

# Issues with Orphan Computations in Distributed Computing Systems

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

V. Sundaram, PhD.  
Former Director, MCA,  
Karpagam College of Engg.,  
Coimbatore, India

## ABSTRACT

In distributed systems, the orphan computations [1] [2] make problems like data inconsistency and wastage of computer resources' usage time. The inconsistency [3][4] of data is a big concern among the users of the distributed systems. Here in this paper, the issues of orphans have been discussed and a case study that we face in our day to day life. The case study actually reveals how a common man gets affected by an orphan computation. The orphan may occur due to abort process, failure node or a failure of communication link between the client and server which participate in the RPC [5][6][7][8]. It is also discussed the situation of deadlock which is caused by orphan computations. The deadlock [9] [10] [11] situation actually locks the resources of the system by unwanted processes and hence affects the overall system throughput.

## Keywords

Remote procedure call, orphan computations, nested transactions, abort orphans, crash orphan, data inconsistency, deadlock, etc.

## 1. INTRODUCTION

The orphan process in distributed systems makes problems like inconsistency of data and wasting of valuable resources' usage time just because of the orphan processes continue to compute at the server site, where the requests of clients are fulfilled, and its results are no longer needed. In the distributed systems the communication is made by remote procedure call mechanism. Here a client process which needs service from the server process sends a request to the server and results of the request is sent by the server after computing client's request. In the mean time if the client which made the request fails by any means like abort process[12] or node failure[13] or even communication link between client and server fails, then the server without knowing these failures at the client site continue to operate on the request and try to send back the result. This type of situations frequently happens in the distributed systems. The inconsistency of the data due to orphan processes may affect very badly to the customers who believes that the distributed systems are highly reliable. For example, how the inconsistency in a bank transaction- which have been discussed in our case study- affects a common man. Many situations we come across in our day to day life like a railway ticket booking or while reserving an air line ticket through internet. In latter cases, before getting the ticket to the customer the client processes fail and the result is that an incomplete transaction as far as the customer is concerned. But, as far as the system is concerned, the transaction is a completed one because of the orphan computation where the orphan is not killed properly when it is born.

i.e., the orphan computation should not be allowed to run in the system. It can be accomplished by simply killing the orphan immediately after their birth. Or we can say that, when a node fails or abort process happens or whenever a link fail happens between the client and server, all the process corresponding to the request made by that particular process should be rolled back. We put forward a mechanism to handle these situations in our previous paper '*time stamp based global log [14] [15] and monitor approach to handles orphans in distributed systems*'.

## 2. HOW ORPHAN PROCESSES ARE BORN?

In distributed systems failures may happen to both processes and communication channels. Failures to processes are discussed here. They are,

### 2.1. Failure of node

The node of a distributed system may fail; eventually the all processes running on that node also fail. The processes which have been initiated by the processes of the failed node at other ends by executing remote procedure calls continue to run without knowing the fact that its parent process is no more at the requesting end.

### 2.2. Abort processes

When a parent process is aborted, but the processes running to fulfill the request made by the abort processes before being aborted continue to compute at server end and its results are unwanted because no parent is waiting for its result.

In the above two scenarios the result is *orphans*, a process continue to run and its results are no longer needed either because of parent process aborted or the parent process crashed. The former one is called *abort-orphan* and the latter one is called *crash-orphan*. It has depicted in the figures 1(a) to 1(c).

