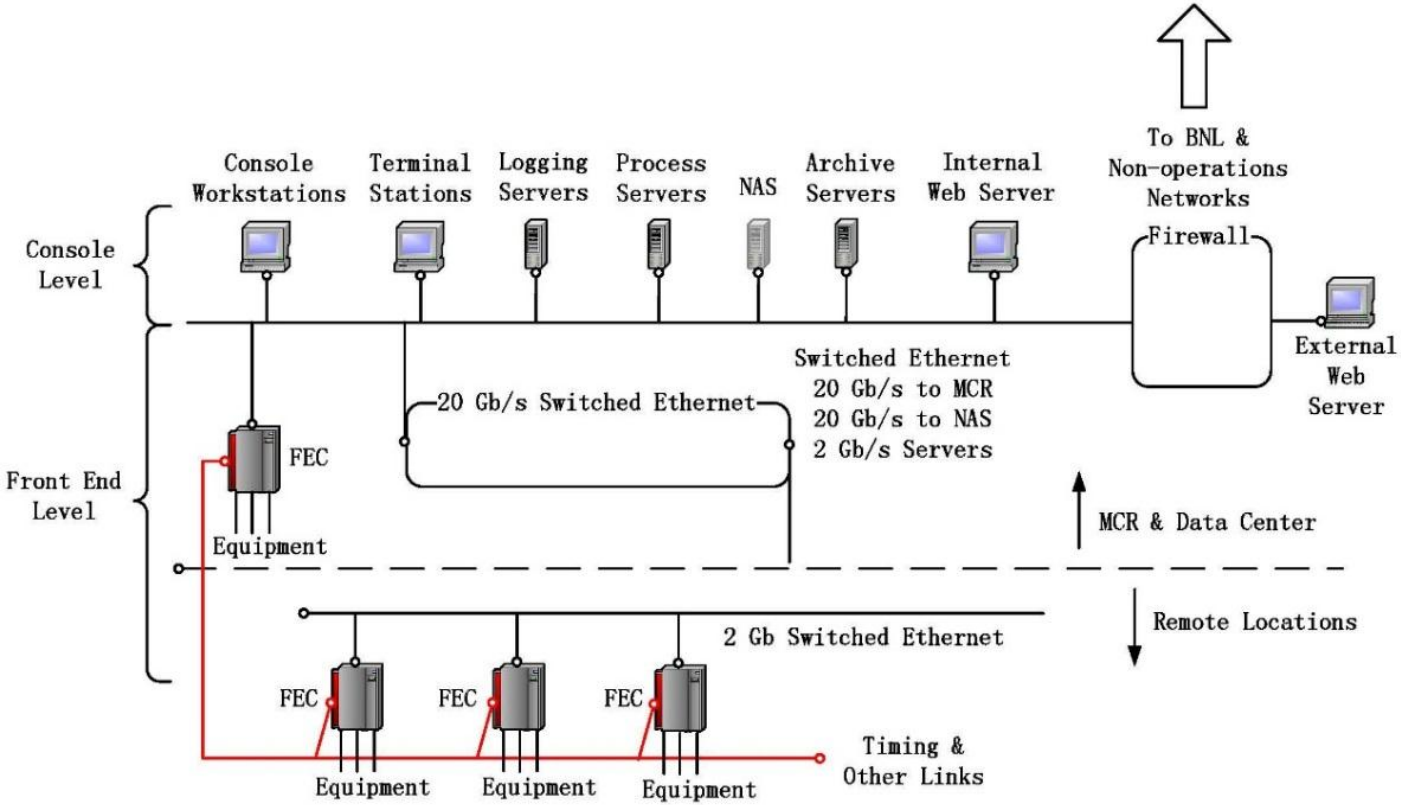
Outlines

PART 1:
Client-Server Problem with Dynamic Server Capacity

