Final Report

FAUL-TOLERANT KEY-VALUE SERVER CLIENT USING MPI

Distributed System

by Nguyễn Lê Tuấn Duy,

Nguyễn Huy An, Nguyễn Thành Gia Hiển,

Lưu Hải Nam

March 2021

Contents

1 INTRODUCTION

1.1 Abstract

One of the most significant tasks of an IT department is to keep mission-critical systems going. While clustering products provide a high-availability solution, the failover mechanism can cause application processing to be disrupted for up to 30 seconds. Users can need to reconnect to the clustered application when it returns on the new node, depending on the client application's configuration, and you'll have to send a technician to fix the failed node if it's on a remote site.

The growth in the number of nodes has been a long-term trend in high-performance distributed systems. As a result, the likelihood of failures in supercomputers and distributed networks grows. As a consequence, fault tolerance becomes a vital function of parallel applications. Fault-tolerant program design and implementation is a difficult challenge. Fault tolerance is a strong property that means that the underlying protocols are theoretically proven. After that, the protocol implementation should be compared to the specification.

The goal of FT-MPI is to create a fault-tolerant MPI implementation that can withstand failures while also providing the application developer with a variety of recovery options rather than simply reverting to a previous checkpoint state.

1.2 Authors and Credits

The work has been undertaken by group number 2, whose members are listed in the following table.

Full name	Student ID
Nguyễn Lê Tuấn Duy	BI9-082
Nguyễn Huy An	BI9-033
Nguyễn Thành Gia Hiển	BI9-099
Lưu Hải Nam	BI9-170

We would like to express our gratitude towards our grateful lecturer, Dr.TRAN Giang Son, who introduced us to Distributed System. Through his lectures, we got to understand the basic concepts and also the applications of the subject in real world problems.

2 STATE OF THE ART

There are several methods which is used to maintain Fault-Tolerant. The two methods below stand out the most

2.1 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.

2.2 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.

3 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.

4 CONCLUSION

Finally, we learn about MPI, which stands for message passing interface and is a networking protocol for programming parallel computers. It is a generic and compact message passing standard. MPI is a message-passing application programming interface that includes protocol and semantic requirements for how its functionality should behave in any implementation.

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

5 References

[1] An introduction to MPI by William Gropp and Ewing Lusk, Argonne National Laboratory

[2] MPI Programming Model: Desert islands Analogy by Henry Neeman