ISSN:0975-9646

Analysis of Different Techniques Used For Fault Tolerance

Jasbir Kaur, Supriya Kinger
Department of Computer Science and Engineering, SGGSWU, Fatehgarh Sahib, India, Punjab (140406)

Abstract- Cloud computing is a synonym for distributed computing over a network and means the ability to run a program on many connected computers at the same time. This phase is also more commonly used to refer to network based services which appear to be provided by real server hardware, which in fact are served up by virtual hardware, simulated by software running on one or more real machines. This paper is based on the survey of types of faults-tolerance and different types of fault-tolerance techniques. There are several methods used to avoid the faults before and after it occur.

General Terms – Cloud Computing, Fault Tolerance, Redundancy, Check pointing, Virtual Machines

1. Introduction

Cloud Computing is a growing research field in these days and it is the next big revolution in computer networks and web provisioning.

1.1 Definition of Cloud Computing

Cloud computing or something being within the cloud is an expression used to describe a variety of computing concepts that involve a large number of computers connected through a real time communication network such as the internet. In science, Cloud computing is a synonym for distributed computing over a network and means the ability to run a program on many connected computers at the same time. The phrase is also more commonly used to refer to network-based services which appear to be provided by real server hardware, which in fact are served up by virtual hardware, simulated by software running on one or more real machines[1]. Such virtual servers do not physically exist and can therefore be moved around and scaled up(or down) on the fly without affecting the end user – arguably, rather like a cloud. The popularity of the term can be attributed to its use in marketing to sell hosted services in the sense of application service provisioning that run client server software on a remote location.

According to NIST “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources(e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction “[2].

The cloud model is composed of five essential characteristics, three service models and four deployment models [3].

1.1.1 Essential Characteristics

- **On-Demand self Service**
  A consumer can unilaterally provision computing capabilities such as server time and network storage as needed automatically without requiring human interaction with each service provider.
- **Broad Network Access**
  Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g.-mobile phones, tablets, laptops and workstations).
- **Resource Pooling**
  The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model with different physical and virtual resources dynamically assigned and re-assigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the specify location at a higher level of abstraction (e.g. country, state or datacenter). Examples of resources include storage processing, memory and network bandwidth.
- **Rapid Elasticity**
  Capabilities can be elastically provisioned and released in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- **Measured Service**
  Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g. storage, processing, bandwidth and active user accounts). Resource usage can be monitored, controlled and reported, providing transparency for both the provider and consumer of the utilized service.

1.1.2 Service Models

- **Software as a Service (SaaS)**
  The capability provided to the consumer is to use the providers applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g. web based e-mail), or a program interface. The consumer does not manage or control the underlying clud infrastructure including network servers, operating systems, storage or even individual application capabilities with the

possible exception of limited user specific configuration settings.

- **Platform as a Service (PaaS)**
  The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network servers, operating systems or storage but has control over the deployed applications and possibly configuration settings for the application hosting environment.
- **Infrastructure as a Service**
  The capability provided to the consumer is to provision processing, storage, networks and other fundamental computing resources where the consumer is able to deploy and run arbitrary software which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage and deployed applications and possibly limited control of select networking components (e.g. host firewalls).

1.1.3 Deployment Models

- **Private Cloud**
  The cloud infrastructure is provisioned for exclusive use by single organization comprising multiple consumers (e.g. business units). It may be owned, managed and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
- **Community Cloud**
  The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g. mission, security requirements, policy and compliance considerations). It may be owned, managed and operated by one or more of the organizations in the community, a third party or some combination of them and it may exist on or off premises.
- **Public Cloud**
  The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed and operated by a business, academic or government organization or some combination of them. It exists on the premises of the cloud provider.
- **Hybrid Cloud**
  The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community and public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g. cloud bursting for load balancing between clouds).

1.2 Advantages of Cloud Computing

- Cloud computing do not need high quality equipment for user and easy to use.
- Cloud computing provides dependable and secure data storage center. No need to worry about data loss or virus.
- Cloud computing can realize data sharing between different equipments.
- Cloud provides nearly infinite possibility for users to use internet.

1.3 Disadvantages of Cloud Computing

- The production or service of cloud computing is not stable and believable.
- The most worrying question is the privacy of cloud computing.
- If we use cloud computing without technical layout, that is very dangerous.

