Abstract Model of Fault Tolerance Algorithm in Cloud Computing Communication Networks

Pardeep Kumar,
Deptt. of CSE & ICT,
Jaypee University of Information Technology,
Waknaghat, solan (H.P)- India
Email: pardeepkumarkhokhar@gmail.com

Shiv Kumar Gupta,
Deptt. Of CSE & ICT,
Jaypee University of Information Technology,
Waknaghat, Solan(H.P)-India
Email: shivku2003@gmail.com

Abstract--We discussed in this paper the modeling behavior of the cloud computing communication network through which processors exchange their messages. The goal is to develop an abstract description of a communication network that effectively enforces a TDMA-style access to the communication medium by avoiding that a "babbling idiot" can interfere with another processor's message broadcast. This ensures that a non-faulty processor will receive the message that was sent by the sender of a given slot. To prevent a faulty processor from speaking outside its designated slots an independent component is necessary that controls the access to the communication medium. In TTP/C, this component is called the guardian. There are two common implementations of a guardian: in a bus topology, every processor is equipped with its own local guardian, whereas in a star topology there is one central guardian per channel.

Keywords: broadcasting guardian, non-faulty, TDMA, TTP/C.

I. INTRODUCTION

Cloud Computing can be thought of as a way to make the world of computer resources seamlessly scalable. "Cloud Computing Services" can involve "a software and server framework (usually based on virtualization)" that uses "many servers for a single software-as-a-service style application or to host many such applications on a few servers." Cloud Computing Services are an emerging network architecture where applications reside on third party servers, managed by private firms that provide remote access through web-based devices. Customers generally do not own the infrastructure. This model of service delivery is in contrast to an architecture in which data and applications typically reside on servers or computers within the control of the end user. Such type of model, which aims at offering distributed, virtualized, and elastic resources as utilities to end users, has the potential to support full realization of "computing as a utility" in the near future [1][2]. Along with the advancements of the Cloud technology, new possibilities for Internet-based applications development are emerging. These new application models can be grouped in to two parties: on one side, there are the cloud service providers that are willing to provide large-scale computing infrastructure at a price based primarily on usage patterns. It eliminates the initial high-cost for application developers of environment set up an application deployment. On the other side there are large-scale software systems providers, which develop applications such as social networking sites and e-commerce, which are gaining popularity on the Internet. These applications can benefit greatly of Cloud infrastructure services to minimize costs and improve service quality to end users.

Previously, development of such applications required acquisition of servers with a fixed capacity able to handle the expected application peak demand, installation of the whole software infrastructure of the platform supporting the application, and configuration of the application itself. But the servers were underutilized most of the time because peak traffic occurs only at specific short time periods. With the advent of the Cloud, deployment and hosting became cheaper and easier with the use of pay-per-use, flexible elastic infrastructure services provided by Cloud providers. Figure 1 shows the hardware and software infrastructure for cloud computing. When these two ends are brought together, several factors that impact the net benefit of Cloud can be observed. Some of these factors include geographic distribution of user bases, capabilities of the Internet infrastructure within those geographic areas, dynamic nature of usage patterns of the user bases, and capabilities of Cloud services in terms of adaptation or dynamic reconfiguration, among others. A comprehensive study of the whole problem in the real Internet platform is extremely difficult, because it requires interaction with several computing and network elements that cannot be controlled or managed by application developers. Furthermore, network conditions cannot be predicted nor controlled, and it also impacts quality of strategy evaluation. Study of such dynamic and massively distributed environments in a controlled and reproducible manner can be achieved with the use of simulation. CloudSim [3] allows modeling and simulation of infrastructures containing Data Centers, users, user workloads, and pricing models. It enables modeling and simulation of typical Cloud infrastructures, even though it has been developed without focusing any specific Cloud provider. Cloud Analyst, Figure 2 built on top of CloudSim, allows description of application workloads, including information of geographic location of users generating traffic and location of data centers, number of users and data centers, and number of resources in each data center. Using this information, Cloud Analyst generates information about response time of requests, processing time of requests, and other metrics. By using Cloud Analyst, application developers or designers are able to determine the best strategy for allocation of resources among available data centers, strategies for selecting data centers to serve specific requests, and costs related to such operations.

