Efficiently Scheduling Cloud Resources for Peak or Urgent Events in Telecom System

Chia-Chun Shih
Billing Information Laboratory
Chunghwa Telecom Laboratories
Taipei, Taiwan
ccshih@cht.com.tw

Chao-Wen Huang
Billing Information Laboratory
Chunghwa Telecom Laboratories
Taipei, Taiwan
huangcw@cht.com.tw

***Abstract*—Call Detail Record or Charging Data Record (CDR) processing is a computation-intensive task, which is traditionally handled by costly high-performance machines. If extra computing resources are not allocated beforehand, it is fragile to handle peak or urgent events. Cloud computing, which offers elastic computing resources, can cure this problem. In this paper, we investigated several Cloud scheduling strategies for minimizing cost in peak and/or urgent CDR processing scenarios while preserving designated Service Level Agreements (SLAs). To be realistic, we assumed that incoming event is unknown to scheduler until it arrives. Results show that a well-performing strategy in normal scenario could be the worst strategy in peak & urgent scenario. Scheduling strategies should be carefully chosen to minimize cost when handling unexpected events in Cloud.**

Keywords—Cloud, CDR processing, Peak event, Urgent event, Event scheduling

I. INTRODUCTION

Call Detail Record or Charging Data Record (CDR) processing is an important function in telecom BSS (Business Support Systems) and involves tasks of collection, conversion, and delivering of CDRs to other BSS (e.g. billing). Millions, if not billions, of CDRs are generated daily, so it requires heavy computing power to process CDRs in telecom billing mediation system.

Traditional telecom systems use costly high-performance machines for this task. It is also preferable to allocate extra computing resources for handling unexpected peak events. However, extra resources incur extra cost. Moreover, re-configuration of computing resources may require human intervention, which also takes cost.

Cloud computing, which offers elastic computing resources, can be adopted to launch extra machines for consuming peak or urgent events. Recently, more and more telecom systems adopt Cloud Computing to handle computation-intensive tasks [4]. Adoption of Cloud Computing can help telecom companies save significant cost on hardware and maintenance.

In this paper, we investigated several Cloud scheduling strategies for minimizing cost in peak and/or urgent CDR processing scenarios while preserving designated SLAs. To be realistic, we assumed that incoming event is unknown to scheduler until it happens, so schedule must be dynamically adjusted to efficiently handle new events. This adjustment may be severe if the incoming event is peak and/or urgent. For simplicity, we also assumed that all virtual machines are homogeneous. Our experimental results show that a well-performing strategy in normal scenario could be the worst strategy in peak and/or urgent scenario.

The remainder of this paper is organized as follows. Section 2 presents related works. Section 3 states the problem. Section 4 describes scheduling strategies. Section 5 shows experimental result. Finally, Section 6 presents our concluding remarks, as well as future directions.

II. RELATED WORKS

Although telecom operators are usually cloud service providers, they could also be cloud service adopters. A survey of MVNO (Mobile Virtual Network Operator) experts shows that most telecom BSS functions could be deployed on the Cloud [13]. Xu et al. further proposes a framework of deploying telecom services in cloud environment [12]. The possibility of cloud-powered BSS has already gained attention in industry [2].

CDR processing is an important function in BSS and involves tasks of collection, conversion, and delivering of CDRs to other BSS systems (e.g. billing). Millions, if not billions, of CDRs are generated daily, so performance and reliability are always critical issues of CDR processing. Siangzhee and Achalakul attempted to improve performance and reliability of CDR processing by a specific job scheduler with dynamic priority assignment [11]. Although this work could be naturally applied to cloud, they didn't try to adopt different algorithms considering characteristics of Cloud environment. Real-time processing of CDRs is important in certain applications, for example, SIM card rental services. Some prior works focus on near real-time processing of CDRs [3] [7]. Compared to these prior works, this work allows SLA-based CDR processing, which can also cover near real-time processing.

