Improving Recovery Block Method for More Reliable Client-Server Communication

Daniil Ageev, Nikita Voinov
Peter the Great St. Petersburg Polytechnic University
Saint Petersburg, Russia
ageev.dyu@edu.spbstu.ru, voinov@ics2.ecd.spbstu.ru

***Abstract*—Nowadays high availability and reliability are essential characteristics of any enterprise client-server architecture. This primarily requires a stable and fault-tolerant operation of the server side which can hardly be guaranteed without possibility to select an appropriate server depending on specific client needs. In this case fault-tolerance is provided by the client side. One of the main principles used for such approach is Recovery Block. Described in this paper is a new algorithm for increasing fault tolerance in client-server communication. It is based on Recovery Block method and allows ranking of available servers with their stability factor and making decision of removing unstable servers considering previous connection attempts.**

I. Introduction

Fault tolerance is the most important criteria of a software system to prevent unscheduled outage of dependent applications. In client-server architectures to provide high level of fault tolerance a failover approach is usually used which means that a client switches to a reserve server in case of any issues with a primary one. Failover can be implemented using Recovery Block method [1]. In this case the probability of a client failure depends on the probability of a server failure.

In this work a new method is proposed which improves standard Recovery Block. The method is based on the algorithm for selecting a specific server among available ones by considering previous failures occurred during the operation of applications.

The described method can be applied for the client-server architecture consisting of the following components:

1) A client permanently connected to a single server. Client stability is defined by results of requests to the server.

2) A set of communication servers located in different data centers worldwide with different intervals for updates and technical maintenance.

Such architecture may lead to unstable operation of the whole system due to possible failures on the server side such as DNS issues for a specific region, database corruption, internal server errors, human mistakes, update issues, etc. The proposed method is aimed to reduce the probability of the communication failure and downtime of the client side.

II. Related works

The considered topic is focused mostly on the aspect of client-server communication but fault tolerance is an extremely broad area providing various algorithms that cover different problem domains.

Jhawar R. et al. [2] introduces innovative perspective aspect of managing without focus on implementation details, but using service layer instead. This allows to satisfy user requirements for fault tolerance. Moreover, clients do not need to adjust their applications for specific environment.

Kumar A. et al. [3] investigates different techniques of fault tolerance which are used in real time distributed systems. This paper is focused on failure types for distributed systems and the mechanism of their detection. Based on the type of a failure different fault tolerance techniques can be used to reduce downtime.

Dhingra M et al. [4] makes the comparative analysis of fault tolerance models and their challenges in cloud computing. Pros and cons of existing techniques of fault tolerance used in cloud are discussed. In general handling system errors while computing in a cloud is a real challenge for software developers.

Verma A. et al. [5] considers modification of classical N-Version systems. The main difference between classical solution is a weight factor for each version. It provides opportunity for a system to rank versions for voting module. The voting mechanism in this case is based on trust factor for each version.

Existing methods often cannot be applied for communication between server and client. Proposed in this paper is a modification of Recovery Block method that can be applied to existing client-servers architectures to increase their reliability.

III. Standard Recovery Block method

Recovery Block is one of the fundamental techniques used for client-server communication. Its standard model is shown at Fig. 1.

When failure occurs, the client either sends a repeated request to the same server or breaks the connection with the current server and makes a new connection to another server. This is a common approach used in many frameworks (Spring,

etc.) In this solution the probability of a system failure is calculated as follows:

$$P_{rb} = \prod_{i=1}^{n} (e_i + t_{1i}) + \sum_{i=1}^{n} t_{2i} e_i \left(\prod_{k=1}^{i-1} e_k + t_{1k}\right) \quad (1)$$

where

e_i – the probability of the *i-th* server failure;

t_{1i} – the probability that for the *i-th* attempt the correct results is considered to be incorrect;

t_{2i} – the probability that for the *i-th* attempt the incorrect results is considered to be correct.

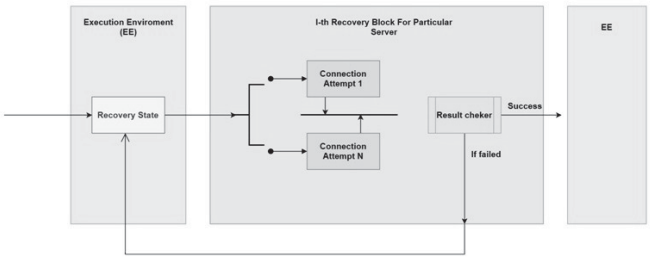

Fig. 1. Recovery Block standard model

One of the main methods used to select the next available server is Round Robin. It goes through all available IP addresses one by one until either the list is over or successful reconnection. There are also modifications of this method, such as ibnamed, which provide servers based on client location and server load.

