International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

SIMULATION OF VIRTUAL SYNCHRONY IN A DISTRIBUTED ENVIRONMENT

Arun.G[1], Dr. Jasmine K.S[2], Dr. Sumithradevi K.A[3]
[1] 5th Sem MCA, Dept of MCA, R.V. College of Engineering, Mysore Road, Bangalore-59
[2] Associate Prof, [3] Prof, Dept of MCA, R.V. College of Engineering, Mysore Road, Bangalore-59
[1]arubca@gmail.com, [2]jkjasmine1@gmail.com, [3]sumithraka@gmail.com

Abstract

Cloud computing is becoming revolutionary these days and hence making the future of IT. Basically all the IT companies data are being shifted on to cloud as it decreases their investment for servers and its maintenance. There are many vendors which provides cloud space for the IT which can store massive data where these clouds are formed by thousands (if not hundreds of thousands) of servers. In such systems, scalability is the main aspect and this, in turn, compels application developers to replicate various forms of information. By replicating the data needed to handle client requests, many services can be spread over a cluster to exploit parallelism. Servers also use replication to implement high availability and fault tolerance[3] mechanisms, ensure low latency, implement caching, and provide distributed[7] management and control. So virtual Synchrony supports these applications and is helpful for a cloud environment. The main objective of the paper is to show how virtual synchrony is feasible for the said purpose. Our work shows the simulation of various scenarios which can be solved using virtual synchrony by considering some sample scenarios in a distributed network environment.

***Keywords*-- Cloud computing, Fault tolerance, co-operative caching, unbreakable tcp/ip, Event handling, peer to-peer communication, Locking, Virtual Synchrony**

I. INTRODUCTION

Virtual synchrony [3] is an inter process message passing technology in a distributed execution model that guarantees a very strong notion of consistency. Applications can create and join groups, associate information with groups , send multicasts[3] to groups (without knowing the current membership), and will see the same events in equivalent orders, The highest event rates are reached when events are very small and sent asynchronously. In such cases an implementation can pack many events into each message it sends. Peak performance also requires network support for multicast, or an efficient overlay multicast. permitting group members to update the group state in a consistent, fault tolerant[1] manner. Virtual synchrony systems allow programs running in a network to organize themselves into process groups [2], and to send messages to groups. Each message is delivered to all the group members, in the identical order, and this is true even when two messages are transmitted simultaneously by different senders. Application design and implementation is greatly simplified by this property: every group member sees the same events (group membership changes, and incoming messages) and in the same order.
Distributed computer systems often need a way to replicate data for sharing between programs running on multiple machines, connected by a network. Virtual synchrony is one of three major technologies for solving this problem.

Each process group has a name, visible within the network. A single application program can connect itself to many groups at the same time. In effect, a process group becomes an abstraction for sharing data, coordinating actions, and monitoring other processes. The term virtual synchrony refers to the fact that applications see the shared data evolve in what seems to be a synchronous manner. This form of synchronization [4] is virtual because the actual situation is more complex than seems to be the case from a programmer's perspective. Like a compiler that sometimes reorders the execution of instructions for higher performance, or an operating system that sometimes stores random access memory on disk, Moreover, although we've described the virtual synchrony model and uses, problems involved with solution in below chapters.

II. ISSUES ADDRESSED WITH VIRTUAL SYNCHRONY

There are six main issues which are addressed in the paper with the help of virtual synchrony. They are :

- Fault Tolerance.
- Event Notification.
- Peer to Peer mechanism.
- Unbreakable TCP/IP Communication.
- Co-Operative Caching.
- Locking.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

2.1. Fault Tolerance

A group can easily support primary-backup forms of fault-tolerance, in which one process performs actions and a second one stands by as a backup. Even fancier is the "coordinator/cohort"[7] model, in which each request is assigned to a different coordinator process. Other processes in the group are ranked to serve as a primary backup, secondary, etc. Using the virtual synchrony model, it is relatively easy to maintain fault-tolerant replicated data in a consistent state. One can then build all sorts of higher level abstractions over the basic replication mechanisms. Virtual synchrony replication is used mostly when applications are replicating information that evolves extremely rapidly.