2. FAULT –TOLERANCE IN CLOUD COMPUTING

Fault-tolerant computing is the art and science of building computing systems that continue to operate satisfactorily in the presence of faults. A fault tolerant system may be able to tolerate one or more fault types including- transient, intermittent or permanent hardware faults, software and design errors, operator errors, or externally induced upsets or physical damage[4]. An extensive methodology has been developed – most dealing with random hardware faults, while a smaller number deal with software, design and operator faults to varying degrees. A large amount of supporting research has been done and reported. Fig. 1 shows the relationships among faults, error and failure.

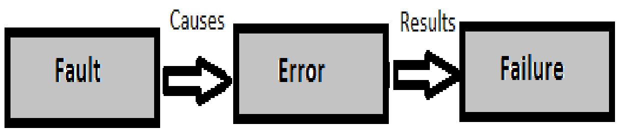

Fig. 1. Generation Path of Failure [5]

Fault tolerance and dependable systems research covers a wide spectrum of applications ranging across embedded real-time systems, commercial transaction systems, transportation systems, and military/space systems. The supporting research includes systems architecture, design techniques, coding theory, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing, and real time processing. These areas often involve widely diverse core expertise ranging from formal logic, mathematics of stochastic modeling, graph theory, hardware design and software engineering.

2.1 Types of Fault-Tolerance

2.1.1 Hardware Fault-Tolerance

The majority of fault-tolerant designs have been directed toward building computers that automatically recover from random faults occurring in hardware components. The techniques employed to do this generally involve partitioning a computing system into modules that act as fault-containment regions. Each module is backed up with protective redundancy so that, if the module fails, other can assume its function[6,7]. Two general approaches to hardware fault recovery have been used: 1) fault masking, and 2) dynamic recovery.

- **Fault Masking**
  Fault masking is a structural redundancy technique that completely masks faults within a set of redundant modules. A number of identical modules execute the same functions, and their outputs are voted to remove errors created by a faulty module. Triple modular redundancy (TMR) is a commonly used form of fault masking in which the circuitry is triplicated and voted. The voting circuitry can also be triplicated so that individual voter failures can also be corrected by the voting process. A TMR system fails whenever two modules in a redundant triplet create errors so that the vote is no longer valid. Hybrid redundancy is an extension of TMR in which the triplicated modules are backed up with additional spares, which are used to replace faulty modules – allowing more faults to be tolerated. Voted systems require more than three times as much hardware as non redundant systems, but they have the advantage that computations can continue without interruption when a fault occurs, allowing existing operating system to be used[7].
- **Dynamic Recovery**
  Dynamic recovery is required when only one copy of computation is running at a time (or in some cases two unchecked copies), and it involves automated self-repair. As in fault masking, the computing system is partitioned into modules backed up by spares as protective redundancy. In the case of dynamic recovery however, special mechanisms are required to detect faults in the modules, switch out a faulty module, switch in a spare, and instigate those software actions (rollback, initialization, retry, restart) necessary to restore and continue the computation. In single computers special hardware is required along with software to do this, while in multicomputer the function is often managed by the other processors[7]. Dynamic recovery is more hardware-efficient than voted systems, and it is therefore the approach of choice in resource-constrained (e.g., low power) systems, and especially in high computing must be maximized. Its advantage is that computational delays occur during fault recovery, fault coverage is often lower, and specialized operating system may be required.

2.1.2 Software Fault-Tolerance

Efforts to attain software that can tolerate software design faults (programming errors) made use of static and dynamic redundancy approaches similar to those used for hardware faults. One such approach, N-version programming, uses static redundancy in the form of independently written programs that perform the same functions, and their outputs are voted at special checkpoints. Another approach called design diversity combines hardware and software fault-tolerance by implementing a fault-tolerant computer system using different hardware and software in redundant channels [7].

3. TECHNIQUES FOR FAULT TOLERANCE

In cloud computing there are two types of fault tolerance:

- Reactive fault tolerance
- Proactive fault tolerance

3.1 Reactive fault tolerance

Reactive fault tolerance means to remove the fault after it occurs. Basically reactive fault tolerance policies reduce the effect of failures on application execution when the failure effectively occurs. There are various techniques which are based on these policies like Checkpoint/Restart, Replay and Retry and so on[5].

- **Checkpointing/Restart**: When a task fails, it is allowed to be restarted from recently checked pointed state rather than from the beginning. It is an efficient task level fault tolerance technique for long running applications.
- **Replication**: Various task replicas are run on different resources, for the execution to succeed till the entire replicated task is not crashed. It can be implemented using tools like Hadoop and AmazonEc2 etc.
- **Job Migration**: During failure of any task, It can be migrated to another machine. This technique can be implemented by using HAProxy.