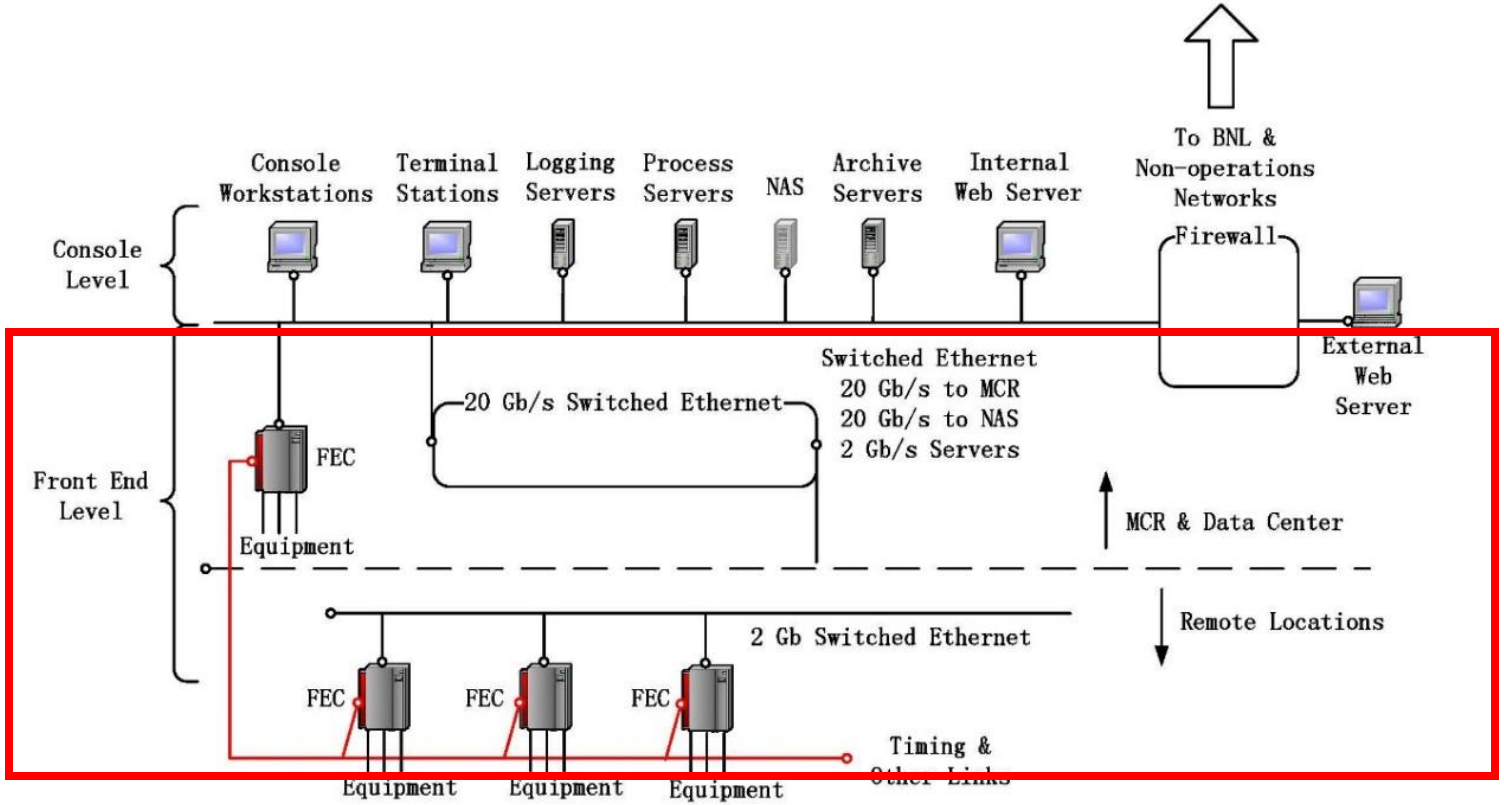
PART 2:
A Regret-Based Procedure and Success-History based Parameter Adaptation Scheme

