Research Journal of Information Technology 2(2): 35-38, 2010
ISSN: 2041-3114

Submitted date: April 30, 2010 Accepted date: June 14, 2010 Published date: September 13, 2010

Using Server Clusterization to Establish Fault-Tolerant Internet Connectivity

[1]O.O. Adeosun, [2]E.R. Adagunodo and [3]E.A. Olajubu
[1]Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria
[2,3]Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria

Abstract: This study discusses the issue of providing tolerance to hardware and software faults in Internet system as well as issues related to clusterization of servers. A replication scheme is presented, and a detailed dependability analysis of this scheme is performed. The proposed model was designed mainly for fault-tolerant internet system where many unrelated applications could compete for hardware and software resources, thereby exhibiting highly varying and dynamic system characteristics. A major feature of the model under consideration is to attempt the adaptive execution of redundant components for a required level of fault tolerance.

Key words: Clusterization, servers, replication scheme, redundant components, dependability analysis, reliability, parallel processing, real-time processing, stochastic modeling

INTRODUCTION

Fault-tolerant computing is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. Fault-tolerance is achieved by applying a set of analysis and design techniques to create systems with dramatically improved dependability. Fault tolerance and dependable systems research covers a wide spectrum of applications ranging across embedded real-time systems, commercial transaction systems, transportation systems and military/space systems - to name a few. The supporting research includes system architecture, design techniques, coding theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real-time processing. These areas often involve widely diverse core expertise ranging from formal logic, mathematics of stochastic modeling, graph theory, hardware design and software engineering.

Replication is one of the oldest and most important topics in the overall area of distributed systems. Whether one replicates data or computation, the objective is to have some group of processes that handle incoming events. If we replicate data, these processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates when we replicate computation, the usual goal is to provide fault-tolerance.

The development of highly dependable computing systems usually requires the combined utilization of a wide range of techniques, including fault tolerance techniques intended to cope with the effects of faults and avert the occurrence of failures or at least to warn a user that errors have been introduced into the state of the system (Anderson, 1985). Clients in high-available systems can re-route transparently faulty requests to another application server (or failover). To implement failover, it requires replicating service on multiple nodes, storing distributed checkpoint and synchronizing replicas.

Using the integrated approach to the incorporation of fault tolerance, some of existing software fault tolerance approaches will be extended to the treatment of both hardware and software faults (hybrid faults). Two typical schemes are taken into account - recovery blocks (Randell, 1975) and Self-Configuring Optimistic Programming (SCOP), an adaptive scheme introduced recently by Bondavalli *et al.* (1993). N-version programming (Avizienis and Chen, 1977) is also chosen as a representative of non-adaptive schemes for the sake of comparison. These architectural solutions specially directed to Internet system are then analyzed with respect to dependability, availability, accessibility and restartability.

Similar works, in particular, Laprie *et al.* (1990) presented a set of hybrid-fault-tolerant architectures and analyzed and evaluated three of them. Their architectures are based on a fixed set of hardware components and not related to the dynamicity of hardware resources available as well as the efficiency issues. Such architectural solutions cannot well match the characteristics of dependable Internet system in which the resources must

Corresponding Author: O.O. Adeosun, Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria

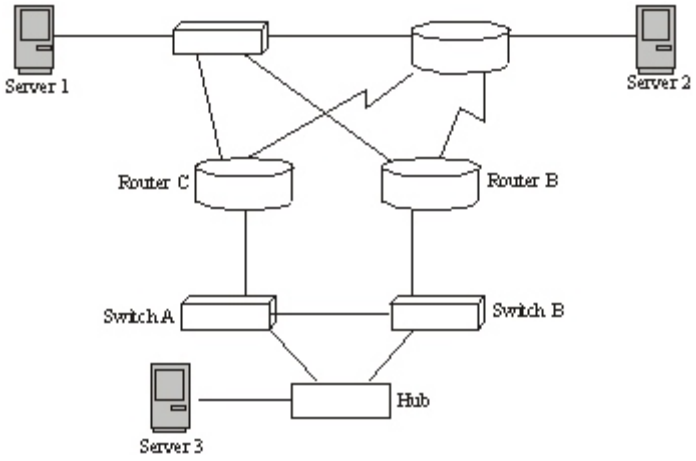

Fig. 1: Replicated servers model

be competed by many unrelated but concurrent service requests. In such varying environments the architectures with the fixed requirement to hardware components are either inefficient or infeasible.

