


RESEARCH ARTICLE | SEPTEMBER 26 2022

## Calculation of the optimal timeout of executing a SQL query for multi-master replication system


V. Babaylov; G. Timofeeva 




*AIP Conf. Proc.* 2522, 060001 (2022)

<https://doi.org/10.1063/5.0100779>






Lock-in Amplifier



Boxcar Averager



Zurich  
Instruments

Boost Your Optics and  
Photonics Measurements

Find out more

# Calculation of the Optimal Timeout of Executing a SQL Query for Multi-master Replication System

V. Babaylov<sup>1,a)</sup> and G. Timofeeva<sup>1,2,b)</sup>

<sup>1</sup>Ural State University of Railway Transport, 66 Kolmogorov str., 620034 Yekaterinburg, Russian Federation

<sup>2</sup>Ural Federal University named after First President of Russia B.N. Yeltsin, 19 Mir str., 620002 Yekaterinburg, Russian Federation

<sup>a)</sup> [vsbabaylov@gmail.com](mailto:vsbabaylov@gmail.com)

<sup>b)</sup> Corresponding author: [Gtimofeeva@usurt.ru](mailto:Gtimofeeva@usurt.ru)

**Abstract.** Many multi-master replication systems are based on a two-phase commit mechanism. One of its problems is waiting for a response from the database servers by the coordinator, which may not come due to problems on the network or on the database server itself, which leads to delays in the activity of the multi-master replication system. Usually, such problems are solved by introducing a timeout, but if its value is too large, the problem may not be noticed in a timely manner, and too small a timeout will be processed for SQL queries, which will cause an increase in the load on the network due to additional service messages. The paper analyzes the execution time of a SQL query on several servers in order to calculate the optimal timeout for a multi-master replication operation. Based on the results obtained, a module for calculating the optimal query execution timeout for multi-master replication systems has been developed.

## BACKGROUND

Multi-master is a database replication technique that allows data to be saved by a group of servers and updated by any member of the group. A multi-master replication system is responsible for applying the data modifications made by each member to the rest of the group and resolving any conflicts that arise when multiple members modify data at the same time. When organizing this type of replication, distributed transactions are used, which adds a new class of problems - Byzantine conflicts [1].

One such conflict is described by the Two Generals Problem, presented in 1975 by Akkoyunlu, Ekanadham, and Huber [2]. The problem statement is as follows:

Two armies, each led by a different general, are preparing to attack a fortified city. The armies are encamped near the city, each in its own valley. A third valley separates the two hills, and the only way for the two generals to communicate is by sending messengers through the valley. Unfortunately, the valley is occupied by the city's defenders and there's a chance that any given messenger sent through the valley will be captured.

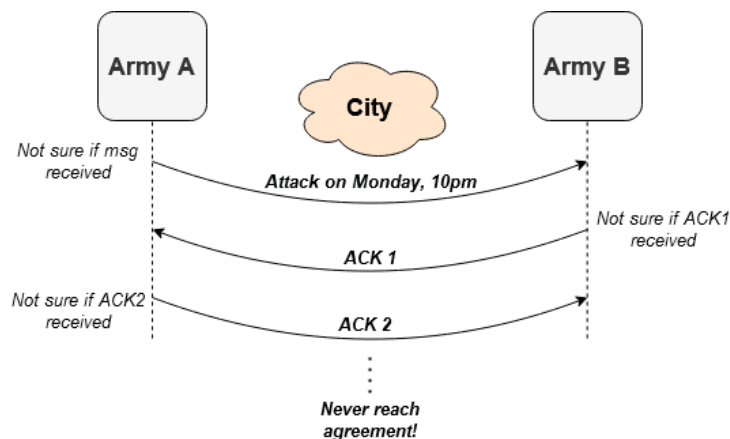
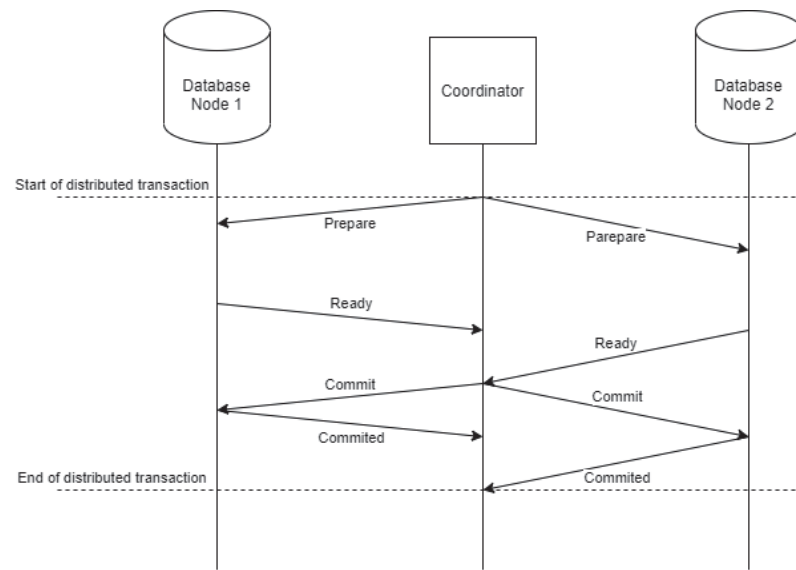


