Time Stamp based global log and monitor approach to handle orphans in distributed systems

Shamsdueen. E†, Dr. V. Sundaram††

†Research Scholar, Karpagam University, Coimbatore, India
††Director, MCA, Karpagam Engg. College, Coimbatore, India

Summary
Orphan handling in distributed system is very important because, the orphans make problems like inconstancy of data[2],[7], wastage of resources[2], and the execution time of the server processes. Moreover, having no orphan state[1] in distributed system is very rare. The timestamp based global log and monitor approach tries to answer the questions like, What happens when the client crashes while the token is being updated? and how the monitor process knows the communication link between client and server which participate in the RPC mechanism is down. In order to tackle these type of situations a deadline is setup with the token which is traveling across the system to find out whether any client which participates in the RPC is down or not. This approach provides such a mechanism that should find out the orphans and killed immediately after the orphan is born. So it prevents the orphans from seeing inconsistent information [4].

Key words
RPC, orphans, Global log, time stamp ,etc.

1. Introduction

RPC is the fundamental communication mechanism for client/server interaction in distributed systems. The client is the initiator of an RPC, and the server provides the implementation of the remotely executed procedure. It is shown in the figure 1.The request message in the figure 1 contains all current input parameters for the procedure call. Conversely, the response message contains all results for the corresponding request produced by the server.

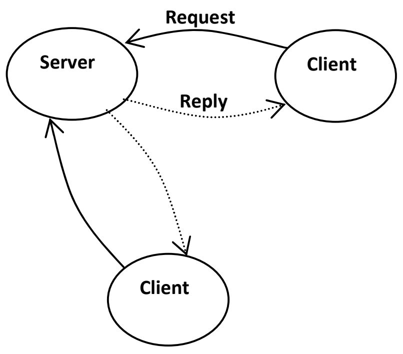

Figure 1. Client/server interaction in Distributed system

Manuscript received August 5, 2011
Manuscript revised August 20, 2011

Figure 2 shows the situation in an RPC where the client that breaks down during the execution of a remote procedure call. It results the orphan execution at the server end.

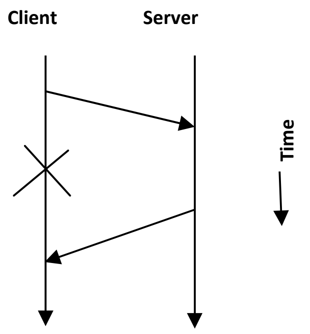

Figure 2. Client Process crash

An orphan process is a process that is being executed at the server and in the meanwhile the initiator (parent) of that process has been crashed down or the parent process has been aborted or the communication link between client and server has been down. In effect, there will be no parent process to wait for the result of an orphan computation. So, the result of such a computation is no more needed either [11]. Moreover, such computations make many problems like,

1. Inconsistency of data,
2. Wastage of resources
3. Wastage of server execution time

Orphan process may also increase the computational cost [3]. So, orphans in distributed systems are to be found out and killed immediately. Many approaches are there in the literature to detect and kill orphans. The main approaches are,
Extermination [10]: orphan is killed by looking into log entry which is made by a client before and RPC is made.
Reincarnation [10]: the way it works is to divide time up sequentially numbered epochs. When a client reboots, it

broadcasts a message to all remote computations on behalf of that the client are killed.
Gentle reincarnation: when an epoch broadcast comes in, each machine checks to see if it has any remote computations, and if so, tries to locate their owner. Only if the owner cannot be found is the computation is killed.
Expiration[9]: In this approach, a deadline shall belong to all RPCs. If the work is not completed within the specified time, then one new deadline shall be requested.
All the above approaches are effective in some aspects and also have some limitations. For example, in the case of extermination, it is good when system contains very less number of RPCs, otherwise the log will be very lengthy and this approach says nothing about the grand orphans-result of nested invocation [6]. In all the above approaches, the orphan is detected and killed only after rebooting the client. In the global log and monitor approach, the orphan is detected immediately after an orphan is born. In this approach, the detection and killing is done by the global monitor by constantly listening the clients who are participating in the RPC. It is done by sending a token through the system and receiving back the token by the global monitor. The monitor process checks for the data structure associated with the token and finds the value of status variable. If the value is 1, then the corresponding client is alive and if it is '0', then the monitor process realize that the particular client is no more there in the system.

2. Time stamp based global log approach

In the global log and monitor approach [5], a time stamp parameter is added in order to answer the problems listed below.

- What happens if token is lost due to downing of a node just after it receives the token and it is being updated? Figure3.

- While the token is being transmitted the communication link fails, figure4, then the result is same, and that is the lost token. How it tackles?

In the above two circumstances, the monitor process does not get the token back and the monitor process cannot identify the server process where the orphans are active.
The only solution to the above problem is to regenerate the token and send it through the network again. Now the question arises here is that in what interval the token to be regenerated? A time stamp is set with the token, so that the token can be regenerated after that deadline of the token.
The deadline of the token can be calculated on the basis of the number of machines of the system, and the channel capacity, and the time required to update the token by each client processes.

The deadline, $D= N*t*\delta$

Where N is the number of systems and 't' is the time needed to update the token and 'δ' is the communication delay and it is depending up on the quality of the communication channel and the network speed.