Cyber Infrastructure: Hardware And Software

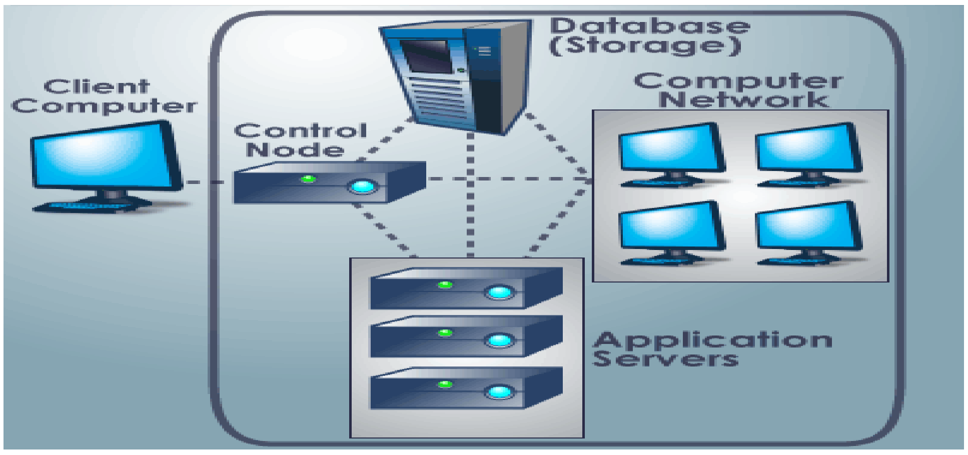

Figure 1. General View of Cloud Computing

Cloud computing is defined as “a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers “One of the most useful features of cloud infrastructures is the ability to automatically scale an infrastructure vertically and horizontally with little or no impact to the applications running in that infrastructure. In truth, useful is an understatement. This feature fundamentally alters IT managers’ relationships to their infrastructures and changes the way finance managers look at IT funding. But the feature is a double-edged sword.

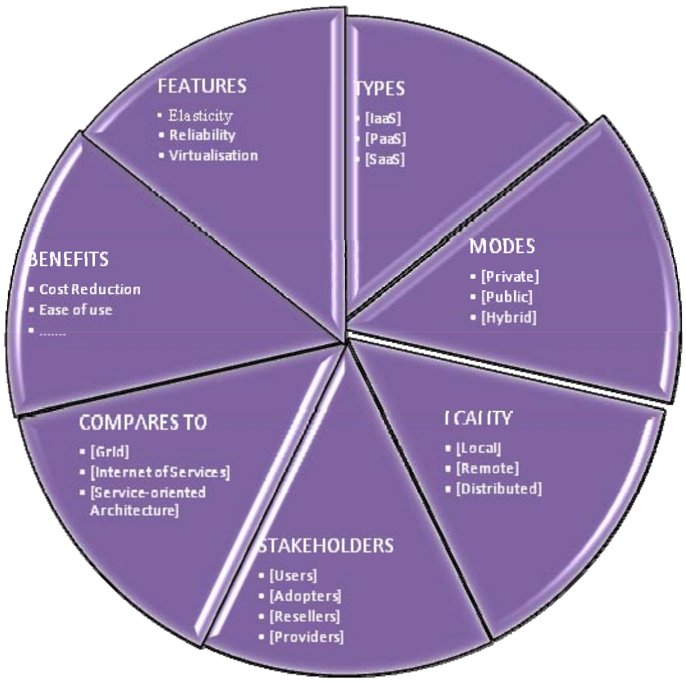

Figure 2. Cloud communication Net

The name cloud computing comes from the fact that in network designs large parts of a network are drawn as a cloud. For example the Internet is most often displayed as clouds. It is displayed like this because I am not sure what is in there, and have no control over it. But when put some data in I get some data out. This is the same with cloud computing, Cloud computing has several layers in which I can operate. The more basic (and most used) layers are: software, platform and infrastructure.