Scheduling has been an active research topic for a long time [10]. Resource limitation is a common assumption in past scheduling research [1]. However, this limitation can be relaxed in Cloud environment. Cloud can be adopted to handle peak workload [9]. To the best of our knowledge, few scheduling researches have tried to head-to-head tackle peak

workload in Cloud environment. Li and Lo attempted to average peak load in Cloud for energy saving issues [8]. However, this work doesn't impose job deadline requirements. Most researches adopt "workload forecasting" to handle oscillate load [5]. Although we don't adopt workload forecasting to handle peak workload, it can be a complement of our paper. Scheduling under uncertainty can complicate scheduling problem in that no algorithm can be optimal under high uncertainty [6]. A good scheduling strategy, accompanies with good resource allocation method [15], can effectively save energy.

III. PROBLEM STATEMENT

This section formulates CDR processing problem, as well as notations used in subsequent sections.

In this paper, CDR processing tasks are called events. For simplicity, capacity requirement of each event is measured in units. Also, continuous timeline is divided into discrete timeslots. Each event has SLA (Service Level Agreement), which is the required number of timeslots for processing this event. SLA is a hard constraint in this paper.

An event is represented as a tuple (*arrival timeslot, capacity requirement in number of units, SLA in number of timeslots*). For example, a tuple (3, 50, 10) means this event arrives at timeslot 3, requires 50 units capacity, and needs to be done in 10 timeslots (before timeslot 13).

This paper assumes events are separable. On the other words, one incoming event can be divided into several independent events. This assumption is reasonable because CDR processing is record-based. One CDR processing task usually involves thousands to millions of records, and in most cases, each record can be processed independently.

This paper assumes events are unpredictable. Incoming event is unknown to scheduler until it happens.

Virtual machines are assumed to be homogeneous. This assumption simplifies the problem in that fitness between events and virtual machines can be discarded.

Like popular Amazon Cloud Service, virtual machines are priced in hourly basis. However, SLAs of urgent CDR processing tasks needs to measured in minutes. To compromise, we adopted five-minute as a timeslot. Therefore, one virtual machine launched in current timeslot is available in subsequent 12 timeslots. This number 12 is defined as a variable *#machine_lifetime.*

All virtual machines are equally priced. Renewed virtual machine has no discount.

The overall objective is to minimize cost under above problem settings. On the other words, the objective is to minimize number of rentals (machine-hour) of virtual machines.

IV. SCHEDULING STRATEGIES

Three categories of scheduling strategies are attempted: *Average Strategy*, *Capacity Reservation Strategy*, and *Pooling Strategy*. Next subsections introduce these strategies.

A. Average Strategy

For each event, *Average Strategy* tries to equally distribute workload to available timeslots. For example, assume current timeslot is 3, and it comes an event A(3, 50, 10). It means event A requires fifty units capacity in ten timeslots, which begin from timeslot 3, and suppose to be done no later than timeslot 12. Following this strategy, scheduler will assign 5 units to each timeslot from timeslot 3 to timeslot 12.

The philosophy behind Average Strategy is simple: avoiding peak workload. Peak workload forces scheduler to launch extra machines and may result in over-provisioned capacity in the future.

At each timeslot, scheduler allocates enough computing capacity for consuming works assigned to this timeslot. Sometimes, it remains residual capacity in current timeslot after consumption of assigned works. The residual capacity will be assigned to other unfinished events in *urgency order*.

The *urgency order* is determined by *urgency score* in decreasing order. The *urgency score* of an event is measured by the following formula:

- If number of remaining timeslots of this event is equal to 1, then *urgency score* = Integer.MAX
- Else *urgency score* = (number of remaining units / number of remaining timeslots)

If two events have equal *urgency score*, the event with more remaining unit priors.