3.1.1 Techniques used for Reactive Fault-Tolerance are:

- FTM Architecture
- MPI Architecture

3.1.1.1 FTM (Fault Tolerance Manager) Architecture

FTM is built to work on top of hypervisor, spanning all the nodes and transversing the abstraction layers of the Cloud to transparently tolerate failures among the processing nodes[4]. Fig. 2 illustrates the architecture of FTM which can primarily be viewed as an assemblage of several Web service components, each with a specific functionality. A brief description of the functionality of all the components along with the rationale behind their inclusion in the framework is given below.

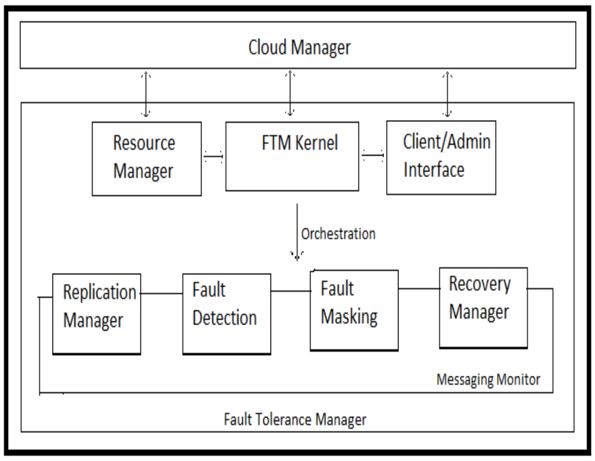

Fig. 2. FTM Architecture

In this architecture Replication Manager replicates users applications such that a redundant copy of the application is available after failure happens. Fault Detection/Prediction Manager is used to detect replica failures. Fault detection services detect the faults immediately after their occurrence and send a notification about the faulty replica to the FTM Kernel to invoke services from the Fault Masking Manager and Recovery Manager. Fault Masking Manager meets high availability demands by recovering or replacing the failed components in the background while the application's execution remains uninterrupted in another instance "replica". A collection of such algorithms that "mask" the occurrence of failures and prevent the faults from resulting into errors is included in this component[4]. Recovery Manager includes all the mechanisms that resumes error prone nodes to a normal operational mode. Messaging Monitor offers the necessary communication infrastructure, It exchange the message among replicas of a replica group and also provide inter-component communication within the framework. Client/Admin interface act as an interface between the end user and FTM. FTMKernel is the central computing component of Fault Tolerance Manager which manages all the reliability mechanisms in the framework. Resource Manager maintains a database which include logging information about the machines in the cloud and provides an abstract**,** simple representation of working state of resources in the form of graph.

3.1.1.2 MPI (Message Passing Interface) Architecture

MPI has become the de facto standard for parallel applications programming[7]. It has a modular, layered architecture which separates the implementation of the high level protocols and functions from the low level mechanisms used for interprocess communication and process management. This is very important because it makes it possible to build a new implementation by rewriting only the functions at the lowest level[9]. The LAM layer provides a framework and run-time environment upon which the MPI layer executes. The MPI layer provides the MPI interface and an infrastructure for direct, process-to-process communication. The MPI library consists of two layers. As shown in Fig.4 the upper layer is independent of the communication sub-system (i.e., MPI function calls and accounting utility functions).

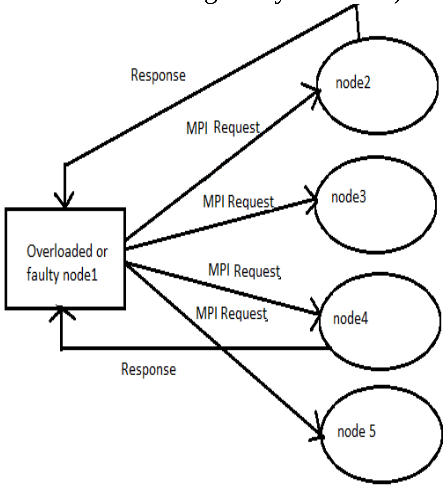

Fig.3 Message Passing Interface

The lower layer consists of a modular framework for collections of Systems Services Interfaces called SSI. One such collection is the MPI Request between the MPI peer processes. Another is the checkpoint/restart (CR), which provides an interface to the back-end checkpointing system that does the actual checkpointing and restart functionality. At the start of execution of an MPI job, the CR SSI determines whether checkpoint/restart support was requested, and if so, the blcr module is selected to run. To support checkpointing, an RPI module must have the ability to generically prepare for checkpoint, continue after checkpoint, and restore from checkpoint. A checkpointable RPI module must therefore provide callback functions to perform this functionality.

