TCP-ABC: From Multiple TCP Connections to Atomic Broadcasting[*]

Zhiyuan Shao, Hai Jin, Wenbin Jiang, and Bin Cheng

Cluster and Grid Computing Lab,
Huazhong University of Science and Technology, Wuhan, 430074, China
zyshao@mail.hust.edu.cn

Abstract. In this paper, we propose a novel scheme, named as TCP-ABC, which replicates the server side TCP connections among multiple server nodes of a cluster. By guaranteeing atomic request delivery, and consensus on responses, this scheme provides the legacy server applications running on the server nodes with multiple active backups in a transparent fashion. By failing the connections over healthy units, the scheme enhances the service and data availability of the cluster. By conducting experiments on the prototype system of a cluster up to four nodes, we find TCP-ABC results in small performance lost while greatly enhances the service and data availability.

1 Introduction

With the popularity of using clusters built with COTS components, more and more efforts need to be done to enhance the availability of the cluster systems. For the considerations of cost and portability, clusters always adopt mature legacy server applications, such as Apache, Q-Mail, to provide the services. Most of these applications follow the client/server model, and use TCP to implement their communication module. However, few of these applications provide active or standby backups to tolerate the faults so as to enhance the availability of a cluster. Although achieving fault-tolerance of the application by totally replacing its communication module sounds feasible, it involves huge effort. The most ideal way to improve fault-tolerance of the application and availability of the cluster is to employ solutions transparent to these legacy applications.

Generally, the availability of a cluster system has two aspects: the service availability and the data availability. Nowadays, front-end solutions, such as LVS [13], are used to achieve the service availability of a cluster, and a series of TCP fault-tolerance schemes [2][7][9], are proposed to do it at finer granularity, i.e., TCP connections. However, few legacy application transparent solutions are forwarded to enhance the data availability for the share-nothing clusters.

Active and semi-active replications methods [12] provide strong data consistency among the copies, which are the most ideal choices to implement the data availability of the clusters. However, both classes of these schemes require support from the

[*] This paper is supported by National 863 Hi-Tech R&D Project under grant No.2002AA1Z2102.

communication layer, i.e., atomic multicasting (broadcasting) [1][4]. In order to be transparent to the legacy applications, converting the TCP connections at the server side to atomic multicasting is the prerequisite of deploying these replication methods.

In this paper, we propose a novel scheme, namely TCP-ABC, which replicates the server side TCP connections among multiple server nodes of a cluster. By guaranteeing atomic request delivery, and consensus on responses, this scheme provides the legacy server applications running on the server nodes with multiple active backups in a transparent fashion. By failing the connections over healthy units, the scheme enhances the service and data availability of the cluster.

We organize this paper as the followings. In section 2, the scenario of research is presented. In section 3, we discuss the mechanisms employed by TCP-ABC during the failure-free phase, and consider the possible failures in section 4. To evaluate this scheme, we conduct experiments on real implementations, and present the results in section 5. In section 6, we present a briefly survey of the related works and conclude the paper in section 7.

2 Scenarios of Research

We take the share-nothing cluster shown in Fig. 1 as the scenario of our research. Among the server nodes, there is a unique primary server and multiple backup servers. The primary server possesses the *Portal IP* of the cluster. All the server nodes in the cluster have their own IP addresses (*IP1, IP2 .... IPn*), which belong to a same private subnet. The switch (or router), which connects the server nodes of the cluster with the outside world, supports IP multicasting (which is widely supported by varieties of network standards today) as well as point-to-point communication.

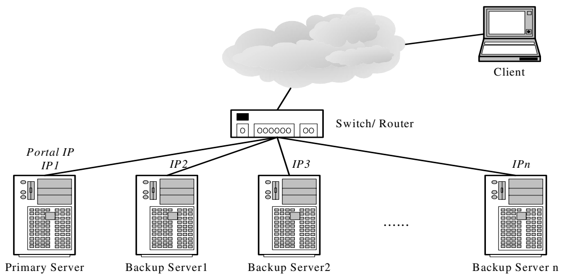

Fig. 1. Scenarios of Research

Data on the server nodes can only be modified by the server side applications by processing the requests of the clients. After processing each request, the server side application sends a response back to the client to indicate the result of the operation (Interactive Communication), and the requests and responses are sent via the established TCP connections. In this paper, we only consider the TCP connections initiated by the clients to the cluster. Regarding the server side applications, we consider only those processing the incoming requests in a non-stop fashion, i.e., the request messages are delivered in the order they are received.

