IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727Volume 7, Issue 5 (Nov. - Dec. 2012), PP 15-18
www.iosrjournals.org

Failure Detection and Revival for Peer-To-Peer Storage Using Mass

V.Vimala Devi [1], K.Sampath Kumar[2], V.Rajesh Kumar[3], A.S.Lakshmi[4]
[1,3,4](Assistant Professor, Department of Computer Science and Engineering)
[1,3,4](P.S.R.Rengasamy College of Engineering for Women, Sivakasi, Tamil Nadu, India)
[2](Head & Professor in Computer Science and Engineering)
[2](PGP College of Engineering and Technology,Namakkal, Tamilnadu)

Abstract: *Sustaining a given level of data redundancy is a basic requirement of peer-to-peer (P2P) storage systems to make certain desired data availability, additional replicas must be created when peers fail. Because the majority of failures in P2P networks are short-lived (i.e., peers return with data intact), reliably distinctive permanent and transient failures, however, is a demanding task, because peers are apathetic to probes in both cases. This paper proposes MASS (Maximum Available Server Selection), an algorithm that detects the failure and redirect the user services by the available server.*

Keywords:*Failure detector, P2P storage, availability, Failure Recovery*.

I. Introduction

Peer-to-Peer (P2P) storage networks aim to aggregate today's resource-abundant computers to form large, decentralized storage systems. Of the abundant P2P storage systems urbanized in recent years, prototypes including Total Recall [3], Friend store [8] and commercial services like Wuala [2] and Clever Safe [1], all of them faced the long-standing problem of enforcing required data availability from participating peers. To reach availability target in unreliable networks, storage systems replicate data across multiple peers (using replication or erasure coding). Even if some replicas (or fragments) become unavailable due to their hosts failing, the object is still available by accessing other online replicas. Over longer periods, the system must produce new replicas as required to reimburse for others lost to peer failures. Since replica production consumes a large amount of bandwidth, maintaining availability incurs a vital cost on storage systems [4]. Intuitively, aggressive replication incurs high bandwidth costs that may cripple the entire system, while reducing the number of replicas generated might result in an unacceptably low level of availability. Therefore, the challenge in building P2P storage systems is to carefully navigate this cost-availability trade-off, by reducing maintenance cost as much as possible while maintaining the desired level of availability.This challenge is further complicated by the fact that peer failures can be either transient or permanent. Data are lost following a permanent failure, and the expected level of redundancy must be restored by creating new replicas. Absolutely not, an object with adequate replicas can stand transient failures without sacrificing availability, given that a peer undergoing a short-lived failure will retort the network and fetch back its stored data. Hence, an ideal system would recognize peer failures and only reproduce data following enduring ones. Undesirably, reliably distinguishing permanent and transient failures turns out to be a discouraging task, since they cannot be distinguished using probes [9].To address this problem, this paper proposes MASS, an algorithm that provides better protection against transient failures. In particular, given a replica group (i.e., all peers hosting replicas of the same object) and downtime of these peers, MASS can detect the number of remaining replicas in the group, i.e., the number of replicas residing on online peers or peers experiencing transient failures. Storage systems can use this detected number of replicas to determine whether data recovery is necessary to reach the desired replication level and redirect the user service with last session page by other online peers.

II. Related work

Types of Failures
A failure occurs when an actual running system deviates from its specified behavior.

A. Muteness Failure

Muteness failures are malicious failures in which a process stops sending messages but might continue to send other messages. When muteness failure occurs the service will stop executing its designed features but might still be able to generate likeness messages such failures cannot be detected by crash failure detectors. Adopting the muteness failure detecting algorithm in which proposes a protocol that forces the monitored service to send "Iam-not-mute" message to the muteness failure detector periodically when service is not mute but stop sending such messages when a muteness failure occurs.

B. Timing Failure

Timing failure occurs when a service response lies outside the specified time interval. Example if the service-hosting machine or network is overloaded or some other resources on which the service depends are overloaded then the service response might be delayed and a timing failure [5] might occur. In order to detect a timing failure recording the time when the conversation between a service pair starts can be adopted. If the service instance cannot return the answer before the specified deadline is regarded as a timing failure. Moreover there are more sophisticated timing failure detectors such as the one reported in which uses group communication to detect timing failure in a quasi-synchronous system or the timely computing base model can [7] deal with timeliness requirements without synchronized clocks.