3.2 Proactive fault tolerance

It refers to avoiding failures, errors and faults by predicting them in advance. Some of the techniques which are based on these policies are preemptive migration, software rejuvenation etc[5].

- **Proactive Fault Tolerance using self healing**: When multiple instances of an application are running on multiple virtual machines, it automatically handles failure of application instances.
- **Proactive Fault Tolerance using preemptive Migration**: Preemptive Migration relies on a feedback-loop control mechanism where application is constantly monitored and analyzed.

3.2.1 Technique used for proactive fault tolerance is: MapReduce Fault Tolerance

MapReduce has been gaining popularity and it has been used at Google extensively to process 20 petabytes of data per day[8]. Yahoo developed its open-source implementation, Hadoop, which is also used in Facebook for production jobs including data import, hourly reports, etc. As shown in fig.4, in MapReduce, input data is split into a number of blocks, each of which is processed by a map task. The intermediate data files produced by map tasks are shuffled to reduce tasks, which generate final data output.

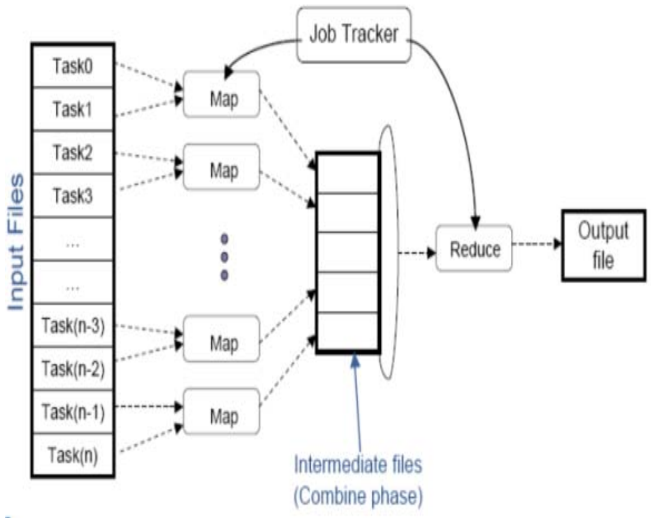

Fig.4 MapReduce Architecture[10]

MapReduce handles failures through re-execution. If a machine fails, MapReduce reruns the failed tasks on other machines. In the effect of a single machine failure on the runtime of a Hadoop job (i.e., a two-stage job) is studied and it is found out to cause a 50% increase in completion time.

4. CONCLUSION AND FUTURE SCOPE

Various different fault-tolerance techniques have been discussed in this paper and concluded that there is need of a more efficient and reliable technique that is also cheaper than the existing techniques. Future research works can explore more on MPI architecture in to present a reliable and less costly technique for fault-tolerance.

REFERENCES

[1] Shuai Zhang, Shufen Zhang, Xuebin Chen, Xuizhen Huo, "Cloud Computing Research and Development Trend" © 2010 IEEE, DOI 10.1109/ICFN.2010.58.

[2] Peter Mell, Timothy Grance, "NIST Definition of Cloud Computing", Sept 2011, National Institute of Standards and technology, Gaithersburg, MD 20899-8930.

[3] Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen and Zhenghu Gong, "The Characterstics of Cloud Computing", © 2010 IEEE, DOI 10.1109/ICPPW.2010.45

[4] Ravi Jhawar, Vincenzo Piuri, Marco Santambrogio, " A Comprehensive Conceptual System-Level Approach to Fault Tolerance in Cloud Computing"© 2012 IEEE, DOI 10.1109/SysCon.2012.6189503.

[5] Amritpal Singh, Supriya Kinger, "An Efficient Fault Tolerance Mechanism Based on Moving Averages Algorithm" © 2013, IJARCSSE, ISSN: 2277 128X.

[6] http://en.wikipedia.org/wiki/Fault-tolerant_computer_system

[7] http://www.cs.ucla.edu/~rennels/article98.pdf

[8] Qin Zheng, "Improving MapReduce Fault Tolerance in the Cloud" © 2010 IEEE.

[9] Ekype Okorafor, "A Fault-tolerant High Performance Cloud strategy for Scientific Computing" ©2011 IEEE, DOI 10.1109/IPDPS.2011.306.

[10] https://www.google.co.in/?gfe_rd=cr&ei=WvM8U5_PL9jBuASr6oHACQ#q=Mapreduce + architecture.