B. Capacity Reservation Strategy (I)

At each timeslot, scheduler adopting *Capacity Reservation Strategy(I)* tries to reserve sufficient capacity for consuming unfinished events. For example, at current timeslot, assume unfinished events totally require 300 machine-timeslots capacity, and launched virtual machines can only provide 250 machine-timeslots capacity in the future. At this timeslot, in order to fill the gap, scheduler will launch extra machines that can totally provide at least 50 machine-timeslots in the future. *Capacity Reservation Strategy (I)* always reserves sufficient capacity for known events. If peak/urgent events come, the reserved capacity could be reallocated to peak /urgent events to reduce impact.

Specifically, at each timeslot, scheduler tries to launch extra machines if the following two conditions are met:

- Current capacity is insufficient to consume all urgent events, whose deadline is one timeslot away from now
- Total capacity is insufficient to consumer all events

The first condition guarantees that all urgent events be finished before deadline. The second condition reserves sufficient capacity for all known events. Please note that condition 2 does *not* guarantee that all known events are well-allocated. Instead, it only ensures that *total* provided capacity is larger than *total* requested capacity. Resources may be over-provisioned in some timeslots and under-provisioned in another timeslots. That is, only condition 2 does *not* guarantee sufficient resources in *all* timeslots.

C. Capacity Reservation Strategy (II)

Capacity Reservation Strategy (I) can be modified to guarantee sufficient resources for *some* events in only *limited* number of subsequent timeslots. This modified strategy is called *Capacity Reservation Strategy (II)*.

Capacity Reservation Strategy (II) guarantees sufficient resources for events whose deadline is in subsequent (*#machine_lifetime*-1) timeslots. A virtual machine is launched at one timeslot can only contribute capacity in subsequent (*#machine_lifetime*-1) timeslots. At current timeslot, it is possible to guarantee resources availability of events that due in subsequent (*#machine_lifetime*-1) timeslots by launching extra virtual machines.

Furthermore, Unlike *Capacity Reservation Strategy (I)*, which aggregative calculates resource requirement, *Capacity Reservation Strategy (II)* calculates resource requirement timeslot-by-timeslot, therefore finer granularity is provided.

D. Pooling Strategy

Pooling Strategy adopts the following rules: 1) Do not launch extra machine unless this machine will be fully occupied; 2) Do not begin to consume events unless postpone may incur higher cost. Launch extra machines if capacity is insufficient for consuming these events.

The following example illustrates the first rule. At timeslot 7, there are four unfinished events: A(4, 20, 8), B(3, 5, 7), C(1, 30, 12), and D(4, 10, 8). Assume that one virtual machine can offer 10 units capacity in a timeslot and *#machine_lifetime* be equal to 5, so one virtual machine can offer 50 units capacity in each rental. Based on these assumptions, how many new virtual machines will *Pooling Strategy* launch at timeslot 7? Fig. 1 shows capacity requirements of timeslot 7-13 if all events are pooled in each timeslot. For example, at timeslot 11, event A, C, and D are pooled, and totally require 60 units (20+30+10) capacity. Note that workload assigned at timeslot 11 can be assigned to timeslots before 11 without violating SLA. So, the 60-unit capacity requirement can be spread to timeslot 7-11, which fully occupies one machine (50 units capacity) launch at timeslot 7. So *Pooling Strategy* will launch one extra machine at timeslot 7. Also note that although this new machine is launched for consuming event A, C, and D, it is not dedicated to consume these three events. If other urgent events come, the machine can be reallocated. But whether urgent events come or not, this machine is guaranteed to be fully occupied.

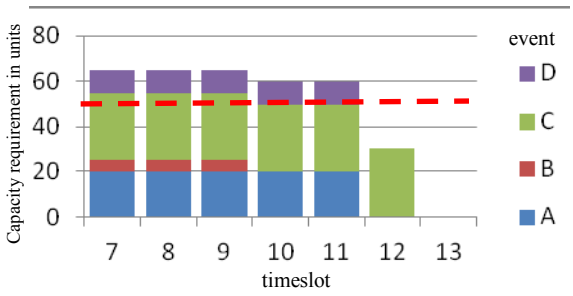

Fig. 1. Capacity requirements of timeslot 7-13 if all events are pooled in each timeslot

The following example illustrates the second rule. At timeslot 3, it comes an event (3, 40, 6), which requires 40 units capacity from timeslot 3 to timeslot 8. Assume one machine can offer 10 units capacity in each timeslot. As shown in Table I, the minimum number of concurrent machines required by this event depends on when consumption starts. For the same events, the later consumption starts, the more possibility that minimum number of concurrent machines increases.

