CHAPTER FIVE: CASE STUDY

5.1 Distributed System

A distributed system is a collection of independent computer that appear to its user as a single coherent system.

This definition has important aspects. The first one is that a distributed system consists of components that are autonomous. A second aspect is that user thinks they are dealing with a single system. This mean one way or the other the autonomous components need to collaborate how to establish this collaboration lies at the heart of developing distributed system.

Concerning the characteristics of distributed system, one important characteristic is that differences between that various computer and the ways is which they communicate are mostly hidden from user. The same holds for the internal organization of the distributed system. Another important characteristic is that user and application can interact with a distributed system in consistent and uniform way, regardless of where and when interaction take place.

In principle, distributed system should also be relatively easy to expand or scale. And a distributed system will normally be continuously available, although perhaps some part may be temporally out of order. User and applications should not notice that part are being replaced or fixed In order to support heterogeneous computers and networks while offering a single system view, distributed system are often organized by means of layer of software-that is , logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating systems and basic communication facilities .

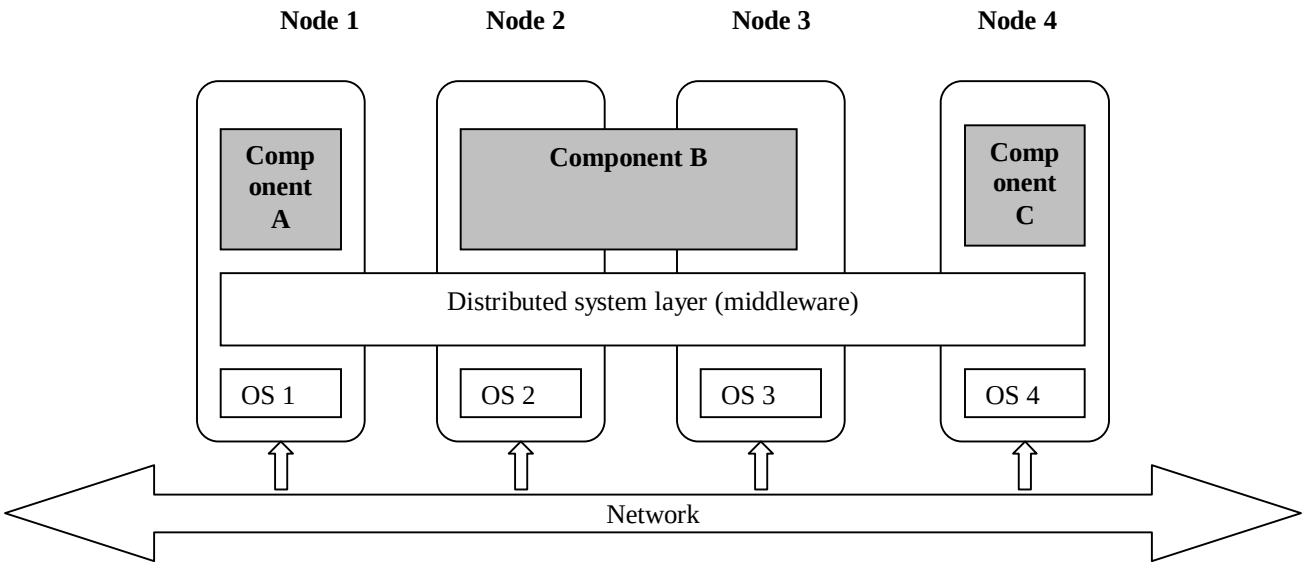

Figure 5-1: The Distributed Systems

5.2 System Architecture

In term of architecture, distributed system are often complex pieces of software of which the components are by definition dispersed across multiple machines. The organization of distributed system is mostly about the software components that constitute the system. Theses software architecture tells us how the various software component are to be organized and how they should interact. The actual realization of a distributed system requires that we instantiate and place software components on real machines. There many different choices that ca be made in doing so. The final instantiation of software architecture is also referred to as system architecture. Software architecture is considering where software component are placed. Deciding on software component, their interaction and their placement leads to an instance of software architecture.