For convenience of discussion, we assume the execution of the application is deterministic (Deterministic), and the server node delivers the received messages if it does not fail (Self-delivery). We assume the network is always available and will not be partitioned. Messages sent from one server node to another will eventually arrive at its destination (Live Network). Moreover, we assume the failures are crashes (Failure Stop) of the server nodes, and after failure, they will never come back. As our scheme can adopt any independent failure detector, we assume the failure detector used in our scheme is eventually perfect [5], i.e., it can diagnose the faults correctly.

3 Failure-Free Phase

Although the servers can obtain the incoming request messages at ease by simply programming the switch [7], guaranteeing the atomicity of request delivery turns difficult. In TCP-ABC, incoming requests are sequenced at the primary server and then propagated, while the responses from the server nodes converge at the primary server to form a unique response. The communication paradigm of TCP-ABC is shown in Fig. 2, where *Pi* (*i* =1~8) and *Bj* (*j* =1~9) are the processing steps at the primary and backup respectively.

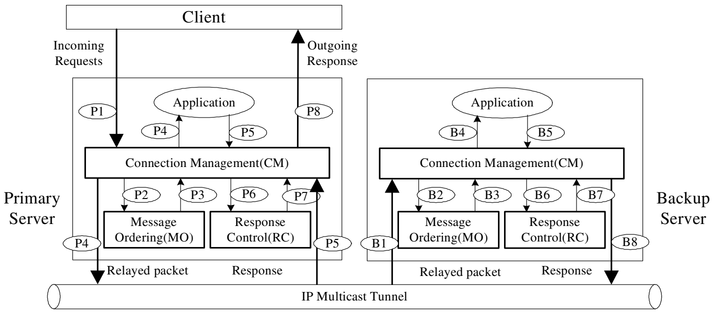

Fig. 2. Communication Paradigm of TCP-ABC

From Fig. 2 we can see that each server node of the cluster consists of *Connection Management* (CM), *Message Ordering* (MO) and *Response Control* (RC) module.

When the primary server receives an incoming TCP request packet from one of the clients, its CM module intercepts the packet and conducts legality check on the packet according to the connections. After that, the packet is given a global ordering number by MO module of the primary server, and then relayed to the backup servers. Section 3.1 will explain the ordering and delivery mechanisms in detail.

When responses are generated, they will be intercepted by the local CM modules and further handled by RC modules to figure out to the ordering number of the incoming request packet the response is for (the response number). Then, the response together with the response number will be sent to the primary server, which will decide the final version. Section 3.2 will explain this procedure in detail.

3.1 Message Ordering and Delivery Strategy

In TCP-ABC, each incoming TCP request packet from the clients is ordered by MO module of the primary server. To explain the ordering method, we illustrate the message exchange pattern of a typical TCP connection [10] in Fig. 3.

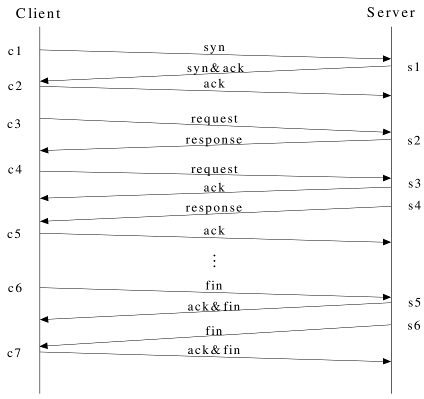

Fig. 3. The Message Exchange Pattern of a Typical TCP Connection

During the connection, MO module of the primary server gives the request packets (*c1*, *c2* … in Fig. 3) from the client the ordering numbers provided they are not the retransmitted packets or pure ACKs. In TCP-ABC, the ordering number grows monotonically and re-folds at a boundary, and fragments of the same request packet are given the same ordering number.