TABLE I. MINIMUM NUMBER OF REQUIRED CONCURRENT MACHINES

Start consumption from timeslot #	Remaining timeslot #	Required Capacity per timeslot	Minimum # of Required Concurrent Machines
3	6	6.66 (40/6)	1
4	5	8.0 (40/5)	1
5	4	10.0 (40/4)	1
6	3	13.33 (40/3)	2
7	2	20.0 (40/2)	2
8	1	40.0 (40/1)	4

This event requires *one* concurrent machine when starting consumption from timeslot 5, but requires at least *two* concurrent machines when starting consumption after timeslot 5. Therefore, *Pooling Strategy* will start consuming this event no latter than timeslot 5, or it may incur risks of launching unnecessary machines.

V. EXPERIEMRNTAL RESULTS

To explore adaptability of these strategies for peak and/or urgent workload, two CDR processing scenarios are tested:

- Peak Scenario: The connection between BSS mediation components and networked equipments is temporarily broken. CDRs are accumulated in networked equipments, and BSS mediation components can not receive CDRs. Once the connection is repaired, accumulated CDRs become peak events. Luckily, it still leaves enough time for handling these peak events. Red bars in Figure 4 shows an example of peak scenario, where events from timeslot 4 to timeslot 9 are accumulated and postponed to timeslot 10.
- Peak and Urgent Scenario: This scenario is identical to peak scenario except that time is not enough. Therefore, many events require much tougher SLAs to catch up with deadline. Red bars in Figure 6 shows an example of peak and urgent scenario, in which events from timeslot 16 to timeslot 21 are accumulated and postponed to timeslot 22. Because timeslot 24 is ultimate deadline, these peak events become urgent.

This experiment only focuses on 24 timeslots. Fewer timeslots can help differentiate these two scenarios from normal scenario.

This experiment assumes each machine offers 10 units capacity in each timeslot. And *#machine_lifetime* is equal to 12. One rental of a virtual machine can totally offer 120 units capacity.

This experiment assumes seven levels of SLAs: 1 timeslot (5 minutes), 3 timeslots (15 minutes), 6 timeslots (30 minutes), 9 timeslots (45 minutes), 12 timeslots (60 minutes), 18 timeslots (90 minutes), and 24 timeslots (120 minutes). Each event randomly picks a SLA.

Capacity requirement of each event is a random integer between 1 unit and 119 units.

At each timeslot, 20 incoming events are randomly generated. Notably, all events are forced to be consumed completely no latter than end of timeline (timeslot 24). Therefore, all generated events due at a timeslot later than timeslot 24 are adjusted to timeslot 24.

We repeated 100 times of experiment with randomly generated events. Each experiment tests performance of the four strategies in three scenarios (normal, peak, peak & urgent). Performance is measured in number of required virtual machine rentals. Following subsections demonstrate experimental results of normal scenario, peak scenario and peak & urgent scenario.

We implemented a custom simulator in Java to test the four scheduling strategies. More implementation details are available in a technical report [14].

A. *Normal Scenario*

Normal scenario excludes connection broken in peak scenario, and acts as a control group. Fig. 2, which is an example of this scenario, shows aggregated capacity requirement of incoming events at each timeslots (left most red bar in each timeslot), as well as how the four strategies consume events at each timeslots.