The ability of communication networks to transfer communication at extremely high speed is used to gather information in large volumes nearly instantaneously and, with the aid of computers, to almost immediately exercise action at a distance. These two unique capabilities form the basis for many existing services and an unlimited number of future network –based services. As we know cloud is shower of multi communication task in the multi field like business, IT-sectors, E-commerce, BIO-tasking, life-science, etceteras touching with computing system, so we called cloud computing. The process of formal analysis can roughly be divided into two tasks: building a formal model of the object under study and formulating adequate correctness requirements, and carrying out the formal proofs that the properties hold for the model. As we are interested in analyzing the fault-tolerant algorithms of TTP/C, a central aspect of the work is the development of a formal model for TTP/C. The starting point is the high-level specification document [4] [5] of TTP/C, which consists of English text, tables, graphics, and some formulae. When developing formal models, a major question to be solved is which details should be part of the model, and which should not. On the one hand, any formal model should reflect the real world as closely as possible. On the other hand, if one tries to map every single aspect into the model, one will most likely fail to do any interesting formal analysis, because the many details would impede focusing on the main aspects.

II. ABSTRACT MODEL OF COMMUNICATION

In order to state the requirements that a guardian needs to satisfy as generally as possible, we introduce an abstract model, which does not restrict the type of the guardian used in the communication network. We split the description of the abstract model into two parts. First, we present a model that describes an ideal communication through a single channel that is controlled by a single guardian. The latter is modeled as an abstract, un-interpreted entity for which certain properties are required to hold. From these properties we prove the central fact for communication: provided that the guardian is non-faulty, all non-faulty processors receive the message sent by the sender of a given slot. In a second step the single-channel model is refined to a model that introduces multiple channels for communication in order to provide fault tolerance. In this model, the guardian, too, is replicated such that there is a separate guardian for each channel. For this model we need to introduce further items that model how the messages that are transported through the various channels result in a single, agreed broadcast. We then show that the abstract requirements stated for the single-channel model hold for the multi-channel refinement. This allows to inherit the main communication property – messages sent are received as long as there is no fault – from the single-channel model [3][4].

A. Single-Channel Communication

The message that is transported through the communication network in a given slot *t* is denoted by an abstract function $broadcast(t)$. Furthermore, for the abstract guardian g we write $NF^t(g)$ to denote that the guardian is non-faulty in slot *t*. The main requirement a non-faulty guardian has to fulfill is that the message sent by the sender of the current slot is broadcast through the communication medium. This also means that a guardian cannot actively generate messages itself, but only forward the message of the sending processor.

Requirement 1.1 (Reception of Messages: fault-free case)

Let *p* denote the sender of slot *t*. If no link failure occurs during slot *t*, all non-faulty processors *q* receive the message sent by *p*:

$$rcvd(t,p) = sent(t,p)$$

Requirement 1.2 (Non-faulty Guardian broadcast)
If the guardian *g* is non-faulty then the message that was sent by the sender of the current slot is broadcast:

$$NF^t(g) \Rightarrow broadcast(t) = sent\big(t, sender(t)\big)$$

A consequence of this requirement is that in case the guardian is faulty any message might be broadcast.

Now we can describe what is expected from a receiving processor. In order to distinguish between a faulty processor that exhibits arbitrary behavior, such as not obeying [5] the TDMA scheme, or generating arbitrary messages, from a processor that simply fails to receive a message, we introduce the fault mode of a *receive-faulty* processor. We use NF_r^t to denote the set of processors that are not receive-faulty in slot *t*. We expect that a processor that is not receive-faulty will receive the message that is broadcast.

Assumption 1.1 (Non-faulty receiver)
In slot *t*, a processor that is not receive-faulty will receive the message broadcast:

$$p \in NF_r^t \Rightarrow rcvd(t,p) = broadcast(t)$$

If the above assumption holds, it is easy to see that all processors that are not receive-faulty will receive the message sent by the current sender, provided the guardian is non-faulty. This is the main property that describes [6] the functionality of a non-faulty guardian.

