Comparison of Global Log and Monitor Approach with Various Orphan Detection Algorithms

Shamsudeen.E[1], Dr. V. Sundaram[2],
[1]*(Research Scholar, Karpagam University, Coimbatore, India)*
[2]*(Former Director, MCA, Karpagam Engg. College, Coimbatore, India)*

ABSTRACT: *Orphans in the distributed systems may waste system resources by holding these for completing the unwanted computations. Many ways are put forward to detect and kill orphans in the distributed system. All the algorithms presented are detecting and killing the orphans only after rebooting the client. The period between the client crash and the client reboot, the orphans are active at the server end. But, the global log and monitor approach detects and kill the orphan at the point of time when they are born. Here a comparison of all approaches with the global log and monitor approach is made based on the various performance parameters like when the orphans are detected and killed, impact of orphans in the systems and how the overall performance of the system affected by the orphans and concludes that the global log approach is the best among other approaches to detect and kill orphans in the distributed systems. Moreover, this approach also deals with the nested orphans while the other approaches do not say anything much about it.*

KEYWORDS: *Global log and monitor approach, Orphans, Remote procedure call*

I. INTRODUCTION

Orphans [1] in the distributed systems are unwanted processes because their presence in the system may lead to inconsistency of data. Orphans are the result of either a parent process crash or abort [2] [3] [4] of parent process that take part in remote procedure call. Parent process crash may be the result of node failure where the parent process runs.

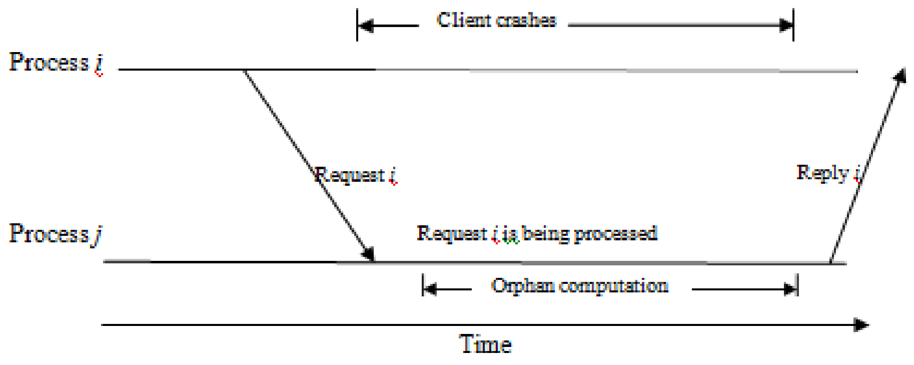


Fig. 1. An example of an Orphan computation

An example of an orphan computation is depicted in the figure, Fig.1, here process *i* sends a request *i* to the process *j* to get some service, unfortunately the request is being executed at the process *j*, the process *i* crashes and no parent is waiting for the reply *i*.

Orphan cause a variety of problems like waste of CPU cycles by running the unwanted processes. The unwanted process may also lock files or otherwise hold valuable resources [5]. After the client crash, if the client reboots and do the same RPC (Remote Procedure call) [6] again, the same computation may be done again and it may lead to inconsistency of data in the system. Nelson [6] proposed four solutions to handle orphans. They are,

- Extermination
- Reincarnation
- Gentle reincarnation
- Expiration

II. COMPARISON OF VARIOUS ORPHAN DETECTION AND KILLING METHODS USED IN THE DISTRIBUTED COMPUTING

The above methods and the method proposed in the global log and monitor approach [7] [8] [9] are thoroughly discussed with various aspects like,

- When orphans are detected and killed in the system?
- Impact of orphans in the system
- How the overall performance of the system affected with orphans.

All the above methods (by Nelson) detect and kill orphans only after rebooting of the client where parent process run. During the interval, between the orphan's birth and the client reboot, the orphan is very much active at server site. In the extermination, before a client stub sends an RPC message, it makes a log entry telling what it is about to do. The log is kept on disk or some other medium that survives crashes. After reboot, the log is checked and the orphan is explicitly killed off.

Here, the killing of orphan is done only after the rebooting of the parent process that made the remote call. Until then, the orphan is active at the server site, just because the fact that the server processes do not know the current status of the parent process that made the RPC to get some services from the server process. It is depicted in the figure given below, Fig.2. Here the client sends the same request twice and gets the results back. Both replies come back while resuming the client after crash. This type of scenario is not a promising one and it makes data in an inconsistent state and the system an unreliable one.