The kinds of applications that would need this model include multiuser role-playing games, air traffic control systems, stock exchanges, and telecommunication switches. Of course, there are other ways to solve the same problems. For example, most of today's online multiuser role-playing games give users a sense that they are sharing replicated data, but in fact the data lives in a server on a data center, and any information passes through the data centers. Those games probably wouldn't use models like virtual synchrony, at present. However, as they push towards higher and higher data rates, taking the server out of the critical performance path becomes important, and with this step, models such as virtual synchrony become valuable. The flexibility associated with this limited form of event reordering permits virtual synchrony platforms to achieve extremely high data rates while still preserving very strong fault-tolerance and consistency guarantees.

2.2. Event Notification

These are interfaces that let applications publish event messages, tagging them with topic names. Applications can subscribe to a topic, or a pattern that matches many topics, specifying a function to be invoked when a matching message is received. Virtual synchrony standardizes the handling of group membership[2]: the system tracks group members, and informs members each time the membership changes, an event called a view change. when installing a new view that adds one or more members to a group. Notice that state transfer can be thought of as an instantaneous event: even if a multicast is initiated concurrently with a membership change, a platform implementing virtual synchrony must serialize the events so that the membership change seems atomic and the multicast occurs in a well- defined view. By using Multicast[7] without knowing its current membership (indeed, without even being a member).

Multicast events are ordered with respect to one- another and also with respect to group view events, and this ensures that a multicast will be delivered to the "correct" set of receivers. Every process sees the same events in the same order, and hence can maintain a consistent perspective on the data managed by the group, but event orderings sometimes deviate from synchrony in situations where the processes in the system won't notice. These departures from synchrony are in situations where two or more events commute.

Another, simpler approach might be to send an "identify yourself" message to the multicast address all the hosts would respond by sending their most recent update message, and perhaps (in the absence of any kind of "registry") a message containing the URI of a file containing more detailed information about them.

2.3. Peer-to-Peer mechanisms

Every member knows the identity of every other member. Groups can also be a useful infrastructure for implementing swarm algorithms[9] in virtual synchrony and can be achieved by using GBCAST[10].

*2.4. Unbreakable TCP connections***.**

The idea here is that a client can make a TCP(transmission control protocol)[3] connection to a whole group. Using multicast, the state of the server-side TCP connection can be replicated, so that even if one server crashes, others are able to take over its roles, transparently. In such situations the implementation might take advantage of the extra freedom (the relaxed ordering) to gain higher performance.

2.5. Locking

Many systems need some form of locking or synchronization mechanism. Locking can easily be implemented on top of a virtual synchrony subsystem. For example, a system can associate a token with each group, and make the rule that to hold the lock, a process must gain "ownership" of the token. A multicast is used to request the lock: every member of the group will thus learn of every request. To release a lock, the holder selects the oldest pending request, and multicasts a message releasing the lock to the process that issued that oldest request. Every process in the group will thus learn that the lock has been passed, and to whom. Similarly, if a lock holder crashes, every process will learn that this happened, and a leader (usually, the oldest non-crashed group member) can take remedial action, then release the lock.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

2.5. Cooperative caching.

Members of a group can share lists of data they have in their caches. This way, if one process needs an object that some other process has a copy of, the group members can help one-another out and avoid fetching the object from a server that might be distant, overloaded or expensive.

III. PROPOSED SOLUTIONS

3.1 Event notification

According to current scenario, we can use Multicasting to achieve Event Notification. Multicasting is an attempt to have the best of two worlds: broadcasting and point-to-point links. The idea is that any given host, in addition to having its own Internet address, can belong to a number of "multicast groups"[2]. Each multicast group has it's own special Internet address, IP addresses in a certain range have been set aside for this purpose. When any host sends a message to a multicast address, that message is sent to all the hosts that belong to that multicast group. In effect, it's like broadcasting [6] to a subnet[9] that spans continents. Multicast delivered to all group members as a single.

3.1.1 Difficulties involved in Multicasting

Atomic multicast protocols [8]- All recipients see the same order for events such as deliveries, failures, recoveries, and etc. But this is expensive and sometimes too strong.

In Reliable multicast [6] - Each member of the group should get the message Reliable point-to-point (TCP) channels don't suffice What, if the sender crashes, or a new process joins during message delivery.