Theorem 1.1 (Correct broadcast non-faulty guardian)
If the guardian is non-faulty then all processors *p* that are not receive-faulty will receive the message sent by the sender of the current slot *t*:

$$NF^t(g) \land p \in NF_r^t \Rightarrow rcvd(t,p) = sent\big(t, sender(t)\big)$$

Proof: Direct consequence of Assumption 1.1 and Requirement 1.2

Note that this property essentially justifies Req. 1.1 introduced in the description of the synchronous system model.
While so far we have only described the desirable situation where no fault occurs; the remainder of the model considers the case when the guardian is faulty. Obviously, in order to achieve the main property that only the current sender is allowed to broadcast in a given slot, we must require [7] that all processors are non-faulty in

case the guardian suffers from a fault. However, the guardian still can fail in such a way that it does not even allow the current sender to send its message. Therefore we state a weaker form of Req. 1.2 and require that either the message of the sender is broadcast, or nothing.

Requirement 1.3 (Non-faulty Processors broadcast)

If all of the processors are non-faulty, then the message that is broadcast will either be the one sent by the current sender, or *null*.

$$(\forall : p \in NF^t) \Longrightarrow broadcast(t) = sent\big(t, sender(t)\big) \quad or \quad broadcast(t) = null$$

In order to provide a reasonable interpretation of message reception we assume that a processor can only receive a message that was also actually broadcast.

Assumption 1.2 (Passive Receiver)

The message received by a processor *p* in slot *t* is either the one that is broadcast or *null*.

$$rcvd(t,p) = broadcast(t) \vee rcvd(t,p) = null$$

Putting the last two properties together we obtain the result that if all processors are non-faulty, then all will either receive [8][9] the message sent by the sender, or nothing.

Theorem 1.2 (Correct broadcast non-faulty processors)

If all the processors are non-faulty then all processors will either receive the message sent by the sender of the current slot or an empty message.

$$(\forall : p \in NF^t) \Longrightarrow rcvd(t,p) = sent\big(t, sender(t)\big) \vee rcvd(t,p) = null$$

Proof: Direct consequence of Assumption 1.2 and Requirement 1.3.

Finally, we want to combine Theorem 1.1 and Theorem 1.2 to yield the main result for the behavior of [10][11] the communication network under the *single fault hypothesis*, which states that a fault does not occur to both the guardian and one of the processors.

Definition 1. (Single Fault)

The communication network is said to satisfy the *single fault hypothesis* if either the guardian is non-faulty, or all processors are non-faulty.

Under the single fault hypothesis the communication network guarantees that all processors will either receive the message sent by the current sender, or nothing.

Theorem 1.3 (Correct broadcast single fault)

If the communication network satisfies the single fault hypothesis then all processors will either receive the message sent by the sender of the current slot or an empty message.

$$g \in NF^t \vee (\forall p : p \in NF^t) \Longrightarrow rcvd(t,p) = sent\big(t, sender(t)\big) \vee rcvd(t,p) = null$$

Proof: Direct consequence of Theorem 1.1 and Theorem 1.2.

B. Multi-Channel Communication

Now we are going to refine the single-channel model and introduce multiple channels, each controlled by a separate guardian. Consequently, *g*(*c*) denotes the guardian of channel *c*, and the function *broadcast*(*t*, *c*) is used to denote the message that is broadcast in channel *c* in slot *t*. [12] [13] TTP/C uses two redundant channels; the model, however, does not restrict the number of channels.

Our goal is to inherit the theorems established for the single-channel model. This can be achieved in two steps. First, we must provide an interpretation for the abstract entities used in the single-channel model. This amounts to describing how the broadcasts of the multiple channels are combined to form a single message broadcast, and what it means for the replicated guardians to be non-faulty. In a second step we then have to show that the requirements stated for the single-channel model are satisfied by the multi-channel refinement.
In order to satisfy Requirement 1.2 of the single-channel model we must assume a similar property of the replicated guardians. Here we require that a non-faulty guardian of a channel *c* enables the message sent by the sender of the current slot to be broadcast via this channel.