C. Omission Failure

When a service fails to send a response or receive a message an omission failure occurs behaves as a communication failure will cause message transmission fail. The simplest way to detect omission failures is to enable the service to provide failure information. If the service can throw a fail to send or fail to receive message exception or send this information to the failure detector then the failure is regarded as an omission failure. D. QoS Failure

A service even if it provides a correct result might still fail to meet the consumers desired level of service fails to satisfy a specified property by the service consumer by a certain level Of QoS constraints. QoS failure can be tracked by matching the given QoS specification with the QoS delivered by the service.

E. Response Failure

Response failure occurs when a service response is incorrect. In general, response failures can be separated into two types. The first type is value failure: the response value is wrong; the second type is state transition failure: the service deviates from the [6] correct flow of control [7]. To detect value failure, voting algorithms can be adopted if multiple service replications are deployed. To detect state transition failure, the service design specification should be available to check whether a service has deviated from its estimated state or not.

F. Partial Failure

For a composed application, a component failure may result in a partial failure of the composed service. Identifying such a partial service failure still remains challenging. For a composed service, due to service internal fault-tolerance policies, partial failure might not be visible externally by a failure detector, which only observes the composed service. In order to discover such partial failures, sensors must be implemented at the atomic component level to way the status information of each atomic component of a composed service. The implementation of the sensor for a component should be based on the failure mode that the sensor is concerned with.

Failures, Fault Tolerance and Dependability

The software and hardware may contain internal or external bugs, errors that can make the run-time services unstable. Computer system shows that bugs are one of the important reasons for system crashes; faults are accepted as inevitable and may lead to a system failure. To improve the critical systems survivability when failures occur used the fault tolerance mechanisms. Fault tolerance is the ability or the property to enable a system to continuously operate correctly when some abnormal internal or external events occur. Dependability is one of the most important issues for computer systems which is a complex attribute, the concept of dependability as the property of a computer system such that reliance can justifiably be placed on the service it delivers. In addition by recording the lifetime information of a system, the systems dependability can be described quantitatively. Dependability of a system can be measured according to the reliability, availability, consistency, usability and security. In order to simplify the measurements which are related to failure detection.Reliability can be clear as the probability that the system will run correctly in a specified operating environment up until time t ($t>0$). Availability can be definite as the probability that the system will be set at time t. stability can be defined as the probability that the system will return to normal operation correctly after a failure has occurred within a specified operating environment within time t. Highly available system which requires the system to be accessible with correct maneuver most of the time or the highly consistent system which requires fast recovery of the system after failures occur. The system comprising a failure detector and a monitored crash-recovery target. It extends failure detectors to take account of failure recovery in the target system. This involves extending QoS measures to include the recovery detection speed and proportion of failures detected. It also extends estimating the parameters of the failure detector to achieve a required QoS to configuring the crash-recovery failure detector. Which investigate the impact of the dependability of the monitored process on the QoS of our failure detector.

III. Proposed Methodology

Peer-To-Peer network systems, various Types of failure may occur through the execution. This paper addresses failure detection and recovery. In addition, many researchers have haggard their attentions on the Crash failure detectors, implementations and crash detection algorithms [8]. However, most previous work on the Crash failure detection is based on the crash-stop or the fail-free assumption and relies on predicting the liveness message transmission behavior and it doesn't consider Redirecting failed user service to which server having lowest user Availability to reduce the overload and estimating a suitable timeout threshold to achieve a better MASS failure detection.

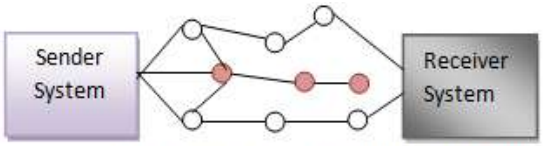

Fig. 1: Network topology with node failures

In contrast, Process or service as recoverable, since many fault-tolerance techniques can be adopted to achieve such recovery. For high level applications, a more realistic crash failure model would be crash-recovery.Fig.1 shows the problem definition where Red Color Node represents the network failure. The solution implements the Sender sends some data to receiver where some nodes exist between networks of sender to receiver, where different port numbers assigned to all these intermediate nodes. When program begins a file accepted by sender to transmit to receiver MASS (Maximum Available Server Selection Policy Algorithm) monitors the transmission for failures. If any failure detected MASS reports the specific node where data lost and recovers the data from previous node. Retransmission from recovered node done by MASS this process continues until all data packets safely reach the destination. For successful delivery to destination all the intermediate nodes maintain a copy of data to ensure data Transmission reliability over network.