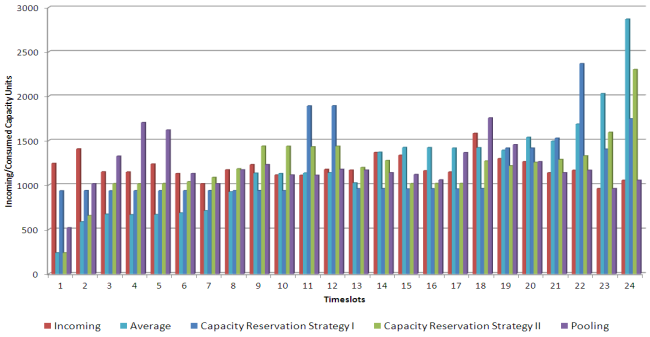

Fig. 2. An example of normal scenario: aggregated incoming capacity requirements at each timeslot (left most red bar in each timeslot), as well as how the four strategies consume events at each timeslots.

Table II summarizes experimental results. Of 100 experiments, *Pooling Strategy* performs best, with 41 times of best performance. *Capacity Reservation Strategy I & II* performs well too. *Average Strategy (Average)*, without any best performance, is the worst strategy of them.

TABLE II. NUMBER OF TIMES EACH STRATEGY PERFORMS BEST IN NORMAL SCENARIO.

Strategy	# of times this strategy performs best
Average Strategy	0
Capacity Reservation Strategy I	20
Capacity Reservation Strategy II	39
Pooling Strategy	**41**

Table III shows performance statistics of the four strategies in number of virtual machine rentals. *Capacity Reservation Strategy (II)* and *Pooling Strategy* are almost equally best in median but *Pooling Strategy* performs more stable.

TABLE III. PERFORMANCE STATISTICS OF EACH STRATEGY IN NORMAL SCENARIO (IN NUMBER OF VIRTUAL MACHINE RENTALS).

	median	**min**	**max**	**Q1**	**Q3**
Average	439	362	492	425	453
Capacity Reservation Strategy I	405	271	483	373	426
Capacity Reservation Strategy II	**383**	**264**	**464**	**362**	404
Pooling	384	329	439	365	**395**

Fig. 3 visualizes the statistics as a box plot.

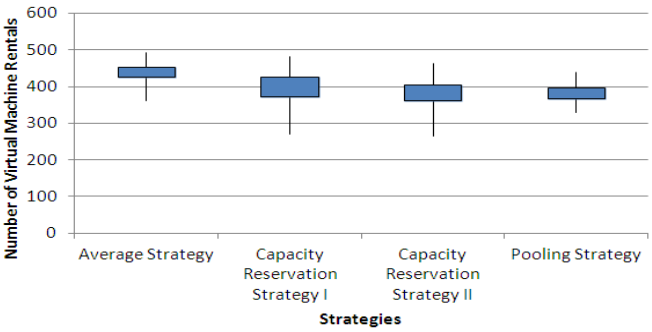

Fig. 3: A box plot visualizes Table III.

B. *Peak Scenario*

In this scenario, events from timeslot 4 to timeslot 9 are postponed to timeslot 10. Fig. 4, which is an example of this scenario, shows aggregated capacity requirements of incoming events at each timeslots (left most red bar in each timeslot), as well as how the four strategies consume events at each timeslots.

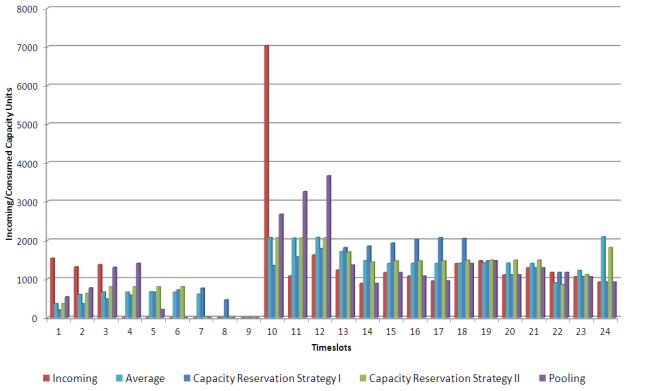

Fig. 4. An example of peak scenario: aggregated incoming capacity requirements at each timeslot (left most red bar in each timeslot), as well as

how the four strategies consume events at each timeslots. Incoming Events from timeslot 4 to timeslot 9 are postponed to timeslot 10.