Fault-Tolerant Internet Connectivity Model: Guerraoui and Schiper (1996) opine that group communication enables encapsulating a set of entities that cooperate to achieve some common service. A group has a logical address, which allows clients ignore the existence of its members. In Fig. 1, a set of replicated servers composes the group. All replicas must provide access to the same methods and have to maintain the same state. To achieve this, there should be strong consistency. This will enable read-one-write-all replicas.

From primary to backup replication: In the replicated service, one of the replicated servers (the primary) executes a transaction locally. Many classical approaches to replication are based on a primary/backup model where one device or process has unilateral control over one or more other processes or devices. For example, the primary might perform some computation, streaming a log of updates to a backup (standby) process, which can then take over if the primary fails, that is, it forwards updates to all other group member (backups) using the total order multicast primitive (TOCAST) (Guerraoui and Schiper, 1996). This primitive ensures that updates are delivered in the same order by all correct processes that work according to their specification. The termination property of the TOCAST assures the distributed system progress despite of failures, as well as its non-blocking characteristics. Typically, the primary waits for all backup answers and returns response to the client.

In order to avoid bottleneck, any replica can be enabled to play the primary role. Backup failure is transparent to the requester, but faulty primaries require achieving failover. In this study, a client detects a faulty-primary using timeout and its stub automatically re-routes a faulty request to an alternative server.

The weakness of primary/backup schemes is that in settings where both processes could have been active, only one is actually performing operations. It is true we are gaining fault-tolerance but spending twice as much money to get this property. An outgrowth of this work was the emergence of schemes in which a group of replicas could cooperate, with each process backup the others, and each handling some share of the workload.

Communication model: We use an asynchronous communication in our model; even overload server can be assumed as fault-suspected because there is no way to distinguish between overload and faulty servers. Also, we use an underneath group-communication layer to provide the needed multicast primitive and also server service. The server service manages the replicated servers and detects fault-suspected servers removing them from the group. The group communication layer operates in the presence of message omission faults, processor crashes and recoveries as well network partitions and merges.

Design approach: As the number of nodes in a distributed computation increases, so does the probability for failure. If one thinks of a system as a collection of functionalities that must perform specific tasks, then the design of a survivable system can be thought of as a multistage process. It should be noted that, in a malicious environment, each stage has its limitations.

In traditional fault-tolerance, tolerating faults is typically achieved utilizing the principle of redundancy.

Information redundancy: Usually considers the inclusion of additional information as the basis for fault recovery. A typical example is an error correction code.

Time redundancy: Relies on multiple executions skewed in time on the same node and is often used to mask omissions.

Spatial redundancy: Uses multiple components, each computing a value, and the final value is derived from a convergence function (e.g., majority voting). The resulting N-modular redundant (NMR) system implements a k-of-N system, which implies that the system functions as long as k or more components are fault free. A typical configuration is a triple-redundant redundancy (TM), which is a 2-of-3 system.

Enabling recovery failures and providing failover service to users: Achieving the proposed Internet fault-tolerant service using replicated servers requires treating client-primary as well primary-backups interaction. The model handles client-primary interaction switching of the client requests to alternative server, when the current service is interrupted. The work also handles primary-backup interaction implementing distributed checkpoints. Recover from a failed server is easier. It just requires re-routing clients' requests.