The centralized architecture is also known as client and server system, the basic of client server model, processes in a distributed system are divided into two groups. A server is a process implementing a specific service, for example, a file system service. A client is a process that requests a service from a server by sending it a request and

subsequently waiting for the server's reply. This client-server interaction, also known as request-reply behavior, the diagram explains that:

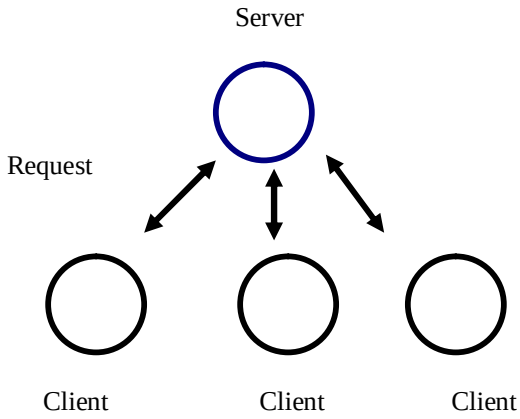

Figure 5-2: Centralized Architecture

5.3 Component-based Software

Component based software engineering (CBSE) promises enhancements in software development due to separation of concern, component reuse, and other component oriented methodologies. Individual reusable components are plugged together via interfaces, which are used for component interaction. Several industrial grade server component architectures have been defined, such as Sun's Enterprise JavaBeans (EJB), Microsoft's COM+ and the CORBA Component Model (CCM) (Schmo et al, 2007). Szyperski defines a component as: ''A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only" (Szyperskiet al, 2002) that means to be able to compose components into applications, each component must provide one or more interfaces. These interfaces form a contract between the component and its environment. The component interfaces clearly define which services a component provides. Also, software usually depends on a specific context, such as particular database schemas or other system resources. To support the composability of components, such dependencies must be explicitly specified (Schmo, 2007).

5.4 Case Study

As a case study we utilize two type of network, the first is peer to pee and the second is cluster network. We map our model into peer to peer form, peer to peer is a kind of network that each node has its own capabilities to act is a kind of decentralize network, cluster network is a group of network that has an specific node controlling the activity.

5.5 Peer to Peer

A peer-to-peer (or P2P) computer network uses diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application. P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files (see file sharing) containing audio, video, data or anything in digital format is very common, and real time data, such as telephony traffic, is also passed using P2P technology.

A pure P2P network does not have the notion of clients or servers but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. A typical example of a file transfer that is not P2P is an FTP server where the client and server programs are quite distinct: the clients initiate the download/uploads, and the servers react to and satisfy these requests.

Peer-to-peer (P2P) systems have become very popular of late, and are widely used for sharing resources, such as music files, software components. Software component reliability and availability is a crucial operation in distributed component based system; and there has been considerable recent work in devising effective algorithm to answer these problems.

Let's say that we have component in each of the node these nodes are connected in peer to peer structure see figure below

5.6 Distributed Architecture

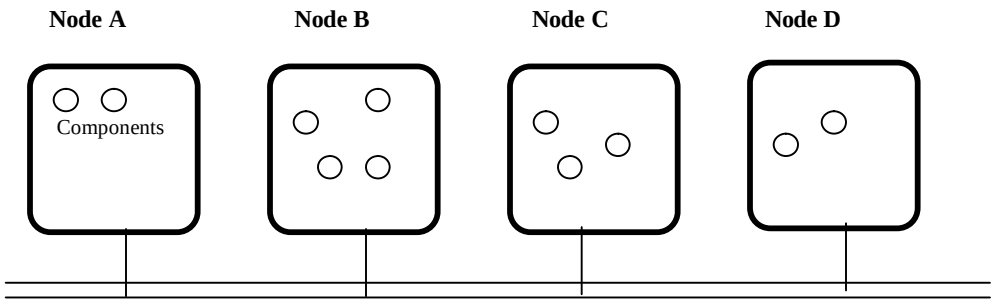

Figure 5-3: Distributed Architecture

By adding our model to the above structure, each node is have the model component which is the agent the detection agent of the monitoring agent, the replication and configuration agent, the prevention agent, the runtime management agent.
The system will resemble the figure 5-4.