Table IV summarizes experimental results. Of 100 experiments, *Capacity Reservation Strategy I* dominates with 77 times of best performance. *Pooling Strategy*, which performs well in normal scenario, becomes the worst strategy. *Average Strategy*, although still performs poor, is relatively better than it performs in normal scenario.

TABLE IV. NUMBER OF TIMES EACH STRATEGY PERFORMS BEST IN PEAK SCENARIO.

Strategy	# of times this strategy performs best
Average Strategy	5
Capacity Reservation Strategy I	**77**
Capacity Reservation Strategy II	18
Pooling Strategy	0

Table V shows performance statistics of the four strategies in number of virtual machine rentals. *Capacity Reservation Strategy (I)* dominates other strategies in all statistics, and *Pooling Strategy* falls behind in all statistics.

TABLE V. PERFORMANCE STATISTICS OF EACH STRATEGY IN PEAK SCENARIO (IN NUMBER OF VIRTUAL MACHINE RENTALS).

	median	min	max	Q1	Q3
Average Strategy	463	392	544	438	488
Capacity Reservation Strategy I	**384**	**291**	**479**	**327**	**426**
Capacity Reservation Strategy II	424	312	550	392	461
Pooling Strategy	490	428	581	466	505

Fig. 5 visualizes the statistics as a box plot.

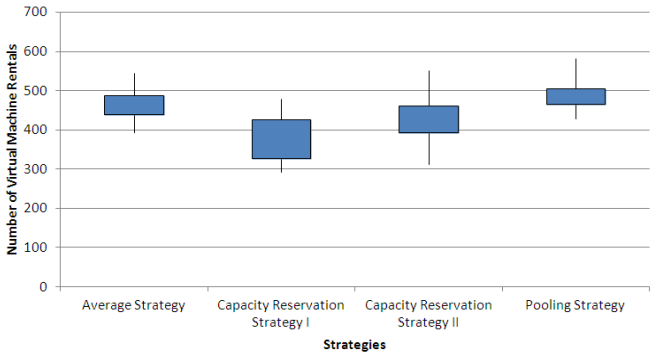

Fig.5. A box plot visualizes Table V.

C. Peak & Urgent Scenario

In this scenario, events from timeslot 16 to timeslot 21 are postponed to timeslot 22. Moreover, these peak events are forced to be consumed completely no latter than timeslot 24. Fig. 6, which is an example of this scenario, shows aggregated capacity requirements of incoming events at each timeslots (left most red bar in each timeslot), as well as how the four strategies consume events at each timeslots.

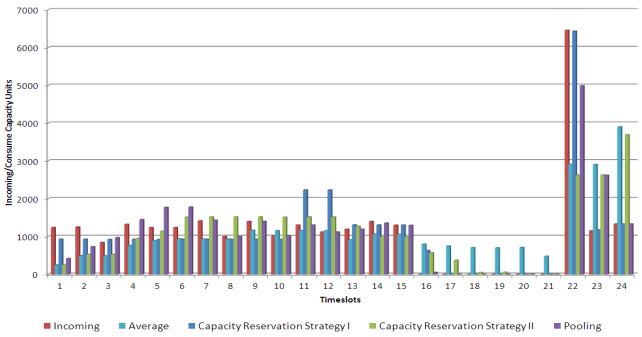

Fig. 6. An example of peak & urgent scenario: aggregated incoming capacity requirements at each timeslot (left most red bar in each timeslot), as well as how the four strategies consume events at each timeslots. Incoming Events from timeslot 16 to timeslot 21 are postponed to timeslot 22.

