Software Engineering for Message Passing Parallel Clusters

Eşme, Taner
Boğaziçi University, Department of Computer Engineering, Software Engineering
Istanbul, Turkey
May 2018

Abstract: In this paper, we briefly address the computer clusters and Beowulf cluster being the first precedent of them. We also discuss the key attributes of a cluster and well-known parallel architectures. We make a list of message passing techniques and describe the most popular message passing methods in a comparable manner. We advert to the testing of message passing parallel programs in a nutshell.

Keywords: Computer Clusters, Beowulf Cluster, Message Passing

1. Introduction

According to Wikipedia [1], message-passing is "a technique for invoking behavior (i.e. running a program) on a computer. The invoking program sends a message to a process and relies on the process and supporting infrastructure to select and invoke the actual code to run". Besides of this definition, according to IBM [2], a computer cluster is "a group of servers and other resources that are connected through hardware, networks, and software to behave as if they were a single system".

Although the date of the first cluster computer invented is unknown, it is known that the development of the cluster computers is underpinned by packet switching networks that conceptually invented by the RAND Corporation in 1962. The packet switching concept made it possible for the Internet in the manner that we know today and the Internet can be considered as the mother of all computer clusters, but the essential reason causing computer clusters to thrive is the development of Parallel Virtual Machine (PVM) software, which was an open source software based on TCP/IP communication, in 1989 [3]. The first computer cluster made out of commodity computers, which were inexpensive network PCs, was Beowulf cluster, which was a NASA project to build a supercomputer [4].

In the sections that follow, we discuss the clusters, their attributes and how we communicate the applications running on these computers accommodated in the cluster.

2. What is Beowulf Cluster?

Beowulf Cluster is a scalable computer cluster that can be built by using the commodity-level computers, a local area network (LAN), and a software to share the processing among the computers in the cluster. It may also be considered as a homemade cluster [6].

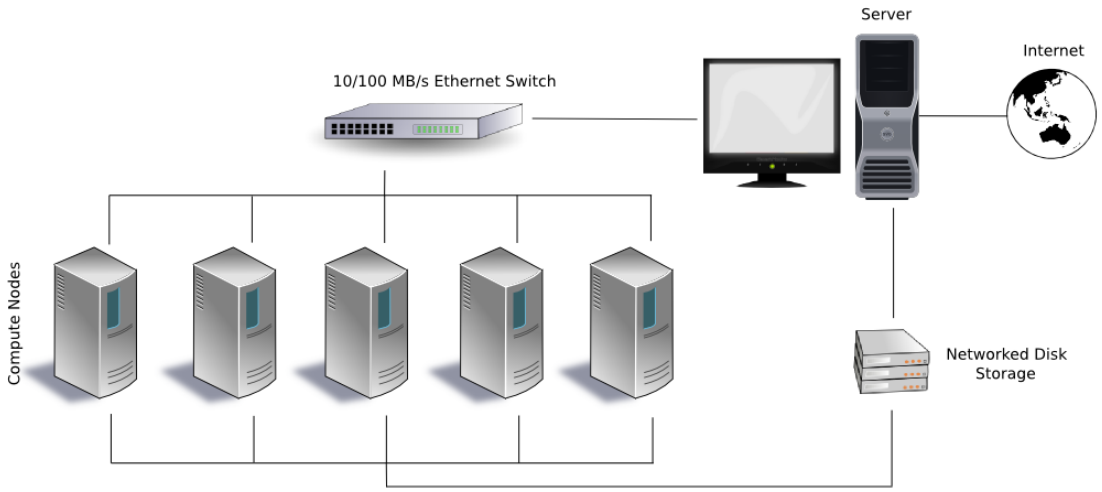

Figure 1 A typical Beowulf Cluster Configuration [5]

The Beowulf configuration that you see above is a system which constitutes of one server node, which controls the whole cluster and is a gateway to the general purpose network of the organization, and five dummy client nodes connected each other via a switch. Also, they have a disk-less configuration, they share the same storage devices. Finally, Beowulf is a technology of clustering computers to form a parallel, virtual supercomputers [6].