Distributed checkpoint implementation: A distributed checkpoint contains all local snapshots placed in all the replicated servers. Each snapshot holds information about the last executed method, the client who requested this method and the server who executed this method. This follows a distributed checkpoint approach, which multicasts a snapshot from a primary to all other servers. Whenever the primary receives a transactional request (using point-to-point communication) from a client, it updates its own state and multicasts synchronization messages to the backups using the TOCAST primitive. The primary verifies if the distributed checkpoint was successfully established (waiting for all backup confirmation messages) and answers the client.

Backups process the synchronization messages and automatically store updates in their own states to establish the distributed checkpoint and to reflect a single distributed global state. If a server fails, clients are guaranteed access to the same data through the backups. When a client-server connection is closed, all servers remove the information about the distributed checkpoint for that client. Storing this information will enable automatic failover during a transaction execution. The non-finished methods will be executed in another server.

Propagating updates to backups: There are two possible strategies to propagate updates: deferred update and immediate update (Wiesmann *et al*., 2000). In deferred update, transactions are processed locally at one server and are forward to the backups at the commit time while the immediate update synchronizes every transaction across all servers.

Implementation Issues: Two OpenSource projects were identified: Java-Groups (Ban, 1999) and JOnAS (Java Open Application Server) (Danes *et al*., 2000). Our replicated server is been developed to match the two OpenSource. In our model, we changed some classes of the JOnAS to include the TOCAST primitive in the server-side. Replicated servers join the group and use this primitive to setting the distributed checkpoint. We implement the distributed checkpoint selecting, at compiling time, updates to be forwarded during the service execution. An update is assumed to be a method without result (it returns a null value). In the client-side, we modify the client's stub to automatically re-route faulty requests.

CONCLUSION

Transactional systems could benefit from group communication to achieve fault tolerance. The systems are more available for service delivery and multicasting just updates provides good performance, eliminating additional communication rounds.

Also, we expect that replication improves the application response time, when compared with non-replicated application servers, by allowing requests to be handled by several nodes rather than one besides eliminating a single point-of-failure. In addition, deployment and redeployment of new and recovered servers are necessary to maintain the Internet availability.

ACKNOWLEDGEMENT

The authors are thankful to Prof. Adetunde, I.A., the Dean of Engineering, University of Mines and Technology, Tarkwa, Ghana, for his advice, valuable comments, suggestions and for making this article publishable.

REFERENCES

Anderson, T., 1985. Resilient Computing Systems, London, UK, Collins Professional and Technical Books.

Avizienis, A. and L. Chen, 1977. On the implementation of N-version programming for software fault tolerance during program execution. COMPSAC 77, Chicago, IL,pp: 149-155.

Ban, B., 1999. JavaGroups User's Guide, Department of Computer Science, Cornell University, pp: 73. http://JavaGroups sourceforge.net/.

Bondavalli, A., F. di Giandomenico and J. Xu, 1993. A cost-effective and flexible scheme for software fault tolerance. J. Comput. Sys. SC. Eng., 8(4): 234-244.

Danes, A., P. Dechamboux, M. Riveill and G. Vandome, 2000. Technologie a Base de Composants EJB Experience et Perspectives Avec JOnAS. OCM 2000, Nantes, Mai 2000, pp: 11-13. Retrieved from: http://www.objectweb.org/jonas/.

Guerraoui, R. and A. Schiper, 1996. Fault_Tolerance by Replication in Distributed Systems. Departement d'Informatique Ecole Polytechnique Federale de Lausanne, 1996.

Laprie, J.C., J. Arlat, C. Beounes, K. Kanoun and C. Hourtolle, 1987. Definition and analysis of hardware-and-software fault-tolerant architectures. IEEE Comput., 23 (7): 39-51.

Randell, B., 1975. System structure for software fault tolerance. IEEE TSE, Vol. SE-1, No. 2, pp: 220-232.

Wiesmann, M., F. Pedone, A. Schiper, B. Kemme and G. Alonso, 2000. Understanding replication in databases and distributed systems. Proceedings of ICDCS 2000, Taipei, Taiwan, R.O.C., April 2000, pp: 264-274.