Weak reliable multicast [3] - We assume that the groups remains unchanged during the given message delivery. We assume also that the sender knows all receivers Message numbering & history buffer at sender suffices.

3.1.2 Proposed Solution 1

UDP, TCP and Multicast are not good enough so we can achieve Event Notification in Virtual Synchrony by using Synchronous Execution Model(Close Synchrony using Asynchronous protocols). Using GBCAST[10](Addition of ABCAST[11] and CBCAST[12]) and also we can implement using Gossip Protocols[13].virtually synchronous groups can't suffer "split brain[4][3]" problems.

When a new member joins a group, it will often need to learn the current state of the group – the current values of data replicated within it. This is supported through a state transfer:

3.2 Fault-tolerance with co-operative caching

Since failures are rare, the effect is that a group with *N* members can potentially handle *N* times the compute load. Yet if a failure does occur, the group can transparently handle it.

Different Types of Failures:-

Crash failure:- A server halts, but is working correctly until it halts

Omission failure:- A server fails to respond to incoming requests

Receive omission:- A server fails to receive incoming messages

Send omission :- A server fails to send messages

Timing failure:-A server's response lies outside the specified time interval

Response failure:- The server's response is incorrect

Value failure:- The value of the response is wrong

State transition failure:- The server deviates from the correct flow of control

Arbitrary failure:- A server may produce arbitrary responses at arbitrary failure times

Virtual synchrony is a weaker model relates to exactly what happens when some process crashes. In virtual synchrony, the group continues executing as if no message was ever sent. After all, there is no *evidence* to the contrary. Example two process p and q have both crashed, so they won't behave in a manner inconsistent with the model. Yet it is possible that q received p's message and delivered it to the application right before the crash. So there is a case in which virtual synchrony seems to lie: it behaves as if no message was sent, and yet the crashed processes might actually have exchanged a message.

3.2.1 Proposed Solution 2

The above scenario never happens in Paxos[5] or transactional systems[5], which makes them a good match for updating database files on a disk. if a server holding any data and that may crashing, any data it collected prior to crashing will still be valid, except to the extent that it missed updates delivered to the other group members while it was down. The cost of this guarantee is, however, quite high.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

Asynchronous Paxos[5], and transactional systems, impose a long delay before any process can deliver a message. First, these platforms make sure that the message reaches all of its destinations, asking them to delay the incoming message before delivering it. Only after the first step is completed are recipients told that it is safe to deliver the message to the application. In one variant on these models, the platform only makes sure that majority receive the message, but the delay is comparable.

3.2.2. Problems involved in Fault tolerance

1. Interference with fault detection in another component. Another variation of this problem is when fault-tolerance in one component prevents fault detection in a different component. For example, if component B performs some operation based on the output from component A, then fault-tolerance in B can hide a problem with A. If component B is later changed (to a less fault-tolerant design) the system may fail suddenly, making it appear that the new component B is the problem. Only after the system has been carefully scrutinized will it become clear that the root problem is actually with component A.
2. Reduction of priority of fault correction. Even if the operator is aware of the fault, having a fault-tolerant system is likely to reduce the importance of repairing the fault. If the faults are not corrected, this will eventually lead to system failure, when the fault-tolerant component fails completely or when all redundant components have also failed.
3. Cost. Both fault-tolerant components and redundant components tend to increase cost. This can be a purely economic cost or can include other measures, such as weight.
4. Inferior components. A fault-tolerant design may allow for the use of inferior components, which would have otherwise made the system inoperable. While this practice has the potential to mitigate the cost increase, use of multiple inferior components may lower the reliability of the system to a level equal to, or even worse than, a comparable non-fault-tolerant system.

IV. Results Obtained And Suggstion For Future Study

In our work, simulation is performed with NS2 to show how we can control the overall issues in the distributed systems and track them too as when and where they have occurred, when they have various clusters of systems and servers connected in the star network topology. Here we explain how the data packets are transferred across various clients and servers, the details are explained briefly below.

4.1. Locking:

For this issue as and when there is a requirement the client sends the request to the server asking for the data and in reply server sends the acknowledgement to the client saying whether the server is free and the data requested is not used by any other client. If so happens that the client requests for the same data which is being shared by other client the server puts the request into the queue and sends the acknowledgement saying the server is busy presently and the request will be handled after the present execution is completed and at that ti.