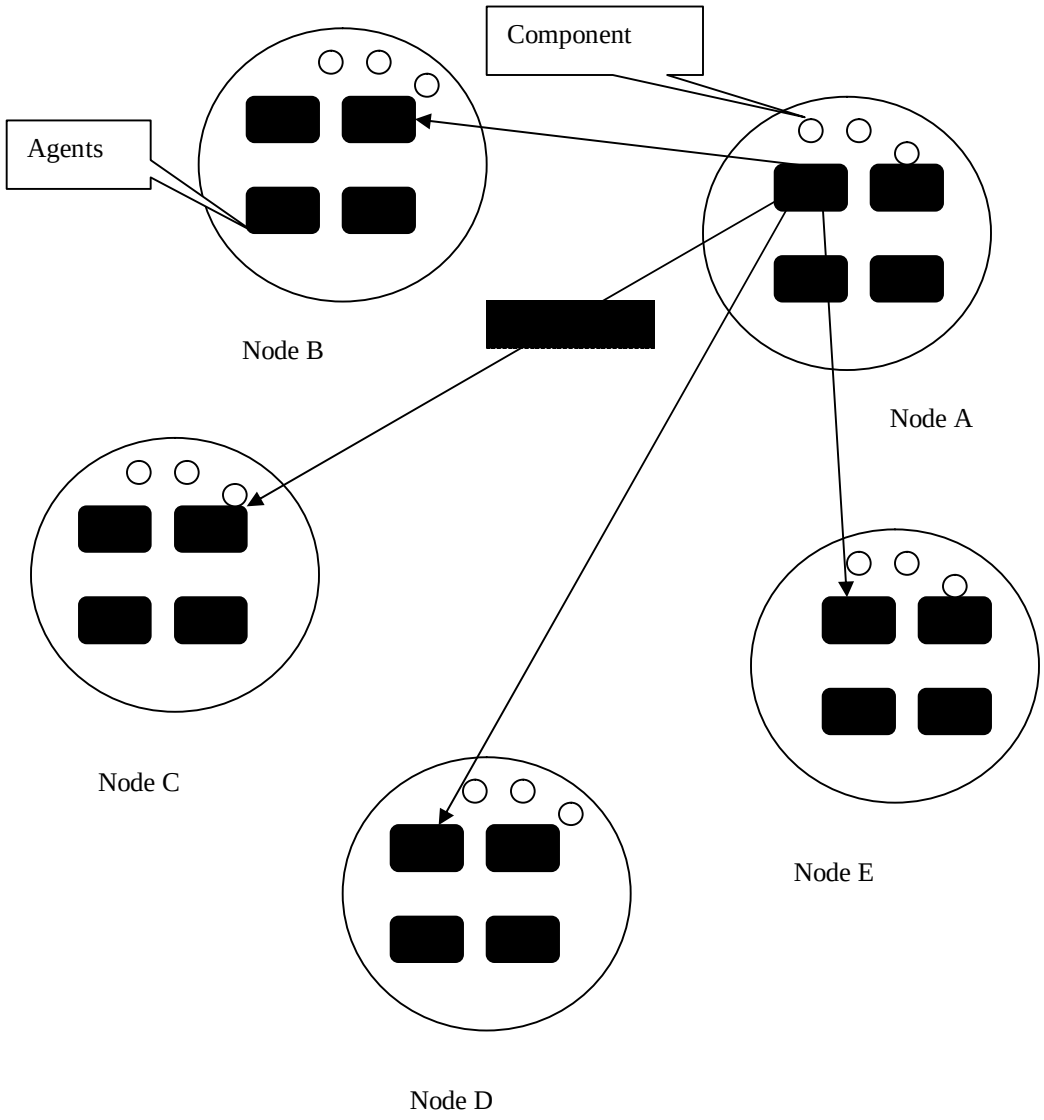

Figure 5-4: Mapping our Approaches to Peer to Peer

In case of any attack to the software component of the network, the monitor or the detector agent will detect any malicious activity; the detection will be as shown in chapter three. After the detection the system would react and accomplish the regeneration part in order to protect the software component.