PART 3:
Simulation Results

RHIC Control System Overview



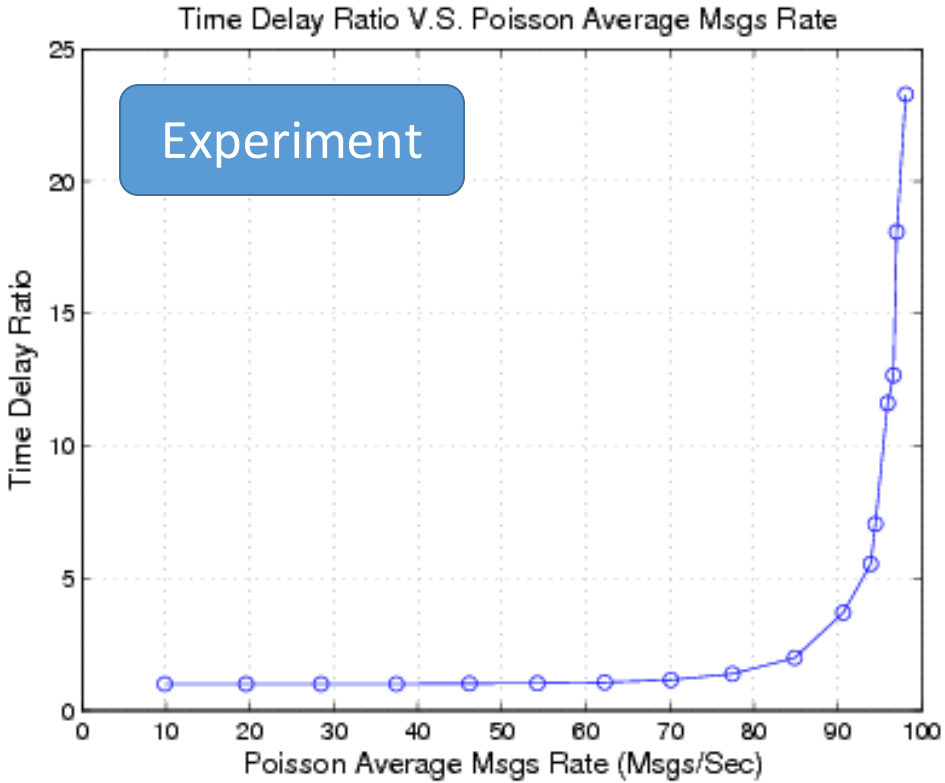
RHIC Control System Overview



Client-Server Problem with Dynamic Server Capacity

- A performance bottleneck in the Front End System…

In the RHIC front end system, every computer acts as a server providing services to a large number of clients. When the number of clients reaches its limit, the system slows down or even crashes.

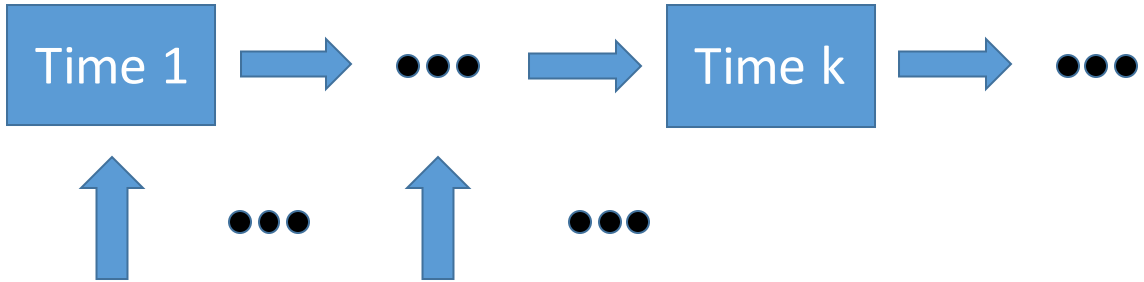


Client-Server Problem with Dynamic Server Capacity

- Moreover, there are asynchronous processes residing on FECs, those processes will share FECs' resources when the information they queried is updated, resulting in a varying available server capacity circumstance.
- One difficulty to deal with this situation is how to regulate clients behaviors so that they can learn the server's limitations and adjust their strategies properly.
- In this work, we consider this problem from a game theory perspective.

Client-Server Game Model – Repeated Game

Repeated game:
A same stage game is played over and over again.



Stage Game

Players	A set of n clients
Actions	Send (S) or Hold (H)
Client i's traffic	L(i)
Server crash cost	-c

	S	H
S	-c, -c	L(1), 0
H	0, L(2)	0, 0

Payoff table of a 2-client example

Client-Server Problem with Dynamic Server Capacity

- Moreover, there are asynchronous processes residing on FECs, those processes will share FECs' resources when the information they queried is updated, resulting in a varying available server capacity circumstance.
- One difficulty to deal with this situation is how to regulate clients behaviors so that they can learn the server's limitations and adjust their strategies properly.
- In this work, we consider this problem from a game theory perspective.
- Our goal is to safely and efficiently route clients' traffic so that server's throughput is maximized, and server crashes as little as possible.

Client-Server Game – Integer Programming

$$\textbf{Max}: \sum_{t=1}^{T} \sum_{i=1}^{n} a_i^t L_i$$

Subject to

$$\sum_{i=1}^{n} a_i^t L_i \leq C_t, \forall t = 1, 2, \ldots, T$$

$$a_i^t \in \{0, 1\}, \forall i = 1, 2, \ldots, n, \forall t = 1, 2, \ldots, T$$

$$L_i \in (0, L_{MAX}], \forall i = 1, 2, \ldots, n$$

- Our goal is to maximize server's throughput.
- The first constraint restricts that the server does not crash during the game.
- The second constraint specifies that clients' actions are binary.
- The third constraint states that the maximum traffic load a client can have is below a threshold L[max].
- It is a NP-hard problem, seeking any optimal solution would be difficult.