4.2. Co-operative Caching:

For this issue many servers are sharing the data to serve the client request here distance matters if the requested servers nearest neighbour having the requested data then nearest neighbour will send that data to client with reliable distance and time.

4.3. Unbreakable TCP/IP

For this issue, we represent the link between the two server and client with the red lines which tell that the built connection is strong and the connection is terminated after the complete transfer of data from server to client

4.4. Event notification

For this issue, we use acknowledgements, where clients sends acknowledgement saying the request sent by the client is received by the server and then client also sends the acknowledgement saying that it has received the data which has been sent by the server.

4.5 Fault tolerance

To solve this issue, there is a backup of the server is made very often with respect to the clock and whenever the system goes down, the data is copied to other nearest neighbour and the server is rebuilt and then the data is sent to the client from where it had paused and the details of the sent data are stored in the nearby server as for the reference.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

4.6 Peer to Peer mechanisms:

For this issue, firstly a particular path is created when the client asks the server to send some information and in reply if server sends the acknowledgement saying it is free, then the connection between the two nodes is established and the data transfer between the two nodes begin and the client sends back the acknowledgement saying it has received the data packets.

Following snapshots describes the simulation outputs.

V. RESULTS OBTAINED

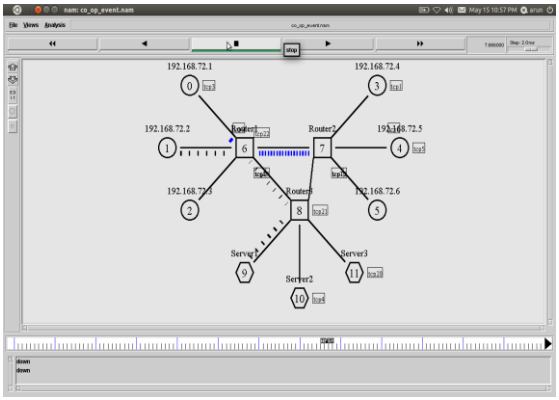

Fig 1: Locking event notification

Figure 1 describes the locking usage in group between client 1,2,3,6,7,8 and server1. i.e if client2 communicating with server1 for requesting a particular resource at that time communication should be locked, no other communication for same purpose will not provided by sending locking event over the group depicted with blue lines event notification. Blue lines shows communication event and black indicates requested information flow with locking, respect to timing constraints refer trace file.

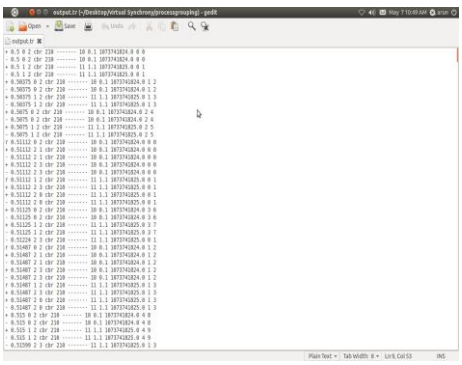

Fig 2: Tracing file

Here in fig 2 trace file describes all scenarios with a time constraint.

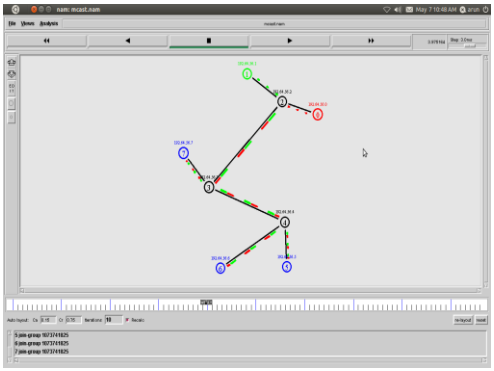

Fig 3: unbreakable tcp/ip

The figure 3 shows unbreakable tcp/ip for reliable communication shown. RED type of communication link is there between a group, with router2 and router 3 depicted with red lines.