First the agent need locate component in an optimum way, the after localization the node send request in order to copy the component.

C_B means that al components of node B

C_A means all component of node A

$C_B = C_{B1} + C_{B2} + C_{B3} + \ldots C_{Bn}$

$C_A = C_{A1} + C_{A2} + C_{A3} + \ldots C_{An}$

If attack happens to node A and try to harm component C_{A1} the agent would interact and send

A *sends* C_{A1} to B

B receive C_{A1} from A

After the regeneration the component would resend back to A

B *sends* C_{A1} to A

A receive C_{A1} from B

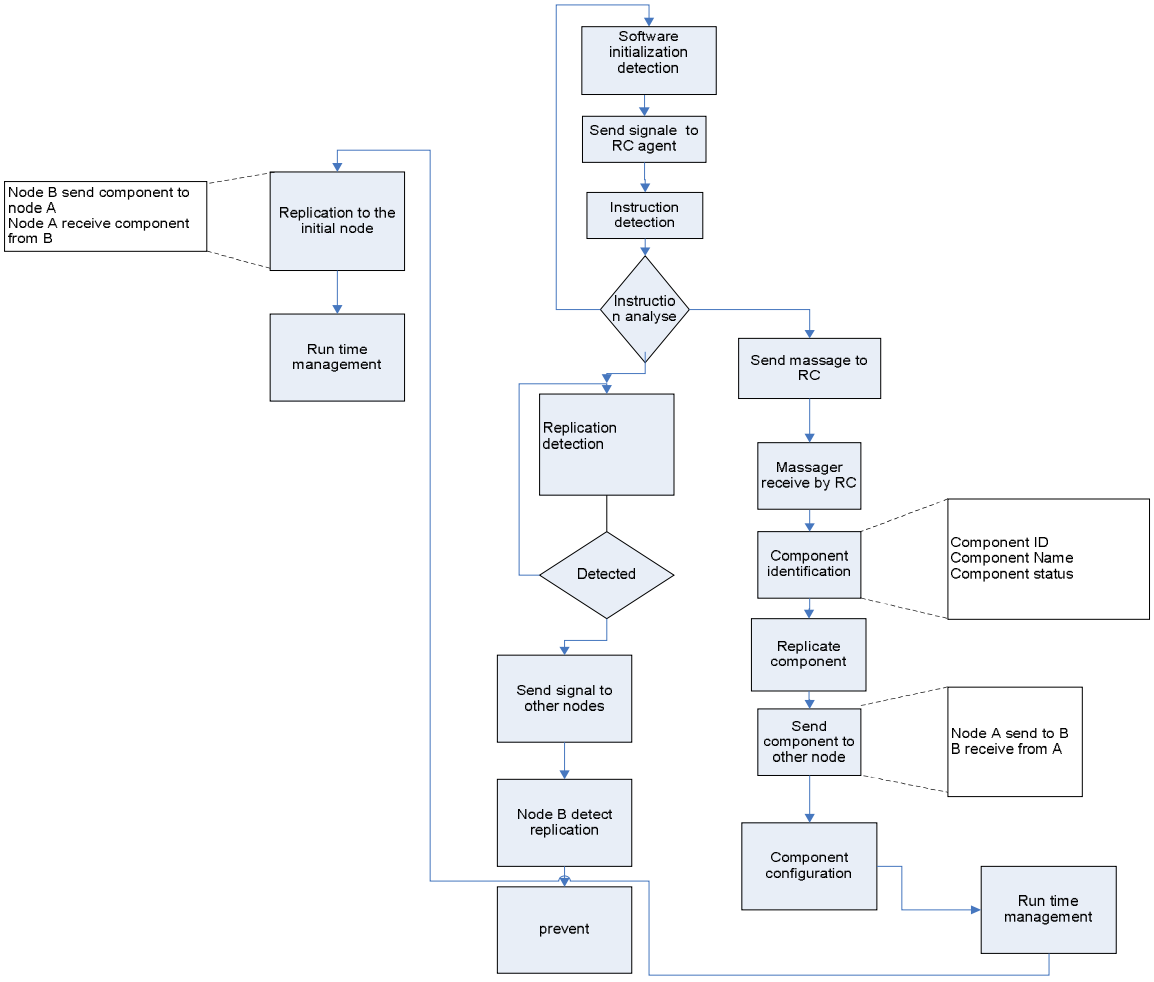

Figure 5-5: Model flow

We present the experiment in the peer to peer model, this experiment show how to apply our approach to self regenerate in the different four scenarios.