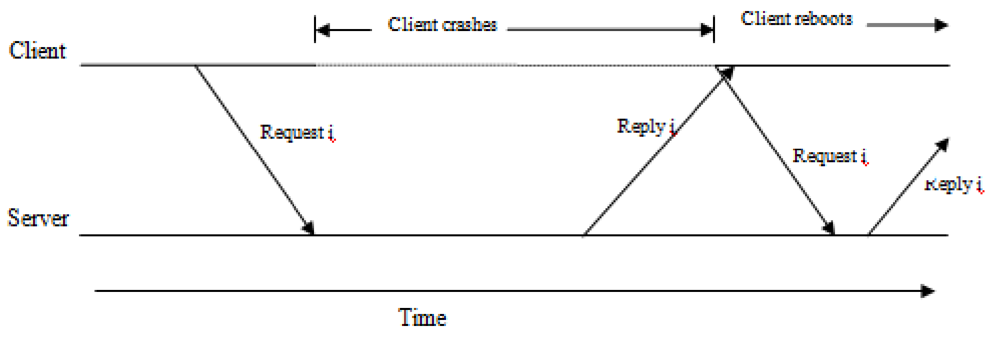


Fig. 2. An instance of extermination method of RPC

In the case of reincarnation, the RPCs do not record in the log; instead divide time up into sequentially numbered epochs. When a client reboots, it broadcasts a message to all machines declaring the start of a new epoch. When such broadcast comes in, all remote computations on behalf of that client are killed. When they report back, their replies will contain an obsolete epoch number making them easy to detect. It is shown in the figure, fig. 3. Here the client sends epoch broadcast throughout the system and the server receives it regardless of the fact that it is meant for it or not. The reply to the client is transmitted by the server if any RPC is there on behalf of the client.

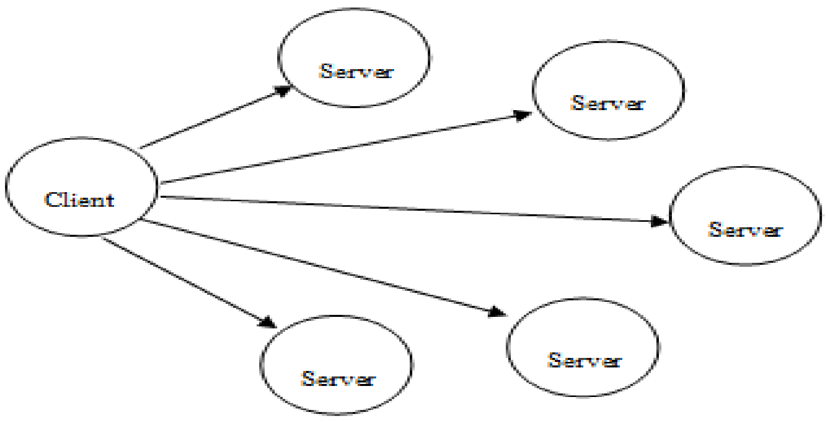
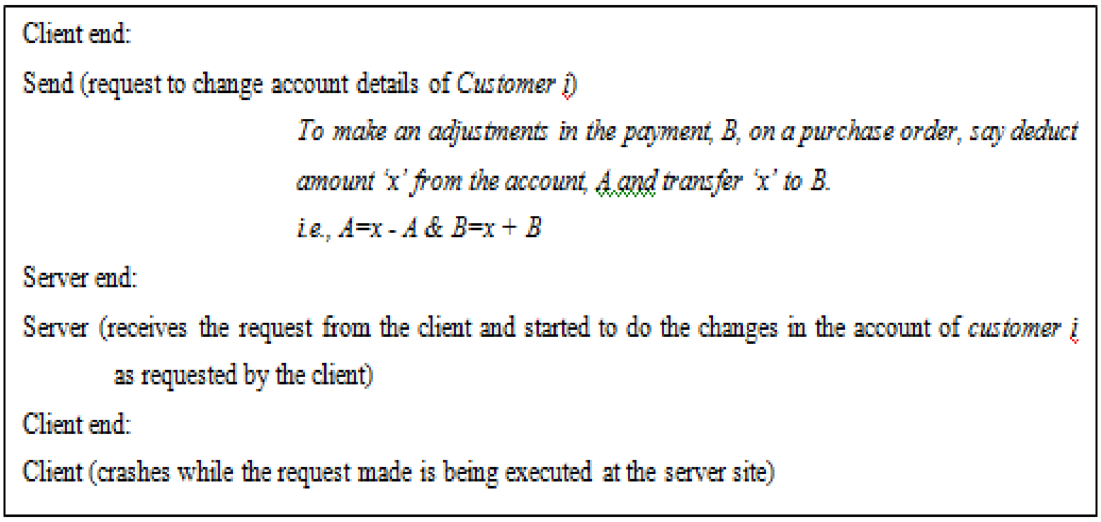