3. What are the attributes of computer clusters?

Computer clusters, which constitute from more than one computer that carry out the same operation, can be made use for different purposes such as supporting a web-service or a web-based application, or for huge scientific calculations. They provide a high-available system perceived as a single [5]. The main purposes of computer clusters are to make the total response-time minimum and throughput maximum [7].

3.1. Computational-intensive purposes

Parallel computing on clustered systems is a quite appropriate solution for the applications such as simulations, IO-intensive operations, and scientific calculations [5] [12].

3.2. Load Balancing

In a cluster, load balancing is an approach to distribute the workload across the computers in a cluster to handle it concurrently. It does not only mean that the computers are used but also

other resources such as disks, network links, processing units can be used. Today's organization can respond to the high number of requests with a lower cost thanks to the load balancing [8].

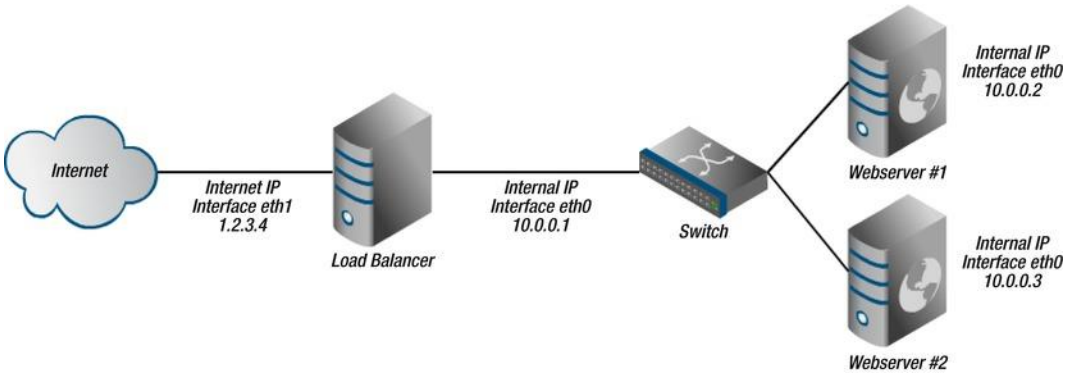

Figure 2 An overview of the setup [8]

3.3. High-Availability

Availability can be stated as a percentage of a system's uptime, which is the amount of time that a system is accessible, in a year [9].

High availability is considered as a characteristic of a system that means the ability to provide a service with a higher uptime than normal. If a system is designed as highly-available, then it is expected from it keeping working in the cases such as being altered, updated, and so on [10].

There are three principles for high-availability [9];

1. Getting rid of the single point of failure
2. Failure detection
3. Reliability or fault tolerance

It is obvious that we need redundant computers to achieve high availability and load balancing, in this case, there is a tremendous requirement for communicating these computers each other to share their state and other data that is necessary to keep the system or the result of computation consistent.

4. What are the parallel architectures?

Today, load balancing and high availability can be the most popular purposes of using the parallel architecture in the industry, but the basic idea behind it is to raise the throughput and to reduce the processing time. We can address two different type of architectures to support parallelism [11];

1. Shared-memory architecture
2. Message passing architecture

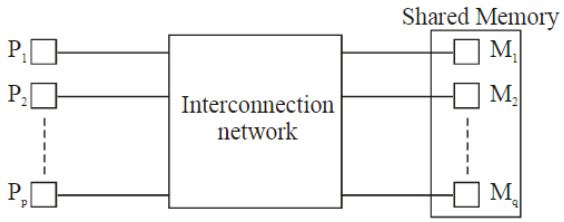

Figure 3 Shared-memory architecture [11]

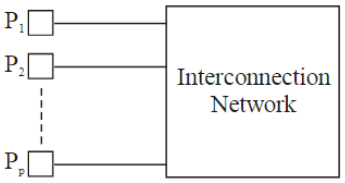

Figure 4 Message passing architecture [11]

First architecture is used for multiprocessor systems to make the processes communicate with each other, the second one is used for multicomputer systems that we call cluster in the prior sections. In the shared-memory architecture, the processes communicate each other via shared variables stored on the shared memories whereas, in the message passing architecture, they communicate by sending and receiving packages among each other through the network [11].

