J. Parallel Distrib. Comput. 73 (2013) 630–640

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Avoiding disruptive failovers in transaction processing systems with multiple active nodes

Gong Su *, Arun Iyengar

IBM T.J. Watson Research Center, Yorktown Heights, NY, 10598, United States

ARTICLE INFO

Article history:
Received 1 May 2012
Received in revised form
8 November 2012
Accepted 18 January 2013
Available online 26 January 2013

Keywords:
Computer-driven trading
Fault tolerance
High availability
Total ordering algorithm
Transaction processing

ABSTRACT

We present a highly available system for environments such as stock trading, where high request rates and low latency requirements dictate that service disruption on the order of seconds in length can be unacceptable. After a node failure, our system avoids delays in processing due to detecting the failure or transferring control to a back-up node. We achieve this by using multiple primary nodes which process transactions concurrently as peers. If a primary node fails, the remaining primaries continue executing without being delayed at all by the failed primary. Nodes agree on a total ordering for processing requests with a novel low overhead wait-free algorithm that utilizes a small amount of shared memory accessible to the nodes and a simple compare-and-swap like protocol which allows the system to progress at the speed of the fastest node. We have implemented our system on an IBM z990 zSeries eServer mainframe and show experimentally that our system performs well and can transparently handle node failures without causing delays to transaction processing. The efficient implementation of our algorithm for ordering transactions is a critically important factor in achieving good performance.

1. Introduction

Transaction-processing systems such as those for stock exchanges need to be highly available. Continuous operation in the event of failures is critically important. Failures for any length of time can cause lost business resulting in both revenue losses and a decrease in reputation. In the event that a component fails, the systems must be able to continue operating with minimal disruption.

This paper presents a highly available system for environments such as stock trading, where high request rates and low latency requirements dictate that service disruptions on the order of seconds in length can be unacceptable. A key aspect of our system is that processor failures are handled transparently without interruptions to normal service. There are no delays for failure detection or having a back-up processor take over for the failed processor because our architecture eliminates the need for both of these steps.

A standard method for making transaction processing systems highly available is to provide a primary node and at least one secondary node which can handle requests. In the event that the primary node fails, requests can be directed to a secondary node which is still functioning. This approach, which we refer to as the primary–secondary approach (which is also known as active–passive high availability), has at least two drawbacks for environments such as stock trading. The first is that stock trading requests must be directed to specific nodes due to the fact that the nodes have local in-memory state information typically not shared between the primary and secondary for handling specific transactions. For example, a primary node handling trades for IBM stocks would have information in memory specifically related to IBM stocks. If a buy or sell order for IBM stock is directed to a secondary node, the secondary node would not have the proper state information to efficiently process the order. The primary node should store enough information persistently to allow stock trading for IBM to continue on another node should it fail. However, the overhead for the secondary node to obtain the necessary state information from persistent storage would cause delays in processing trades for IBM stock which are not acceptable. The second problem with the primary–secondary approach is that there can be delays of several seconds for detecting node failures during which no requests are being processed. For systems which need to be continuously responsive under high transaction rates, these delays are a significant problem. Therefore, other methods are desirable for maintaining high availability in transaction processing systems which handle high request rates and need to be continuously responsive in the presence of failures.

Our system handles failures transparently without disruptions in service. A key feature of our system is that we achieve redundancy in processing by having multiple nodes executing transactions as peers concurrently. If one node fails, the remaining ones simply continue executing. There is no need to transfer control to a secondary node after a failure because all of the

* Corresponding author.
E-mail addresses: gongsu@us.ibm.com (G. Su), aruni@us.ibm.com (A. Iyengar).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2013.01.007

nodes are already primaries. A key advantage to our approach is that after a primary failure, there is no lost time waiting for the system to recover from the failure. Other primaries simply continue executing without being slowed down by the failure of one of them.

One of the complications with our approach is that the primaries can receive requests in different orders. A key component of our system is a method for the primaries to agree upon a common order for executing transactions, known as the total ordering, without incurring significant synchronization overhead. We do this by means of a limited amount of shared memory accessible among the nodes, and a simple but efficient synchronization protocol.

The overall concept of the primary–primary approach has been proposed previously [25] and is also known as active–active high availability. However, previous methods proposed for achieving total ordering among requests are often complex and not wait-free. In our work, we show how to achieve total ordering of requests using a relatively simple wait-free protocol. In addition, we have implemented and thoroughly tested our system using a stock trading application. A considerable effort is needed to go from ideas proposed in past papers to an efficient working system.

