Corrigendum to “Overload Control for Scaling WeChat Microservices”

Hao Zhou
Tencent Inc.
China
harveyzhou@tencent.com

Yong Wang
Tencent Inc.
China
darwinwang@tencent.com

Rui Gu
Columbia University
New York, USA
ruigu@cs.columbia.edu

Ming Chen
Tencent Inc.
China
mingchen@tencent.com

Xiaobin She
Tencent Inc.
China
stevenshe@tencent.com

Beng Chin Ooi
National University of Singapore
Singapore
ooibc@comp.nus.edu.sg

Qian Lin
National University of Singapore
Singapore
linqian@comp.nus.edu.sg

Sifan Liu
Tencent Inc.
China
stephenliu@tencent.com

Junfeng Yang
Columbia University
New York, USA
junfeng@cs.columbia.edu

This is in response to recent feedback from some readers, which requires some clarifications regarding our adaptive admission control algorithm published in [1]. For a better understanding, we want to amend the algorithm to make its detailed strategies easier to interpret and correct some minor errors found in the original paper.

Algorithm 1 summaries the procedures of adaptive admission control in DAGOR. Each server maintains an array of counters, each of which corresponds to a compound admission level indexed by $(\mathcal{B}, \mathcal{U})$. Each counter counts the number of the incoming requests associated with the corresponding business and user priorities. DAGOR periodically adjusts the admission level of load shedding as well as resets the counters, and the period is consistent to the window size for overload detection. For each period, if an overload is detected, the server sets the expected amount of admitting requests in the next period to be α (in percentage) less than that in the current period; otherwise, the expectation of request amount in the subsequent period is increased by β (in percentage) of the incoming requests in the current period. Given the expected amount of admitting requests, the admission level is calculated with respect to the maximum prefix sum in the histogram near that amount, and is adjusted based on its current state and whether or not an overload is detected. Let $\mathcal{B}^*$ and $\mathcal{U}^*$ be the optimal settings of the admission levels of business priority and user priority respectively. The optimal setting of the compound admission level $(\mathcal{B}^*, \mathcal{U}^*)$ is determined by the constraint such that the sum of counters with

$(\mathcal{B}, \mathcal{U}) \leq (\mathcal{B}^*, \mathcal{U}^*)^1$ is just below (resp. exceeding) the expected amount when overload is indicated (resp. suppressed).

Note that even if the system is not overloaded at some point, some requests may still be discarded due to the current settings of admission levels, especially when the system is undergoing recovery from a previous overload.

REFERENCES

- [1] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. 2018. Overload Control for Scaling WeChat Microservices. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SoCC '18, October 11–13, 2018, Carlsbad, CA, USA

© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6011-1/18/10...\$15.00
<https://doi.org/10.1145/3267809.3267823>

¹For two admission levels $(\mathcal{B}_1, \mathcal{U}_1)$ and $(\mathcal{B}_2, \mathcal{U}_2)$, we have $(\mathcal{B}_1, \mathcal{U}_1) < (\mathcal{B}_2, \mathcal{U}_2)$ if $\mathcal{B}_1 < \mathcal{B}_2$, or $\mathcal{B}_1 = \mathcal{B}_2$ but $\mathcal{U}_1 < \mathcal{U}_2$.

Algorithm 1: Adaptive admission control.**Global:** Range of admission levels of business priority $\mathcal{B}_L \dots \mathcal{B}_H$ **Global:** Range of admission levels of user priority $\mathcal{U}_L \dots \mathcal{U}_H$ **Global:** Incoming request counters $C[\mathcal{B}_L \dots \mathcal{B}_H][\mathcal{U}_L \dots \mathcal{U}_H]$ **Global:** Incoming request counter N **Global:** Admitted request counter N_{adm} **Global:** Compound admission level $(\mathcal{B}^*, \mathcal{U}^*)$ **Procedure** ResetHistogram(): /* at the beginning of period */ $N, N_{\text{adm}} \leftarrow 0$ **foreach** $c \in C$ **do** $c \leftarrow 0$ **Input:** Service request r **Procedure** UpdateHistogram(r): $N \leftarrow N + 1$ $C[r.\mathcal{B}][r.\mathcal{U}] \leftarrow C[r.\mathcal{B}][r.\mathcal{U}] + 1$ **if** $r.\mathcal{B} < \mathcal{B}^*$ **or** $(r.\mathcal{B} = \mathcal{B}^* \text{ and } r.\mathcal{U} \leq \mathcal{U}^*)$ **then** $N_{\text{adm}} \leftarrow N_{\text{adm}} + 1$ **Input:** Boolean flag f_{ol} indicating overload**Procedure** UpdateAdmitLevel(f_{ol}): /* at the end of period */ $N_{\text{prefix}} \leftarrow N_{\text{adm}}$ **if** $f_{\text{ol}} = \text{true}$ **then** $N_{\text{exp}} \leftarrow (1 - \alpha) \cdot N_{\text{adm}}$ **while** $N_{\text{prefix}} > N_{\text{exp}}$ **and** $(\mathcal{B}^*, \mathcal{U}^*) > (\mathcal{B}_L, \mathcal{U}_L)$ **do** $\mathcal{U}^* \leftarrow \mathcal{U}^* - 1$ **if** $\mathcal{U}^* < \mathcal{U}_L$ **then** $\mathcal{B}^* \leftarrow \mathcal{B}^* - 1$ $\mathcal{U}^* \leftarrow \mathcal{U}_H$ $N_{\text{prefix}} \leftarrow N_{\text{prefix}} - C[\mathcal{B}^*][\mathcal{U}^*]$ **else** $N_{\text{exp}} \leftarrow N_{\text{adm}} + \beta \cdot N$ **while** $N_{\text{prefix}} < N_{\text{exp}}$ **and** $(\mathcal{B}^*, \mathcal{U}^*) < (\mathcal{B}_H, \mathcal{U}_H)$ **do** $\mathcal{U}^* \leftarrow \mathcal{U}^* + 1$ **if** $\mathcal{U}^* > \mathcal{U}_H$ **then** $\mathcal{B}^* \leftarrow \mathcal{B}^* + 1$ $\mathcal{U}^* \leftarrow \mathcal{U}_L$ $N_{\text{prefix}} \leftarrow N_{\text{prefix}} + C[\mathcal{B}^*][\mathcal{U}^*]$