For message passing architecture, many approaches have emerged to support to increased computational need and network-based programming because of the reasons such as the growth of WEB, the popularity of the personal computers and high-speed networks. Some of these approaches are RPC (Remote Procedure Call), DCE (Distributed Computing Environment), DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), and message passing like PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). PVM and MPI are considered as the most popular programming approaches for message passing architecture [12].

4.1. Parallel Virtual Machine (PVM)

PVM is a software that is used by the computers that are connected to each other through the network to solve the CPU and memory intensive computational problems. It is also designed to be used on the different types of the machine such as UNIX and Windows [13].

PVM is designed around a virtual machine concept that underpins the scalability, portability, heterogeneity and fault tolerance being its the fundamental features. The performance is less important for PVM than it being in MPI because of its research-focus and the slow internet connectivity in the 1990s. If you need to run an application parallel on the different platforms, PVM API can be a better choice for your application [14].

4.2. Message Passing Interface (MPI)

MPI interface, which is a de facto standard for the communication between the processes running parallel, is a solution for containing all of the message passing methods within a standard because, at the beginning of the development of parallel processors, every vendor had their own implementation for message passing. Therefore, MPI, rather than PVM, specifies the standards, which are implemented by using TCP/IP [12], for the communication of the computers in a cluster [14].

MPI allows the programmers who develop parallel applications to choose one of the communication options of point-to-point or collective. It is possible to send and receive messages

between the processes and between the process groups that are used in collective communication [15].

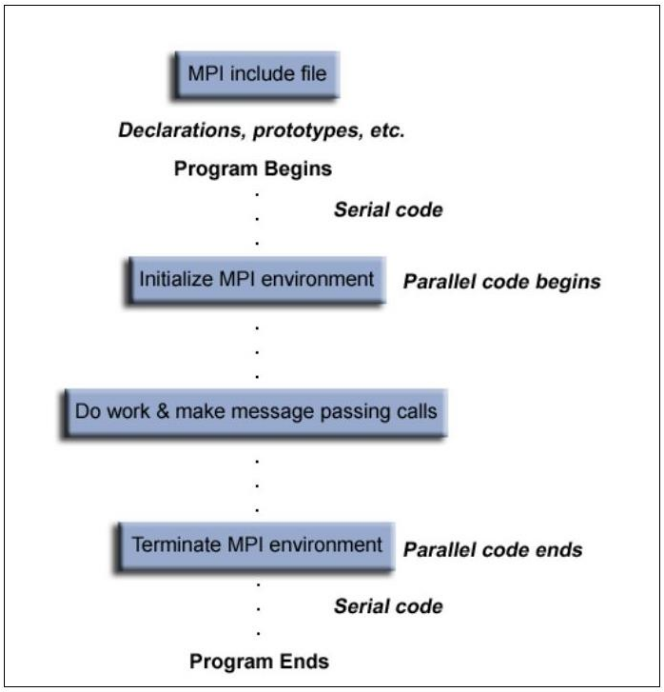

Figure 5 General Structure of an MPI program [16]

5. How can we test a parallel program?

Parallel programs are harder to test than the sequential ones because of their features such as indeterminism, uncertainty, and concurrency. Although there are some tools to simulate and debug the parallel programs, there is no testing tool that can be used actively to create test data, testing criteria and to assess the testing coverage but there are works to propose new ways [17] to test the message passing parallel programs. Besides testing, it is possible to find the solutions to verify and validate the message passing parallel programs in the literature [18] [19].