Table VI summarizes experimental results. *Average Strategy* is absolutely the best, wins 98 best performance out of 100 experiments. *Capacity Reservation Strategy II* and *Pooling Strategy*, which are outperforming in other scenarios, becomes bad strategy in this scenario.

TABLE VI. NUMBER OF TIMES EACH STRATEGY PERFORMS BEST IN PEAK & URGENT SCENARIO.

Strategy	# of times this strategy performs best
Average Strategy	**98**
Capacity Reservation Strategy I	0
Capacity Reservation Strategy II	2
Pooling Strategy	0

Table VII shows performance statistics of the four strategies in number of virtual machine rentals. *Average Strategy* dominates other strategies in all statistics.

TABLE VII. PERFORMANCE STATISTICS OF EACH STRATEGY IN PEAK & URGENT SCENARIO (IN NUMBER OF VIRTUAL MACHINE RENTALS).

	median	min	max	Q1	Q3
Average Strategy	**480**	**418**	**566**	**455**	**501**
Capacity Reservation Strategy I	521	441	612	490	545
Capacity Reservation Strategy II	765	490	966	714	841
Pooling Strategy	809	685	895	783	836

Fig. 7 visualizes the statistics as a box plot.

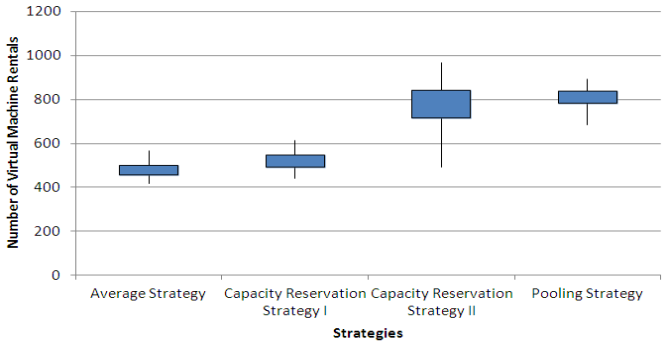

Fig.7. A box plot visualizes Table VII.

D. Discussions

Table VIII summarizes relative performance of the four strategies in the three scenarios. The best, i.e., least expensive strategy is set to 0.00% in each scenario.

In normal scenario, the largest difference is only up to 14.62%. In peak scenario, the largest difference grows to 27.60%. In the toughest peak & urgent scenario, the difference grows further up to 68.54%. It shows that strategy selection especially matters in tougher scenario.

No strategy can outperform others in every scenario. To minimize cost, scheduler could alter strategy if peak and/or urgent scenario is met. If the alternation is not available, *Capacity Reservation Strategy I*, which is the most stable strategy, is suggested.

Capacity Reservation Strategy II and *Pooling Strategy*, although perform well in normal scenario, could be trapped in peak & urgent scenario.

TABLE VII. RELATIVE PERFORMANCE OF EACH STRATEGY IN EACH SCENARIO.

	normal scenario	**peak scenario**	**peak & urgent scenario**
Average Strategy	-14.62%	-20.57%	**0.00%**
Capacity Reservation Strategy I	-5.74%	**0.00%**	-8.54%
Capacity Reservation Strategy II	**0.00%**	-10.42%	-59.38%
Pooling Strategy	-0.26%	-27.60%	-68.54%

VI. CONCLUDING REMARKS AND FUTURE DIRECTIONS

This paper studies four strategies for scheduling CDR processing events with hard SLAs in Cloud environment to minimize cost. Especially, this paper examines adaptability of these strategies in peak scenario and peak & urgent scenario. Experimental results show that no strategy performs best in every scenario. The best performing strategy in normal scenario could perform very poorly in peak & urgent scenario. But in average, *Capacity Reservation Strategy I* performs most stably. To minimize cost, scheduler could alter strategy if peak and/or urgent scenario is met.