FIGURE 1. Illustration of Two Generals Problem



**FIGURE 2.** General scheme of two-phase commit

While the two generals have agreed that they will attack, they haven't agreed upon a time for attack. It is required that the two generals have their armies attack the city at the same time in order to succeed, lest the lone attacker army will die trying. They must thus communicate with each other to decide on a time to attack and to agree to attack at that time, and each general must know that the other general knows that they have agreed to the attack plan (Figure 1).

Reaching such an agreement in the case of a reliable communication channel is quite simple - one message with the time of the start of the assault and one message confirming consent is enough. However, according to the condition of the problem, the communication channel is not reliable.

At the moment, the solution to this problem is considered impossible, therefore, when developing distributed systems, an approach is used that does not completely eliminate channel unreliability, but reduce it to an acceptable level.

One method for maximizing the reliability of message transmission in a distributed transaction is the two-phase commit mechanism (Figure 2). Within the framework of this mechanism, among the participating servers, there must be a node responsible for coordinating the conduct of a distributed transaction. Such a node is called a coordinator. This mechanism includes two stages - preparation and commit [3].

However, this method is not ideal due to the unreliability of equipment and communication channels. Table 1 below summarizes the main issues encountered when executing a distributed transaction using two-phase commit [4].

One of the methods for tracking such problems is to introduce a timeout for waiting for a response, but this implies the question - how to calculate the timeout as efficiently as possible so that there are not too many unnecessary service messages, but at the same time small enough to track emerging problems as quickly as possible?

**TABLE 1.** Two-phase commit problems

Step	Effect for coordinator	Effect for node
Node not received message "prepare"	Cannot commit the transaction because it has not received confirmation from one of the nodes	Cannot start a transaction because it has not received the appropriate command
Coordinator not received message "ready" from one of the nodes	Cannot commit the transaction because it has not received confirmation from one of the nodes	Cannot commit the transaction because it has not received the appropriate command
Node not received message "commit"	Cannot commit the transaction because it has not received confirmation from one of the nodes	The node cannot commit the transaction because it has not received the appropriate command. The rest of the nodes have committed a transaction that the coordinator cannot commit.
Coordinator not received message "committed" from one of the nodes	Cannot commit the transaction because it has not received confirmation from one of the nodes	All nodes have committed a transaction that the coordinator will not commit

## RESEARCH METHODS AND RESULTS

The study of this moment was carried out using the example of the stage of committing a transaction as the longest stage of its implementation.

**TABLE 2.** Results of 1st data collect stage

Node	Maximum execution time, sec.	Minimum execution time, sec.	Average execution time, sec.
Node 1	135.86	111.94	122.76
Node 2	105.01	92.37	99.09
Node 3	144.12	128.30	135.61
Node 4	79.86	76.82	77.80
Node 5	78.64	65.47	72.15
Node 6	13.34	2.38	8.15
Node 7	0.31	0.14	0.20
Node 8	70.20	47.57	57.13

In the course of the study, it was planned to collect data on the duration of the “heavy” query in the contact center database. Queries were executed in the PostgreSQL 9.6 DBMS, and the load on these databases was carried out by the contact center software Naumen Contact Center. Data collection was carried out from eight contact center

**TABLE 3.** Results of 2nd data collect stage

No	Time	No	Time	No	Time	No	Time	No	Time
1	111.94	11	119.91	21	118.41	31	133.18	41	136.52
2	128.30	12	129.17	22	136.80	32	123.08	42	138.22
3	113.18	13	115.00	23	122.10	33	141.94	43	132.40
4	131.82	14	136.90	24	107.24	34	128.17	44	118.54
5	132.12	15	112.67	25	109.10	35	138.62	45	125.74
6	141.00	16	133.06	26	130.05	36	121.93	46	123.54
7	133.39	17	113.96	27	117.89	37	126.38	47	119.75
8	132.62	18	138.18	28	114.13	38	131.72	48	130.99
9	135.86	19	115.66	29	128.79	39	115.52	49	99.97
10	132.43	20	144.12	30	123.43	40	133.55	50	131.22