The result show the interaction of agent in the regeneration in four scenario vs the number of node and number of message needed to accomplish a whole system function. As the number of nodes increases the agent interaction messages increase as well, in addition the time of the regeneration decrease as the number of node increase. The figure below depicts the scenario number one in an ACL message vs number of nodes.

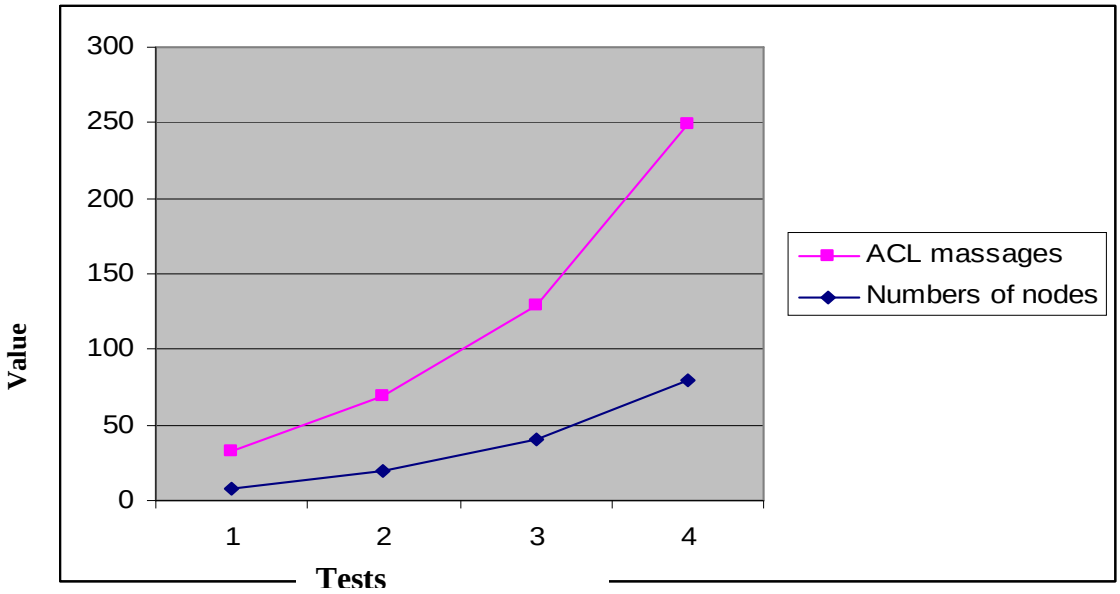

Figure 5-6: The ACL messages VS numbers of nodes

Comparatively in the four scenarios we test certain number of node vs ACL message interaction in the four scenarios.