The proposed method allows to consider additional information stored at the client side for selecting a specific server with maximum probability for successful reconnection.

IV. The Proposed Improvement of Recovery Block Method

According to (1), successful reconnection at the *i-th* attempt means that the downtime depends only on the total number of all reconnections and can be calculated as follows:

$$t_{downtime} = \sum_{i=1}^{n} t_i \quad (2)$$

where

t_i – downtime while reconnection to the *i-th* server.

To reduce the downtime the number of reconnections shall be minimized. This can be achieved by increasing the probability of successful transfer to the new server.

Investigation of the reason of unstable client-server connection at the specific moment is a very complicated task which may consume a lot of resources. Alternative way provided by the proposed method is a special algorithm for tracking and ranking available servers according to the probability of their failures.

As the main task is minimization of reconnection attempts, the client shall keep historical data of all previous attempts. This can be implemented either by storing it in client local database, local file or on the servers. Further considered is data storage in a local file to simplify the method description. A special JSON format is used to keep information about all available servers (Fig. 2).

```
{
  "ipv4": string,
  "ipv6": string,
  "successProbability": double,
  "totalSuccessAttempts": long,
  "totalFailureAttempts": long,
  "totalAttempts": long,
  "serverVersion": string,
  "errorReasons": [
    {
      "statusCode": int,
      "operationType": string, ... custom operation type that defined by client, also can be defined as PUT/POST/DELETE and etc. requests
      "successProbability": double,
      "successAttempts": long,
      "failureAttempts": long,
      "attempts": long
    },
    ... can be described any a different set of errors like Unauthorize, Internal Server Error, Gateway Timeout and etc.
  ]
}
```

Fig. 2. Data stored for a specific server

Based on this data a client can find servers with less failures according to connection history. This allows to reduce the number of unsuccessful attempts and consequently the overall downtime. Besides that, the stored information contains types of operations which failed. As each operation may involve different functionality of a server application (which can be temporary unavailable or unstable for other operations), it is proposed to split the ranking by operation types, so that the client can use the most appropriate server at specific time.

Assuming that failures in client-server communication occur on the server, their distribution can be tried to be predicted based on the results of requests execution. Thus the proposed solution based on the statistics of communication with servers may help to reduce downtime and failures due to selection of the most appropriate servers among the available ones.

The proposed algorithm is shown at Fig. 3 and contains the following steps:

1) Create a new event which requires a request to the server and specify the type of this event.

2) Select the first server in the list of available servers.

3) Connect to the server. If successful, go to step 9. Otherwise go to step 4.

4) Define type of the failure received from the server.

5) Increment the counter of specific failure type for the server and overall failure counter, and update the file with connections history.

6) Create a new object for the server in accordance with specific failure, if this object does not still exist.

7) Select a new server with maximal probability of successful connection for a specific operation from the list of available servers. If all servers with historical data about failures of this type were tried, connect to other servers starting from those which have maximal overall probability of successful connection.

8) In case of failure of all operations, the event is put into retriable state. In this state each N time slots the client tries to repeat the algorithm until successful implementation of the operation. After some limit K is reached, the event is removed.

9) In case of successful operation, its data is stored in the file with connections history. The current server is used until any further failure.

10) Each T time slot update ranking of servers in accordance with probability of successful communication with specific client.

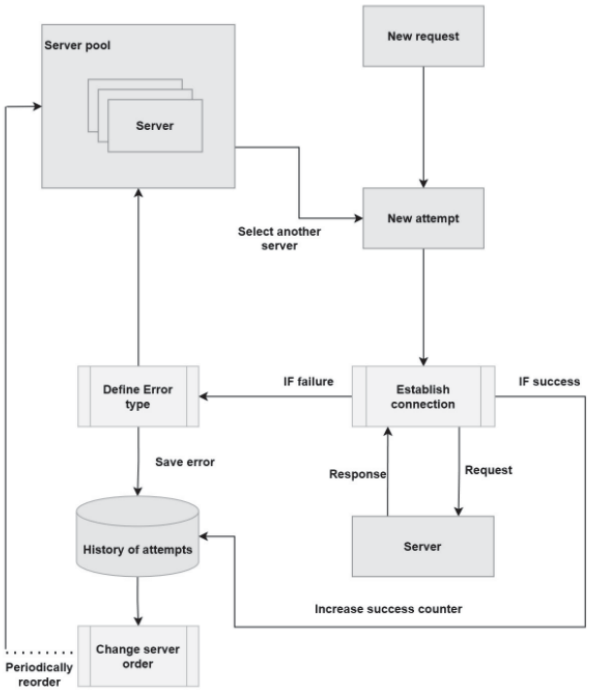

Fig. 3. Scheme of the proposed algorithm

Servers ranking is calculated based on probability of successful requests and failure types in response on specific requests. Specifying of a failure can be implemented either dynamically based on the server response or statically at the client side where a specific function corresponds to each operation. For example, operations can be grouped by modules used in server infrastructure, such as NFS volume (send/delete files), database operation (add/remove system), etc.