No	Time	No	Time	No	Time	No	Time	No	Time
51	127.57	61	127.06	71	133.54	81	130.93	91	111.35
52	92.33	62	151.92	72	110.60	82	134.15	92	123.97
53	130.37	63	145.28	73	130.57	83	112.83	93	135.93
54	137.55	64	127.56	74	128.35	84	123.15	94	117.42
55	127.10	65	122.39	75	128.84	85	120.63	95	131.60
56	133.84	66	134.22	76	102.26	86	119.76	96	118.07
57	123.73	67	135.18	77	114.44	87	120.71	97	130.03
58	148.01	68	130.22	78	137.03	88	125.83	98	148.58
59	121.98	69	122.41	79	139.83	89	121.16	99	129.01
60	117.32	70	114.51	80	151.66	90	132.69	100	125.15

installations with similar hardware characteristics and different loads (Table 2). It can be seen that the data between the servers is significantly different, as a result of which two installations with the highest query execution time were selected from the participating servers, namely, nodes 1 and 3.

When examining the data collected at the second stage (Table 3), a hypothesis was put forward about the normal distribution of the query execution time. Pearson's chi-squared test was used to prove this hypothesis with parameters  $m = 126.71$  and  $\sigma = 10.89$ . Figure 3 shows a histogram of frequencies and a line of theoretical distribution.

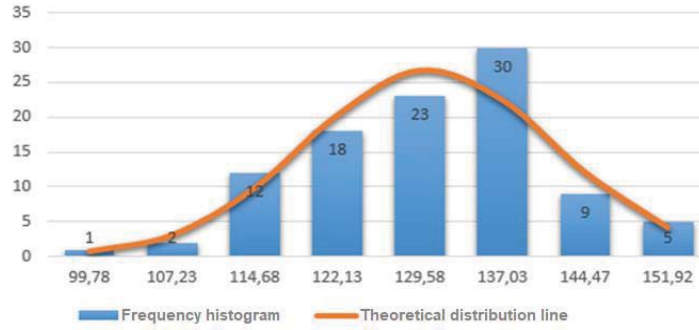
Based on the results of comparing the calculated empirical value (1) and the tabular critical value (2), the hypothesis of a normal distribution is accepted with a significance level of 0.05.

$$\chi^2_{\text{emp}} = 5.26 \quad (1)$$

$$\chi^2_{\text{crit}} = 11.0705 \quad (2)$$

The normal distribution of query execution time allows us to turn to the three sigma rule: the probability of a random variable deviating from its mathematical expectation by more than three standard deviations is practically zero.

Thus, if you set the query execution timeout to a value equal to three standard deviations, only extremely heavy queries will go beyond it. In most cases, its activation will mean problems in the operation of the System.



**FIGURE 3.** Histogram of frequencies and a line of theoretical distribution

However, this option is inherently redundant, and if timely detection of emerging problems is more important than the temporary loss from triggering a timeout for normal requests, another approach to obtaining the optimal timeout value is better suited.

For this, a timeout optimization function  $F(x)$  was created. It represents the sum of two terms:  $F_1(x)$  - costs associated with the maximum response time to an emergency, duration  $x$  sec. and  $F_2(x)$  - losses associated with exceeding the time  $x$  when executing a request. These costs are proportional to the probability that the duration of the query execution  $T$  will exceed the selected waiting time  $x$ :  $P\{T > x\}$  and depend on the type of distribution and its parameters. For the normal distribution, we get:

$$F_2(x) = p \cdot s \cdot P\{T > x\} = p \cdot s \cdot \left(0,5 - \Phi\left(\frac{x-m}{\sigma}\right)\right), \quad (3)$$

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_0^z e^{-\frac{z^2}{2}} dz, \quad (4)$$

Here  $p$  is the probability of an emergency,  $s$  is the total estimated number of database queries.

Thus, we get the optimization problem:

$$F(x) \rightarrow \min_{x>0}, \quad (5)$$

$$F(x) = F_1(x) + F_2(x) = l \cdot x + p \cdot s \cdot \left(0,5 - \Phi\left(\frac{x-m}{\sigma}\right)\right), \quad (6)$$

where  $l$  is temporary losses when a timeout was triggered per one executed query.

Before optimization, the initial values of parameters were determined. The initial parameters and their values are given in Table 4.

**TABLE 4.** Initial parameters for optimization function

Parameter	Value
Probability of an emergency, $p$	0.1
Temporary losses when a timeout was triggered per one executed query, $l$	0.15
Total estimated number of database queries, $s$	10000

Further, using these parameters, as well as the distribution parameters, the optimization function (5) was compiled, the graph of which is shown in Figure 4. With the optimal timeout, the value of the function will be minimal. With the initial data described earlier, the optimal timeout value turned out to be 165.4 seconds.