Regarding the pure ACK request packets, such as *c5* in Fig. 3, we give them a special ordering number that does not fall in the range of ordinary ordering numbers. When received by the server nodes, they are simply delivered if no request packets are pending before them. MO module of primary server will have the FIN packets as *c6* in Fig. 3 ordered before dissemination. If a server node receives the final request packet, i.e., *c7* in Fig. 3, and makes sure that all other nodes have also received the packet, the resources used by the corresponding connection will be reclaimed.

After being properly ordered, each incoming TCP request packet (except for the pure ACK and *c7*) forms a *decision message* as <*n*, *m*, *p*>, where *n* denotes the ordering number allocated by the primary, *p* denotes the request packet while *m* denotes the connection number the packet belongs to. The backup servers in TCP-ABC receive the decision messages by a monotonically increasing order. If decision message is received in disrupted order, the backup server will stop message delivering and send NAK messages to the primary server for retransmissions, which requires the primary to log incoming requests. As communication is interactive, the size of buffer used for logging on the primary should be the number of connections, and this buffer is replicated among all the backup servers to tolerate faults.

In TCP-ABC, all the server nodes of the cluster only deliver the decisions by a monotonically increasing order. Before delivering, the server nodes should make sure the decision is *stable*, i.e., all the others have received the decision. TCP-ABC requires all the backups to send a positive ACK message with the ordering number to other nodes after having received a decision. Each node delivers the decision only after having gathered all corresponding positive ACKs from the backups. As receiving ACK message with higher ordering number from a backup, each backup employs a simple time-out mechanism to retransmit the positive ACK message with the latest ordering number to guarantee the reliable dissemination of its positive ACK messages.

Theorem 1. *TCP-ABC guarantees the atomicity of message delivery for request packets at the server nodes.*

Proof. A multicast protocol is atomic if it satisfied three properties: *Self-delivery*, *All-or-nothing* and *Message ordering*. Self-delivery is assumed in section 2. Since a fixed sequencer (i.e., the primary) is used to order all incoming requests, which means TCP-ABC satisfied FIFO ordering. In case a decision message is lost at some nodes, the rest of the server nodes can delivery this decision only after the decision is received by all the server nodes, as they cannot receive all the positive ACKs. If one of server nodes crashes on the fly, the employed failure detector will eventually confirm the failure, exclude the server node from the cluster, and awake the rest of the server nodes. By this way, the all-or-nothing property is satisfied.

3.2 Response Control (Consensus)

In active replication schemes (e.g. [3]), with the deterministic assumption, server nodes always send their responses directly back to the clients, and the client picks up the fastest one. This method, however, does not fit TCP-ABC, as if it was employed, the processing and communication speed will be decided by the fastest node, and the slower nodes will lose pace. In TCP-ABC, a consensus on the responses at each turn of the iterations of communication is required to synchronize the server nodes.

Response numbers are used to differentiate the iterations. TCP-ABC computes the response number by comparing the ACK number of the response packet and the sequence number of the request packets in history. Response packets of the server nodes together with their individual response numbers will converge at the primary, which decide the final version of response for each response number by comparing the response packets with the same response number. TCP-ABC drops the pure ACK responses of the backups, and sends only those of the primary back to the clients.

With this consensus stage, TCP-ABC actually implements a semi-active replication mechanism to guarantee the data consistency of the replicas [12].

4 Failures

There are two typical types of failures in our scheme: failure of backup server and that of the primary. Crash failure of one of the backups makes the cluster stop working temporarily, since the rest healthy server nodes cannot receive the positive ACKs for the decisions and the responses from the failed backup. System continues to work until the failure detector diagnoses the failure, and after that, server nodes in the cluster will not wait messages from it anymore.

TCP-ABC handles the failure of primary by electing a new primary server among the healthy backups. The one with highest ordering number will be chosen as the new primary so as to keep the existing ordering number of the decisions. If more than one backup satisfy this criteria, the one with the smallest private IP address wins the election. Portal IP address of the cluster will be bound to the NIC of the new primary (IP-takeover). The retransmission mechanism of TCP assures that the unacknowledged requests of the clients will arrive at the new primary.