Fig. 3. Broadcasting epochs to all servers

A slight change is made to the above idea, another method put forward by Nelson called gentle reincarnation. The protocol works as follows, when epoch broadcast comes in, each machine checks to see if it has any remote computations, and if so, tries to locate their owner. The computation is killed only if the owner of a particular computation cannot be found. Finally Nelson proposes expiration method to handle orphans in distributed systems. In this method, each RPC is given a standard amount of time, *T*, to do the job. If it cannot finish the work on time, it must explicitly ask for another quantum of time, which is a nuisance. On the other hand, if after a crash the client waits for a time *T* before rebooting, all orphans are sure to be gone. The problem to be solved here is choosing a reasonable value of T in the face of RPCs with wildly differing requirements.

In the global log and monitor approach, a global log mechanism is introduced. While an RPC is made by a process, it is recorded in the global log and it 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, it also be recorded 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 with a time stamp which evolves round the network and passes through all the clients who participates in the RPCs. If a node crash occurs, then the monitor associated with the global log 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 gets rebooted. It is done by sending the token to all processes that made RPCs and each process that made RPC receives the token and looks for its process id in the data structure and updates the status variable associated with the token. 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 associated with the token will not be updated. After evolving round throughout the system, the token returns back to the monitor. The monitor checks the token and finds the failed processes by looking into the status variable. Then the monitor finds the orphan process associated with the aborted processes or processes which run on crashed nodes by looking into the log entry and sends message to kill them.

Below it is discussed a simple RPC scenario where a client sends a request to a server asking to do some computations.



Client end:

Send (request to change account details of *Customer i*)

To make an adjustments in the payment, B, on a purchase order, say deduct amount 'x' from the account, A and transfer 'x' to B.

i.e., A=x - A & B=x + B

Server end:

Server (receives the request from the client and started to do the changes in the account of *customer i* as requested by the client)

Client end:

Client (crashes while the request made is being executed at the server site)

An RPC where a client and a server participates

In the extermination method the RPC is logged in the log provided and checked after rebooting the system. After checking the log, the client resends the request to the same server to get the request fulfilled just because the fact that for the first request made by the client did not get any reply from the server. Before getting any reply from the server, the client crashes down and in the mean time the server continue to carry on the computation invoked by the client and does the necessary deductions, whatsoever, in the account of the *customer i*. The *customer i*'s account is deducted to x-A, and the same amount is credited towards the purchase, B.

Unfortunately, the client is not aware of the deductions made by the server since it did not get any result from the server and simply do the same request once again. The server once again execute the same computations and deduct the same amount from the *customer i's* account again. Instead of deducting the account of the said customer once, due to allowing the orphan to continue at server until the client gets rebooted, the amount is deducted twice from the account. If the orphan had been killed immediately after their birth, the result should have been the desired one. i.e., the account of the customer will be deducted only once. Extermination says nothing about the grand orphans- computations further does RPCs by the server themselves- or further descendants that are very difficult or impossible to locate.

In the above mentioned RPC scenario, if the reincarnation method is used? Then too, the problem of time delay to get rebooted the client is still exist. During this period, the orphans are active at the server site to fulfill the request made by the client. Here, valuable system resources are wasted until the epoch broadcast comes in. To fulfill the same request, the client has to resend the same message again to the server and the client process has to forcefully wait until the result of RPC back in. It affects the entire performance of the system. Moreover, the broadcast overhead in the system due to the epoch numbers broadcasted throughout the system regardless of the client and server who participate in the RPC.

In the case of gentle reincarnation, the only difference is that when epoch broadcast comes in, each machine checks to see if it has any remote computations, if so, tries to locate their owner, the epoch broadcast sends again after the rebooting of the crashed client. During the period of time the orphan is on at server node and holding all resources like CPU cycles, databases, etc. As a result of this the data may be in an inconsistent state, because the client does not aware the request made it earlier is still running at server end or not and hence the deduction at customer account is done. By the time of rebooting the client and sending epoch broadcast, the orphan may complete its work and all deductions as in the above scenario has been completed. In that case, the server will not have anything to do with epoch broadcast since, there is no process on behalf of it is being run. At the same time, the computation on the *customer i's* account is done. The reliability of the system is questioned if this type situation arises.