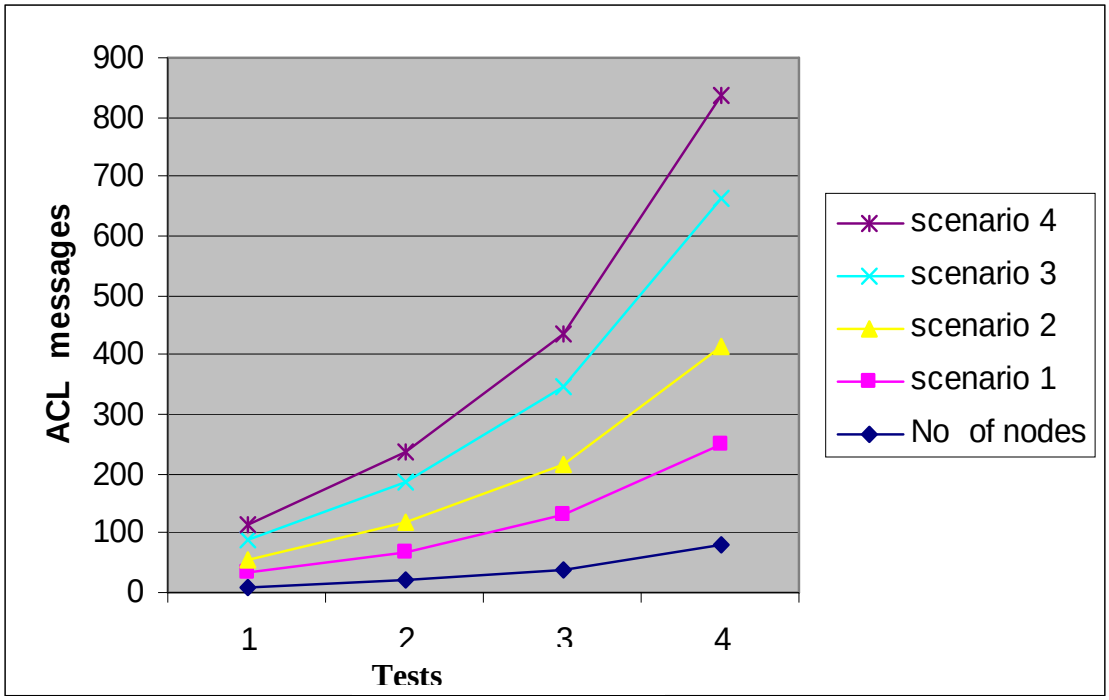

Figure 5-7: The ACL messages VS numbers of nodes in the four scenarios

5.5 Cluster Network

A computer cluster is a group of linked computers, working together closely so that in many respects they form a single computer. The components of a cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve performance and/or availability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or availability (Bader et al)

In this section we describe the component-based systems software that has actually been deployed on our cluster. It consists of an infrastructure for component interaction, as show in the figure below