5 Performance Evaluation

To evaluate performance of TCP-ABC, we implement a prototype with a cluster up to four server nodes. In section 5.1, we will discuss the penalty on communication. In section 5.2, we will discuss the performance of MySQL cluster, which employs TCP-ABC to achieve high availability. The server nodes of the cluster are PC servers running Redhat Linux with kernel version 2.4.7-10, the hardware configuration is Intel Pentium III 1GHz CPU, 512MB Memory and 100Mbps Intel EEPro NIC. The client machines are PCs running Windows 2000 Professional (service pack 4) with hardware of Intel Celeron 1.7GHz CPU, 512MB Memory and RTL8139A NIC. We use 3COM 100Mbps switch to connect the clients and the server nodes.

5.1 Communication Penalty

In Fig. 4, we compare the performance of TCP connections under different cluster configurations. TCP-ABC is used when there is more than one server node. The *round trip time* (RTT) between the client and the cluster is used to demonstrate the latency of communication, and Netpipe-2.4 [11] is used as the benchmark.

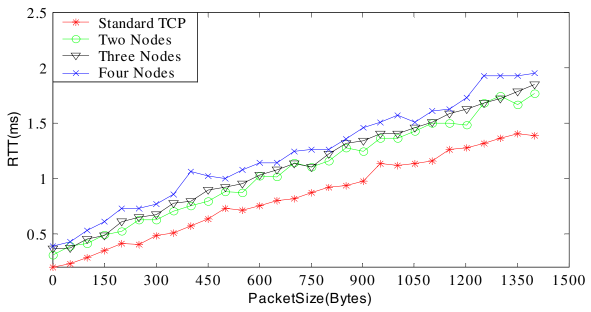

Fig. 4. Communication Penalty of TCP-ABC

From Fig. 4, we can see that when there are two server nodes in the cluster, the latency increases about 20~30% compared with that of the standard TCP. The latency increase is due to message ordering operations on the primary and the time paid at waiting for the positive ACKs and responses from the backup.

When the number of server nodes increases to four, the latency turns higher than that of two. But from Fig. 4, we can observe that compared with that of two nodes, the latency of four nodes only increases about near 10%. The increment is resulted for more time spent on waiting for the positive ACKs and responses from the backup servers.

5.2 Performance of MySQL Cluster

As an open source database management system, MySQL server [8] has been gaining more and more users around the world. In common installations, it is used as backend server providing data to the other server nodes. However, the crash of MySQL server will result in unavailability of the whole cluster. Build-in program of MySQL package can provide the users with a standby backup, and the failover mechanism is not automatic and seamless. We use TCP-ABC to provide multiple active replicas for MySQL server. In our experiment, MySQL server 3.23.41 runs on the server nodes, and the client machine connects to the cluster with MySQL ODBC 3.51.10.

A thread is invoked at the client to create and drop 1000 tables, each of which has ten integer fields. A *test table* with ten integer fields is created for further experiments. We invoke another thread to insert and delete 10000 rows into and out from the test table. The performance of update is obtained by updating a random row within the test table for 1000 times. For all these tests, average response time is obtained to indicate the performance, shown in Fig. 5.

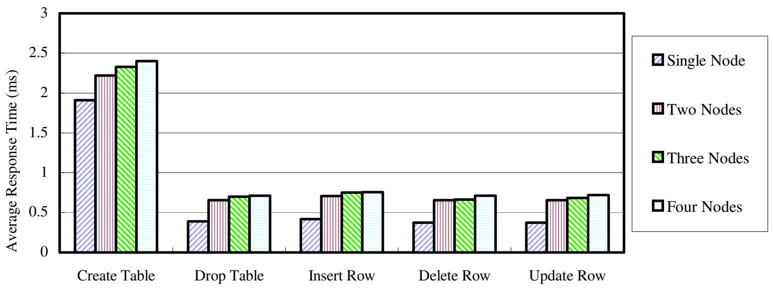

Fig. 5. Performance of MySQL Cluster on Update Operations

Fig. 5 shows that creating of table is the most time consuming. This is because MySQL server needs I/O operations when creating new files to hold newly created tables. The other update operations cost less time since the file is always open before operation. Larger sacrifice on the performance of the update operations consuming less time than those consuming more can be observed. Since penalty put on communication can be better masked by the time consumed on the operations. This means, to the complex operations (e.g., updates on multi-table), the sacrifice is less than that of the simple ones in the experiments.