Requirement 1.4 (Non-faulty Guardian broadcast)

If the guardian *g*(*c*) of channel *c* is non-faulty, then channel *c* will broadcast the message that was sent by the sender of the current slot:

$$NF^{t}(g(c)) \Rightarrow broadcast(t, c) = sent(t, sender(t))$$

As messages are broadcast through a number of redundant channels we must describe what is meant be *the* message broadcast in a given slot [14][15]. To this end, we need to consider the case where the channels contain different messages, either due to a sending fault of the sender, or due to a faulty processor speaking outside its slot. The following definition captures the notion of a *unique broadcast*: all channels either hold one and the same message, or nothing.

Definition 2 (Unique broadcast)

We call a message *m uniquely broadcast* in slot *t*, denoted $unique_{1}bcast(t$, *m*), if the following two clauses hold:

There is a channel *c* through which *m* is broadcast in slot *t*:

$$\exists c : broadcast(t, c) = m$$

All channels broadcast the same message, or *null*:

$$\forall c_1, c_2, m_1, m_2 : broadcast(t, c_1) = m_1 \wedge broadcast(t, c_2) = m_2 \\ \Rightarrow m_1 = m_2 \vee m_1 = null \vee m_2 = null$$

As a simple consequence of this definition we see that if the guardians of all channels are non-faulty then the message sent by the sender is uniquely broadcast.

Corollary 1.1 (Non-Faulty guardian unique broadcast)

If the guardians of all channels are non-faulty, then the message sent by the sender in slot *t* will be uniquely broadcast.

$$\left(\forall c : NF^{t}(g(c))\right) \Rightarrow unique_{bcast\left(t, sent\left(t, sender(t)\right)\right)}$$

Proof: Simple consequence of Req. 1.4: as the guardians are non-faulty, the message sent by the sender is broadcast on all channels and hence is uniquely broadcast.

The above result indicates how the replicated channels and guardians can be combined to form an ideal, single-channel model [16][17]. First, we define that a message *m* is the broadcast of a slot *t*, denoted *broadcast*(*t*) = *m*, if *m* is uniquely broadcast in slot *t*. Second, the replicated guardians can be regarded as a single component, which is considered non-faulty if all of its constituents are.

Definition 3. (Single-Channel Instantiation)

The multi-channel model can be regarded as a single-channel system as follows:

The broadcast in a slot *t* is that message *m* that is uniquely broadcast:

$$broadcast(t) = \; m \; \Leftrightarrow unique_{bcast(t,m)}$$

The system of guardians *g* is considered non-faulty, if the guardians of all channels are:

$$NF^t(g) \Leftrightarrow \left(\forall c : \; NF^t\big(g(c)\big)\right)$$

Given these definitions it is straightforward to show that Requirement 1.2 of the single-channel model holds: From Corollary 1.1 we have

$$\forall c : \quad [\![NF]\!]\,^{\uparrow}t\;(g(c))) \;\Longrightarrow unique_{\downarrow}bcast(t, sent(t, sender(t)))$$

This translates using *Defn.* 3. to

$$NF^t(g) \Rightarrow broadcast(t) = \; sent\big(t, sender(t)\big)$$

In a similar way one can prove the other requirement, Req. 1.3 of the single-channel model, if we can assume that in case of only non-faulty processors each channel either broadcasts the message sent by the current sender, or nothing.

Requirement 1.5 (Non-faulty Processors broadcast)

If all of the processors are non-faulty, then the message that is broadcast through a channel *c* will either be the one sent by the current sender, or *null*.

$$(\forall p : p \in \; NF^t) \Longrightarrow broadcat(t, c) = \; sent\big(t, sender(t)\big) \; or \; broadcast(t, c) = \; null$$

Having established its requirements, we can inherit Theorems 1.1, 1.2, and 1.3 of the single-channel model for the multi-channel model.

CONCLUSION

It is easy to see that all processors that are not receive-faulty will receive the message sent by the current sender, provided the guardian is non-faulty. This is the main property that describes the functionality of a non-faulty guardian. We have only described the desirable situation where no fault occurs; the remainder of the model considers the case when the guardian is faulty. Obviously, in order to achieve the main property that only the current sender is allowed to broadcast in a given slot, we must require that all processors are non-faulty in case the guardian suffers from a fault. If we can assume that in case of only non-faulty processors each channel either broadcasts the message sent by the current sender, or nothing.