Example of such failures specification is shown at Fig. 4. In case of a specific operation failure, the most appropriate server is firstly searched for in the list of ranked probabilities of this type of operations. If the list for specific operation expires, then return to the general server list. Such ranking allows to select the most appropriate servers for current tasks and avoid requests to irrelevant servers.

V. Results

The proposed method was compared with standard Round Robin method while modeling failures for miscellaneous system components during 1 hour and calculating system downtime.

Obtained results shown in the Table I prove effectiveness of the proposed method. It can be even more significant when failures occur systematically for the same servers on a long time slot lowering their position in the ranking list.

However the current algorithm contains a drawback. While collecting a large number of operations, occurrence of new systematic failures with servers in the top of the list may lead to the situation when their probability of successful communication will reduce very slow even with a large number of such failures. This is because of large number of accumulated requests. This can be solved by resetting information about number of requests, but storing the current probability value.

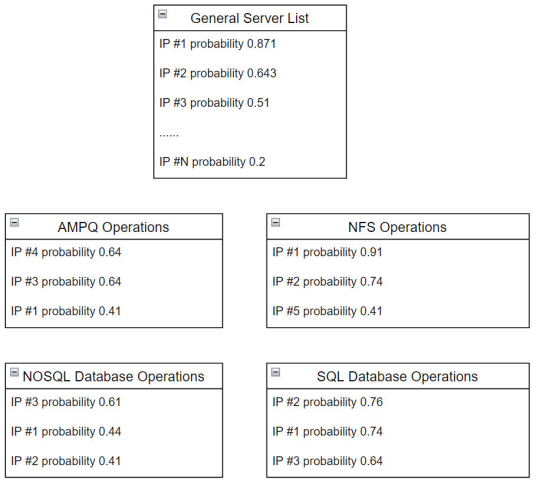

Fig. 4. Grouping available servers by operation types

TABLE I. Experimental results

System component	Downtime for proposed algorithm, seconds	Downtime for Round Robin method, seconds
NFS	274.2	301.7
SQL Database	255.6	273.0
NOSQL Database	253.1	261.3
AMPQ	281.7	306.2

VI. Conclusion

The paper describes a new method improving standard Recovery Block solution for reliable client-server communication. The improvement consists in selecting a specific server among available ones by considering previous failures occurred during the operation of applications. This is implemented based on servers ranking which is calculated based on probability of successful requests and failure types in response on specific requests. Experimental results were compared with existing approach and proved effectiveness of the proposed method for reduction of client downtime.

However the proposed approach has a limitation for the client - it cannot be thin and should store server statistic to provide more accurate data. Furthermore, the ranking process creates extra work for the client in addition to its main functionality.

A possible way to improve the proposed method is to share the statistic of server failures with a set of similar clients. It

shall allow clients to use the most relevant servers, but it also creates a challenge for servers at the top of the list to hold new workload without failures that may affect corresponding list.

REFERENCES

[1] B. Randell, "System Structure for Software Fault Tolerance", *in IEEE Trans on Software Engineering*, vol.1(1), June 1975, pp.220-232.

[2] R. Jhawar, V. Piuri, and M. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing", *in Systems Conference (SysCon)*, Mar.2012, pp. 1-5.

[3] A. Kumar, R.S. Yadav, and A.J. Ranvijay, "Fault tolerance in real time distributed system", *Int. J. Comput. Sci. Eng.*, vol.3, 2011, pp. 933–993.

[4] M. Dhingra, and N. Gupta, "Comparative analysis of fault tolerance models and their challenges in cloud computing", *in International Journal of Engineering and Technology*, vol.6(2), 2017, pp. 36–40.

[5] A. Verma, A. Ghartaan, and T. Gayen, "Review of Software Fault-Tolerance Methods for Reliability Enhancement of Real-Time Software Systems", *International Journal of Electrical and Computer Engineering*, vol.6(3), Jun.2016, pp. 1031-1037.