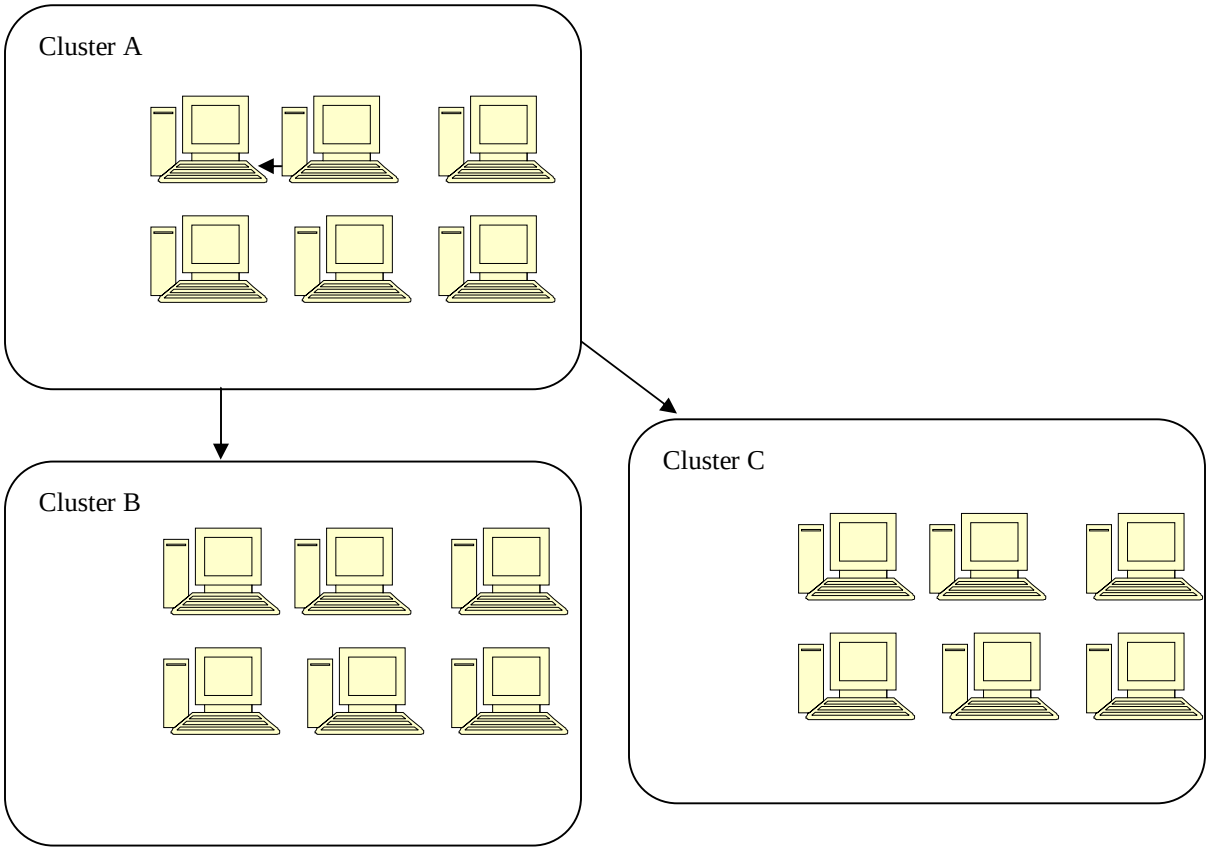

Figure 5-7: The Cluster System deployment

As we can see in cluster system we can say the interaction of agents in the cluster system is less than the one in the peer to peer system according to the figure below.

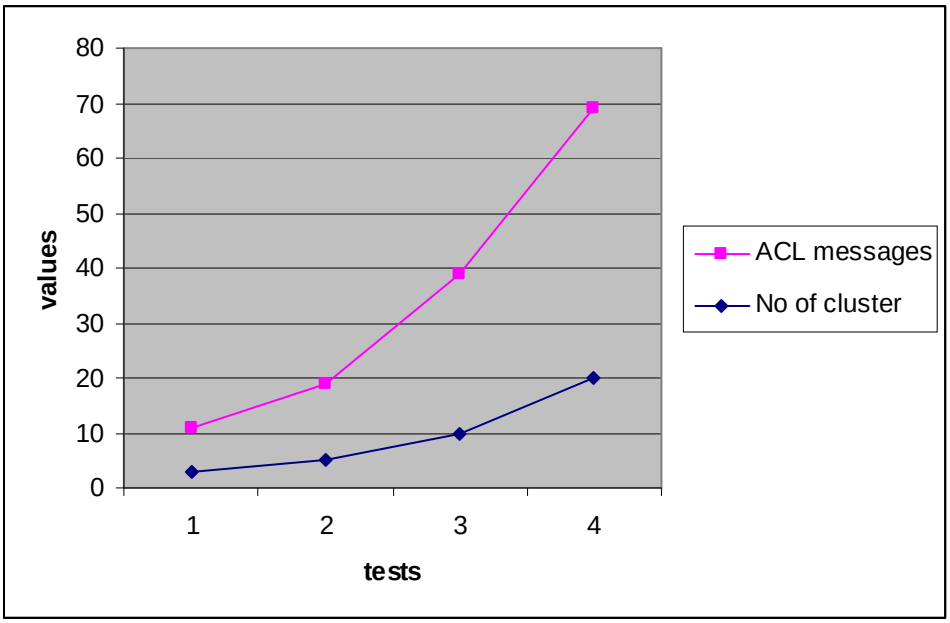

Figure 5-7: The ACL Messages VS Numbers of Nodes in the four Scenarios

5.6 Summary

Throughout this chapter we consider two key case studies, the first is using the peer to peer system and the second is in a cluster environment. Component based in a peer to peer environment are getting significant attention at the present time, therefore the availability of the component is very essential. Our approach is tested in the peer to peer environment to see how it vigorously self regenerate the component. The second case study is the cluster system, cluster network now is used for high bandwidth and latency and also used for high availability, and we show our model in a cluster environment and present considerable results.