An Approach for Orphan Detection

Shamsudeen. E
Research Scholar, Karpagam University,
Coimbatore, India

Dr. V. Sundaram
Director, MCA, Karpagam Engg. College,
Coimbatore, India

ABSTRACT
In distributed systems, node crashes [1] and abort processes may result orphan computations: computations are still active and its results are no longer needed [5]. Orphans are undesirable because they waste system resources [3] and may make inconsistent data [4]. In this paper we present a new approach called global log and monitor approach to cope up with orphan in a more effective way. By this method orphans can be killed immediately after the node crash or abort process happens. No need to wait until the nodes get rebooted as the other methods does. This approach deals with the problem of grand orphans [2] as well. The grand orphans are by orphans who make further RPCs. i.e., the grand orphans are the result of nested transactions.

Key words
RPC (Remote procedure call), distributed systems, orphan, Global log.

1. INTRODUCTION
In distributed system, the communication between two processes can be implemented by RPC, sending a request by a client to server to do some computations and waiting for the reply from the server. Here failure may occur due to communication link faults or due to node crash. In RPC, if the client requests something from the server and before gets back the reply from the server, the client crashes. It makes the computation continue to execute at server site with no parent process waiting for it. Such computations are called crash-orphan [1]. Another type of orphan occurs by aborting the parent process called abort-orphan. It is happened when a process made an RPC and before getting back the reply of computations from the server, the parent process aborted. i.e., the calling process no longer exists.

2. PREVIOUS APPROACHES
Nelson proposed four solutions to cope with the orphan problem [2]. Extermination [6]: Orphan is killed by looking into log entry which is made by a client before an RPC is made. After reboot, the log is checked and the orphan is explicitly killed off. The problems here are cost of logging per each RPC because of each client keeps separate log and killing of orphan is done only after rebooting of client. Furthermore, it may not even work, since orphans themselves may do RPCs, thus creating grand orphans or further descendants that are difficult or impossible to locate.

Reincarnation: the way it works is to divide time up into sequentially numbered epochs. When a client reboots, it broadcasts a message to all remote computations on behalf of that the client are killed. The problem with this method is its broadcast overhead in network and resulted traffic.

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 there owner. Only if the owner cannot be found is the computation killed

Expiration: In this approach, a deadline shall belong to all RPC. If the work is not completed within the specified time, then one new deadline shall be requested. At this point calculation of deadline is important because different RPCs need different types of services, hence different amount of time. The problem with this is that, suppose that orphan locks one or several resources and a timeout occurs, the process is killed suddenly, the locked resources are remained as locked for ever.

3. GLOBAL LOG MONITOR APPROACH
In this approach, we introduced a global log mechanism. While an RPC is being made by a process, it should be logged in the global log. This global entry keeps details of all RPCs made by the processes of the distributed system. The global log keeps updated and if a server makes an RPC, i.e., a nested transaction [2], it also be logged in the global log. The global log also monitors the processes which made RPCs so that node crashes can easily be detected by sending a token message which evolves round the network and passes through all the clients who send RPCs. If a node crash occurs, then the global log server immediately sends a message to corresponding processes where the orphan process and nested orphans being run and kill them off. Here, the orphans are killed immediately after the node crash. No waiting until the crash node get rebooted.

4. HOW IT WORKS?
A token contained the details of process id, and its status - whether it is alive (1) or not (0). The process id reveals the process and on which node it runs. The status variable will have a value ‘1’ or ‘0’. The value ‘1’ indicates the process is very much alive. i.e., there is neither a node crash nor process abort. If the status variable value is ‘0’, then it says that the process is not alive. The default value of the status variable is set as ‘0’ for every process. The Global log monitor sends the token to all processes who made RPCs and each process who made RPC receives the token and looks for its process id in the data structure and updates the status variable to ‘1’. After this the token is transmitted to the next process and does the same. If a process is no more after making RPC because of node crash or process abort, then its status variable will not be updated, i.e., status variable value will remain ‘0’. After evolving round, the token returns back to the monitor. The monitor checks the token

and finds the failed processes by looking into the data structure. For example, the data structure r_u_there (3, 0) says that the process with id 3 is not alive. Then the monitor will find the orphan process associated with the aborted processes or processes which run on crashed nodes by looking into the log entry and send message to kill them. In the above example all the orphan computations will be killed which are initiated by the process 3. The sending token, evolving the token round the nodes, returning the token back to the monitor, and killing of orphans goes on.