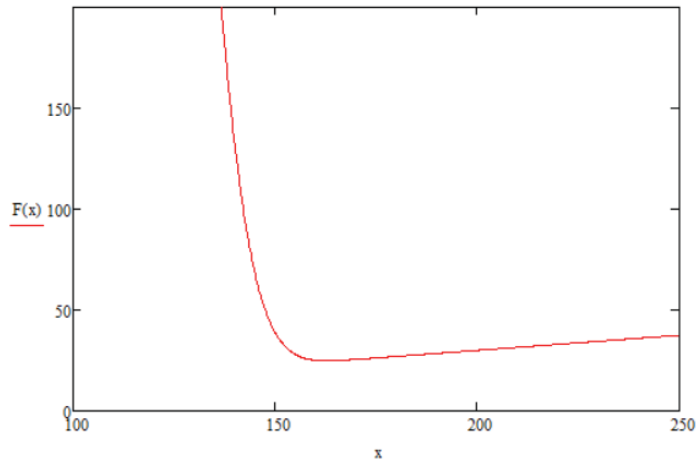


FIGURE 4. Graph of optimization function

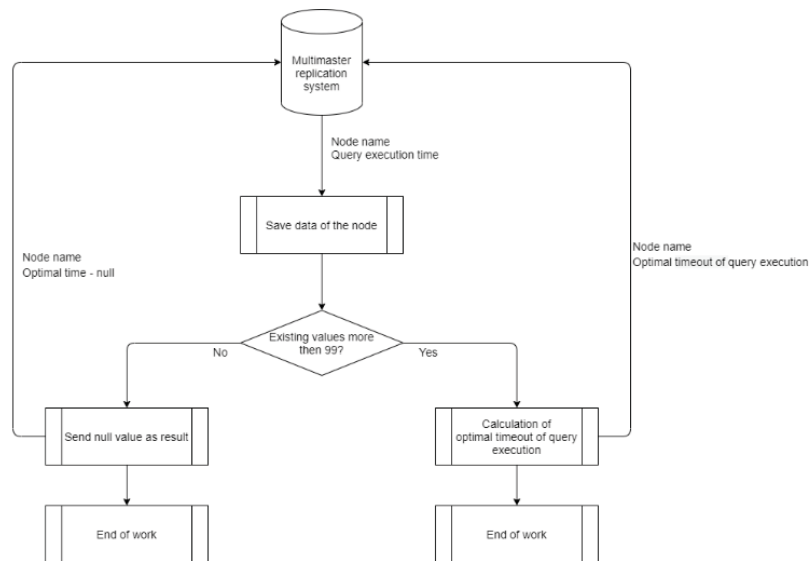


FIGURE 5. Block diagram of the module work

The results obtained can be used to develop a module that calculates floating timeouts for multi-master replication systems depending on the duration of requests by participating servers.

## RESULTS IMPLEMENTATION

On Figure 5, one can see an example of how the module works in the form of a block diagram. Upon completion of each request, the module accepts as input the name of the server on which the request was executed, as well as the execution time of that request, and returns the estimated optimal timeout time for this database server. If the number of values received during the module operation is less than 100, the module returns an empty value.

Let us move on to the implementation. Despite the fact that during the operation of the module all the values will be stored in the cache of the module, it was decided to use Redis as a system for long-term storage of arrays of values [5], so that when the module is restarted, there is no need to collect data for its operation from scratch.

When developing the module, it was decided to omit the stage with the study of the distribution normality, since this hypothesis had already been accepted earlier. Thus, the work of the module is only to find a value equal to three standard deviations for the samples contained in the system.

The resulting module can be adapted for PostgreSQL version 9.6 and higher and can be viewed on github: <https://github.com/SunLightFS/execution-timeout-counter>

## CONCLUSIONS

The study of the normal distribution of the query execution time allowed us to develop a module for calculating the optimal query execution timeout for multi-master replication systems. The resulting module minimizes the previously considered problems of multi-master replication systems based on the two-phase commit protocol, which allows expanding their scope, as well as serviceability and reliability.

## REFERENCES

1. M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *JACM* **27**, 228-234 (1980).
2. E. Akkoyunlu, K. Ekanadham, and R. Huber, "Some Constraints and Trade-offs in the Design of Network Communications," in *Proceedings of the Fifth ACM Symposium on Operating Systems Principles*, Vol. 9, pp. 67-74 (November 1975).
3. M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*, 1st edn (O'Reilly Media, 2017), pp. 352-37.
4. P. Bernstein and E. Newcomer, *Principles of Transaction Processing* (Elsevier, Amsterdam, 2009), 400p.
5. S. Chen, X. Tang, H. Wang, and H. Zhao, Towards scalable and reliable in-memory storage system: A case study with Redis, *2016 IEEE Trustcom/BigDataSE/I SPA*, pp. 1660–1667 (2016).