In the expiration method, a quantum of time is given to the RPC to complete. But, how the calculation of the time is done? Since different computations need different time interval to complete. Moreover, if the given time is not enough to complete the specified task, the computation can demand more time to complete, otherwise, the forceful termination of the process is done. If it does so, some transactions may be incomplete or some may possess lock on some resources, then the locks may remain forever there in the system. The incomplete transaction may produce inconsistent data and the incomplete transactions should be rolled back which may result extra overhead to the entire and hence low performance.

In the above mentioned RPC scenario, after doing the deduction from the *customer i'*s account, the RPC is forcefully killed just because of the time quantum allotted is expired, and then the payment to the purchase order, B would not be transferred. Again unreliable transaction is the outcome and this method is not a promising one.

All the above four solutions discussed by Nelson never tells any word about nested transactions or grand orphans and what happens this type of grand orphans or deeply nested transactions occurs in the system. In the global log and monitor approach, the orphans are killed immediately after their birth. No wait is done to get the system rebooted. So, there is no damage in the system in the sense that orphans do not make any inconsistency in the data because they are killed when they are born. But in the approaches discussed earlier no orphans are killed immediately after their birth. They have been killed only after the client gets rebooted it causes not only the inconsistency of data but, also the locking of valuable resources such as files, databases, and other hardware. Moreover, in the global log approach, since the orphans are killed when they are born and the system do not work with computations that are no longer needed by any parent process, hence the performance of the system would not be affected.

The global log and monitor approach accomplish this great difference from all other previous approaches by killing the orphans when they are born. It can be done by recording all RPCs in a global log that can be kept somewhere that survives the crash and the monitor process is associated with it looks into the entire RPCs in the system and sends a token throughout the system to find out whether the clients participated in the RPC are alive or not. If it is not alive, the corresponding orphans would be detected and killed. The token should have a time stamp, during this period this token should come back to the monitor process, otherwise the token should be regenerated and send it again and allow it to keep revolve round the system.

In the above mentioned RPC scenario, if this approach is used to take care of the orphans, then deduction and transfer will be intact and the system shows high reliability, data are consistent and hence the performance also very high.

III. CONCLUSION

In the above comparison, it is clear that the parameters discussed are quite enough to realize the domination of global log and monitor approach over other orphan detection and killing approaches. It is also noticed that the importance of the orphan processes to be detected and killed in the distributed systems. The global log and monitor approach kills the orphans immediately after their birth; no wait is done there to kill the orphans. Hence the wastage of resources and inconsistency of data can be reduced in the entire systems. While the other approaches takes some time to detect and kill the orphans; the time to get rebooted the client processes. So, during that the period of time the orphans are very much at the server site and hence the wastage of resources and deadlock in the system due to holding valuable resources by these unwanted computations. It may affect the performance and reliability of the entire system.

REFERENCES

[1] Fabio Panzieri and Santhosh K. Shrivastava, Rajdoot: A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing, IEE transactions on software engineering. VOL 14. NO. I. JANUARY 1988

[2] G. F. Coulouris, J. Dollimore, *Distributed Systems. Concepts and Design.* Addison-Wesley Publishing Company.

[3] A. Silberschatz, *Operating System Concepts,* Addison-Wesley Publishing Company, 1985.

[4] A. Tanenbaum, R. Van Renesse, *Distributed Operating Systems*, Computing Surveys, Vol. 17, No.4, December 1985.

[5] Shamsudeen. E and Dr. V Sundaram. *Issues with orphan computations in distributed computing systems*. International Journal of Computer Applications, pp-7-9, Vol 62-N020, Jan. 2013, Published By Foundation of Computer Science.

[6] A. Birrell, B. Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, Vol. 2, No. 1, February 1984, Pages 39-59.

[7] Shamsudeen. E and Dr. V Sundaram. *An Approach for Orphan Detection*. International Journal of Computer Applications 10(5):28–30, November 2010. Published By Foundation of Computer Science.

[8] Shamsudeen. E and Dr. V Sundaram. *A protocol to detect and kill orphan processes in distributed computer systems*. Journal of Computer Engineering (IOSRJCE), pp 12-16, Sept-Oct.2012.

[9] Shamsudeen. E and Dr. V Sundaram. *Time stamp based global log and monitor approach to handle orphans in distributed systems*. International Journal of Computer Science and Network Security, pp 123-125, Vol. 11 No., Aug 2011.