The key contributions of this paper include the following:

- We show how the primary–primary approach can be used for transaction processing applications such as stock trading in which the primary nodes must agree upon a common order for processing the requests.
- We have developed and implemented a new efficient *wait-free* algorithm for nodes in a distributed environment receiving messages in different orders to agree on a total ordering for those messages. This algorithm is used by our system to determine the order for all nodes to execute transactions and makes use of a small amount of shared memory among the nodes. The total ordering algorithm imposes little overhead and proceeds at the rate of the fastest node; it is not slowed down by slow or unresponsive nodes.
- We have implemented our approach on an IBM z990 zSeries. Experimental results show that our system achieves fast recovery from failures and good performance. Average latencies for handling transactions are well below 10 ms. The efficient total ordering algorithm is a critically important factor in achieving this performance.

2. System architecture

Our system makes use of multiple nodes for high availability. Each node contains one or more processors. Nodes have some degree of isolation so that a failure of one node would not cause a second node to fail. For example, they run different operating systems and generally do not share memory to any significant degree. In our implementation, nodes can communicate and synchronize via a small amount of shared memory.

2.1. Traditional primary–secondary architecture

For environments such as stock trading, response times have to be extremely fast. Therefore, state information needed to perform transaction processing is cached in the main memories of nodes handling transactions. A key drawback of the primary–secondary approach of having a back-up node take over in case the primary node fails is that the back-up node will not have the necessary state information in memory in order to restart processing right away. There are also delays in detecting failures. A common method for detecting failures is to periodically exchange heart beat messages between nodes and listen for failed responses. It is generally not feasible to set the timeout period before a node is declared failed to too small an interval (e.g. less than several seconds) due to the risk of erroneously declaring a functioning node down. This means that it often takes several seconds to detect a failure. The delays that would be incurred in detecting the failure of a primary node and getting a secondary node up and running by obtaining the necessary state information from persistent storage are thus often too high using this conventional high availability approach.

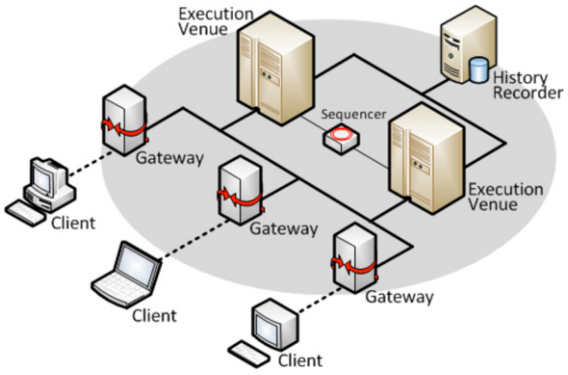

Fig. 1. Primary–primary architecture.

For this reason, it is essential to have at least two nodes with updated in-memory data structures for handling orders for the stock. That way, if one of the nodes fails, the other node will still be functioning and can continue handling trades for the stock.

One way to achieve high availability would be to have a primary node handling requests for a stock in a certain order and to have the primary node send the ordered sequence of requests that it is processing to a secondary node. The secondary node then executes the transactions in the same order as the primary node but a step or two behind the primary node. The secondary node would avoid performing many updates to persistent storage already performed by the primary node since the whole reason for the secondary node executing transactions is to keep its main memory updated.

While this approach eliminates some of the overhead of simply having a cold standby taking over for the primary, it still incurs some overhead for both detecting the failed primary node and handling the failover from the primary node to the secondary node. As we mentioned previously, detecting the failed primary node can take several seconds. The secondary node also needs to figure out exactly where the primary node failed in order to continue processing at exactly the right place. If the failover procedure is not carefully implemented, the secondary node could either repeat processing the primary node has already done or leave out some of the processing the primary node performed before failing; either of these two scenarios results in incorrect behavior.

Our system avoids the problems of both detecting failures and transferring control from a failed node to a back-up node by having multiple primary nodes executing the same sequence of transactions as peers concurrently. Normally, two primary nodes would be sufficient. If failure of more than one node within a short time period is a concern, more than two primaries can be used. In the event that a primary node fails, the remaining primaries keep executing without being hindered by the failed primary. We now describe our architecture in more detail.

2.2. Our primary–primary architecture

We depict the overall primary–primary stock exchange trading architecture in Fig. 1.

An electronic stock exchange, as illustrated by the shaded ellipse area in the diagram, typically consists of 3 tiers,

- *Gateways* (GW) collect buy/sell requests from clients (e.g., traders and brokers) and perform preprocessing such as validation.