Outlines

PART 1:
Client-Server Problem with Dynamic Server Capacity

PART 2:
A Regret-Based Procedure and Success-History based Parameter Adaptation Scheme

PART 3:
Simulation Results

A Regret-based Learning Procedure

- We adopted a discrete regret-based procedure to regulate clients' behaviors.
- Clients applying this procedure will play strategies which are more profitable according to their history.
- Clients can learn their environment gradually based only on their own information.
- However, it only works good in a static environment…
- To accommodate the dynamic aspect in our system, we proposed a success-history based parameter adaptation scheme.

Effect of Server Crash Cost

- We leverage server crash cost to help to regulate clients' behaviors.
- The general idea is high server crash cost will cause clients to pay more when server crashes, hence suppress their intentions to send traffic, and vice versa.
- More precisely, we modify a parameter called Crash Cost Factor (CCF) - α to control server crash cost. It is proportional to a client's amount of traffic load.
- For client i:

$$c_i = \alpha L_i$$

note that different clients have different crash cost, which are proportional to the amount of traffic they hold.

Effect of Server Crash Cost

- We leverage server crash cost to help to regulate clients' behaviors.
- The general idea is high server crash cost will cause clients to pay more when server crashes, hence suppress their intentions to send traffic, and vice versa.
- More precisely, we modify a parameter called Crash Cost Factor (CCF) to control server crash cost. It is proportional to a client's amount of traffic load ($c_i = \alpha L_i$).
- While clients applying this scheme, they keep monitoring the amount traffic they successfully route to the server (amount of effective traffic).
- Whenever an increment occurs, clients record the corresponding CCF value (α) as a successful value.

Memory Structure

Index	1	2	...	$H-1$	H
S_{CCF}	$S_{CCF,1}$	$S_{CCF,2}$	...	$S_{CCF,H-1}$	$S_{CCF,H}$
S_{wt}	$S_{wt,1}$	$S_{wt,2}$	...	$S_{wt,H-1}$	$S_{wt,H}$

- The memory structure is shown above. The first row stores CCF values, the second row stores the increment of effective traffic correspondingly.
- An index k decides the position in the memory to update. k is increased whenever a new element is inserted. When k > H, k is reset to 1.

Algorithm Procedure

- Clients check their effective amount of traffic periodically;
- Whenever there is an increase, they record the CCF values and the corresponding increment, which will be used to generate new CCF values.
- When clients need to update their CCF values, with probability p (adaptive probability):
  - If the memory is not full, clients choose a CCF value from a predefined set randomly.
  - If the memory is full, clients generate a new CCF using a weighted mean of all values in the memory.
- With probability 1-p, clients explore new CCF values.

Algorithm Features

- Clients use the amount of effective traffic as a metric to update parameters, which helps to optimize server's throughput, and also preserves the trend of changes of CCF values due to the varying server capacities.
- Clients always have chances to explore new CCF values, which helps them to adapt to new server capacity changes.
- Clients use weighted mean to generate new CCF values, so more profitable values in the memory will have larger impact to the new CCF values, which facilitates the algorithm's convergence rate.

