The Technique of Time Synchronization Algorithms in Distributed File System

Abstract

Time synchronization is critical because every aspect of managing, securing, planning and debugging a network involves determining when events happen. Without synchronized time, accurately correlating log files between these devices is difficult, even impossible. To reduce confusion in shared file systems, it is important for the modification times to be consistent, regardless of what machine the filesystems are on. Synchronized clocks are interesting because they can be used to improve performance of a distributed system by reducing communications. This paper discusses about time synchronization techniques. An approach is made from a centralized emblematic network file system which is dependent on a dedicated file server which leads to several routine issues and consistency disorder. This design suggests a system which is scalable and distributed which can be called as “server-less distributed storage system” and also self-synchronization in distributed file-systems was taken as a domain of study in this paper. The most efficient algorithm called ? synchronization algorithm best in terms of time and communication complexity is discussed.

Key words - Information device, Cluster Tree, Communication Network, Transmission range.

Motivation

Time synchronization plays a critical role in Distributed file system which is used regularly because of its efficiency in data and storage sharing options on a network compared to other options.

Objective

Journal work is on mainly discussing a common simulation technique that helps user to carve an algorithm that gives an intuition that it is running in a synchronous network.

Introduction

Distributed system is a system architecture model whose applications are scattered in various places and connected to each other through a computer network. It is commonly used because of the independent failure and shared resources. If a system interruption occurs, the interference only affects the isolated system and not the overall system. The advantages include which support of each file-volume here it is a single copy is provides for read and write whilst all others are “read only replication”. A file volume is represented using a spanning tree. In a network where nodes communicate by message passing. It is radically different from the classical sequential problem, although the most basic approach resembles Bor?vka's algorithm. One important application of this problem is to find a tree that can be used for broadcasting. If the cost for a message to pass through an edge in a graph is significant, a Minimum Spanning Tree can minimize the total cost for a source process to communicate with all the other

processes in the network. Example of a MST: The minimum spanning tree of a planar graph. Each edge is labeled with its weight, which here is roughly proportional to its length.

These are used to construction and maintenance of this spanning tress is carried out using synchronizer algorithms. The term self-stabilization refers to the recovery from any many synchronizer algorithms. Furthermore such type of systems bears the occurrence of frequent and unexpected faults and delivers the system from the unpredictable data corruption and loss. The contained system in the distributed system should be monitored and maintained all the times using synchronization. Synchronization is performed on the system by conducting files detection first and repairing the files if they are identified as unsynchronized. The unsynchronized files emerge due to the addition of features or improvements of the distributed system. Some improvements on this system, such as repairing the file system application in the source server, should be deployed throughout the distributed system. Sometimes, this deployment process cannot be performed properly because of network interruptions, power problems, full hard drive capacity, corrupted files, etc. This happens because files on the distributed system can be unsynchronized from the existing files in the source server. Detecting different files can be done by using a cryptographic hash function namely, SHA1.

Literature Review

In distributed systems, time is a key issue. In reality, each node of such a network ought to have its own internal clock and the time can float uniquely in contrast to one hub to another for a few reasons like temperature varieties or quartz properties[1]. This issue has been outstanding for a long time and has prompted the formation of the NTP protocol[2]. In MANETs [3] (Mobile Ad-Hoc Networks), nodes are moving, involving various disconnections in the network over time. In other words, in a such network, it isn't conceivable to guarantee that a communication is always possible between two nodes. Subsequently, making one (or several) of these nodes a central time synchronization agent (as for NTP) is not suitable.

This paper deals with distributed algorithms in two networks models namely synchronous network model and synchronous network model. The asynchronous network is likewise called as “point to point communication network”. It is well explained with the help of the network and E represents the bi-directional non-interfering communication channels existing between them. The processors have no shared memory and each node has a discrete characteristic. The task of each node is to process messages obtained from the neighbors, to perform local computations and to send messages to the neighbors. These actions are carried in very little less time. The messages which carry only a specified quantity of information are supposed to have constant length, the messages reaches within finite but unpredictable time.

In the case of a synchronous network, messages are sent only at integer times of a global clock. Each processor has access to this global clock. The maximum number of messages that can be sent at a specific pulse and over a communication link is limited to one. The performance of the algorithms is measured using the time and communication complexities. The communication complexity is the total number of messages sent during the execution of pulses transferred from its initial time until its extinction. A characteristic fact in communication networks is the trade-off that occurs between the communication and time [4]. The proposed system design comprises of a synchronizer that is efficient enough to run any synchronous algorithm in any asynchronous network. A new pulse is produced at a node after it has received

all the messages of the synchronous algorithm that has been sent to those nodes by its neighbors at the preceding pulses.

Approach

The alpha synchronizer

In round r, the synchronizer at p sends p's message (tagged with the round number) to each neighbor p' or no-msg(r) if it has no messages. When it collects a message or no-msg from each neighbor for round r, it delivers all the messages. It's easy to see that this satisfies the local synchronization specification. This produces no change in time but may drastically increase message complexity because of all the extra no-msgs flying around.

The beta synchronizer

Centralize detection of message delivery using a rooted directed spanning tree (previously constructed). When p' receives a round-r message from p, it responds with ack(r). When p collects an ack for all the messages it sent plus an OK from all of its children, it sends OK to its parent. When the root has all its acks and OKs, it broadcasts go. Receiving go makes p deliver the queued round-r messages. This works because in order for the root to issue go, every round r message has to have gotten an acknowledgment ? all round r messages are waiting in the receivers' buffers to be delivered. For the beta synchronizer, message complexity increases slightly from M to 2M+2(n-1), but time complexity goes up by a factor proportional to the depth of the tree.

The gamma synchronizer

The gamma synchronizer combines the alpha and beta synchronizers to try to get low blowups on both time complexity and message complexity. The essential idea is to cover the graph with a spanning forest and run beta within each tree and alpha between trees. Specifically:

- Every message in the underlying protocol gets acked (including messages that pass between trees).
- When a process has collected all of its outstanding round-r acks a, it sends OK up its tree.
- When the root of a tree gets all acks and OK, it sends ready to the roots of all adjacent trees (and itself). Two trees are adjacent if any of their members are adjacent.
- When the root collects ready from itself and all adjacent roots, it broadcasts go through its own tree.

As in the alpha synchronizer, we can show that no root issues go unless it and all its neighbors issue ready, which happens only after both all nodes in the root's tree and all their neighbors (some of whom might be in adjacent trees) have received acks for all messages. This means that when a node receives go it can safely deliver its bucket of messages. Message complexity is comparable to the beta synchronizer assuming there aren't too many adjacent trees: 2M messages for sends and acks, plus O(N) messages for in-tree communication, plus O(Eroots) messages for root-to-root communication. Time complexity per synchronous round is proportional to the depth of the trees: this includes both the time for in-tree communication, and

the time for root-to-root communication, which might need to be routed through leaves.

Comprehensive Analysis

Synchronizer Time Message Spanning Tree

Alpha Low Time Complexity High Message Complexity No

Beta High Time Complexity Low Message Complexity Yes

Gamma Fairly Low Time Complexity Fairly Low Message Complexity Yes

Conclusion

Self-synchronization can to be obtained using synchronization algorithms in the distributed file systems. The synchronizer ? has been proved to be a resourceful technique in any synchronization network for designing distributed algorithms. This algorithm lead a path to low complexity algorithms.