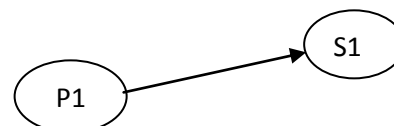


Fig.1 (a). The process P1 sends a request to server S1



**Fig.1(b). While server S1 computes the result the client P1 crash or abort of parent process happens**



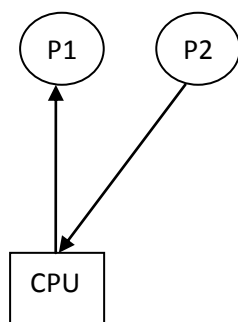
**Fig.1(c). After computation, the server sends back the result, but no client is to receive the result**

Here P1 is the client process which requests a service from the server S1. The client and server may be on the same system or in the different systems. In Fig.1(c), the process P1 is shown with dotted lines and fade color indicates that the client is down while the result is being communicated. That is the result of an orphan process is no longer needed by any one. In all the above cases, there is no client waiting for the result from the server. These types of data transactions make the data in an inconsistent state.

### 3. PROBLEMS WITH ORPHAN PROCESSES

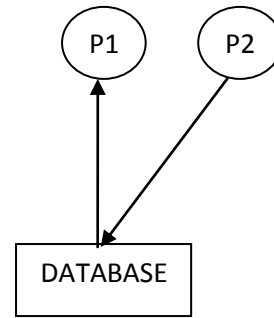
#### 3.1. Wastage of computer resources

Orphans cause two undesirable problems. First, they waste resources- the orphan process is active without knowing the parent process is no more there at the requesting end. The resources held by the orphan processes are remain locked and even it may lead to the problem of deadlock for a period of time as shown in the figures 2(a) & 2(b) below. Here no preemption happens until the orphan computation completes or the orphan process is killed.



**Fig.2(a). Deadlock by no preemption of CPU**

In Fig.2(a)., CPU is held by the orphan process P1 and the process P2 is waiting for the CPU. Unfortunately, the CPU can be allocated to P2 only after execution of P1 or after deliberate killing of process P1.



**Fig.2(b). Deadlock situation by locked Database**

Whereas in Fig.2 (b)., the database is locked by the orphan process P1 and the process P2 wants a lock on the same. The lock cannot be granted for P2 because P1 is having a lock with database. So, a deadlock occurs till completion of the execution of P1. Forced preemption can be possible only by killing the orphan process P1 and hence all the resources held by P1 can be freed.

The resources locked by the orphan computations not only the CPU but may also be the memory, databases etc. More importantly the valuable CPU time to compute the unwanted computation of the orphan process is wasted. So, the orphan processes slow down the entire performance of the distributed systems.

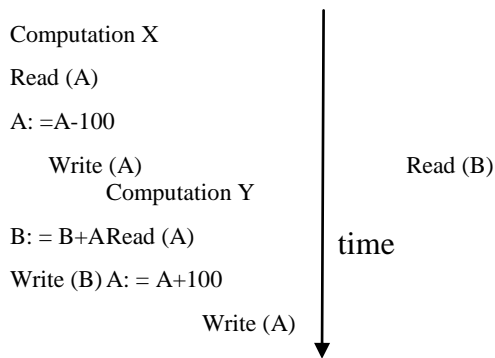
#### 3.2. Data inconsistency

As it has been discussed above, the orphan process leaves data in an inconsistent state. Data inconsistency exists when different and conflicting versions of the same data appear in different places of the system. Data inconsistency creates unreliable information, because it will be difficult to determine which version of the data is correct.

Consider a situation where two data variables (Figure 3), A and B are accessed from the database by two different computations, say X and Y, which are initiated by two different processes. After the initiation of the request X, its parent process is crashed and the computation X becomes an orphan and continue to execute and makes changes in the values of A and B. After some time, as shown in the fig. 3, the computation Y starts and it reads the value of A and does its calculations. In the meanwhile the transaction Y is a successful one and it has read the value of A which has just updated by the computation X. But, being an orphan process, the transaction X should be rolled back. Now the value of A becomes an inconsistent one. Again, the computation Y works with the old value of A and subsequent transactions should produce unexpected and unwanted results. See, what a disastrous situation made by a single orphan!