Outlines

PART 1:
Client-Server Problem with Dynamic Server Capacity

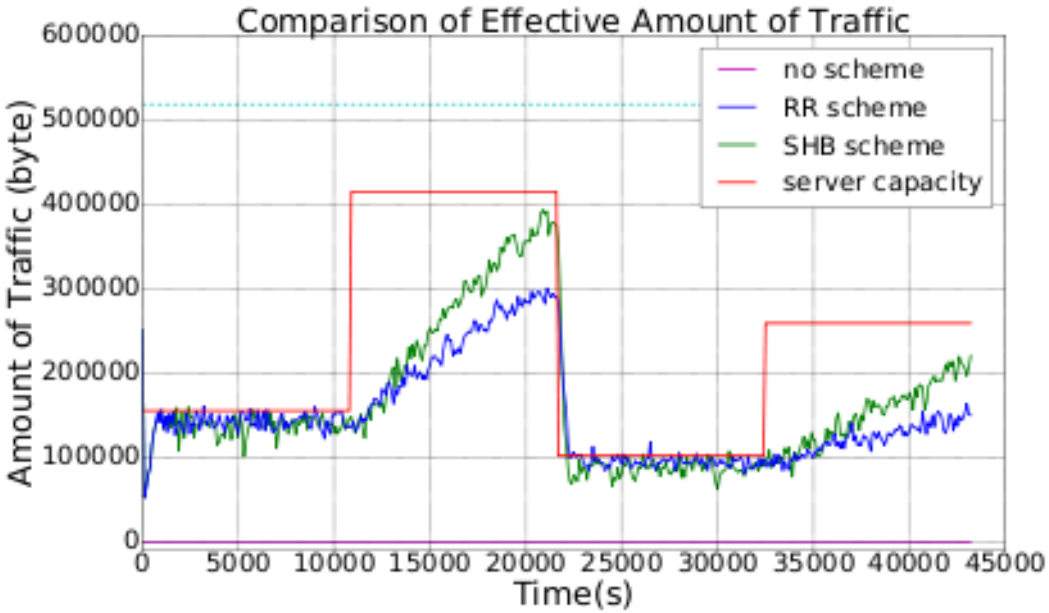
PART 2:
A Regret-Based Procedure and Success-History based Parameter Adaptation Scheme

PART 3:
Simulation Results

Simulation Settings

Parameter	Value
Number of clients	500
Simulation length	12 hours
Stage game length (one time slot)	1 second
Clients message rate	1 msgs/sec
Maximum traffic load	$256 * 8$ bytes
Server capacity variation period	3 hours
Crash cost factor options	$1, 5, 10, \ldots, 100$
Memory size	20
Statistic period	5 minutes
Adaptive probability	0.9
Normal variance	3

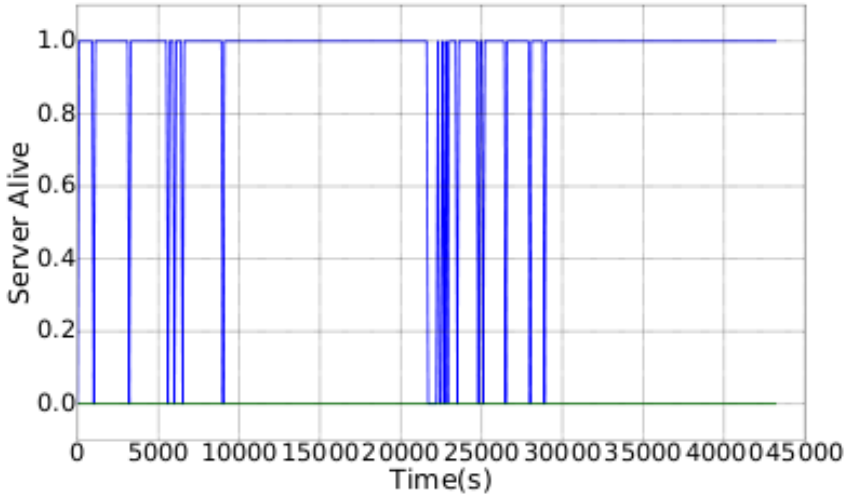
Effective Amount of Traffic



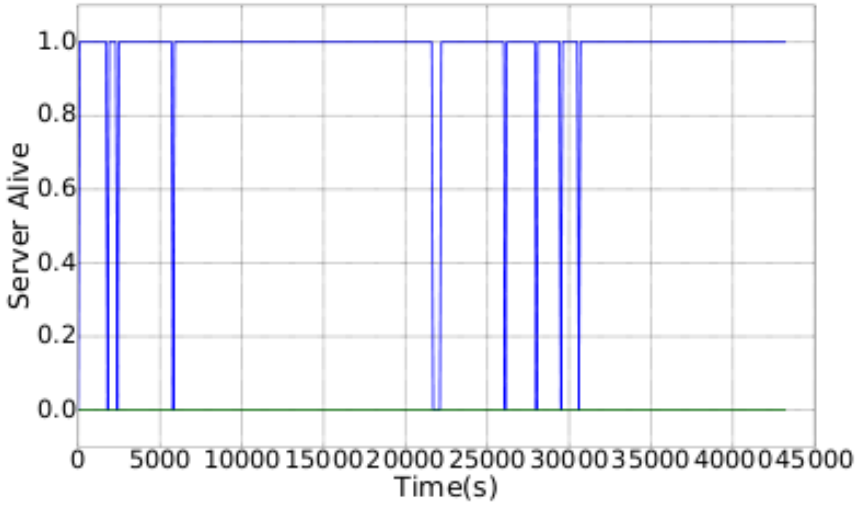
If there is no regulation on clients' behaviors, they will send traffic all the time. The actual amount of traffic from them (the top line) are always greater than the server capacity, resulting in a all-0 effective amount of traffic (the bottom line).