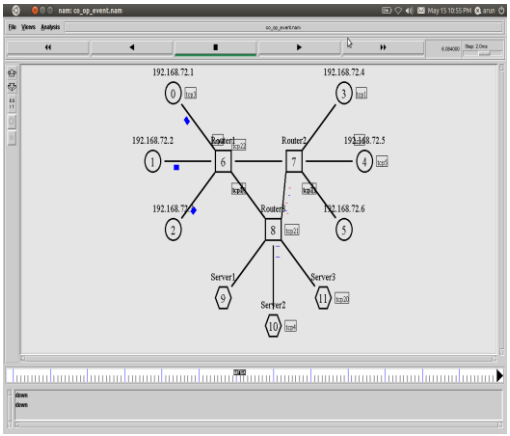

Fig 4: Co-operative caching with fault tolerance

In cooperative caching, server1 failed after some time then it is nearest neighbour i.e server2 will create back up and communication will resume between previous group router 3,router 2 and client 3, failure depicted by red lines and resumed communication showed with blue lines .Server 1 to client 3 communication is happening at the same time server 2 creating the backup of server 1 in figure 5.4,And also if one client asks for a resource at the same time if another server holds that then by considering time constraints nearest server will serve the requested resource it will be tracked in trace file. Trace files will be generated for each activity.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

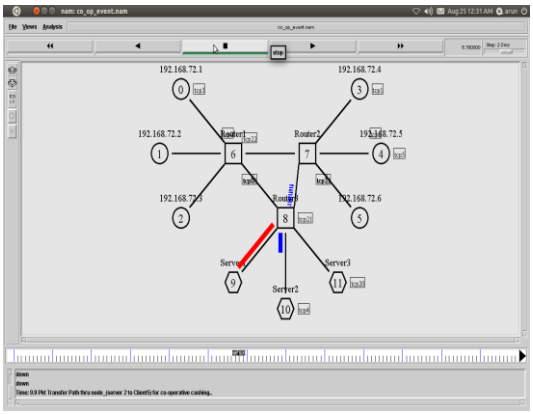

Fig 5: Peer to Peer communication

In figure 5, peer to peer communication is achieved with process group1,group2 and group3.group1 contains node 1,2,3,7,4,6 and 5 communication showed with green lines. Group2 with nodes 0,2,3,4,5,6 by red lines.

VI. CONCLUSION

It's important that all the systems remain synchronized; in other words we can achieve the virtual synchrony over distributed system architecture when all the issues are resolved based on the main six issues and that is shown how it could be possible in this paper. The existing protocols should be used wherever possible, and NTP (the Network Time Protocol) should be sufficient for this purpose. Due to insufficient time, we bound ourselves to minimum set of rules applied on the operations. Generalizing the concept, we can make many improvements and a full fledged package can be thought of.

Hence, this paper allows engineers, researchers to test scenarios that might be particularly difficult or expensive to emulate using real hardware- for instance, simulating a scenario with several nodes or experimenting with a new protocol in the network. We conclude that protocols implementing virtual synchrony can achieve high update and group membership event rates and can support some of the very demanding uses in the distributed system.

REFERENCES

[1] K.P. Birman and T. Joseph. Nov. 1987 "Exploiting virtual synchrony in distributed systems".

[2] K.P. Birman Dec. 1993 "The process group approach to reliable distributed computing". Communcations of the ACM (CACM) 16:12.

[3] K.P. Birman. Springer Verlag 1997. Reliable Distributed Systems: Technologies, Web Services and Applications.

[4] Leslie Lamport 1998 "The part-time parliament". ACM Transactons on Computing Systems (TOCS), 16:2.

[5] K. P. Birman July 1999 "A review of experiences with reliable multicast".

[6] Gregory V. Chockler, Idit Keidar, Roman Vitenberg 2001 "Group communication specifications: a comprehensive study". ACM Computing Surveys 33:4.

[7] Andrew S. Tanenbaum, Maarten van Steen 2002 Distributed Systems: Principles and Paradigms, 2nd Edition.

[8] Andre Schiper July 2005 "Practical Impact of Group Communication Theory.

[9] Gredler, Hannes; Goraiski, Walter 2005. The complete IS-IS routing protocol. Springer. pp.1. ISBN 1-85233-822-9.

[10] spinroot.com/spin/Workshops/ws02/pascoe.

[11] www.cse.chalmers.es/edu/cource/.../lib/12_Atomic Broadcast.

[12] www.acad.ro/sectii2002/proceedings/doc2012-3/10-Babamir