In the future, we plan to develop solutions for more complicated and more realistic scenarios for Cloud-Computing-powered CDR processing. In this paper, we assumed machines are homogeneous. But in real world, Cloud vendors usually provide several options of machines. Heterogeneous machines complicate this problem in that fitness between events and machines should be considered. Moreover, we can also consider situations where in-place upgrade is possible. That is, if capacity is insufficient, we can choose to upgrade machines instead of launching new machines.

Implementing a mixed strategy is also of our future directions. Since no strategy can fit each scenario, it will be beneficial if scheduler can alter strategy based on environmental change. It is also interesting to adopt evolutionary approach, as well as workload forecasting, to enhance the scheduler.

ACKNOWLEDGMENT

Thanks C.W. Cheng, J.Y. Jeng, and other colleagues for their support.

REFERENCES

[1] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: notation, classification, models and methods," *European Journal of Operational Research*, vol. 112, pp. 3-41, 1999.

[2] Y. Chunlei, "Exploring the cloudified BSS," *Huawei Communicate*, vol. 61, pp. 31-33, 2011. (available at: http://www.huawei.com/uk/static/hw-094158.pdf)

[3] M. Cochinwala, and E. Panagos, "Near real-time call detail record ETL flows," *Enabling Real-Time Business Intelligence, Lecture Notes in Business Information Processing*, vol. 41, pp. 142-154, 2010.

[4] J. Gabrielsson, O. Hubertsson, I. Más, and R. Skog, "Cloud computing in the telecommunications," *Ericsson Review*, pp. 29-33, vol. 1/2010, 2010. (available at: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2010/cloudcomputing.pdf)

[5] S.K. Garg, S.K., Gopalaiyengar, and R. Buyya, "SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud," *Proceedings of 11th International conference on Algorithms and Architectures for Parallel Processing*, vol. 1, pp. 371-384, 2011.

[6] W. Herroelen, and R. Leus, "Project scheduling under uncertainty: survey and research potentials," *European Journal of Operational Research*, vol. 165, no. 2, pp. 289-306, 2005.

[7] D. Kar, P. Misra, P. Bhattacharjee, and A. Mukherjee, "Real-time telecom revenue assurance," *Proceedings of the Seventh International Conference on Digital Telecommunications*, pp.130-135, 2012.

[8] X. Li, and J.-chung Lo, "Pricing and peak aware scheduling algorithm for cloud computing," *Proceedings of IEEE PES Innovative Smart Grid Technologies Europe Conference*, pp.1-7, 2012.

[9] M. Mattess, C. Vecchiola, and R. Buyya, "Managing peak loads by leasing cloud infrastructure services from a spot market" *Proceedings of IEEE 12th International Conference on High Performance Computing and Communications*, pp.180-188, 2010.

[10] S.S. Panwalkar, and W. Iskander, "A survey of scheduling rules" *Operations Research*, vol. 25, no. 1, pp. 45-61, 1977.

[11] N. Siangzhee, and T. Achalakul, "Performance and reliability improvement of the call detail record processing system: A case study from a telecommunication enterprise," *TENCON 2009 - 2009 IEEE Region 10 Conference*, pp.1-5, 2009.

[12] H. Xu, M. Song, J. Peng, and Q. Yu, "Research on telecom service deployment in cloud environments," *Proceedings of the 5th International Conference on Pervasive Computing and Applications (ICPCA)*, pp.189-194, 2010.

[13] R. Yrjo, and D. Rushil, "Cloud computing in mobile networks – case MVNO," *Proceedings of the 15th International Conference on Intelligence in Next Generation Networks (ICIN)*, pp.253-258, 2011.

[14] C. Shih, "Technical Report -- Scheduling Algorithms for Peak or Urgent Events in Telecom System," available at: http://sync.hamicloud.net/_oops/chiachunshih/hzt, 2013.

[15] L. Guan, Y. Wang, and Y. Li, "A dynamic resource allocation method in IaaS based on deadline time," *Proceedings of the 14th Asia-Pacific Network Operations and Management Symposium (APNOMSS)*, pp. 1-4, 2012.