Some instances of token passing between monitor and client are given below, figures1-3.

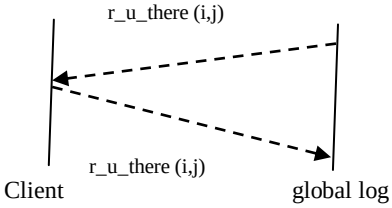

Figure 1. Token passing between a client and global log

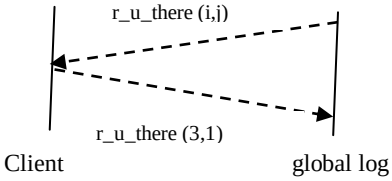

Figure 2. Token passing between and Global log and a client identified as 3 with status variable value set to '1'

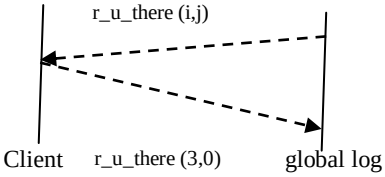

Figure 3. Token passing between Global log and a client identified as 3 with status variable value set to '0'.

(In the above diagrams the value of i = 0, 1, 2….n-1 and value of j = 0 or 1).

A single run of the token is illustrated below in the figure 4. It illustrates that the process2 and process n-1 are aborted or the node on which the process run is crashed. Process 0, 1 and n-2 are alive. In this case the computation initiated by the processes 2 and n is to be killed by sending messages to the corresponding servers where the orphan processes are active. The locations of the orphans can easily be found by looking into the global log entry where all the details about the RPC are kept.

r_u_there (0,1)
r_u_there (1,1)
r_u_there (2,0)
..
r_u_there (n-2,1)
r_u_there (n-1,0)

Figure 4. A sample run of the token message between the global log and clients numbered 0 to n-1.

5. BENEFITS

No inconsistency of data, because when the node crash or process abort occurs the killing of orphan processes are done by sending message by the global log monitor .

No wastage of resources either because killing of orphans occurs immediately after the crash or abort of processes.

Nested transactions can be easily handled because all nested transactions are logged when it happens at the global log. The comparison between the previous approach and the global log monitor approach is listed in the table1.

Table 1. Comparison between the previous approaches and global log approach

Name of the method	Advantages	Disadvantages
Extermination		No mechanism to handle grand orphans. Killing of orphan is done only after rebooting the client; it leads to inconsistency of data and wastage of resources. Nothing is said about abort-orphans.
Reincarnation		Overhead due to broadcast traffic.
Global log	All orphans including grand orphans killed properly. Killing is done immediately after node crash or abort process, hence no inconsistency of data and wastage of resources. Message is sent only to locations where orphans present, hence no network traffic.	

6. CONCLUSION

In this paper we present the roll back recovery methods for orphan and we present our method and its advantages over other methods. By our method the data consistency can be ensured in an effective way, because the orphans are killed immediately after the node crash or abort orphan occurs. This method effectively handles the problem of grand orphans. It is done by

making entry in the global log whenever an RPC occurs. Hence the orphans can be located and killed very easily with no matter whether it is an orphan or a grand orphan. Since the orphans are killed immediately after the node crash or abort process occur, the valuable system resources cannot be wasted either.

7. REFERENCES

[1] 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

[2] Andrew S. Tenenbaum, "Distributed Systems", Prentice-Hall, 2003.

[3] Joachim Baumann, Kurt Rothermel, "The Shadow Approach: An Orphan Detection Protocol for Mobile Agents", Institute for Parallel and Distributed High-Performance Systems, Stuttgart, Germany.

[4] Maurice P. Herlihy, Martin S. Mckendry, " Timestamp-Based Orphan Elimination", IEEE transaction on Software Engineering, Vol. 15, No.7, 1990

[5] 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.

[6] Pradeep K. Sinha, "Distributed Operating Systems-Concepts and Design", Prentice Hall, 2008.