Suppose a situation where the computation X makes another RPC to get a service from any other server and the server itself again makes another request to any other server and hence a chain of transactions or we say nested transactions. Then, after all these chain of requests the transaction X becomes an orphan and the situation becomes more vulgar. It is concluded that the inconsistency made by an orphan process is exponential!

The only solution is to kill the orphan immediately after their birth. Not make any wait to kill them!



**Fig.3. Two parallel computations X and Y and X becomes an orphan during its execution**

#### 4. PROBLEM WITH BANKING TRANSACTION: A CASE STUDY OF ORPHAN COMPUTATIONS IN DISTRIBUTED SYSTEMS.

All of us are well aware of our banking transaction system in our country. It is a good example of distributed computing system. Here the databases are distributed and users are getting real feeling of single system. Here consider a scenario where you are using the ATM machine to withdraw some money from your valuable savings bank account. While you are about to collect the money from the ATM machine, the machine fails due to any problems like network failure or any technical failure and you do not able to get the money back. The result here is that the amount you needed to withdraw has already been deducted from the account and the same time you haven't received the money. This type of situation might have been faced by us in our day to day life. To get back our money back into our account we have to manually inform the corresponding bank branch where we have the account. It is the real problem of orphan computation. See, this type situation-deducting the amount from account without receiving the same at the user end- can be avoided by implementing a correct orphan detection and killing mechanism in the system. If someone is there to look over the entire scenario and to see whether the entire transaction is completed or not? If it is not completed the transaction, it should be rolled back and avoid going back the user to the branch to get credited the account back.

It is put forward a scenario where the orphan processes are the villains. Here the orphan process not only made inconvenience to the customer but it wasted the valuable time of the server by computing an uncompleted transaction and also wasted the associated resources like memory, database, CPU etc.

#### 5. CONCLUSION

It is concluded that the orphans in distributed systems are not a desirable one at all. Moreover, it makes many disastrous situations in our common man's life as we have seen in our case study. Interestingly, the common man only blames the ATM machine or banking system or the computer itself as they never know the orphan computation is behind in this game.

It is also discussed that how orphan processes are making problems with computer resources like CPU, databases etc. and how they lead to deadlock situations in the system. Eventually, as we have seen, the orphan processes slowdowns the performance of the computer. So, the orphan computations should be detected and killed whenever they are born. Never make any wait to kill them.

#### 6. REFERENCES

- [1]. Randy Chow, Theodore Johnson, Distributed Operating Systems and Algorithm Analysis, Pearson Education, India, 2009
- [2]. Andrew S. Tenenbaum, "Distributed Systems-Principles and Paradigms", Prentice-Hall, 2003.
- [3]. Maurice Herlihy, Nancy Lynch, Michael Merritt, William Weihl, On the Correctness of Orphan Management Algorithms, Journal of the Association for Computing Machinery, Vol 39, No.4, October 1992 pp 881-930.
- [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 India, 2008.
- [7]. Tanenbaum, A.S., Distributed Operating Systems, Pearson Education, India, 1995
- [8]. A.D. Birrell and B.J. Nelson, "Implementing remote procedure calls", ACM Trans. Comput. Syst., Vol. 2. no.1, pp. 39-59, Feb 1984.
- [9]. Kai Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw Hill International Edition.
- [10]. Ramez Elmasri, shamkant D Navathe, Fundamentals of Database Management Systems, 5<sup>th</sup> Edition, Pearson Educations, New Delhi.
- [11]. Bipin C. Desai, An Introduction to Database Systems, Galgotia Publications Pvt. Ltd, New Delhi, 2000.
- [12]. 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.
- [13]. 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.
- [14]. Shamsudeen. E, V. Sundaram, An Approach for Orphan Detection, International journal of computer applications, Vol.10.No.5, 2010.pp28-29.
- [15]. Shamsudeen. E, V. Sundaram, Time Stamp Based Global log and monitor approach to handle orphans in distributed systems, international journal of computer science and network security, Vol 11 No.8, 2011, pp 123-125.