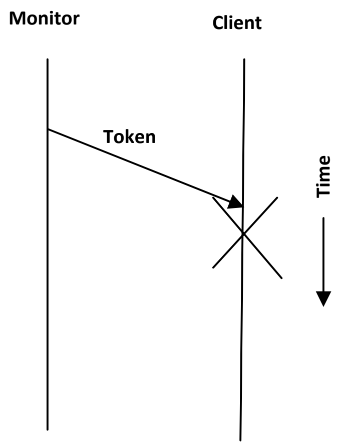

Figure 3. Client crashes down while the token is being updated

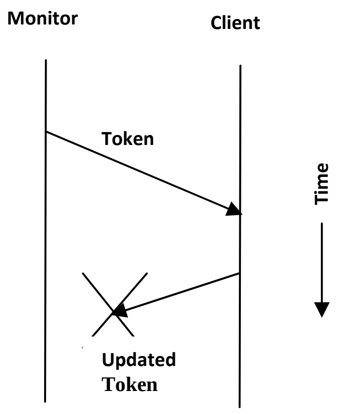

Figure 4. Communication link fails while sending back the token

Here we assume that the't' is same for all processes. The deadline is low when number of system is less in number and network provides very high speed.

3. The Protocol

Monitor process: sends the token, t1, through the system to know whether the clients are alive or not by looking into the value of the status variable associated with the token. If variable value=1, it indicates that the client is alive and if value is 0, it indicates that the client is no more.

Case 1:
Client: after receiving the token, t1, the client updates the status variable of the token to 1, indicates that the client is alive, and passes the token to the monitor process or to the next node of the system.
Case 2:
Client process: after receiving the token, t1, the client updates the token. While updating the token the client crashes. The result is that the monitor process does not get the token back or the token would not be passed to the next node(who participates in the RPC) of the system.
Case 3:
The communication link with the client fails while the token is possessed by the client. In this case too, the token would not be back to the monitor process.
Monitor: In Cases 2&3, after the deadline, $D = N*t*\delta$, the monitor process regenerates the token and sends the token again through the system and gets back the system.

4. Conclusion

The solution to the problems, the client crash and communication link fail, is to regenerate the token sent by the monitor process. The regeneration of the token can be done by any algorithm which is used in the network systems. The regenerated token is again sent through the system and find out the parent process which are active and which are down. The orphans of the corresponding down processes are killed by sending appropriate messages to the server processes. If the communication link of the system is very much prone to fail, then this time stamp based approach is not suitable because the token would not reach back to the monitor process in all occasions. This causes regeneration of the token again and again. By doing so the orphans of other crashed process cannot be identified. i.e., the orphans of other process still active at their servers. So, this approach is suitable when the communication link is almost reliable.

References

[1] Min Ma, Shiyao Jin, Chaoqun Ye, and Xiaojian Liu, "Dynamic Fault Tolerance in Distributed Simulation System", Springer-Verlag Berlin Heidelberg, ICCS, Part1, LNCS 3991, pp. 769-776, 2006
[2] A.D. Birrell and B.J. Nelson, "Implementing remore procedure calls", ACM Trans. Comput. Syst., Vol. 2. no.1, pp. 39-59, Feb 1984.
[3] Jaochim Baumann and Kurt Rothermel, "The Shadow Approach: An Orphan Detection Protocol for Mobile Agents", Springer-Verlag London Ltd, Personal Technologies (1998) 2: pp. 100-108
[4] Valerie Issarny, Gilles Muller, and Isabelle Puaut. "Efficient Treatment of Failures in RPC Systems". Proc. 13th Symposium on Reliable Distributed Systems, pp. 170-180. IEEE Comp. Society Press, 1994.
[5] Shamsudeen. E and Dr. V Sundaram. Article: An Approach for Orphan Detection. International Journal of Computer Applications 10(5):28–30, November 2010. Published By Foundation of Computer Science.
[6] S. Pleisch, A. Kupsys, and A. Schiper. "Preventing Orphan Requests in the Context of Replicated Invocation". In Proc. 22nd Symp. on Reliable Distributed Systems, pages 119–129, 2003.
[7] M. P. Herlihy and M. S. McKendry, Time-Stamp based orphan elimination, IEEE Transactions on Software Engineering, vol. 15, no. 7, pp. 825-831, 1989.
[8] Maurice Herlihy, Nancy Lynch, Michael Merritt, and William Weihl. On the correctness of orphan elimination algorithms. In Proceedings of the 17th Annual IEEE Symposium on Fault- Tolerant Computing, July 1987.
[9] M. Jahanshahi, K. Mostafavi, M.S. Kordafshari, M. Ghlipour, A.T. Haghighat, " Two new Approaches for Orphan Detection", Proc. IEEE 19th International Conference on Advanced Information Networking and Applications (ANAI"05), 2005
[10] Pradeep K. Sinha, "Distributed Operating Systems-Concepts and Design", Prentice Hall, 2008.
[11] Fabio Panzieri, Santosh K. Shrivastava, "A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing" Proc. IEEE Transaction on Software Engineering, Vol. 14, No. 1, 1988.