REFERENCES

[1] Weiss, “Computing in the Clouds,” netWorker, vol. 11, pp. 16-25, Dec. 2007.
[2] Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). Advances in Information Technologies: The Business Challenge, IOS Press, 1997.
[3] TTTech. Time-Triggered Protocol TTP/C High-Level Specification Document. Available on request at http://www.tttech.com/technology/specrequest.html, 2002.

[4] R. Buyya, R. Ranjan, and R. N. Calheiros, “Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities,” Proc. of the 7th High Performance Computing and Simulation Conference (HPCS 09), IEEE Computer Society, June 2009.

[5] M. Kaashoek and A. Tanenbaum. “Group Communication in the Amoeba Distributed Operating System”. In Proc. of the 11th Intl. Conf. on Distributed Computing Systems, pp. 222–230, May 1991.

[6] Rakesh kumar katare and Shiv Kumar Gupta “Realization through ABFT in Cloud computing” on International Conference on Challenges of Globalization: Strategies for Competitiveness, 2011.

[7] Pardeep Kumar and Shiv Kumar Gupta **“**PBX Design Over Integrated Services Digital Network Interface” IJARCS, Volume 2, No. 2, ISSN No. 0976-5697 , Mar-Apr 2011.

[8] P. Verissimo, A. Casimiro, and L. Rodrigues. “Cesiumspray: A Precise and Accurate Global Time Service for Large-Scale Systems”. Journal of Real-Time Systems, 12(3):243–294, 1997.

[9] W. Steiner and M. Paulitsch. “The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault- Tolerant Systems”. Intl. Conf. on Distributed Computing Systems 2002, pp 329–336, July 2002.

[10] Powell. Group Communication – Introduction. Communications of the ACM, 39(4), April 1996.

[11] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership Algorithms for Multicast Communication Groups. In Proc. of the 6th Intl. Workshop on Distributed Algorithms (WDAG-6), volume 647 of Lecture Notes in Computer Science, pp. 292–312. Springer-Verlag, November 1992.

[12] Hoare. Communicating Sequential Processes. Prentice Hall, 1985.

[13] W. Jia, J. Kaiser, and E. Nett. RMP: Fault-Tolerant Group Communication. IEEE Micro, 16(2):59–67, April 1996.

[14] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, Network Flows, Prentice Hall, Inc by 1993.

[15] R. Ramaswami and K. K. Parhi, “Distributed Scheduling of Broadcasts in a Radio Network, INFOCOM 1989”, pp. 497- 504, April 1989.

[16] W. Zirwas et. al., “Broadband multi hop networks with reduced protocol overhead,” European Wireless Conference, 2002.

[17] H. Kopetz and G. Bauer. “The Time-Triggered Architecture”. Proc. of the IEEE, Special Issue on Modeling and Design of Embedded Software, October 2001.

AUTHORS PROFILE

Pardeep Kumar did his B.Tech degree from Kurukshetra University in 2004 and M.Tech degree from Guru Jambheshwar University of Science & Technology, Hisaar, Haryana in 2007. He has been associated with Mody Institute of Technology & Science, Sikar (Raj.)-India for one year. Currently he is working with Jaypee University of Information Technology, Solan (H.P)-India. He is a member of International Association of Engineers(IAENG) and its societies of Data Mining and Computer Science. His area of research is related to Knowledge Discovery in Databases(KDD) and Machine learning.

Shiv Kumar Gupta has done M. Sc (Mathematics), M.C.A., M. Phil (CS). He started his career as a technical staff dept. of computer science and egineering and information communication technology from Jaypee University of Information Technology, Waknaghat, Solan. Shiv Kumar Gupt**a** is life membership of “MATERIALS RESEARCH SOCIETY OF INDIA”. His areas of research interests are cloud computing, algorithm based fault tolerance ,parallel computing, data minig and communication network reliability.