This method is most useful in cases of frequent interaction between the client and the same server set as the status for a given server set stored in the client is in that case always maintained up-to-date.The main problem addressed in this system is maximizing the probability of successful operation with the current transmission, thereby minimize the average number of attempted servers until success.The algorithm proposed in solves the problem by exploiting the dynamically obtained information on the last server access moments and the corresponding activity status of servers in a server set. The user can work without any disturbance when one server fails doing its work the other server will replace it and perform the same task.

The concept is covered by following processes

- User
- Admin
- Healing

A. User Process

User to register as a valid user by Submitting their profile to the Administrator. Once the User will be registered the user can send mail and View the Received Mail. The Send item can store the mail send by the users.

B. Admin Process

Admin can do the Web service configuration. The admin had the power to validate Server settings and Connection. If any problem occurs to the user, admin can resolve through availability server by using this configuration.

C. Healing Process

Through this process admin can interchange the servers for the users. Upon failure detection, the fail-over mechanism resends the SIP request to the new server selected according to an SSP. This may potentially be repeated until all servers have been attempted. The goal of the proposed algorithm is to reduce transaction control time. The MA SSP makes use of the assumption that the server whose last known up time is closest to the actual time, is most likely to be up at the actual time.

MASS is a simple extension of round robin. It assigns a certain weight to each server. The weight indicates the server's processing capacity. This SSP may also be dynamic if it can evaluate individual servers' capacities and their loads occasionally.

Algorithm

Step 1: When server gets failed that time MASS Algorithm is enabled.

Step 2: Finding Available Server
Step 3: Detecting User Availability of Each & Every Server
Step 4: Redirecting failed user service to which server having lowest user
Availability
Step 5: Suppose if available server having same user means we find out highest bandwidth server
Step 6: The Redirected Server starts service user lost session page.

IV. Conclusion and Future Work

The maximum availability (MA) SSP is projected to recover session control performance in scenarios with server and communication failures. Through this system more users can access a server at a time if the server gets busy the user will be responded by the availability server and all the process is done through a single domain. In future, this will be updated to response the users while accessing various domains.

References

[1] CleversafeInc.,*http://www.cleversafe.org/dispersed-storage*,2011.
[2] Wuala, Peer-to-peer storage system, *http://wua.la/en/home.html*, 2010.
[3] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G.M. Voelker,"Total Recall: System Support for Automated Availability Management," *Proc. Symp. Networked Systems Design and Implementation* (NSDI '04), 2004.
[4] C. Blake and R. Rodrigues, "High Availability,Scalable Storage,Dynamic Peer Networks: Pick Two," *Proc. Ninth Workshop Hot Topicsin Operating Systems* (HotOS '03), 2003.
[5] R.Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems," *IEEE Trans. Software Eng., vol. 13, no. 1, pp 23-31*, Jan. 1987.
[6] D. Manivannan and M. Singhal, "A Low-Overhead Recovery Technique Using Quasi Synchronous Check pointing," *Proc. IEEE Int'l Conf. Distributed Computing Systems*, pp. 100-107, 1996.
[7] J.C. Laprie, A. Avizienis, and H. Kopetz. Dependability: Basic Concepts and Terminology. *Springer-Verlag New York, Inc. Secaucus, NJ*, USA, 1992.
[8] D.N. Tran, F. Chiang, and J. Li, "Friendstore: Cooperative Online Backup Using Trusted Nodes," *Proc. First Workshop Social. Network Systems (SocialNets '08*), 2008.
[9] H. Weatherspoon, B.G. Chun, C. So, and J. Kubiatowicz, "*Long-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach,"* Computer, 2005.
[10] Protector: A Probabilistic Failure Detector for Cost-Effective Peer-to-Peer Storage Zhi Yang, Jing Tian, Ben Y. Zhao, Wei Chen, and Yafei Dai, Member, *IEEE Transaction on parallel and distributed systems*, VOL. 22, NO. 9, Sep 2011