Server Crash Percentage

46% less crashes



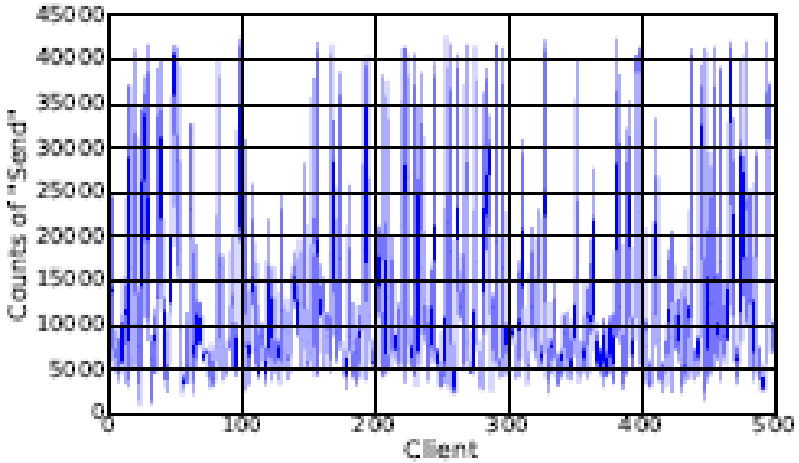
(a) Server alive time using the regret-based scheme.



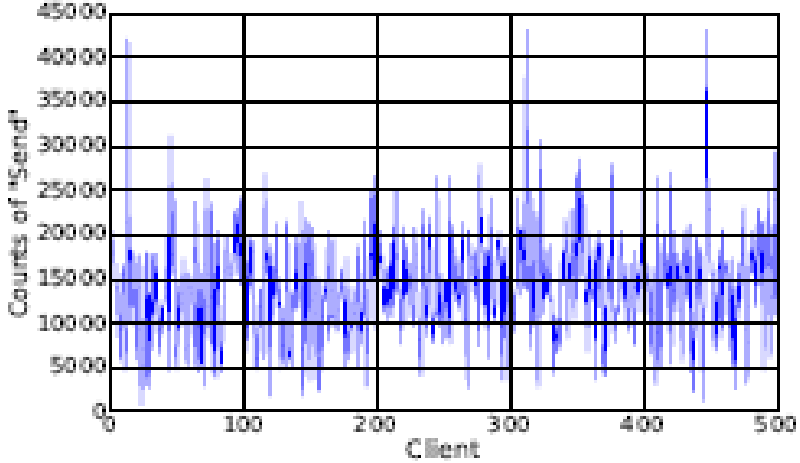
(b) Server alive time using the success-history based scheme.

Counts of “Send” for Every Client

45.5% less variations



(a) Counts of “*Send*” for each client using the regret-based scheme.



(b) Counts of “*Send*” for each client using the success-history based scheme.

Statistics Table

Scheme	Crash Probability Period #				
	1	2	3	4	All
SHB	0.0446	0.0	0.0815	0.0	0.0315
RR	0.0643	0.0	0.1693	0.0	0.0584

Scheme	Effective Traffic Avg. ($\times 10^5$) Period #				
	1	2	3	4	All
SHB	1.295	2.700	0.795	1.495	1.569
RR	1.285	2.273	0.778	1.225	1.393

- With the proposed SHB scheme:
  - Clients can adapt to server capacity changes faster, which means they can utilize the server's resources more efficiently;
  - The server has higher throughput and crashes less;
  - It helps to promote a more equal user experience.

Scheme	"*Send*" Count Std.
SHB	6326.699
RR	11599.990

Conclusions

- In this work, we analyze a real world performance bottleneck in the RHIC control system using game theory approach;
- We model it as a repeated game, formulate it as an integer programming problem, and point out its difficulty;
- We provide a basic solution by adopting a regret-based procedure, and propose a success-history based parameter adaptation scheme to better accommodate the dynamic server capacity scenario in our system;
- Simulation results show that both schemes can significantly improve system performance. Moreover, compared with the regret-based scheme the proposed success-history based scheme can result in a notably higher server throughput and lower crash probability.

Q & A

Thank You!