UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
UNDERGRADUATE SCHOOL

PROJECT

FAUL-TOLERANT KEY-VALUE SERVER CLIENT USING MPI

REPORT

Submitted by

Name of students	Student ID
Nguyen Minh Duc	USTHBI7-038
Nguyen Minh Duong	USTHBI7-048
Vu Xuan Huy	USTHBI7-084
Le Canh Duong	USTHBI7-047
Pham Van Duan	USTHBI7-034

Hanoi, April 2020

Content

I. INTRODUCTION

Keeping mission-critical applications running is one of an IT department's most important responsibilities. Although clustering products provide an effective high-availability solution, the failover process can disrupt application processing for 30 seconds or longer. Depending on the client application's design, users might have to reconnect to the clustered application when it resumes on the new node, and if the failed node sits on a remote site, you'll have to dispatch a technician to repair it.
A long trend in the high performance distributed systems is the increase of the number of nodes. As a consequence, the probability of failures in supercomputers and distributed systems also increases. So fault tolerance becomes a key property of parallel applications. Designing and implementing fault tolerant software is a complex task. Fault tolerance is a strong property which implies a theoretical proof of the underlying protocols. The protocol implementation should then be checked with respect to the specification.

II. OBJECTIVE

The aim of FT-MPI is to build a fault-tolerant MPI implementation that can survive failures, while offering the application developer a range of recovery options other than just returning to some previous checkpoint state.

III. STATE OF THE ART

There are several methods which is similar to Faul-Tolerant. The two methods below stand out the most.

1. High Availability

A High Availability system is one that is designed to be available 99.999% of the time, or as close to it as possible. Usually this means configuring a failover system that can handle the same workloads as the primary system.

In VMware, High Availability works by creating a pool of virtual machines and associated resources within a cluster. When a given host or virtual machine fails, it is restarted on another VM within the cluster**.** In Azure, admins use the Resiliency feature to create High Availability, backup, and Disaster Recovery as well by combining single VMs into Availability Sets and across Availability Zones. In either case, the hypervisor platform is able to detect when a machine is failing and needs to be restarted elsewhere.

For physical infrastructure, High Availability is achieved by designing the system with no single point of failure; in other words, redundant components are required for all critical power, cooling, compute, network, and storage infrastructure.

One example of a simple High Availability strategy is hosting two identical web servers with a load balancer splitting traffic between them and an additional load balancer on standby. If one server goes down, the balancer can direct traffic to the second server (as long as it is configured with enough resources to handle the additional traffic). If one load balancer goes down, the second can spin up.

The load balancer in this situation is key. High Availability only works if you have systems in place to detect failures and redirect workloads, whether at the server level or the physical component level. Otherwise you may have resiliency and redundancy in place but no true High Availability strategy.
Fault tolerant systems provide an excellent safeguard against equipment failure, but can be extraordinarily expensive to implement because they require a fully redundant set of hardware that needs to be linked to the primary system. High availability architecture is much more cost effective, but also bring with them the possibility of costly downtime, even if that downtime only lasts for a few moments.

2. Disaster recovery

Disaster Recovery goes beyond Faul-Tolerant or High Availability and consists of a complete plan to recover critical business systems and normal operations in the event of a catastrophic disaster like a major weather event (hurricane, flood, tornado, etc), a cyberattack, or any other cause of significant downtime. High Availability is often a major component of Disaster Recovery, which can also consist of an entirely separate physical infrastructure site with a 1:1 replacement for every critical infrastructure component, or at least as many as required to restore the most essential business functions.

Disaster Recovery is configured with a designated Time to Recovery and Recovery Point, which represent the time it takes to restore essential systems and the point in time before the disaster which is restored (you probably don’t need to restore your backup data from 5 years ago in order to get back to work during a disaster, for example).

A Disaster Recovery platform replicates your chosen systems and data to a separate cluster where it lies in storage. When downtime is detected, this system is turned on and your network paths are redirected. Disaster Recovery is generally a replacement for your entire data center, whether physical or virtual; as opposed to High Availability, which typically deals with faults in a single component like CPU or a single server rather than a complete failure of all IT infrastructure, which would occur in the case of a catastrophe.

While high availability and fault tolerance are exclusively technology-centric, disaster recovery encompasses much more than just software/hardware elements. HA and FT focus on addressing the isolated failures in an IT system. DR, on the contrary, deals with failures of a much bigger scope, as well as the consequences of such failures. Incorporating high availability or fault tolerance cannot ensure protection from disasters, but both of them can complement disaster recovery strategies in an efficient manner.

IV. METHOD

The program will contain the client the server using MPI. The client can create, get, delete the key and value through the MPI to the server. Also the program can handle the error during implement the action. That means it's able the program to continue operating properly in the event of the failure.

All of MPI programs begin with **MPI_Init** and end with **MPI_finalize**. Plus, some of component we need to know the program:

MPI_Open_port (MPI_Info info, char *port_name)
Establish an address that can be used to establish connections between groups of MPI processes

MPI_Comm_accept (const char *port_name, MPI_Info info, int root, MPI_Comm comm, MPI_Comm *newcomm)
Accept a request to form a new intercommunicator:

- port_name: port name (string, used only on root)
- info: Limplementation-dependent information (handle, used only on root)
- root: rank in comm of root node (integer)
- comm: intracommunicator over which call is collective (handle)

MPI_Comm_connect (const char *port_name, MPI_Info info, int root, MPI_Comm comm, MPI_Comm * newcomm)

MPI_Send, to send a message to another process

MPI_Send (void *data_to_send, int send_count, MPI_Datatype send_type, int destination_ID, int tag, MPI_Comm comm);

- data_to_send: variable of a C type that corresponds to the send_type supplied below
- send_count: number of data elements to be sent (nonnegative int) send_type: datatype of the data to be sent (one of the MPI datatype handles) destination_ID: process ID of destination (int)
- tag: message tag (int)
- comm: communicator (handle)

MPI_Recv, to receive a message from another process

MPI_Recv (void *received_data, int receive_count, MPI_Datatype receive_type, int sender_ID, int tag, MPI_Comm comm, MPI_Status *status);

- received_data: variable of a C type that corresponds to the receive_type supplied below
- receive_count: number of data elements expected (int)
- receive_type: datatype of the data to be received (one of the MPI datatype handles)
- sender_ID: process ID of the sending process (int)
- tag: message tag (int)
- comm: communicator (handle)
- status: status struct (MPI_Status)

The receive_count, sender_ID, and tag values may be specified so as to allow messages of unknown length, from several sources (MPI_ANY_SOURCE), or with various tag values (MPI_ANY_TAG).

The first part of the program is the connect between the client and the server. The server will open the port that client can connect by MPI_Open_port and then we add MPI_Comm_accept to allow the client can connect the server. The client can connect to the server when use MPI_Comm_connect. After the connection is established. The client can send data to the server by use MPI_Send and the server receive the date from client by use MPI_Recv, also with the case server send data to client.

It is key-value stored, we need to make a key-value store to the server by using struct with C or C++. The client will choose the options with key-value such as create a key and a value, delete the key, get the value from the key.

V. CONCLUSION

In the end, we can know the MPI, it is message pasing interface, a standardized and portable message passing standard, is a communication protocol for programming parallel computers. MPI is a message-passing application programer interface, together with protocol and semantic specifications for how its features must behave in any implementation. And fault tolerant MPI that survices the crash of n-1 processes in a n-process job and if required, can respawn/restart them.

In this project, we can apply MPI to make the simple client and server, understand how to implement it.