6. Conclusions

When we think of today's computational requirements and new trends such as big data, and data analytics [20], it is an obvious fact that the necessity of the parallel programming will be raising indispensably day after day. Therefore, the parallel programming skills and the architectural know-how of cluster computers can become one of the dominant career trends.

References

[1] *Message Passing*. (2018). Retrieved from https://en.wikipedia.org/wiki/Message_passing

[2] *Clustering: A basis 101 tutorial*. (2018). Retrieved from https://www.ibm.com/developerworks/aix/tutorials/clustering/clustering.html

[3] *History of computer clusters.* (2018). Retrieved from https://en.wikipedia.org/wiki/History_of_computer_clusters

[4] *Overview – History* (2018). Retrieved from http://www.beowulf.org/overview/history.html

[5] *Computer Cluster* (2018). Retrieved from https://en.wikipedia.org/wiki/Computer_cluster

[6] *Beowulf Cluster* (2018). Retrieved from https://en.wikipedia.org/wiki/Beowulf_cluster

[7] Werstein, Paul & Situ, Hailing & Huang, Zhiyi. (2007). Load Balancing in a Cluster Computer. 569 - 577. 10.1109/PDCAT.2006.77.

[8] Membrey, Peter & Plugge, Eelco & Hows, David. (2012). Practical Load Balancing. *Apress*.

[9] *High Availability.* (2018). Retrieved from https://en.wikipedia.org/wiki/High_availability

[10] Piedad, Floyd & Hawkins, Michael. (2001). High Availability: Design, Techniques, and Processes. *Prentice Hall*.

[11] Z Mihu, Ioan & Caprita, Horia & Gellert, Arpad. (2004). Parallel Programming using MPI Library on Message-Passing Architectures. Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE. 49. 1224-600.

[12] Fadlallah, Ghassan & Lavoie, Michel & Dessaint, Louis-A. (2000). Parallel computing environments and methods. 2-7. 10.1109/PCEE.2000.873591.

[13] *PVM: Parallel Virtual Machine* (2018). Retrieved from https://www.csm.ornl.gov/pvm/

[14] A. Geist, G & Kohl, James & M. Papadopoulos, P. (1998). PVM and MPI: A comparison of features. Calculateurs Paralleles. 8.

[15] Clarke, Lyndon & Glendinning, Ian & Hempel, Rolf. (1996). The MPI Message Passing Interface Standard. Int. J. Supercomput. Appl.. 8. 10.1007/978-3-0348-8534-8_21.

[16] Morrison, Carlos. (2017). Build Supercomputers with Raspberry Pi 3. *Packt*.

[17] Souza, Simone & Vergilio, Silvia & Lopes de Souza, Paulo & Simão, Adenilso & Bliscosque Goncalves, Thiago & de Melo Lima, Alexandre & Ceolin Hausen, Alexandre. (2005). ValiPar: A Testing Tool for Message-Passing Parallel Programs.. 386-391.

[18] Santos, César & Martins, Francisco & Vasconcelos, Vasco. (2015). Deductive Verification of Parallel Programs Using Why3. Electronic Proceedings in Theoretical Computer Science. 189. 10.4204/EPTCS.189.11.

[19] Lopez A., Hugo Andres & Marques, Eduardo & Martins, Francisco & Ng, Nicholas & Santos, César & Vasconcelos, Vasco & Yoshida, Nobuko. (2015). Protocol-Based Verification of Message-Passing Parallel Programs. ACM SIGPLAN Notices. 50. 10.1145/2858965.2814302.

[20] Dobre, Ciprian & Xhafa, Fatos. (2014). Parallel Programming Paradigms and Frameworks in Big Data Era. International Journal of Parallel Programming. 42. 10.1007/s10766-013-0272-7.