6 Related Works

Atomic Multicast schemes [1][4] and View Synchronous Communication [6] are two important communication abstractions that have been extensively considered in the context of asynchronous fault-tolerant distributed systems. However, besides the disadvantages for practical applications, such as heavy-weighted, prolonged delivery time and complexity, both of these two abstractions take stateless communication protocols (i.e. UDP) as their basis. This inevitably jeopardizes the transparency if they are applied to the legacy applications using TCP.

TCP Fault-tolerance Schemes [2][7][9] were proposed within the past a few years. Most of them were implemented by providing primary server that actually handled the connection with an active fully replicated backup. However, these schemes suffered some common drawbacks, such as long failover time [2], unreasonable assumption on the processing speed of replicas [7], heavy load on the primary [9]. Moreover, these schemes considered only the service availability.

7 Conclusions

In this paper, we propose a scheme to replicate the server side TCP connections among multiple server nodes of a cluster so as to make failover at TCP connection granularity possible. By guaranteeing atomic request delivery, and consensus on responses, a semi-active replication mechanism is formed to guarantee the data consistency of the server nodes. By conducting experiments on the prototype system of a cluster up to four nodes, especially the MySQL cluster, we find our scheme results in small performance lost while greatly enhances the service and data availability.

References

1. D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia, "The Totem multiple-ring ordering and topology maintenance protocol", *ACM Transactions on Computer Systems*, May 1998, 16(2):93-132
2. L. Alvisi, T. C. Bressoud, A. El-Khashab, K. Marzullo, and D. Zagorodnov, "Wrapping Server-Side TCP to Mask Connection Failures", In *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, USA, 2001, pp.329-337
3. Y. Amir, D. Dolev, P. M. Melliar-Smith, and L. E. Moser, "Robust and Efficient Replication using Group Communication", *Technique Report CS94-20*, Institute of Computer Science, Hebrew University, 1994
4. K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast", *ACM Transactions on Computer Systems*, 1991. 9(3):272-314
5. T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, March 1996, 43(2):225-267
6. G. V. Chockler, I. Keidar, and R. Vitenberg, "Group Communication Specifications: A Comprehensive Study", *ACM Computing Surveys*, December 2001, 33(4):1-43
7. M. Marwah, S. Mishra, and C. Fetzer, "TCP Server Fault Tolerance Using Connection Migration to a Backup Server", In *Proceedings of the 2003 IEEE International Conference on Dependable Systems and Networks (DSN)*, San Francisco, CA, USA, 2003, pp.373-382
8. MySQL server, http://www.mysql.com
9. Z. Shao, H. Jin and B. Chen, J. Xu, and J. Yue, "HARTS: High Availability Cluster Architecture with Redundant TCP Stacks", In *Proceedings of the International Performance Computing and Communication Conference (IPCCC)*, Phoenix, Arizona, USA, 2003, pp.255-262

10. W. R. Stevens, *TCP/IP illustrated. Volume 1: The protocols*, Addison-Wesley, 1994
11. Q. O. Snell, A. Mikler, and J. L. Gustafson, "Netpipe: A Network Protocol Independent Performace Evaluator", In *Proceedings of IASTED International Conference on Intelligent Information Management and Systems*, June 1996, pp.196-204
12. M. Wiesmann, F. Pedone, A. Schiper, and B. Kemme, "Understanding replication in databases and distributed systems", In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Taipei, Taiwan, 2000, pp.264-274
13. W. Zhang, "Linux Virtual Server for Scalable Network Services", In *Proceedings of Ottawa Linux Symposium*, Ottawa, Canada, 2000, pp.212-221